Tipos de Datos Estructurados

1. Definición

A continuación se presentan diferentes definiciones de tipos de datos estructurados:

Una estructura de datos o tipo de dato estructurado es un tipo de dato construido a partir de otros tipos de datos. Los tipos de datos estructurados pueden estar formados por otros tipos de datos estructurados, pero los componentes de su nivel mas inferior están formados por datos de tipo simple (Cobo, 2009).

Entendemos por tipo de dato estructurado la agrupación de varias informaciones bajo un mismo identificador. Por decirlo así, son tipos de datos compuestos por otros tipos de datos, que pueden ser todos iguales, o distintos (Badenas, Coltell y Llopis, 2001).

Los tipos de datos estructurados son colecciones de variables, posiblemente de diferentes tipos de dato, conectadas de varias maneras (Aho, 1983).

2. Clasificación

Existen diferentes formas de clasificar los tipos de datos abstractos, entre ellas se encuentran (Cobo, 2009):

2.1. Según los tipos de datos que lo conforman:

***Homogéneos*:** Cuando todos los tipos elementales que lo conforman son del mismo tipo.
***Heterogéneos*:** Cuando todos los tipos elementales que lo conforman no son del mismo tipo.

2.2. Según su almacenamiento:

***Internos*:** Cuando se almacenan en la memoria del computador.
***Externos*:** Cuando se almacenan en la memoria secundaria del computador.

2.3. Según su uso de la memoria:

***Estáticos*:** Cuando se define su tamaño en memoria antes de que se ejecute el programa y no puede ser modificado durante la ejecución del mismo.
***Dinámicos*:** Cuando su tamaño en memoria puede variar durante la ejecución de un programa.

3. Tipos de datos estructurados mas Comunes

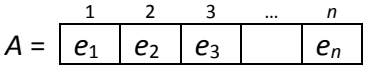
En las siguientes secciones se definirán dos de los tipos de datos estructurados más comunes: Arreglos y registros.

4. Arreglos

Un arreglo es una estructura de datos con elementos relacionados del mismo tipo (Deitel y Deitel, 1996). En una arreglo cada elemento individual se puede identificar por su posición dentro de la colección, la cual es relativa al primer elemento (Sebesta, 2006).

Todos los elementos del arreglo se numeran de forma consecutiva, en la mayoría de los casos comenzando desde el número 1. Estos números son los subíndices del arreglo, los cuales permiten acceder de forma directa a cualquier elemento dentro del arreglo.

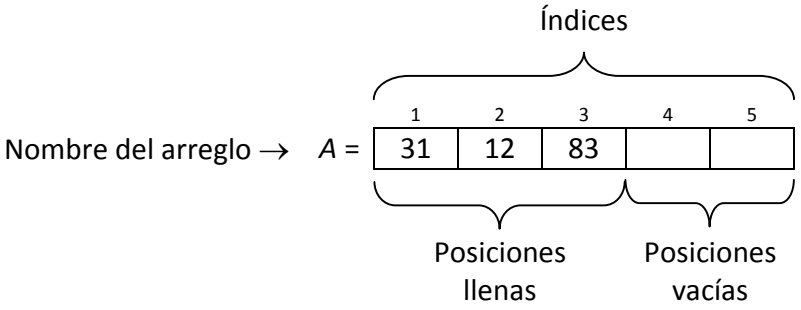
Gráficamente un arreglo se puede ver como sigue:



Donde *A* es el identificador del arreglo, los números 1, 2, 3, …, *n* son los subíndices y e_i ($1 \leq i \leq n$), son los elementos del arreglo.

Ejemplo:

Arreglo de 5 posiciones, con 3 posiciones llenas y dos posiciones vacías.



4.1. Características

- Son estructuras de datos homogéneas.
- Todos sus datos son referenciados mediante un mismo identificador.
- Se almacenan en posiciones de memoria contiguas, esto implica que la posición de memoria más baja corresponde a la del primer elemento del arreglo y la más alta a la del último elemento.
- Es una estructura finita.
- Cada elemento posee una única posición.

4.2. Tipos

Los arreglos se pueden clasificar en: Arreglos unidimensionales y arreglos multidimensionales.

Arreglos Unidimensionales

Un arreglo unidimensional también llamado vector, es un conjunto de elementos del mismo tipo organizados en forma de lista de elementos consecutivos. Estos arreglos cuentan con un solo subíndice para determinar la posición de un elemento en la estructura. Gráficamente:

	1	2	3	…	*n*
A =	e_1	e_2	e_3		e_n

Arreglos Multidimensionales

Los arreglos multidimensionales son aquellos en los cuales la posición de sus elementos es determinada por más de un subíndice.

