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:

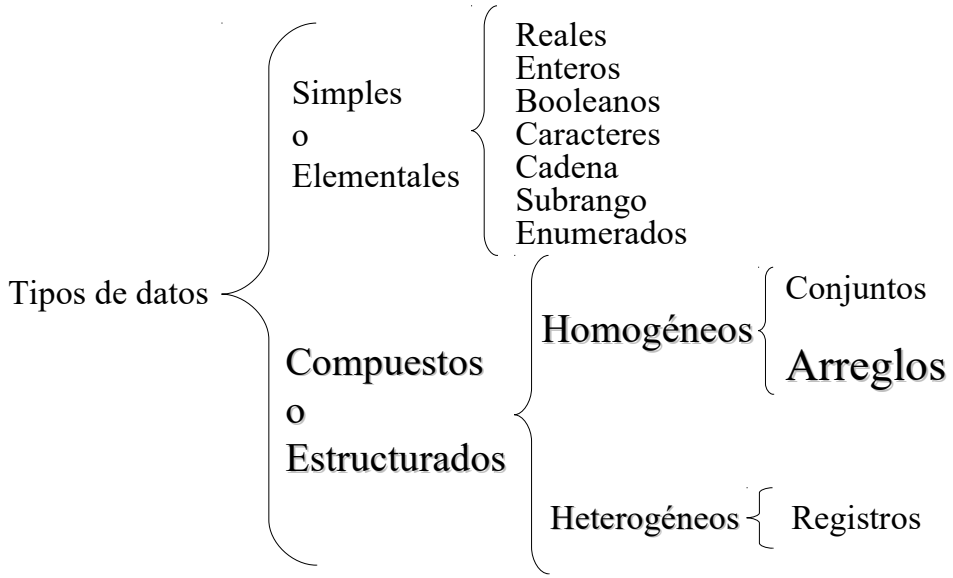
- posee varias componentes, cada una de las cuales puede ser un tipo simple u otra estructura de datos;
- existe una relación entre los distintos componentes que la dota de un cierto significado y utilidad;
- se manipulan mediante una serie de operaciones.

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

- Problema: un curso posee 150 alumnos, de los cuales se necesita la nota de cada uno, para luego informar 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** (tiene un límite), **homogénea** (elementos del mismo tipo) y "**ordenada"** (se sabe quien es el primero, el segundo, el tercero, etc) de elementos.

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

Columnas

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

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 0 (C, Java) o del 1 como primera posición (Pascal).

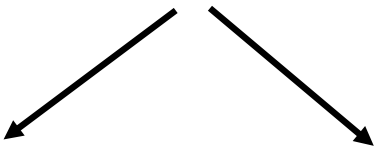
Arreglos

- 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

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

Columnas

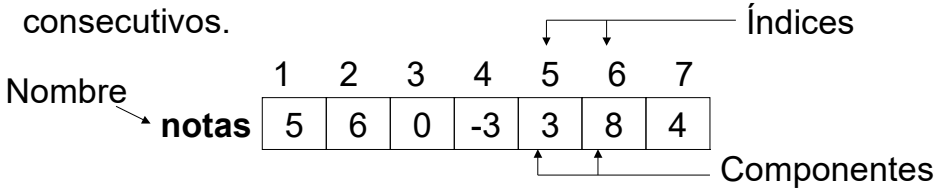
puntaje / Filas	1	2	3	4
1	4	6	-2	1
2	7	8	-4	0
3	9	6	7	3

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
TArr = **arreglo** [1..NMax] **de** TElem // tipo
notas ∈ TArr // variable

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] // variable

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

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

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

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]}

`Entrada:a[i]` } *o* `Entrada:e` `a[i] ← e`

{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

Se necesita almacenar 100 letras.
Nota: resolver utilizando arreglos y estructura iterativa repetir.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  letras ∈ TLetras
  indice ∈ Z
Inicio

Fin
```


Ejemplo I

Se necesita almacenar 100 letras.
Nota: resolver utilizando arreglos y estructura iterativa mientras.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  letras ∈ TLetras
  indice ∈ Z
Inicio

Fin
```

Ejemplo I

Se necesita almacenar 100 letras.
Nota: resolver utilizando arreglos y estructura iterativa para.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  letras ∈ TLetras
  indice ∈ Z
Inicio

Fin
```


Ejemplo I

Se necesita almacenar 100 letras, y luego informarlas en el mismo orden en que fueron obtenidas.
Nota: resolver utilizando arreglos y estructura/s iterativa/s a elección.

