Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 9

Tipo de Dato Estructurado Homogéneo: Arreglos

Tipos de datos

Retomando lo que hemos visto sobre tipos de datos al comienzo del curso, podemos recordar que los algoritmos generalmente operan sobre datos de distinta naturaleza (números, letras, palabras, símbolos, etc.) y por lo tanto, los programas que implementan dichos algoritmos necesitan representarlos de alguna manera.

De allí que definimos a un *tipo de dato* como *una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos*.

Los tipos de datos se caracterizan por:

- Un rango de valores posibles,
- Un conjunto de operaciones realizables sobre ese tipo
- Una representación interna.

Por ello se dice que un tipo es el **conjunto de valores posibles que puede tomar una variable**.

Tipos de datos
Simples y Estructurados

La solución de cualquier problema informático requiere conocer y saber utilizar herramientas y técnicas tales como la algorítmica y la estructura de datos.

Por medio de la algorítmica podemos diseñar una serie de operaciones que ejecutadas en el orden establecido llevarán a la solución del problema. Por otra parte, es necesario definir las características de los datos con los que trabajemos y una serie de operaciones que manipulen estos datos de forma eficiente, para que el algoritmo que opera sobre ellos pueda ser analizado y llevado a la práctica de una forma clara.

Para facilitar la manipulación de los distintos datos se define por un lado el concepto de **tipo de datos** y por otro el de **estructura de datos**.

Tipos de datos
Simples y Estructurados

Como vimos, un **tipo de datos** es un conjunto de valores posibles junto con unas operaciones para su manipulación.

Tipos de datos como los enteros, reales y caracteres se denominan simples o elementales ya que toman valores atómicos. Esto significa que una variable de estos tipos tan sólo puede contener un valor en cada momento.

Sin embargo, existen otros tipos de datos que pueden contener más de un valor y que se denominan **tipos compuestos**, **estructurados**, **estructuras de datos** o simplemente **estructuras**.

Tipos de datos
Simples y Estructurados

Una **estructura de datos** es un tipo de datos con las siguientes características:

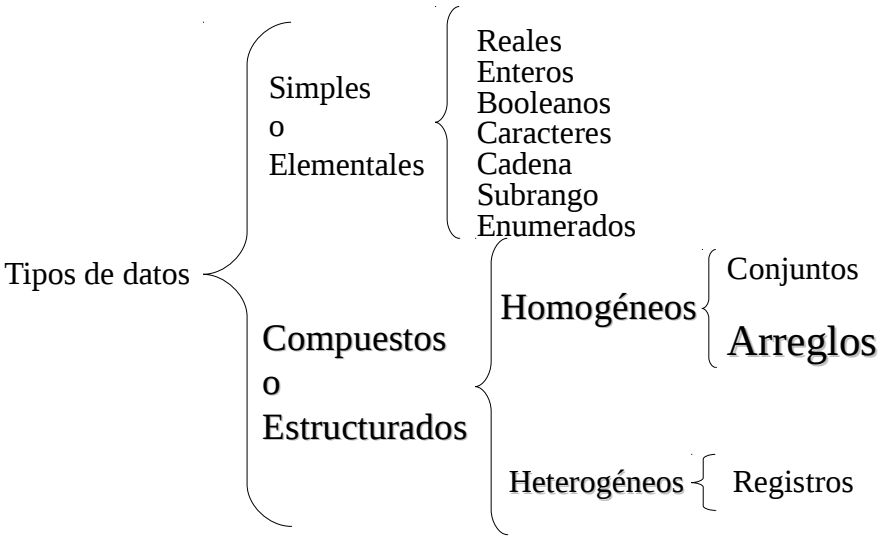
- posee varias componentes, cada una de las cuales puede ser un tipo simple u otra estructura de datos;
- existe una relación entre los distintos componentes que la dota de un cierto significado y utilidad;
- se manipulan mediante una serie de operaciones.

Tipos de datos
Simples y Estructurados

Los objetos del mundo real generalmente son complejos. Por ello para poder representar esos objetos en programas que solucionen problemas se necesita de herramientas que faciliten dicha tarea.

Es por ello existen tipos de datos **Simples**, o *Elementales*, y **Estructurados**, o *Compuestos*. Estos tipos de datos permiten representar en una computadora las características principales de los objetos del mundo real.

