Algoritmia y Programación

Facultad de Ingeniería

Arreglos Unidimensionales

Universidad Autónoma de Occidente

Cali - 2016

www.uao.edu.co

Objetivo de Aprendizaje

Que el estudiante conozca sobre el procesamiento y uso de arreglos y cómo su utilización es necesaria cuando se va a manejar conjuntos de muchos datos.

Definición:

Un arreglo es un tipo especial de variable que permite almacenar más de un valor al mismo tiempo.

En todo caso, la cantidad de valores es un valor finito.

CARACTERÍSTICAS

Estructura de almacenamiento estático; o sea, su máximo tamaño se fija cuando se lo crea. Permite referenciar varios datos con un mismo nombre, que se distinguen entre ellos mediante un subíndice.

Es homogénea; es decir, contiene valores de un solo tipo de dato.

Su aplicación se enmarca al uso masivo de datos, mediante la utilización de los ciclos.

Particularidades

Todos los arreglos tienen un tamaño máximo que es definido cuando se crean.

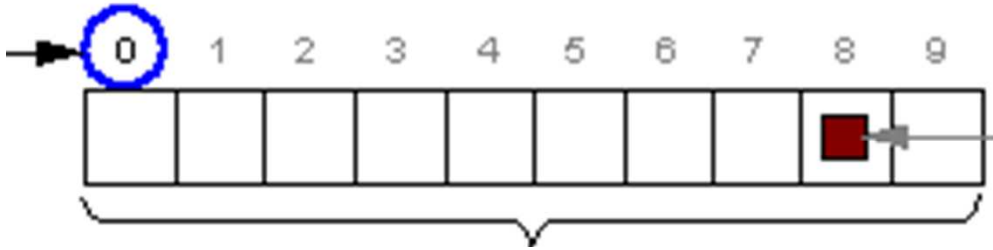
La información dentro de los arreglos se reconoce por un índice ó posición, en la que se almacenan los valores.

El primer valor se guarda en el índice ó posición **0** y el último en el índice ó posición **tamaño-1**.

ARREGLOS DE 1 DIMENSIÓN - VECTORES

Nombre del Arreglo: datos

primer subíndice



subíndices

Arreglo de longitud 10

Elemento en el subíndice 8 : datos[8]

Todos los elementos del Arreglo comparten el nombre

Estrategia Pedagógica

Aprender Haciendo

Manos a la obra

EJEMPLO:

Una empresa de comidas rápidas vende OnLine su producto estrella "Pastelitos Demenciales". Para medir la efectividad del servicio de ventas por internet, cada vez que se realiza una venta, la cantidad de pasteles vendidos es almacenada para su posterior análisis.

En el transcurso de un día, la empresa necesita conocer la cantidad total de pasteles vendidos, la cantidad promedio, la mayor cantidad y la menor.

SOLUCIÓN:

Supongamos que se hicieron 10 ventas en un día.

Actividades:

SOLUCIÓN:

Supongamos que se hicieron 10 ventas en un día.

Actividades:

Dibuje un arreglo de tamaño 10, enumere cada posición iniciando con el índice 0.

SOLUCIÓN:

Supongamos que se hicieron 10 ventas en un día.

Actividades:

Dibuje un arreglo de tamaño 10, enumere cada posición iniciando con el índice 0.

Las cantidades vendidas fueron: 32, 25, 7, 421, 24, 48, 60, 95, 300 y 91.

SOLUCIÓN:

Supongamos que se hicieron 10 ventas en un día.

Actividades:

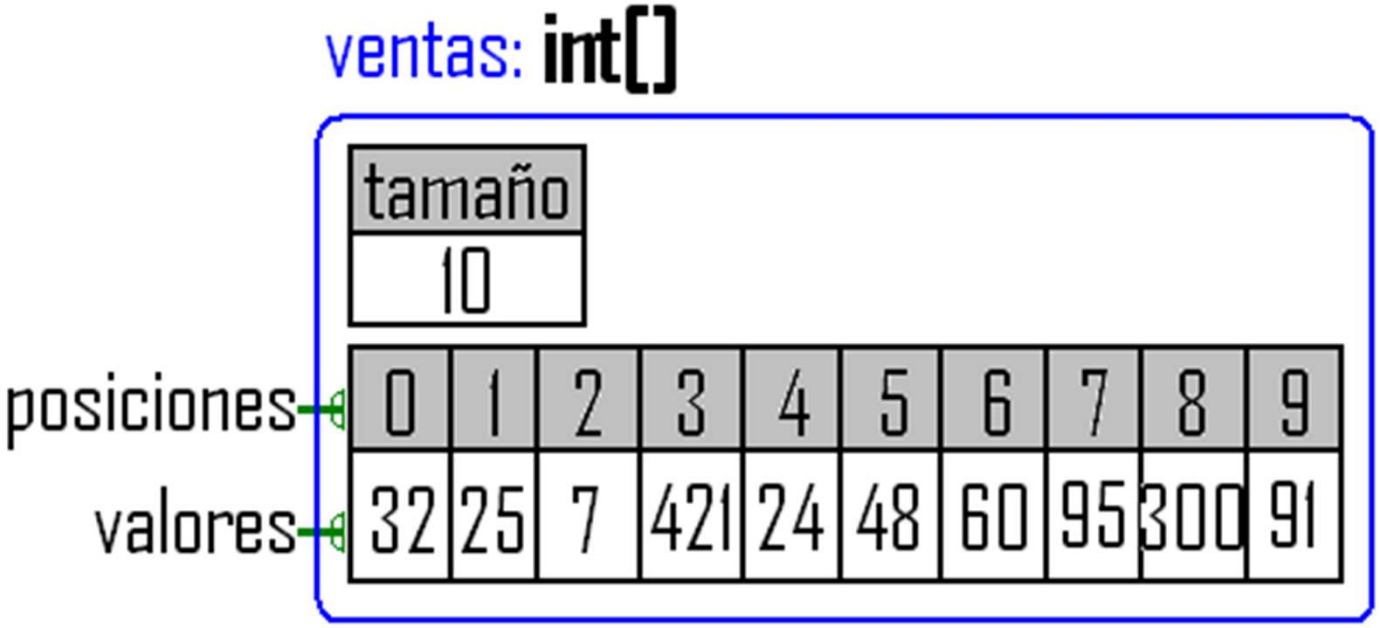
Dibuje un arreglo de tamaño 10, enumere cada posición iniciando con el índice 0.

Las cantidades vendidas fueron: 32, 25, 7, 421, 24, 48, 60, 95, 300 y 91.

Asigne los valores en el arreglo.

SOLUCIÓN:

El arreglo con las cantidades vendidas queda así:



SOLUCIÓN:

Ahora, en Visual **C#**:

```
int[ ] ventas = new int[ 10 ] ;
ventas[ 0 ] = 32 ;
ventas[ 1 ] = 25 ;
ventas[ 2 ] = 7 ;
        ...
```

CONSULTA A REALIZAR

En **C#**:

¿Qué es *try – **catch***?

¿Para qué sirven las **Excepciones**?

RECORRER UN ARREGLO DE 1 DIMENSIÓN - CONTEO

```
public int recorrerContar( String[ ]  nombres ) {
    int contador = 0 ;
    for( int i = 0 ; i < nombres.Length ; i ++ ) {
        if( nombres[ i ].Equals( "Laura" ) ) {
            contador ++ ;
        }
    }
    return  contador ;
}
```

También:

```
public int recorrerContar( String[ ]  nombres ) {
      int contador = 0 ;
      for( int i = 0 ; i < nombres.Length ; i ++ ) {
           if( nombres[ i ]  ==  "Laura" ) {
                contador ++ ;
          }
       }
       return  contador ;
}
```

CONSTRUIR UN REPORTE DEL ARREGLO DE 1 DIMENSIÓN

```
public String reporteArreglo( String[ ]  nombres ) {
        String reporte = "" ;      // Inicie con un reporte vacío.
        for( int q = 0 ; q < nombres.Length ; q ++ ) {
                reporte += nombres[ q ] + "\n" ;
                //reporte += nombres[ q ] +
                        //System.Environment.NewLine ;
        }
        return  reporte ;
}
```

OPERACIONES BÁSICAS CON ARREGLOS DE 1 DIMENSIÓN

Búsqueda.

Contar elementos con algún patrón.

Mayor, Menor.

Ordenar.

BÚSQUEDA SOBRE ARREGLOS DE 1 DIMENSIÓN

Verificar si en un arreglo unidimensional se encuentra almacenado un dato solicitado por el usuario.

Búsqueda secuencial.

