Arreglos Unidimensionales

Programación Lógica

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

1. Arreglos unidimensionales

Una estructura de datos es una colección de datos organizados de una cierta manera para que puedan ser utilizados eficientemente. Las estructuras de datos son muy importantes en los sistemas de computadora. Los tipos de datos pueden ser simples y estructurados. Se recomienda revisar la clasificación que plantea Joyanes Aguilar (2008) en *Fundamentos de programación* para analizar una descripción más completa.

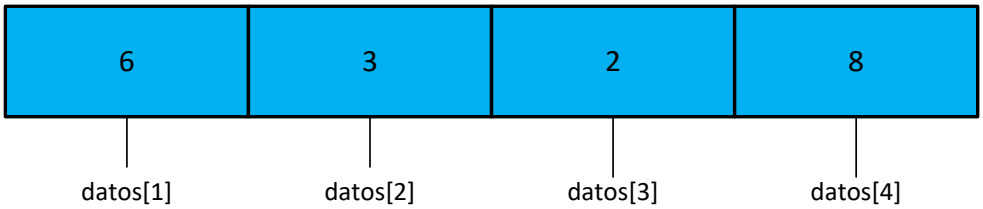
Los datos estructurados se clasifican en estáticos y dinámicos. En las estructuras estáticas, la memoria se gestiona en tiempo de compilación, mientras que, para las estructuras dinámicas, la memoria se gestiona en tiempo de ejecución. La forma de generar el almacenamiento implica ventajas y desventajas. Ya que ambos tipos de estructuras existen en los lenguajes reales, el uso más conveniente de una u otra dependerá de los requerimientos del problema que resolver.

Un arreglo es una estructura de datos estática y representa un conjunto finito y ordenado de elementos del mismo tipo (homogéneos). Los arreglos pueden ser unidimensionales, también llamados vectores (o arreglos lineales), bidimensionales (matrices) o multidimensionales. Todo arreglo tiene asociado un nombre o identificador y cada elemento del arreglo es referenciado por la posición que ocupa. Para referenciar a cada posición del arreglo, se utilizan índices.

Los arreglos se utilizan cuando es necesario manipular varios valores como parte de un conjunto. Se podrían guardar valores en un conjunto de variables simples, pero esto es más complejo de manipular. Por ejemplo, en lugar de tener 1000 variables de tipo *cadena de caracteres* para guardar los nombres de personas, es más sencillo crear un arreglo de 1000 elementos de tipo *cadena de caracteres*, ya que toda la información se maneja con el mismo nombre de variable y los elementos se identifican de acuerdo con un subíndice asociado al vector.

Los vectores se pueden representar gráficamente como muestra la Figura 1, en donde se representa un arreglo unidimensional llamado *datos* de cuatro elementos. El identificador del arreglo es utilizado para todos los elementos del conjunto, y el valor del índice asociado identifica en forma precisa a los elementos del arreglo. En este ejemplo los elementos del arreglo son números enteros.

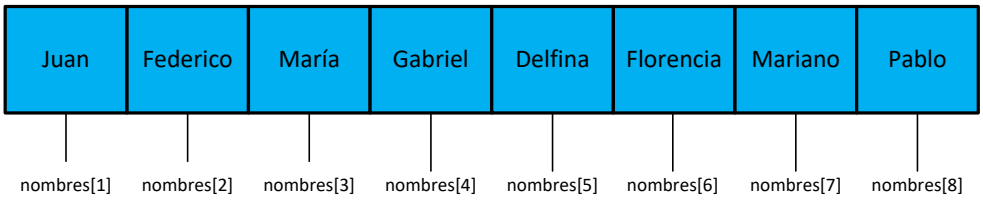
Figura 1: Representación gráfica de un arreglo unidimensional (vector)



Fuente: elaboración propia.