Tipos de datos



Arreglos - Motivación

- Queremos almacenar las notas de todos los alumnos de un curso de 150 alumnos, para luego preguntar la nota más alta del curso, la más baja, el promedio, etc...
- ¿Cómo hacemos esto?

Una primera alternativa es declarar una variable por cada alumno.

Pero hay 150 alumnos!!!

Arreglos - Motivación

Otro problema: aunque nos diéramos el tiempo para declarar 150 variables, ¿cómo podríamos entregar la nota a un usuario que lo solicita? Por ejemplo, un docente quiere la nota del Alumno Juarez. ¿Cómo sabemos cual es la nota de Juarez?

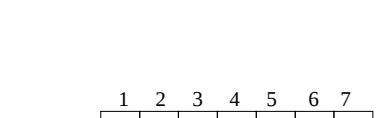
¿Y si se agrega un alumno más al curso?¿Alcanzan las 150 variables?

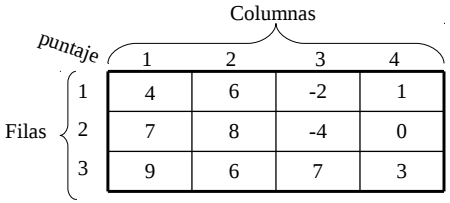
¿Qué podemos hacer?

Como todas las variables son del mismo tipo, necesitaríamos algo que nos permita "contener" una serie de variables del mismo tipo. ¿Existe eso?

Arreglos

Definición: un arreglo es una colección **finita**, **homogénea** y "**ordenada"** de elementos.





Tiene dos partes importantes: las ***componentes*** y los ***índices***.

Los ***componentes*** hacen referencia a los elementos y los ***índices*** a la posición donde se encuentra el elemento.

Para poder referirnos a los datos dentro de una posición del arreglo, se especifica el nombre del arreglo y el número de posición del elemento. Las posiciones generalmente se cuentan a partir del uno o del cero como primera posición (en Pascal es desde 1, C y Java desde 0).

Arreglos

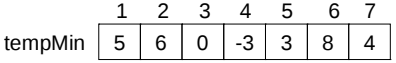
- Físicamente, los arreglos son un grupo de posiciones contiguas en memoria relacionadas entre sí por el hecho de que todas tienen el mismo nombre y los datos que contiene son todos del mismo tipo.

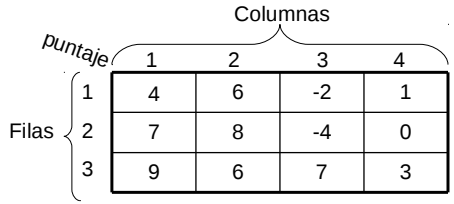
- Son entidades estáticas ya que conservan el mismo tamaño durante toda la ejecución del programa.

Arreglos

Los tipos de arreglos más utilizados son

Unidimensionales Bidimensionales



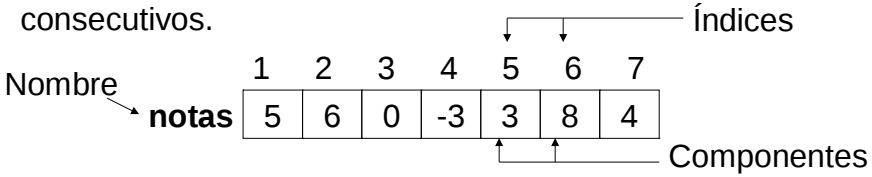


También hay Tridimensionales. Algunos lenguajes permiten más dimensiones aún.

Arreglos Unidimensionales

La estructura más simple es el arreglo de una dimensión, también llamado *vector*.

El arreglo está formado por una sucesión de elementos consecutivos.



Para referirse a una componente de un vector se utilizará el nombre y un *índice*.

El índice se encierra entre corchetes ([índice]).

Ejemplo: **notas[5]**

Declaración de arreglos

Para declarar un arreglo se debe, preferentemente, declarar un nuevo tipo previamente.
Se le da un nombre al tipo, y además se define su tamaño y el tipo de datos (caracteres, enteros, etc.) que van a contener los arreglos que se definan en el futuro.
Luego se declaran todas las variables del tipo arreglo que sean necesarias.