```
Algoritmo MisLetras
Léxico
  Max = 100
  TElem = Caracter
  TLetras = arreglo[1..Max] de TElem
  letras ∈ TLetras
  indice ∈ Z
Inicio

Fin
```


Ejemplo II

Se necesita almacenar no más de 100 letras.

```
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

Se necesita almacenar no más de 100 letras, como máximo, y luego informarlas en el mismo orden en que fueron obtenidas.

```
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

Se necesita almacenar no más de 100 letras, como máximo, y luego informarlas en el orden **inverso** en que fueron obtenidas.

```
Algoritmo MisLetrasInverso
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 V

En un Bingo se registran los números que van saliendo hasta que haya un ganador, o se saquen 30 números. Luego se debe informar los números que salieron.

```
Algoritmo Bingo
Léxico



Inicio



Fin
```

Ejemplo VI

Se necesita almacenar a lo sumo 80 vocales y luego informar la vocal que más veces se repite.

Aclaración: cada vocal debe ser almacenada en mayúscula.

```
Algoritmo Vocales
Léxico



Inicio



Fin
```

Ejemplo VII

Se necesita almacenar **n** números enteros (150 como máximo), y luego informarlos en orden inverso al que fueron obtenidos.

```
Algoritmo OrdenInverso
Léxico



Inicio



Fin
```

Ejemplo VII

Se necesita almacenar **n** números enteros (150 como máximo), y luego informarlos en orden inverso al que fueron obtenidos.

```
Algoritmo OrdenInverso
Léxico
  Max = 150
  TElem = Z
  TNumeros = arreglo[1..Max] de TElem
  TData = <a ∈ TNumeros, cant ∈(0..Max+1)>
  datos ∈ TData
  indice ∈ Z
Inicio
  i ← 1
  Entrada:datos.cant
  mientras i <= datos.cant hacer
    Entrada:datos.a[i]
    i ← i+1
  fmientras
  // informar Los datos ingresados, del último al primero
  mientras i > 1 hacer
   Salida:datos.a[i-1]
   i ← i-1
  fmientras
Fin
```


Ejemplo VIII

Desarrollar una acción que permita almacenar hasta 250 notas (entre 1 y 10) de estudiantes.

```
Algoritmo Notas
Léxico
  Max = 250
  TElem = (1..10)
  TNumeros = arreglo[1..Max] de TElem
  TData = <a ∈ TNumeros, cant ∈(0..Max+1)>
  misNotas ∈ TData




Inicio
  cargarNotas(misNotas)
Fin
```


Ejemplo VIII

Desarrollar una acción que permita almacenar hasta 250 notas (entre 1 y 10) de estudiantes.

```
Algoritmo Notas
Léxico
  Max = 250
  TElem = (1..10)
  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
    // cantidad de notas a cargar
    Entrada:notas.cant
    para (i←1, i<=notas.cant, i←i+1) hacer
      // obtener cada nota
     Entrada:notas.a[i]
    fpara
  Facción

Inicio
  cargarNotas(misNotas)
Fin
```


Ejemplo IX

Desarrollar una acción que permita almacenar hasta 250 evaluaciones. Cada evaluación está compuesta por el nombre del estudiante y la nota.

```
Algoritmo Evaluaciones
Léxico
  Max = 250
  TElem =
  TNumeros = arreglo[1..Max] de TElem
  TData = <a ∈ TNumeros, cant ∈(0..Max+1)>
  misNotas ∈ TData
Inicio




Fin
```

Ejemplo X

Se necesita almacenar a lo sumo 10 vocales (sin superar los 255 caracteres), luego informar la cantidad de vocales almacenadas, 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
Léxico