Los arreglos multidimensionales más comunes son los de dos dimensiones, llamados arreglos bidimensionales o matrices, cuyos elementos se encuentran distribuidos en filas y columnas. Gráficamente:

		1	2	3	…	*n*
	1	$e_{1,1}$	$e_{1,2}$	$e_{1,3}$		$e_{1,n}$
	2	$e_{2,1}$	$e_{2,2}$	$e_{2,3}$		$e_{2,n}$
A =	3	$e_{3,1}$	$e_{3,2}$	$e_{3,3}$		$e_{3,n}$
	…					
	m	$e_{m,1}$	$e_{m,2}$	$e_{m,3}$		$e_{m,n}$

4.3. Declaración

En lenguaje pseudoformal, un arreglo se declara como un tipo definido por el usuario, en el bloque de **tipo** del algoritmo de la siguiente forma:

```
T = arreglo [dim1,..., dimn] de T0
```

Donde:

- *T* es el identificador del nuevo tipo
- dim_i ($1 \leq i \leq n$) es un intervalo de tipo ordinal que especifica, para cada dimensión, los índices del primer y del ultimo elemento del arreglo.
- T_0 es el identificador de cualquier tipo (elemental o definido por el usuario) que especifica el tipo de todos los elementos del arreglo.

Después de declarar el tipo *T*, la declaración de variables de este tipo es la misma declaración de variables conocida:

```
T variable_1, variable_2, ..., variable_n
```

Ejemplos:

a) Para un arreglo unidimensional:

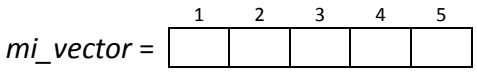
Declaración de tipo:

```
vector = arreglo [1..5] de entero
```

Declaración de una variable:

```
vector mi_vector
```

Gráficamente la variable *mi_vector* del ejemplo se puede visualizar como:



b) Para un arreglo bidimensional:

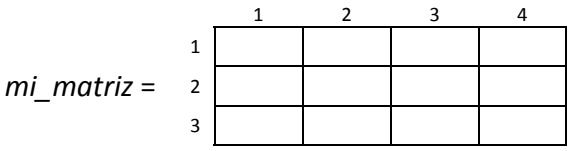
Declaración de tipo:

```
matriz = arreglo [1..3, 1..4] de entero
```

Declaración de una variable:

```
matriz mi_matriz
```

Gráficamente la variable *mi_matriz* del ejemplo se puede visualizar como:



4.4. Acceso

Para acceder a los elementos de un arreglo basta con colocar el identificador de la variable seguido de una lista de índices, $expr_1$, $expr_2$, ..., $expr_n$, entre corchetes, de la siguiente forma:

```
mi_arreglo[expr1, expr2, ..., exprn]
```

Cada índice, $expr_i$ ($1 \leq i \leq n$) va a determinar la posición del elemento en la *i*-ésima dimensión, y puede ser de tres tipos: Constante, variable, o expresión.

Ejemplo:

Para acceder a un elemento de la variable *mi_matriz* del ejemplo anterior se coloca:

```
mi_matriz[2, 3]       // Índices constantes
mi_matriz[i, j]       // Índices variables
mi_matriz[i+2, 3*j]   // Índices tipo expresión
```

4.5. Operaciones

Existen diferentes operaciones que pueden ser realizadas sobre arreglos: Lectura, escritura, inicialización, eliminación e inserción, entre otras. A continuación se presentan algunas de las operaciones mas comunes que pueden ser realizadas sobre arreglos unidimensionales y bidimensionales.

Para explicar las operaciones se asumen las siguientes declaraciones de constantes tipos y variables:

```
const
    N ← 5
    M ← 10
tipo
    vector = arreglo [1..N] de entero
    matriz = arreglo [1..N, 1..M] de entero
var
    vector v
    matriz m
    entero i, j, k
```

Operaciones sobre Arreglos Unidimensionales

a) Lectura y Escritura

```
// Lectura de elementos
para i ← 1 hasta N hacer
    leer(v[i])
fpara

// Escritura de elementos
para i ← 1 hasta N hacer
    escribir(v[i], " ")
fpara
```

b) Inicialización

```
// Inicializa el arreglo en 0
para i ← 1 hasta N hacer
    v[i] ← 0
fpara
```

c) Eliminación

Para la eliminación se asume que el vector tiene *j* posiciones llenas y se desea eliminar el elemento en la posición *k*.

```
// Elimina el valor de la posición k del arreglo
para i ← k hasta j - 1 hacer
    v[i] ← v[i+1]
fpara
j ← j - 1
```