Notación algorítmica

nombreTipo = **arreglo** [límiteInferior..límiteSuperior] **de** tipo //tipo
nombreVar ∈ nombreTipo //variable

Ejemplo:
TElem = Z //tipo
NMax = 7 //constante
TArre = **arreglo** [1..NMax] **de** TElem //tipo
notas ∈ TArre //variable

Declaración de arreglos

Pascal

TYPE nombreTipo = **ARRAY** [límiteInferior..límiteSuperior] **OF** tipo;
VAR nombre: nombreTipo;

El máximo en Pascal es 256 componentes

Ejemplo:

CONST NMax = 7;
TYPE
TElem = INTEGER;
TArre = **ARRAY** [1.. NMax] **OF** TElem;
VAR notas: TArre;

Declaración de arreglos

En el caso de ser necesario saber la cantidad de elementos que contiene el arreglo, debe definirse una variable adicional para almacenar dicha cantidad.

Notación algorítmica

nombreTipo = **arreglo** [límiteInferior..límiteSuperior] **de** tipo //tipo
nombreVar ∈ nombreTipo //variable
nombreCant ∈ [límiteInferior-1..límiteSuperior+1]

Ejemplo:
TElem = Z //tipo
NMax = 7 //constante
TArre = **arreglo** [1..NMax] **de** TElem //tipo
notas ∈ TArre //variable
cant ∈ [0..NMax+1]

Declaración de arreglos

Pascal

TYPE nombreTipo = **ARRAY** [límiteInferior..límiteSuperior] **OF** tipo;
VAR
nombre: nombreTipo;
nombreCant: [límiteInferior-1..límiteSuperior+1]

Ejemplo:
CONST NMax = 7;
TYPE
TElem = INTEGER;
TArre = **ARRAY** [1.. NMax] **OF** TElem;
VAR
notas: TArre;
cant: [0..NMax+1]

Declaración de arreglos

La propuesta anterior se puede mejorar utilizando un registro para contener el arreglo junto a la cantidad elementos que contiene.

Notación algorítmica

nombreTipo = **arreglo** [límiteInferior..límiteSuperior] **de** tipo //tipo
nombreRegistro = <campoArreglo ∈ nombreTipo, nombreCant ∈ [límiteInferior-1..límiteSuperior+1]>
nombreVar ∈ nombreRegistro //variable

Ejemplo:
TElem = Z //tipo
NMax = 7 //constante
TArre = **arreglo** [1..NMax] **de** TElem //tipo
TData = <a ∈ TArre, cant ∈ [0..NMax+1]>
notas ∈ TData //variable

Declaración de arreglos

Pascal

TYPE nombreTipo = **ARRAY** [límiteInferior..límiteSuperior] **OF** tipo;
VAR
nombre: nombreTipo;
nombreCant: [límiteInferior-1..límiteSuperior+1]

Ejemplo:
CONST NMax = 7;
TYPE
TElem = INTEGER;
TArre = **ARRAY** [1.. NMax] **OF** Telem;
TData = **RECORD** a: TArre; cant: [0..NMax+1] **END**;
VAR
notas: TData;
cant: [0..NMax+1]

Declaración de arreglos

Otra forma, que no es recomendable debido a que obstaculiza la modularización (el pasaje de parámetros, por ejemplo).

Notación algorítmica

Nombre ∈ **arreglo** [límiteInferior..límiteSuperior] **de** tipo

Ejemplo:
notas ∈ **arreglo** [1..7] **de** Z

Pascal

VAR nombre: **ARRAY** [límiteInferior..límiteSuperior] **OF** tipo;

Ejemplo:
VAR a: **ARRAY** [1..20] **OF** INTEGER;

Asignación, Carga y Consulta

- **Asignación**

{ei: A=[x_1, x_2, ..., x_i, ..., x_{n-1}, x_n]}

A**[i]** ← e

{ef: A=[x_1, x_2, ..., e_0, ..., x_{n-1}, x_n]}

- **Asignación desde entrada estándar**

{ei : A=[x_1, x_2, ..., x_i, ..., x_{n-1}, x_n]}

leer(e)

A**[i]** ← e

O leer(A**[i]**)

