GUÍA DE TRABAJO – ARREGLOS UNIDIMENSIONALES

1. INFORMACIÓN DE LA ASIGNATURA

Asignatura:	Lógica y Representación I			
Código:	2554208		**Tipo de Curso:**	Obligatorio
Plan de Estudios:	5		**Semestre:**	1
Créditos: 3	**TPS: 6**	**TIS: 3**	**TPT: 64**	**TIT: 32**

2. OBJETIVO

Unidad:	Estructuras de Datos Estáticas - Arreglos Unidimensionales
Objetivo de la asignatura:	Desarrollar en el estudiante la habilidad para plantear soluciones lógicas a problemas computacionales que deriven en la implementación de programas en un lenguaje de programación
Resultados de aprendizaje abordados:	▪ Usar las estructuras de datos para almacenar y manipular los datos asociados a un problema computacional ▪ Escribir algoritmos y programas que solucionen problemas computacionales
Contenido temático:	▪ Introducción a los arreglos unidimensionales ▪ Declaración, almacenamiento y acceso de datos en arreglos unidimensionales ▪ Ordenamiento de arreglos: burbuja, selección, e inserción. ▪ Búsqueda lineal y búsqueda binaria ▪ Visualización de los datos almacenados en arreglos unidimensionales

3. DESARROLLO TEÓRICO

Las variables permiten almacenar la información en posiciones de memoria a las cuales se puede acceder por medio de su identificador. Sin embargo, en muchos casos prácticos, se requiere almacenar y procesar varios valores, lo que implica el uso de un gran número de variables y dificulta el uso de ciclos. Por ejemplo, si se necesita leer 10 números, calcular su promedio y encontrar cuántos de esos números están por encima del promedio; el algoritmo debe leer estos 10 números, luego calcular el promedio para después comparar cada uno de los números con el promedio. Por lo tanto, es necesario guardar los 10 números en variables diferentes.

¡Sería necesario declarar 10 variables y escribir repetidamente las mismas líneas de pseudocódigo 10 veces! Sin embargo, hay una forma más organizada y eficiente de lograr esto, usando arreglos unidimensionales o vectores.

3.1 ASPECTO BÁSICOS SOBRE LOS ARREGLOS

A continuación, veremos lo que es un arreglo e introduciremos las operaciones básicas que podemos aplicar sobre estos.

3.1.1 DEFINICIÓN DE ARREGLO

Un arreglo es una variable que contiene un número fijo y finito de valores, todos del mismo tipo. Por ejemplo, supongamos que deseamos almacenar 10 números enteros. Intuitivamente, para hacer esto deberíamos declarar 10 variables individuales de tipo entero, por ejemplo:

```
# Declaración de 10 variables de tipo entero
Entero x0, x1, x2, x3, x4, x5, x6, x7, x8, x9
```

¡Ahora imagínese que no son 10 número enteros, sino 1000! Sería bastante trabajo hacer lo mismo 1000 veces, ¿no?

Los arreglos nos ayudan con ese problema puesto que en lugar de crear las 10 o 1000 variables independientes se crea una sola variable, a la que en este ejemplo llamamos `x`:

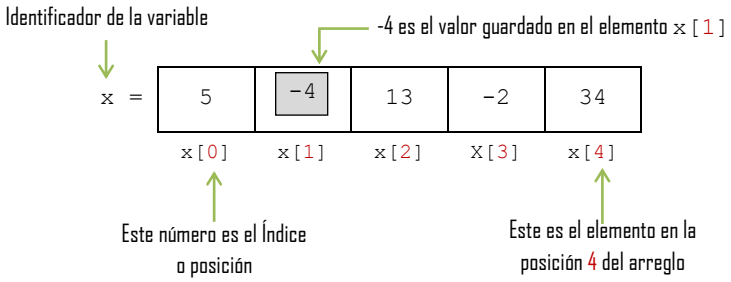
```
# Declaración de una variable que es un arreglo que almacena 10 enteros
Entero x(10)
```

En la declaración anterior, `x` es el identificador de la variable y el número entre *paréntesis* indica cuántos valores se pueden almacenar en el arreglo. Recuerde que el identificador de toda variable siempre debe seguir unas recomendaciones: debe empezar con una letra o un guion bajo (_), los siguientes caracteres pueden ser una combinación de letras, números y guiones bajos, pero no puede tener espacios ni caracteres especiales.