Inicio

Fin

Arreglos unidimensionales en C

```
/* OrdenInverso
Max = 150
TElem = Z
TNumeros = arreglo[1..Max] de
TElem
TData = <a ∈ TNumeros, cant ∈
(0..Max+1)>
datos ∈ TData
indice ∈ Z
*/
#define N 150
struct data{
      int cant;
      int a[N];
};
struct data datos;
int i;
```

```
/* Notas
Max = 250
TElem = (1..10)
TNumeros =
arreglo[1..Max] de TElem
TData = <a e TNumeros,
cant ∈ (0..Max+1)>
misNotas ∈ TData
*/

#define N 250
struct data{
      int cant;
      int a[N];
};
struct data misNotas;
```

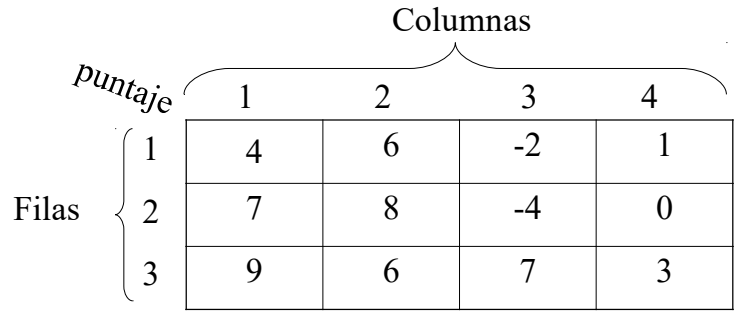

Ejercicios

Implementar en C los ejemplos VII y VIII.

Arreglos Bidimensionales o Matrices

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**.



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]}

```
Entrada:a[2,1]
```

o

```
Entrada: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
```


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

Ejemplo I

Se necesitan almacenar los nombres que representan a los alumnos sentados en el aula 79 del Pabellón 2. El aula 79 contiene 8 hileras de 20 sillas cada una.

```
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

Se necesitan almacenar los nombres que representan a los alumnos sentados en el aula 79 del Pabellón 2. El aula 79 contiene 8 hileras de 20 sillas cada una.

```
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
     //nombre del alumno sentado en la fila i y columna j del aula 79
     Entrada:aula79[i,j]
     j ← j+1
    fmientras
    i ← i+1
 fmientras
Fin
```

Declaración de matrices

En el caso de ser necesario saber la cantidad de elementos que contiene la matriz, 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

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

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.

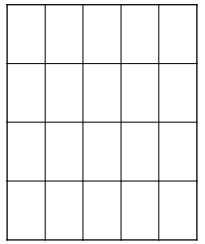
Algoritmo Aula79

Ejemplo III

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

Ejemplo IV

Un piso rectangular contiene 20 balsosas de largo y 30 de ancho. Cada baldosa posee un número (entre 0 y 1) que identifica su dureza. Es necesario almacenar los datos de cada una de las baldosas.



```
Algoritmo Piso
Léxico
  Fila = 20
  Columna = 30
  TNumReales = arreglo[1..Fila, 1..Columna] de R
  i, j ∈ Z
  piso ∈ TNumReales
Inicio

  i←1
  mientras i <= Fila hacer
    j ← 1
    mientras j <= Columna hacer
      Entrada:piso[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

Un piso rectangular contiene 20 balsosas de largo y 30 de ancho. Cada baldosa posee un número (entre 0 y 1) que identifica su dureza. Es necesario almacenar los datos de cada una de las baldosas y luego calcular el promedio de dureza de todo el piso.

```
Algoritmo DurezaPromedio
Léxico
 N = 20
 M = 30
 TNumReales = arreglo[1..N, 1..M] de R
 acum, promedio ∈ R
 i, j ∈ Z
 piso ∈ TNumReales
Inicio

  i←1
  acum←0
  mientras i <= N hacer
    j ← 1
    mientras j <= M hacer
      acum ← acum + piso[i,j]
      j ← j+1
    fmientras
    i ← i+1
 fmientras
 promedio ← acum/(N+M) //dureza promedio
 Salida:promedio
Fin
```

Ejemplo V

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

Arreglos bidimensionales en C

```
/* DurezaPromedio
N = 20
M = 30
TNumReales = arreglo[1..N, 1..M] de R
TData = <a ∈ TNumReales, cantFila ∈(0..N+1), cantCol ∈
(0..M+1)>
i, j ∈ Z
piso ∈ TData
*/

#define N 20
#define M 30
struct data{
      int cant;
      int a[N][M];
};
struct data piso;
int i,j;
```


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.