{ef: A=[x_1, x_2, ..., e_0, ..., x_{n-1}, x_n]}

- **Consulta**

{ei: A=[x_1, x_2, ..., x_i, ..., x_{n-1}, x_n] }

A**[i]**

{ef: A=[x_1, x_2, ..., x_i, ..., x_{n-1}, x_n] }

Ejemplos

A[4] ← 29

A[6] ← A[5] + 1

A[i] ← A[4] + A[6]

A[i] ← A[i+1] + A[i-1]

A[i+1] ← 45 + A[6]

A[i-1] ← A[i] + 3

Ejemplo I

Desarrollar un algoritmo que permita al usuario ingresar 10 letras en un arreglo.

```
Algoritmo MisLetras
Léxico
  Max = 10
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  letras ∈ TLetras
  cant ∈ [0..Max+1]
  indice ∈ Z
Inicio

Fin
```


Ejemplo I

Desarrollar un algoritmo que permita al usuario ingresar 10 letras en un arreglo.

```
Algoritmo MisLetras
Léxico
  Max = 10
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  TData = <a ∈ TLetras, cant ∈ [0..Max+1]>
  letras ∈ TData
  indice ∈ Z
Inicio

Fin
```


Ejemplo II

Desarrollar un algoritmo que permita al usuario ingresar letras en un arreglo, no más de 100.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  TData = <a ∈ TLetras, cant ∈ [0..Max-1]>
  letras ∈ TData
  indice ∈ Z
Inicio

Fin
```

Ejemplo III

Desarrollar un algoritmo que permita al usuario ingresar 100 letras, como máximo, en un arreglo y luego mostrar todo el contenido del mismo en el mismo orden en que fueron ingresados.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  TData = <a ∈ TLetras, cant ∈ [0..Max-1]>
  letras ∈ TData
  indice ∈ Z
Inicio

Fin
```


Ejemplo IV

Desarrollar un algoritmo que permita al usuario ingresar hasta 200 letras en un arreglo y luego mostrar todo el contenido del mismo en el orden inverso al que fueron ingresados.

Ejemplo V

Desarrollar un algoritmo que permita al usuario cargar en un arreglo de hasta 255 caracteres, a lo sumo 10 vocales, luego informar la cantidad de vocales ingresadas en el arreglo y finalmente cambiar cada vocal por la letra que le sigue en el abecedario (por ejemplo, en lugar de a o A debe colocar b o B).

```
Algoritmo Encriptación
```


Ejemplo VI

Desarrollar un algoritmo que permita al usuario cargar en un arreglo de caracteres a lo sumo 80 vocales y luego informar la vocal que más veces se repite.

Aclaración: independientemente de la forma en que el usuario ingrese las vocales, en el arreglo deben ser almacenadas en mayúscula.

Ejemplo VII

Desarrollar un algoritmo que permita al usuario cargar un arreglo con 30 números enteros, en un arreglo para 120 números, y luego visualizarlos en orden inverso.

```
Algoritmo OrdenInverso
Léxico



Inicio



Fin
```


Ejemplo VII

Desarrollar un algoritmo que permita al usuario cargar un arreglo con 30 números enteros, en un arreglo para 120 números, y luego visualizarlos en orden inverso.

```
Algoritmo OrdenInverso
Léxico
  Max = 120
  TElem = Z
  TNumeros = arreglo[1..Max] de TElem
  TData = <a ∈ TNumeros, cant ∈ [0..Max-1]>
  datos ∈ TData
  indice ∈ Z
Inicio
  i ← 1
  datos.cant ← 130
  mientras i <= datos.cant hacer
    escribir (“Ingrese un número entero”)
    leer(datos.a[i])
    i ← i+1
 fmientras
 escribir (“Los datos ingresados, del último al primero, son:”)
 mientras i > 1 hacer
   escribir (datos.a[i-1])
   i ← i-1
 fmientras
Fin
```


Ejemplo VII

Desarrollar un algoritmo que permita al usuario cargar un arreglo con 30 números enteros, en un arreglo para 120 números, y luego visualizarlos en orden inverso.

Ejercicios

- Implementar en Pascal.
- Modificar el algoritmo usando la estructura **para** y luego implementar en Pascal.
- Modularizar la carga del arreglo y la visualización en dos acciones diferentes.