Declaremos de nuevo el arreglo y almacenemos en él los valores enteros 5, -4, 13, -2 y 34. El seudocódigo para hacerlo es el siguiente:

```
# Declaramos un arreglo para almacenar número enteros
Entero x(5)

x[0] = 5
x[1] = -4
x[2] = 13
x[3] = -2
x[4] = 34
```

En este ejemplo, el arreglo `x`, puede verse de manera gráfica así:



Como se muestra, un arreglo es una estructura dividida en celdas en las se puede guardar un valor. Cada celda es un elemento del arreglo y para acceder a ellos se utiliza un índice (al que también se le llama posición). Se debe tener presente que la numeración de los índices comienza con 0. Es por esto que para acceder al primer elemento se utiliza el índice 0 (`x[0]`), mientras que para acceder al segundo elemento se utiliza el índice 1 (`x[1]`).

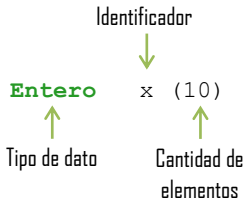
Importante: recuerde que en un arreglo todos los datos que se almacenan deben ser del mismo tipo. Además, Es importante que tenga presente que la numeración de los índices inicia en el valor 0. Por esta razón el primer elemento del arreglo está en la posición 0 y, en el caso del ejemplo dado, el último elemento estará en la posición 4, es decir es el elemento `x[4]`.

3.1.2 Declaración de Arreglos

Para usar un arreglo en un programa, es necesario declarar la variable de referencia, especificar el tipo y el tamaño de este. El tamaño hace referencia a la cantidad de elementos que va a tener el arreglo y se especifica usando paréntesis en la declaración de la variable. Esta es la forma general para declarar un arreglo en pseudocódigo:

```
<Tipo_de_dato> identificador_del_arreglo(cantidad_de_elementos)
```

En la sección anterior ya utilizamos la declaración de un arreglo, cuando creamos la variable x:



Importante: cuando se declara un arreglo se debe indicar el tipo de dato de los elementos y el número de celdas (o casillas) que tendrá el arreglo, es decir, su tamaño. Recuerde, el tamaño de un arreglo no se puede modificar después de creado.

Cuando se conocen los elementos que se almacenarán en un arreglo, podemos declararlo enumenrando, entre llaves y separados por comas, los valores que se almacenarán en este. Por ejemplo, declaremos un arreglo de cinco números enteros que ya se conocen:

```
# Así se declara un arreglo con valores predefinidos
Entero mi_arreglo = {7, 12, -5, 4, 2}
```

Gráficamente, el arreglo que hemos creado se ve así:

mi_arreglo =	7	12	-5	4	2
	0	1	2	3	4

Note que el primer valor en las llaves (que es el número 7) se almacena en la posición 0, el siguiente (que es el 12) se almacena en la posición 1 y así sucesivamente.

3.1.3 Almacenando valores en un arreglo

Para almacenar un valor en un arreglo se debe usar el identificador de la variable, la posición en la que se va a almacenar el valor (entre corchetes **[ ]**) y el valor a guardar:

```
identificador_del_arreglo[posición] = valor
```

Veamos un ejemplo. A continuación, se declara un arreglo denominado `arr` en el que se almacenaran 4 número reales:

```
# Declaramos un arreglo para almacenar 4 valores reales
Real arr(4)

# En cada posición del arreglo almacenamos un valor
arr[0] = 3.5
arr[1] = 2.8
arr[2] = -5.6
arr[3] = -0.5
```

Gráficamente, el arreglo resultante se ve así:

arr =	3.5	2.8	-5.6	-0.5
	0	1	2	3

3.1.4 ACCEDIENDO A LOS VALORES EN UN ARREGLO

Para acceder a un valor almacenado en un arreglo basta con usar el identificador de la variable y el índice del elemento al que queremos acceder:

```
identificador_del_arreglo[posición] = valor
```

Veamos en un ejemplo. Calculemos el promedio de los números almacenados en el arreglo `arr` que usamos en el punto anterior.

```
# Recuerde que para calcular el promedio primero debemos sumar los números
# Por eso declaramos dos variables, una para almacenar la suma y otra para el promedio
Real suma = 0, promedio

# Acumulamos en la variable suma todos los valores del arreglo
# Por lo tanto, aquí accedemos a cada elemento del arreglo usando su índice en corchetes
suma = suma + arr[0]
suma = suma + arr[1]
suma = suma + arr[2]suma = suma + arr[3]

# Calculamos el promedio
promedio = suma/4
```