BÚSQUEDA SOBRE ARREGLOS DE 1 DIMENSIÓN

```
public String buscarDato( String[ ]  conjuntoDatos, String datoBuscar ) {
    String mensaje = "Sí se encontró el dato solicitado" ;
    for( int k = 0 ; k < conjuntoDatos.Length ; k ++ ) {
        if( conjuntoDatos[ k ].Equals( datoBuscar ) ) {
                return  mensaje ;      // Retorne el mensaje.
         }
    }
    return "No se encontró el dato solicitado" ;
}
```

También:

```
public String buscarDato( String[ ]  conjuntoDatos, String datoBuscar ) {
    String mensaje = "Sí se encontró el dato solicitado" ;
    for( int k = 0 ; k < conjuntoDatos.Length ; k ++ ) {
        if( conjuntoDatos[ k ]  ==  datoBuscar ) {
                return  mensaje ;      // Retorne el mensaje.
         }
    }
    return "No se encontró el dato solicitado" ;
}
```

EJERCICIO

Implemente un método en C# que determine si algún estudiante tuvo como definitiva una nota superior a 4.5, en el curso de Algoritmia y Programación.

CONTAR ELEMENTOS QUE CUMPLEN UNA CONDICIÓN

¿Cuántos estudiantes aprobaron la asignatura Física 1?

```
// Con ciclo for:
public  int  cuantosGanaron( double[ ]  notas ) {
        int  aprobaron = 0 ;
        for( int j = 0 ; j < notas.Length ; j ++ ) {
                if( notas[ j ] >= 3.0 ) {        // En el rango 0.0 a 5.0
                   aprobaron ++ ;
                }
        }
        return aprobaron ;
}
```

CONTAR ELEMENTOS QUE CUMPLEN UNA CONDICIÓN

¿Cuántos estudiantes reprobaron la asignatura Inglés I?

```
// Con ciclo while:
public  int  cuantosPerdieron( double[ ]  notas ) {
        int  reprobaron = 0,  pos = 0 ;
        while( pos  <  notas.Length ) {
                if( notas[ pos ]  <  3.0 ) {     // En el rango 0.0 a 5.0
                   reprobaron ++ ;
                }
                pos ++ ;
        }
        return reprobaron ;
}
```

EJERCICIO

Implemente un método en C# que cuente el número de estudiantes del curso Álgebra Lineal que tuvieron un desempeño sobresaliente (superior a 4.0).

ELEMENTO MAYOR O MENOR

Si se conoce el dominio ó rango de los datos:

Por ejemplo, si las notas están entre 0.0 y 5.0.

No se conoce el dominio ó rango de los datos:

Por ejemplo, si de las poblaciones de varias ciudades no se conocen sus valores mínimo y máximo.

ELEMENTO MAYOR O MENOR

Se conoce el dominio ó rango de los datos:

La variable que guardará el mayor se inicia en el valor más bajo.

La variable que guardará el menor se inicia en el valor más alto.

De allí en adelante, se compara el dato a procesar con lo que se lleva como mayor o como menor.

Cada vez que se encuentre un nuevo valor mayor o menor, estas variables (mayor y menor) deben actualizar sus valores.

ELEMENTO MAYOR

Conocido el dominio ó rango de un conjunto de valores, encuentre el mayor valor.

```
public  double  mayorValorConociendoRango( double[ ]  notas )  {
        double  mayor = 0.0 ;          // Si notas está en el rango 0.0 a 5.0
        for( int j = 0 ; j < notas.Length ; j ++ ) {
                if( notas[ j ]  >  mayor ) {
                   mayor = notas[ j ] ;
                }
        }
        return  mayor ;
}
```

ELEMENTO MENOR

Conocido el dominio ó rango de un conjunto de valores, encuentre el menor valor.

```
public  double  menorValorConociendoRango( double[ ]  notas )  {
        double  menor = 5.0 ;          // Si notas está en el rango 0.0 a 5.0
        for( int k = 0 ; k < notas.Length ; k ++ ) {
                if( notas[ k ]  <  menor ) {
                   menor = notas[ k ] ;
                }
        }
        return  menor ;
}
```

ELEMENTO MAYOR O MENOR

No se conoce el dominio ó rango de los datos:

Las variables que guardarán el mayor y el menor valor, se deben iniciar en el primer valor conocido.

De allí en adelante, se compara el dato a procesar con lo que se lleva como mayor o como menor.

Cada vez que se encuentre un nuevo valor mayor o menor, estas variables (mayor y menor) deben actualizar sus valores.