Ejemplo VIII

Desarrollar una acción que permita al usuario cargar un arreglo hasta 30 notas, en un arreglo para 256 números.

```
Algoritmo Notas
Léxico
  Max = 256
  TElem = Z
  TNumeros = arreglo[1..Max] de TElem
  TData = <a ∈ TNumeros, cant ∈ [0..Max-1]>
  misNotas ∈ TData
  Acción cargarNotas (resultado notas ∈ TData)
  Léxico local
    i ∈ Z
  Inicio
    escribir (“Cuantas notas va a cargar?”)
    leer(notas.cant)
    para i desde 1 hasta notas.cant paso 1 hacer
      escribir (“Ingrese la nota ”, i)
      leer(notas.a[i])
    fpara
  Fin
Inicio
  cargarNotas(misNotas)
Fin
```

Arreglos Bidimensionales

Los arreglos de dos dimensiones, también llamados *matrices*, se utilizan para representar tablas de valores.

Se requieren dos índices, uno para las **filas** y otro para las **columnas**.

Columnas

puntaje / Filas	1	2	3	4
1	4	6	-2	1
2	7	8	-4	0
3	9	6	7	3

Declaración de matrices

Para declarar una matriz se debe, preferentemente, declarar un nuevo tipo previamente.
Se le da un nombre al tipo, y además se define su tamaño y el tipo de datos (caracteres, enteros, etc.) que van a contener los arreglos que se definan en el futuro.
Luego se declaran todas las variables del tipo arreglo que sean necesarias.

Notación algorítmica

nombreTipo = **arreglo** [límiteInfFila..límiteSupFila,
límiteInfCol..límiteSupCol] **de** tipo //tipo
nombreVar ∈ nombreTipo //variable

Ejemplo:
TElem ∈ Z //tipo
NMaxFila = 7 //constante
NMaxCol = 10 //constante
TMat = **arreglo** [1..NMaxFila, 1..NMaxCol] **de** TElem //tipo
notas ∈ TMat //variable

Declaración de matrices

Pascal

TYPE nombreTipo = **ARRAY** [límiteInfFila..límiteSupFila,
límiteInfCol..límiteSupCol] **OF** tipo;
VAR nombre: nombreTipo;

El máximo en Pascal es 256 componentes para fila y 256 para columna.

Ejemplo:

```
CONST
NMaxFila = 7;
NMaxCol=10;
TYPE
TElem = INTEGER;
TMat = ARRAY [1.. NmaxFila,1..NMaxCol] OF TElem;
VAR  notas: TMat;
```


Declaración de matrices

En el caso de ser necesario saber la cantidad de elementos que contiene el arreglo, deben definirse dos variables adicionales para almacenar la cantidad filas y columnas.

Notación algorítmica

nombreTipo = **arreglo** [límiteInfFila..límiteSupFila,
límiteInfCol..límiteSupCol] **de** tipo //tipo
nombreVar ∈ nombreTipo //variable
nombreCantFila ∈ [límiteInfFila-1..límiteSupFila+1]
nombreCantCol ∈ [límiteInfCol-1..límiteSupCol+1]

Ejemplo:
TElem ∈ Z //tipo
NMaxFila = 7 //constante
NMaxCol = 10 //constante
TMat = **arreglo** [1..NMaxFila, 1..NMaxCol] **de** TElem //tipo
notas ∈ TMat //variable
cantFila ∈ [0..NMaxFila+1]
cantCol ∈ [0..NMaxCol+1]

Declaración de matrices

Pascal

TYPE nombreTipo = **ARRAY** [límiteInfFila..límiteSupFila,
límiteInfCol..límiteSupCol] **OF** tipo;
VAR
nombre: nombreTipo;
nombreCantFila: [límiteInfFila-1..límiteSupFila+1]
nombreCantCol: [límiteInfCol-1..límiteSupCol+1]

Ejemplo:

CONST
NMaxFila = 7; NMaxCol=10;
TYPE
TElem = INTEGER;
TMat = **ARRAY** [1.. NmaxFila,1..NMaxCol] **OF** TElem;
VAR
notas: Tmat; CantFila: [0..NMaxFila+1]; CantCol: [0..NMaxCol+1]

Declaración de matrices

La propuesta anterior se puede mejorar utilizando un registro para contener el arreglo junto a la cantidad elementos que contiene.