Juguemos ahora un poco con los índices de dos arreglos:

```
# Declaramos dos arreglos de numero enteros, cada uno con 8 elementos.
Entero A = {2, 5, -1, 0, 7, 10, 17, -3}
Entero B = {0, 1, 15, -8, 5, 7, -2, 0}

B[2] = A[0]*3 # En la posición 2 de B se almacena el valor 6
B[3] = B[3] + A[4] # En la posición 3 de B se almacena el valor -1
B[A[3]] = A[5] DIV B[1] # En la posición 0 de B se almacena el valor 10
B[A[6] MOD A[5]] = A[A[7]+A[5]] + B[2] * -3 # En la posición 7 de B se almacena el valor -19
```

Al terminar estas operaciones, el arreglo B queda así:

B =	10	5	6	-1	5	7	-2	-19
	0	1	2	3	4	5	6	7

Importante: El índice o posición de un arreglo siempre es un número entero, es decir, no existe un elemento en la posición 1.5 en el arreglo B y tampoco existe un elemento en la posición 2.0.

3.1.5 EJEMPLOS DE OPERACIONES BÁSICAS CON ARREGLOS

Con base en lo anterior, existen dos operaciones básicas que son el almacenamiento de valores en un arreglo y el acceso a los mismo. No obstante, existen otras operaciones sobre los arreglos que se construyen con base en esas dos operaciones básicas. Veamos algunos ejemplos …

Ejemplo – Llenando un Arreglo

- Desarrolle un algoritmo que pida al usuario 10 números y los almacene en un arreglo.

Para el desarrollo de este ejercicio se requiere pedir 10 veces un valor entero a un usuario. Es decir, necesitamos hacer 10 veces la misma operación. Es por esto que la forma más rápida de escribir la solución a este problema es utilizando un ciclo 10 veces pida un valor al usuario y lo almacene en un arreglo.

```
# Declaramos las variables que requerimos
Entero x(10), i

# Usamos un ciclo Para ya que sabemos cuántas veces se van a repetir las instrucciones
Para i=0 hasta 10 incremento 1 Haga
  Escribir("Ingrese un número entero: ") # Mostramos un mensaje al usuario
  Leer(x[i]) # El valor que ingresa el usuario se almacena en la posición i de arreglo
Fin_Para
```

Tenga en cuenta que al usar la expresión `x[i]`, estamos accediendo al elemento `i` del arreglo, y como `i` toma valores del `0` al `9` en el ciclo, entonces estaríamos accediendo a todos los elementos del arreglo `x`.

Ejemplo – Recorriendo un Arreglo

- Desarrolle un algoritmo que muestre los números pares que se han almacenado en el arreglo x del ejemplo anterior.

Para solucionar este problema debemos recorrer el arreglo, es decir, debemos acceder a cada elemento del arreglo para verificar si el valor almacenado en él es par para mostrarlo. Como necesitamos acceder a todos los elementos del arreglo, debemos hacer uso de un ciclo.

```
# En este caso usaremos un ciclo Mientras, para variar :)
Escribir("Los número pares que hay en el arreglo son: ")
i=0
Mientras (i<10) Haga
  # Accedemos a la posición i del arreglo para verificar si el número es par y mostrarlo
  Si (x[i] MOD 2 == 0) Entonces
    Escribir(x[i])
  Fin_Si
  i=i+1
Fin_Mientras
```

En este caso, para mostrar un elemento del arreglo debemos cerciorarnos de que el elemento es par. Es por esto que usamos un condicional en el que verificamos si el residuo de la división del elemento `x[i]` con 2 es igual a 0.