ELEMENTO MAYOR

Sin conocer el dominio ó rango de un conjunto de valores, encuentre el mayor valor.

```
public double mayorValorSinConocerRango( double[ ]  valores ) {
        double  mayor  =  valores[ 0 ] ;        // Suponga que el primero es el mayor.
        for( int k = 1 ; k < valores.Length ; k ++ ) {
                if( valores[ k ]  >  mayor ) {
                   mayor  =  valores[ k ] ;    // Actualice el mayor.
                }
        }
        return  mayor ;
}
```

ELEMENTO MENOR

Sin conocer el dominio ó rango de un conjunto de valores, encuentre el menor valor.

```
public double menorValorSinConocerRango( double[ ]  valores ) {
        double  menor  =  valores[ 0 ] ;        // Suponga que el primero es el menor.
        for( int p = 1 ; p < valores.Length ; p ++ ) {
                if( valores[ p ]  <  menor ) {
                   menor  =  valores[ p ] ;    // Actualice el menor.
                }
        }
        return  menor ;
}
```

EJERCICIO

Implemente en C#: (**a**) un método que reciba un arreglo de notas de un curso de Algoritmia y Programación, para calcular la nota promedio del curso; (**b**) otro método que reciba el arreglo de notas y la nota promedio, para determinar cuántas notas son mayores que la nota promedio y (**c**) el método principal, que procese el arreglo, llame los métodos y muestre los resultados.

Ordenamientos

ORDENAMIENTOS

Consiste en dar organización particular a los datos dentro del arreglo. El ordenamiento numérico puede ser ascendente o descendente. El de cadenas de caracteres, es alfabético.

Métodos populares:

Burbuja, Inserción, Selección,

QuickSort, ShellSort, MergeSort.

METODO DE ORDENAMIENTO BURBUJA

Conjunto de valores a ordenar

12	-5	4	-8

Primera pasada

12	-5	4	-8	→	-5	12	4	-8
-5	12	4	-8	→	-5	4	12	-8
-5	4	12	-8	→	-5	4	-8	12

Segunda pasada

-5	4	-8	12		
-5	4	-8	12	→	-5 -8 4 12

No hay intercambio

Tercera pasada

-5	-8	4	12		
-5	-8	4	12	→	-8 -5 4 12
-8	-5	4	12		
-8	-5	4	12		

No hay intercambio

No hay intercambio

No hay intercambio

Cuarta pasada

-8	-5	4	12
-8	-5	4	12
-8	-5	4	12

No hay intercambio

No hay intercambio

No hay intercambio

ALGORITMO DE ORDENAMIENTO: BURBUJA

```
public double[ ] ordenamientoNumericoAscendente( double[ ] vector ) {
        double temp = 0.0 ;
        for( int i = 0 ; i < vector.Length - 1 ; i ++ ) {
                for( int j = 0 ; j < vector.Length – 1 - i ; j ++ ) {
                        if(vector[ j ] > vector[j + 1] ) {
                                temp = vector[ j ] ;    // Intercambio.
                                vector[ j ] = vector[j + 1] ;
                                vector[j + 1] = temp ;
                        }
                }
        }
        return vector ;
}
```

ALGORITMO DE ORDENAMIENTO: BURBUJA

```
public double[ ] ordenamientoNumericoDescendente( double[ ] au ) {
        double auxi = 0.0 ;
        for( int i = 0 ; i < au.Length - 1 ; i ++ ) {
                for( int j = 0 ; j < au.Length – 1 - i ; j ++ ) {
                        if(au[ j ] < au[j + 1] ) {
                                auxi = au[ j ] ;          // Intercambio.
                                au[ j ] = au[j + 1] ;
                                au[j + 1] = auxi ;
                        }
                }
        }
        return au ;
}
```

ALGORITMO DE CLONAMIENTO NUMÉRICO

```
// Método para clonar un arreglo numérico recibido en el
// parámetro:
public double[ ] clonarArregloNumerico( double[ ] array ) {
        double[ ]  clon  =  new  double[ array.Length ] ;
        // Hago una copia del parámetro (array):
        for( int k = 0; k < clon.Length; k ++ ) {
                clon[ k ] = array[ k ] ;
        }
        return  clon ;
}
```

ALGORITMO DE CLONAMIENTO ALFABÉTICO

```
// Método para copiar un arreglo de cadenas de caracteres
// recibido en el parámetro:
public String[ ]  clonarArregloDeCadenas( String[ ]  vector ) {
        String[ ]  copia = new String[ vector.Length ] ;
        // Hago un clon del parámetro (vector):
        for( int n = 0; n < copia.Length; n ++ ) {
                copia[ n ] = vector[ n ] ;
        }
        return  copia ;
}
```