Notación algorítmica

nombreTipo = **arreglo** [límiteInfFila..límiteSupFila, límiteInfCol..límiteSupCol] **de** tipo
nombreRegistro = <campoMatriz ∈ nombreTipo, nombreCantFila ∈ [límiteInfFila-1..límiteSupFila+1], nombreCantCol ∈ [límiteInfCol-1..límiteSupCol+1]>
nombreVar ∈ nombreRegistro //variable

Ejemplo:

TElem ∈ Z //tipo
NMaxFila = 7 //constante
NMaxCol = 10 //constante
TMat = **arreglo** [1..NMaxFila, 1..NMaxCol] **de** TElem
TData = <m ∈ TMat, cantFila ∈ [límiteInfFila-1..límiteSupFila+1], cantCol ∈ [límiteInfCol-1..límiteSupCol+1] //tipo
notas ∈ TData //variable

Declaración de matrices

Pascal

TYPE nombreTipo = **ARRAY** [límiteInfFila..límiteSupFila,
límiteInfCol..límiteSupCol] **OF** tipo;
nombreRegistro = **RECORD** campoMatriz: nombreTipo; nombreCantFila: [límiteInfFila-1..límiteSupFila+1]; nombreCantCol: [límiteInfCol-1..límiteSupCol+1]
END;
VAR
nombre: nombreTipo;

Ejemplo:

CONST
NMaxFila = 7; NMaxCol=10;
TYPE
TElem = INTEGER;
TMat = **ARRAY** [1.. NmaxFila,1..NMaxCol] **OF** Telem;
TData = **RECORD** m: TMat; cantFila: [límiteInfFila-1..límiteSupFila+1]; cantCol: [límiteInfCol-1..límiteSupCol+1] **END**;
VAR
notas: TData;

Asignación, Carga y Consulta

• Asignación

{ei: A=[x_{11}, x_{12}, ..., x_{1i}, ..., $x_{1(n-1)}$, $x_{1n,}$ x_{21}, x_{22}, ..., x_{2i}, ..., $x_{2\ (n-1)}$, x_{2n}]}
A**[**2,1**]** ← e
{ef: A=[x_{11}, x_{12}, ..., x_{1i}, ..., $x_{1(n-1)}$, $x_{1n,}$ e, x_{22}, ..., x_{2i}, ..., $x_{2\ (n-1)}$, x_{2n}]}

• Asignación desde entrada estándar

{ei: A=[x11, x12, ..., x1i, ..., x1(n-1), x1n, x21, x22, ..., x2i, ..., x2 (n-1), x2n]}
leer(e)
A[2,1] ← e
{ef: A=[x11, x12, ..., x1i, ..., x1(n-1), x1n, e, x22, ..., x2i, ..., x2 (n-1), x2n]}

• Consulta

{ei: A=[x_{11}, x_{12}, ..., x_{1i}, ..., $x_{1(n-1)}$, $x_{1n,}$ x_{21}, x_{22}, ..., x_{2i}, ..., $x_{2\ (n-1)}$, x_{2n}]}
A**[**i,j**]**
{ef: A=[x_{11}, x_{12}, ..., x_{1i}, ..., $x_{1(n-1)}$, $x_{1n,}$ x_{21}, x_{22}, ..., x_{2i}, ..., $x_{2\ (n-1)}$, x_{2n}]}

Ejemplos

A[1, 4] ← 29

A[1, 6] ← A[1, 5] + 1

A[i, j] ← A[i, 4] + A[i, 6]

A[i, j] ← A[i+1, j] + A[i-1, j]

A[i+1, j] ← 45 + A[6, j]

A[i-1, j+1] ← A[i, j] + 3

Ejemplo I

Desarrollar un algoritmo que permita cargar una matriz de nombres que representa a los alumnos sentados en el aula 79 del Pabellón 2. El aula 79 contiene 8 filas de sillas por 20 columnas.

```
Algoritmo Aula79
Léxico
  Fila = 8
  Columna = 20
  TAula = arreglo[1..Fila, 1..Columna] de Cadena
  i ∈ Z
  j ∈ Z
  aula79 ∈ TAula
Inicio
```