Ejemplo – Encontrando el elemento de mayor valor

- Desarrolle un algoritmo que muestre cuál es el mayor valor almacenado en el arreglo e indique en qué posición está este

Este es un problema clásico que tiene muchas aplicaciones cuando se trabaja con arreglos. La idea principal es buscar el número más grande del arreglo y la posición que ocupa. Como debemos ***buscar***, eso implica recorrer el arreglo y por tanto necesitamos un ciclo. Adicionalmente, se requieren dos variables: una para almacenar el mayor y otra para almacenar su posición. Veamos la solución.

```
# Primero declaramos las variables que requerimos
Entero valor_mayor, posicion_mayor, i

# En este caso usamos un ciclo mientras, por eso iniciamos la variable antes del ciclo
i=0

# Asumimos que el mayor está en la posición 0 del arreglo, y lo usaremos para las comparaciones
Valor_mayor = x[0]
Posición_mayor = 0

Mientras (i<10) Haga
  # Si el elemento i es mayor que valor_mayor
  # Lo reemplazamos porque hemos encontrado un nuevo mayor
  Si (x[i] > valorMayor) Entonces
    valor_mayor = x[i]
    posicion_mayor = i
  Fin_Si
  i=i+1
Fin_Mientras

# Al final mostramos el mayor y su posición:
Escribir("El mayor valor almacenado en el arreglo es ", valor_mayor)
Escribir("Y este se encuentra en la posición ", posición_mayor ," del arreglo")
```

Considere las siguientes situaciones y de una respuesta, acompañada de una prueba de escritorio:

- ¿Qué pasa si en el arreglo el mayor se repite 3 veces?
- ¿Qué hay que modificar para que el algoritmo en lugar de encontrar el mayor encuentre el menor?

Pasemos a un par de ejemplos de aplicación de este tipo de algoritmos en física y matemáticas.

Ejemplo – Calcular la Trayectoria de un Proyectil

La trayectoria que describe un proyectil es el movimiento parabólico que sigue el proyectil al ser sometido a una velocidad inicial y a las fuerzas inherentes al medio en el que se desplaza, principalmente la fuerza de la gravedad.

Con base en lo anterior haga un algoritmo que, a partir de la velocidad inicial (en `x` e `y`) y el ángulo de tiro, se calcule tanto la posición del proyectil (en x e y) como el tiempo transcurrido, durante 20 unidades de tiempo. Esta información se debe almacenar en tres vectores.

En este caso debemos recordar que en el movimiento parabólico tienen dos componentes: una para el movimiento rectilíneo del proyectil, es decir el movimiento en el eje x, y otra para un movimiento con aceleración constante, que es el movimiento en y.

En este caso, la ecuación de movimiento en el eje x es:

$$x = x_0 + v_x \cdot t$$

Y la ecuación de movimiento en el eje y es:

$$y = y_0 + v_{0y} \cdot t + \frac{1}{2} \cdot a_y \cdot t^2$$

Al ser un movimiento parabólico, la aceleración en y es la gravedad, es decir que $a_y = 9.8$. La velocidad en x (v_x) y la velocidad inicial en y (v_{0y}) son datos que debe dar el usuario, al igual que la posición inicial en x e y. Procedemos entonces con la solución.

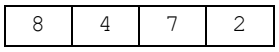
```
# Primero declaramos las variables que requerimos. Los arreglos deben tener 20 elementos
Real x(20), y(20), t(0), vx, v0y, delta_T, x0, y0, i

# Paso 1: Pedimos al usuario los valores de inicio
Escribir("Ingrese la coordenada en X del punto de partida del proyectil: ")
Leer(x0)

Escribir("Ingrese la coordenada en Y del punto de partida del proyectil: ")
Leer(y0)

Escribir("Ingrese la velocidad del proyectil en X en m/s: ")
Leer(vx)

Escribir("Ingrese la velocidad del proyectil en Y en m/s: ")
Leer(v0y)

Escribir("Cada cuántos segundos quiere calcular la posición del proyectil? ")
Leer(delta_T)

# Paso 2: Creamos el ciclo, el cual debe tener 20 repeticiones
Para i=0 hasta 20 incremento 1 Haga
  # Paso 3: calculamos t, de acuerdo con el valor de delta_T:
  t[i] = delta_T*i;

  # Paso 4: calculamos X con base en la fórmula de movimiento rectilíneo
  x[i] = x0 + vx*t[i]

  # Paso 5: calculamos Y con base en la fórmula de movimiento con aceleración constante
  y[i] = y0 + v0y*t[i] + (1/2)*(9.8)*(t[i]^2)

  # Paso 6: mostramos la posición del proyectil
  Escribir("En t=",t[i]," seg., el proyectil está en el posición x=",x[i]," y=",y[i])

Fin_Para
```

3.2 Ordenamiento de Arreglos

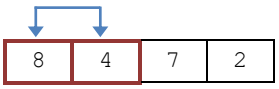
El ordenamiento es una operación común cuando trabajamos con arreglos. Existen diversos algoritmos para ordenar los elementos de un arreglo; sin embargo, en esta sección sólo presentamos los 3 algoritmos más simples que son: el método de la burbuja, el método de selección y el método de inserción.