Ejemplo:

Sea el vector *v* el siguiente:

	1	2	3	4	5
v =	31	12	19	83	

Se desea eliminar la posición *k* = 2.

i = 2 ⇒ *v* =

1	2	3	4	5
31	19	19	83	

i = 3 ⇒ *v* =

1	2	3	4	5
31	19	83	83	

d) Inserción

Para la inserción se asume que el vector tiene *j* posiciones llenas y se desea insertar un elemento en la posición *k*. Es importante verificar que todavía existen posiciones vacías antes de realizar la inserción en el vector.

```
// Inserta el valor val en la posición k del arreglo
para i ← j hasta k en -1 hacer
    v[i+1] ← v[i]
fpara
v[k] ← val
j ← j + 1
```

Ejemplo:

Sea el vector *v* el siguiente:

	1	2	3	4	5
v =	31	12	19	83	

Se desea insertar el valor *val* = 22 en la posición *k* = 2. El proceso de inserción mostrado anteriormente haría lo siguiente:

			1	2	3	4	5
i = 4	⇒	*v* =	31	12	19	83	83
i = 3	⇒	*v* =	31	12	19	19	83
i = 2	⇒	*v* =	31	12	12	19	83
v[2] ← 22	⇒	*v* =	31	22	12	19	83

Operaciones sobre Arreglos Bidimensionales

Para realizar operaciones sobre arreglos bidimensionales es necesaria la utilización de dos ciclos anidados, uno que se encargue del movimiento por filas y otro que se encargue del movimiento por columnas.

a) Lectura y Escritura

```
// Lectura
para i ← 1 hasta N hacer
    para j ← 1 hasta M hacer
        leer(m[i, j])
    fpara
fpara

// Escritura
para i ← 1 hasta N hacer
    para j ← 1 hasta M hacer
        escribir(m[i, j], " ")
    fpara
    escribir("\n")
fpara
```

b) Inicialización

```
// Inicializa todos los valores de la matriz a 0
para i ← 1 hasta N hacer
    para j ← 1 hasta M hacer
        m[i, j] ← 0
    fpara
fpara
```

5. Algoritmos de Búsqueda

Con frecuencia es necesario determinar si un arreglo contiene un cierto valor, es decir, verificar la existencia de un elemento dentro de la estructura de datos. El proceso de encontrar en un arreglo un elemento un particular, se denomina búsqueda.

El resultado de un algoritmo de búsqueda puede ser:

a) Un valor lógico indicando si existe o no el elemento buscado,
b) Un número entero que indique la posición del elemento en el arreglo (si no existe devuelve un valor negativo, o cualquier valor fuera del intervalo definido para el arreglo) o
c) Un número entero, que indique la cantidad de veces que se encuentra el elemento buscado.

En esta sección se estudian dos técnicas de búsqueda, la técnica simple de búsqueda secuencial y una técnica más eficiente llamada búsqueda binaria.

5.1. Búsqueda Secuencial

Este algoritmo busca de forma secuencial un elemento a lo largo del arreglo, sin importar si el arreglo se encuentra ordenado o no. Si el arreglo es pequeño el algoritmo resulta eficiente, de lo contrario resulta ineficiente. Si la búsqueda se va a realizar sobre un arreglo previamente ordenado es mejor utilizar una búsqueda binaria.

```
algoritmo Busqueda_Secuencial
const N ← 100
tipo vector = arreglo [1..N] de entero //Cualquier tipo
inicio
    var
          entero i, num
          logico encontrado
          vector v
    // Lectura de elementos
    para i ← 1 hasta N hacer
          leer(v[i])
    fpara
    // Petición del número a buscar
    escribir("Introduzca el número a buscar: ")
    leer(num)
    // Búsqueda Secuencial
    encontrado ← falso
    i ← 1
    mientras(¬encontrado ∧ i ≤ N) hacer
          si(v[i] = num) entonces
                encontrado ← verdadero
          sino
                i ← i + 1
          fsi
    fmientras
    si(encontrado) entonces
          escribir(num, " fue encontrado en ", i)
    sino
          escribir(num, " no fue encontrado")
    fsi
fin
```