}

```
Fin
```


Ejemplo I

Desarrollar un algoritmo que permita cargar una matriz de nombres que representa a los alumnos sentados en el aula 79 del Pabellón 2. El aula 79 contiene 8 filas de sillas por 20 columnas.

```
Algoritmo Aula79
Léxico
  Fila = 8
  Columna = 20
  TAula = arreglo[1..Fila, 1..Columna] de Cadena
  i ∈ Z
  j ∈ Z
  aula79 ∈ TAula
Inicio
  i←1
  mientras i <= Fila hacer
    j ← 1
    mientras j <= Columna hacer
     escribir("Ingrese el nombre del alumno sentado en la fila ",i," y columna ",j," del aula 79" )
     leer(aula79[i,j])
     j ← j+1
    fmientras
    i ← i+1
 fmientras
```

}

```
Fin
```


Ejemplo I

Resolver el mismo problema anterior pero definiendo la matriz dentro de un registro.

```
Algoritmo Aula79
```


Ejemplo II

Modificar el algoritmo para modularizar en una acción la carga de la matriz de nombres que representa a los alumnos sentados en el aula 79 del Pabellón 2. El aula 79 contiene 8 filas de sillas por 20 columnas.

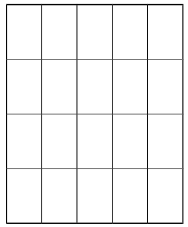
```
Algoritmo Aula79
```

Ejemplo III

Modificar el algoritmo anterior para que luego de cargar la matriz de nombres, muestre todo su contenido por pantalla.

Ejemplo IV

Desarrollar un algoritmo que permita cargar una matriz de números reales. El tamaño de matriz es de 4 filas y 5 columnas.



```
Algoritmo CargaMat
Léxico
  Fila = 4
  Columna = 5
  TNumReales = arreglo[1..Fila, 1..Columna] de R
  i, j ∈ Z
  matriz ∈ TNumReales
Inicio

  i⟵1
  mientras i <= Fila hacer
    j ⟵ 1
    mientras j <= Columna hacer
     escribir("Ingrese el número real que será insertado en la posición [",i,",",j,"] de la matriz " )
     leer(matriz[i,j])
     j ⟵ j+1
    fmientras
    i ⟵ i+1
 fmientras

Fin
```


Ejemplo IV

Resolver el mismo problema anterior pero definiendo la matriz dentro de un registro.

Ejemplo V

Desarrollar un algoritmo que, dada una matriz de números reales y tamaño 9 x 7, ya cargada, permita hallar el suma de todos los números almacenados e informarlo por pantalla.

```
Algoritmo SumaMat
Léxico
 N = 9
 M = 7
 TNumReales = arreglo[1..N, 1..M] de R
 acum ∈ R
 i, j ∈ Z
 matriz ∈ TNumReales
Inicio

  i⟵1
  acum⟵0
  mientras i <= N hacer
    j ⟵ 1
    mientras j <= M hacer
     acum ⟵ acum + matriz[i,j]
     j ⟵ j+1
    fmientras
    i ⟵ i+1
 fmientras
 escribir ("la suma de toda la matriz es: ", acum)
Fin
```

Ejemplo V

Resolver el mismo problema anterior pero definiendo la matriz dentro de un registro.

Bibliografía

- Scholl, P. y J.-P. Peyrin, “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”, Barcelona, Ed. Masson, 1991.

- Lucas, M., J.-P. Peyrin y P. Scholl, “Algorítmica y Representación de Datos. Tomo 1: Secuencia, Autómata de estados finitos”, Barcelona, Ed. Masson, 1985.

- Watt, David, “Programming Language Concepts and Paradigms“, Prentice-Hall International Series in Computer Science (1990).

- Biondi, J. y G. Clavel, “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”, 2° ed., Barcelona: Masson, 1985.

- Clavel, G. y Biondi, J., “Introducción a la Programación. Tomo 2: Estructuras de Datos”, 2° ed., Barcelona: Masson, 1985.

- De Guisti, A. “Algoritmos, datos y programas. Con aplicaciones en Pascal, Delphi y Visual Da Vinci. Prentice Hall.

- Joyanes Aguilar, L., “Programación en Turbo Pascal”. Mc Graw Hill, 1993.