El número de elementos que se pueden asociar a un vector se llama *rango del vector*. En el caso de la Figura 1, el rango del vector llamado *datos* es 4. Los arreglos, en general, pueden contener datos numéricos y no numéricos. El vector de la siguiente figura contiene cadena de caracteres que representan nombres de personas.

Figura 2: Representación gráfica de un arreglo unidimensional (vector)



Fuente: elaboración propia.

El rango o tamaño del arreglo se debe indicar previamente a su uso dentro de un programa. En pseudocódigo, un arreglo unidimensional se declara de la siguiente forma:

Algoritmo *declaracion_arreglo*
tipo
    array*[límite inferior...límite superior]* ***de*** *<tipo de dato>: <nombre estructura arreglo>*
var
    <nombre estructura arreglo>: <nombre variable de tipo arreglo>
inicio
……

Ejemplo:

Algoritmo *declaracion_arreglo*
tipo
    array*[1...4]* ***de*** *entero: arreglo_enteros*
var
    arreglo_enteros: datos
inicio
……

Se debe notar que, en el pseudocódigo indicado, hay una sección llamada *tipo*, en la cual se declaran las estructuras de datos de arreglos y otros tipos de datos definidos por el usuario. Se usa la palabra reservada *array* seguido de un par de corchetes con el rango o tamaño del arreglo indicados por el límite inferior y superior. A continuación, se indica el tipo de dato de los elementos del arreglo y el nombre que tiene la estructura de datos definida. En el ejemplo, la estructura se llama *arreglo_enteros*. En la sección *var*, se declaran las variables que se utilizarán en el algoritmo, y en este caso se deben declarar las variables que tendrán la estructura del arreglo definido. En el ejemplo, el algoritmo utilizará una variable de nombre *datos* que tiene la estructura del arreglo.

Cada elemento de un vector se puede procesar como si fuese una variable simple. Por ejemplo, *datos[4]* ← *8* almacena el valor entero 8 en el vector *datos* en la posición número 4. Si se desea guardar el valor entero "*Federico*" en la segunda posición del vector nombres, la instrucción es nombres[2] ← "*Federico*".

La instrucción *escribir(datos[4])* muestra por pantalla el contenido de la posición 4 del vector llamado *datos*.

En el siguiente ejemplo, se crean dos vectores distintos con la misma estructura definida.

```
Algoritmo declaracion_arreglo_2
tipo
   array[1...10] de entero: lista
var
   lista: origen, destino
   entero: indice
inicio
……
```

En este ejemplo, se declaran dos arreglos distintos definidos por las variables *origen* y *destino*. De la misma forma que se declara la variable *indice* de tipo de dato *entero*, también se declaran las variables *origen* y *destino* de tipo de dato *lista*.

El límite inferior no siempre tiene que iniciar en 1, pero es lo más frecuente. Puede ser definido con cualquier otro valor entero, siempre y cuando el rango definido sea correcto. Las siguientes referencias son válidas:

P[0], P[1], P[2], P[3], P[4], P[5]

En este caso, el rango del arreglo *P* es 6, y a la posición 1 le corresponde el subíndice 0.

P[-10], P[-11], P[-12], P[-13], P[-14], P[-15]

Aquí el rango del arreglo *P* es 6, y a la posición 1 le corresponde el subíndice -10, a la posición 2 le corresponde el subíndice -11, y así sucesivamente.

Las operaciones que pueden realizarse con vectores son:

- asignación;
- lectura/escritura;
- recorrido;
- ordenamiento;
- búsqueda.