5.2. Búsqueda Binaria

El algoritmo de búsqueda binaria o dicotómica se usa en arreglos grandes que se encuentran previamente ordenados.

```
algoritmo Busqueda_Binaria
const N ← 100
tipo vector = arreglo [1..N] de entero //Cualquier tipo
inicio
    var
        entero num, inferior, superior, mitad, pos
        logico encontrado
        vector v
    // Lectura de elementos
    para i ← 1 hasta N hacer
        leer(v[i])
    fpara
    // Petición del número a buscar
    escribir("Introduzca el número a buscar: ")
    leer(num)
    // Búsqueda Binaria
    encontrado ← falso
    inferior ← 1
    superior ← N
    pos ← 0
    mientras(¬encontrado ∧ inferior ≤ superior) hacer
        mitad ← (inferior + superior) div 2
        si(v[mitad] = num) entonces
            encontrado ← verdadero
            pos ← mitad
        sino
            si(v[mitad] < num) entonces
                inferior ← mitad + 1
            sino
                superior ← mitad – 1
            fsi
        fsi
    fmientras
    si(encontrado) entonces
        escribir(num, " fue encontrado en ", pos)
    sino
        escribir(num, " no fue encontrado")
    fsi
fin
```

6. Algoritmos de Ordenamiento: El método de la Burbuja

El método de la burbuja de es uno de los métodos de ordenamiento más fáciles. En este algoritmo se compara cada elemento del arreglo con el siguiente (por parejas), si no están en el orden correcto, se intercambian entre si sus valores. El valor mas pequeño "flota" hasta la parte superior del arreglo como si fuera una burbuja en un vaso de refresco con gas (Joyanes, 2001).

A continuación se presenta el método de la burbuja en lenguaje pseudoformal.

```
algoritmo Burbuja
const N ← 100
tipo vector = arreglo [1..N] de entero //Cualquier tipo
inicio
    var
        entero i, j, aux
        vector v
    // Lectura de elementos
    para i ← 1 hasta N hacer
        leer(v[i])
    fpara
    // Método de la burbuja
    para i ← 1 hasta N - 1 hacer
        para j ← 1 hasta N – 1 hacer
            si(v[j] > v[j+1]) entonces
                aux ← v[j]
                v[j] ← v[j+1]
                v[j+1] ← aux
            fsi
        fpara
    fpara
fin
```

7. Registro

Un registro, también llamado tupla es un conjunto heterogéneo de elementos, en el cual cada elemento individual es identificado por su nombre (Sebesta, 2006). Los componentes de un registro se denominan campos. Los nombres de los campos se denominan identificadores de campo o selectores (Martínez y Rosquete, 2009).

7.1. Declaración

En el lenguaje pseudoformal, un registro se declara como un tipo, en la sección **tipo** del algoritmo de la siguiente forma:

```
T = registro
    T1 id_campo1
    T2 id_campo2
    ...
    Tn id_campon
fregistro
```

Ejemplo:

Si se quieren relacionar los datos correspondientes a una persona, se podría construir un registro de la siguiente forma:

```
persona = registro
    cadena nombre, apellido
    entero cedula
fregistro
```

Luego, para declarar una variable de tipo persona se coloca lo siguiente:

```
persona alguien
```

Donde *alguien* es el identificador de una variable de tipo *persona*.

7.2. Acceso

Para acceder a los diferentes campos dentro de una variable de tipo registro se utiliza el operador de selección de campo punto (**.**). Para el ejemplo anterior, si se quiere asignar un valor a la cédula de la persona correspondiente a la variable *alguien* se colocaría lo siguiente:

```
alguien.cedula ← 12345678
```

8. Referencias

Aho, A., Hopcroft, J. E. (1983). Data Estructures and Algorithms. Addison-Wesley.

Badenas C., J., Coltell S., O. y Llopis B., J. L. (2001). Curso Práctico de Programación en C y C++. Universitat Jaume.

Cobo, A. (2009). Programar desde un Punto de Vista Científico. Madrid, España: Visión Libros.

Deitel H. M. y Deitel P. J. (1995). Cómo Programar en C/C++ (Segunda ed.). Naucalpán de Juárez, México: Prentice Hall Hispanoamericana S.A.

Joyanes A., L. (2001). Programación en C. Metodología, algoritmos y estructura de datos. Madrid, España: McGraw-Hill/Interamericana de España, S.A.U.

Martínez, A., Rosquete, D. (2009). NASPI: Una Notación Estándar para Programación Imperativa. Télématique.

Sebesta, R. W. (2006). Concepts of Programming Languages (Séptima ed.). Massachusetts, USA: Pearson Education.