Las operaciones de lectura y escritura de los elementos de un arreglo lineal, generalmente, se realizan con estructuras repetitivas. Por ejemplo, si se desea mostrar por pantalla el contenido de todo un arreglo, se usará una estructura repetitiva, como se muestra a continuación:

```
Algoritmo recorrer_arreglo
tipo
   array[1...10] de entero: lista
var
   lista: datos
   entero: indice
inicio
      mostrar("Ingresar los 10 elementos del arreglo")
      desde indice ← 1 hasta 10 hacer
            leer(datos[indice])
      fin-desde

      mostrar("Se muestran los elementos del arreglo")
      desde indice ← 1 hasta 10 hacer
            escribir(datos[indice])
      fin-desde
fin
```

Ejercicios de aplicación

1) Implementar un algoritmo que permita ingresar 10 números enteros por teclado, los guarde en un arreglo unidimensional, para luego recorrerlo y mostrar los valores que ocupan las posiciones pares.

```
algoritmo arreglo_posiciones_pares
tipo
    array[1...10] de entero: lista_numeros
var
    entero: i
    lista_numeros: datos
inicio
    mostrar("Ingresar los 10 elementos del arreglo")
    desde i ←1 hasta 10 hacer
        leer(datos[i])
    fin-desde

    mostrar("Se muestran los elementos de las posiciones pares del  arreglo")
    desde i ←2 hasta 10 inc 2 hacer
        mostrar(datos[i])
     fin-desde
fin
```

2) Implementar un algoritmo que permita ingresar 10 números enteros por teclado, los guarde en un arreglo unidimensional, para luego recorrerlo y mostrar los valores positivos que sean pares.

```
algoritmo arreglo_positivos_pares
tipo
    array[1...10] de entero: lista_numeros
var
    entero: i
    lista_numeros: datos
inicio
    mostrar("Ingresar los 10 elementos del arreglo")
    desde i ←1 hasta 10 hacer
        leer(datos[i])
    fin-desde

    mostrar("Se muestran los elementos positivos pares")
    desde i ←1 hasta 10 hacer
        si (datos[i] > 0) y (datos[i] mod 2 == 0) entonces
            mostrar(datos[i])
        fin-si
     fin-desde
fin
```

3) Implementar un algoritmo que permita ingresar 10 números enteros por teclado, los guarde en un arreglo unidimensional, para luego recorrerlo y

guarde los valores positivos en otro arreglo con la misma estructura. Se debe mostrar el contenido del segundo arreglo.

```
algoritmo arreglo_copiar_positivos
tipo
    array[1...10] de entero: lista_numeros
var
    entero: i, indice_destino
    lista_numeros: origen, destino
inicio
    mostrar("Ingrese 10 números enteros para cargar el arreglo origen")
    para i ← 1 hasta 10 hacer
        leer(origen[i])
    fin-para

    mostrar("Copiando números positivos de origen en nuevo arreglo destino...")
    indice_destino ← 0
    para i ← 1 hasta 10 hacer
        si (origen[i] > 0) entonces
            indice_destino ← indice_destino + 1
            destino[indice_destino] ← origen[i]
        fin-si
    fin-para

    mostrar("Mostrando arreglo destino")
    para i ← 1 hasta indice_destino hacer
        mostrar(destino[i])
    fin-para
fin
```

4) Implementar un algoritmo que permita ingresar 10 caracteres por teclado, los guarde en un arreglo unidimensional, para luego recorrerlo y muestre la cantidad de letras mayúsculas.

```
algoritmo arreglo_cantidad_letras_mayusculas
tipo
    array[1...10] de caracter: lista_caracteres
var
    entero: i, cantidadLetrasMayusculas
    lista_caracteres: datos
inicio
    mostrar("Ingrese 10 caracteres")
    para i ← 1 hasta 10 hacer
        leer(datos[i])
    fin-para

    cantidadLetrasMayusculas ← 0
    para i ← 1 hasta 10 hacer
        si (datos[i] >= 'A') Y (datos[i] <= 'Z') entonces
            cantidadLetrasMayusculas ← cantidadLetrasMayusculas + 1
        fin-si
    fin-para
```

```
    mostrar("La cantidad de letras mayúsculas del arreglo es: ", cantidadLetrasMayusculas)
fin
```

Referencias

Joyanes Aguilar, L. (2008). *Fundamentos de programación. Algoritmos, estructura de datos y objetos*. Madrid, ES: McGraw-Hill.