3.2.1 Método de la burbuja

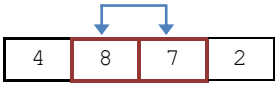
El método de la burbuja es un método simple de ordenamiento en el que múltiples veces se comparan las parejas de elementos adyacentes del arreglo, intercambiándolos cuando estos no están en el orden correcto. Veamos cómo funciona el algoritmo antes de hacer la codificación de este. Suponga que tenemos el siguiente arreglo de números enteros y deseamos ordenarlo de menor a mayor.



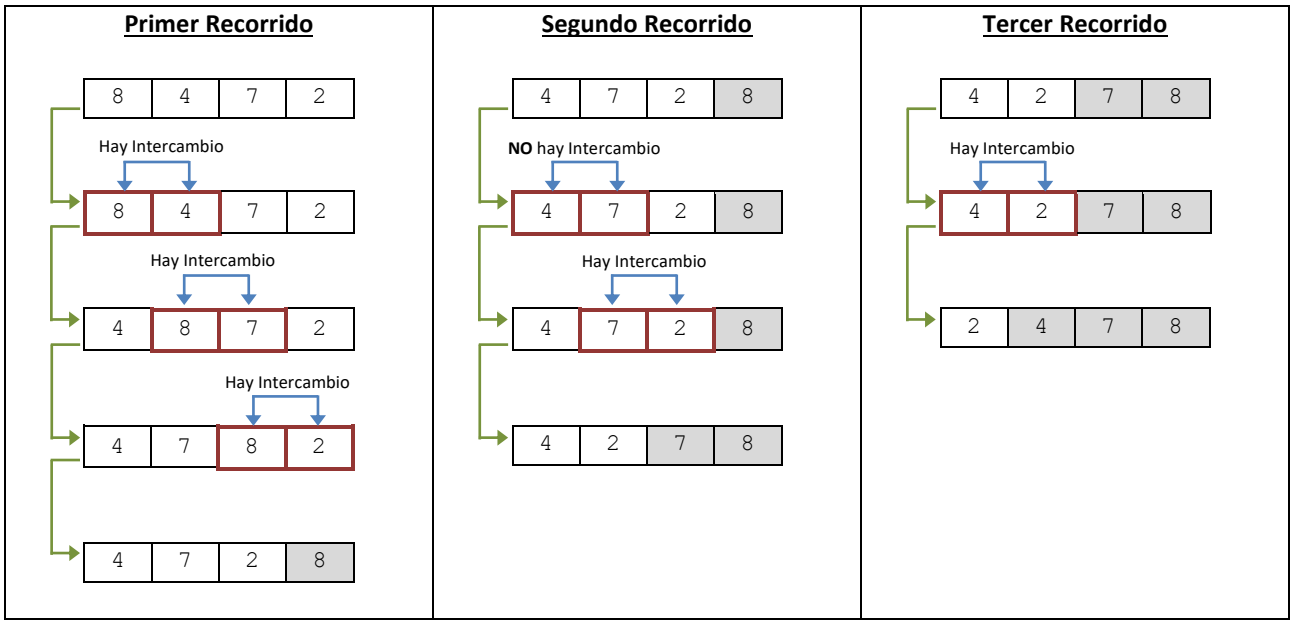
En este caso, una vez ordenado el arreglo el menor debe estar en la primera posición y el mayor en la última. Para lograr esto comparamos los elementos del arreglo de a dos en dos.



Al comparar esos dos elementos vemos que no están en el orden que deseamos, así que los intercambiamos y pasamos a comparar la siguiente pareja:



Nuevamente, hacemos la comparación y como el menor está a la derecha intercambiamos los dos valores. Esta operación se debe hacer múltiples veces, haciendo tantos recorridos del arreglo, como elementos tenga este. Veamos cómo queda el ejemplo completo:



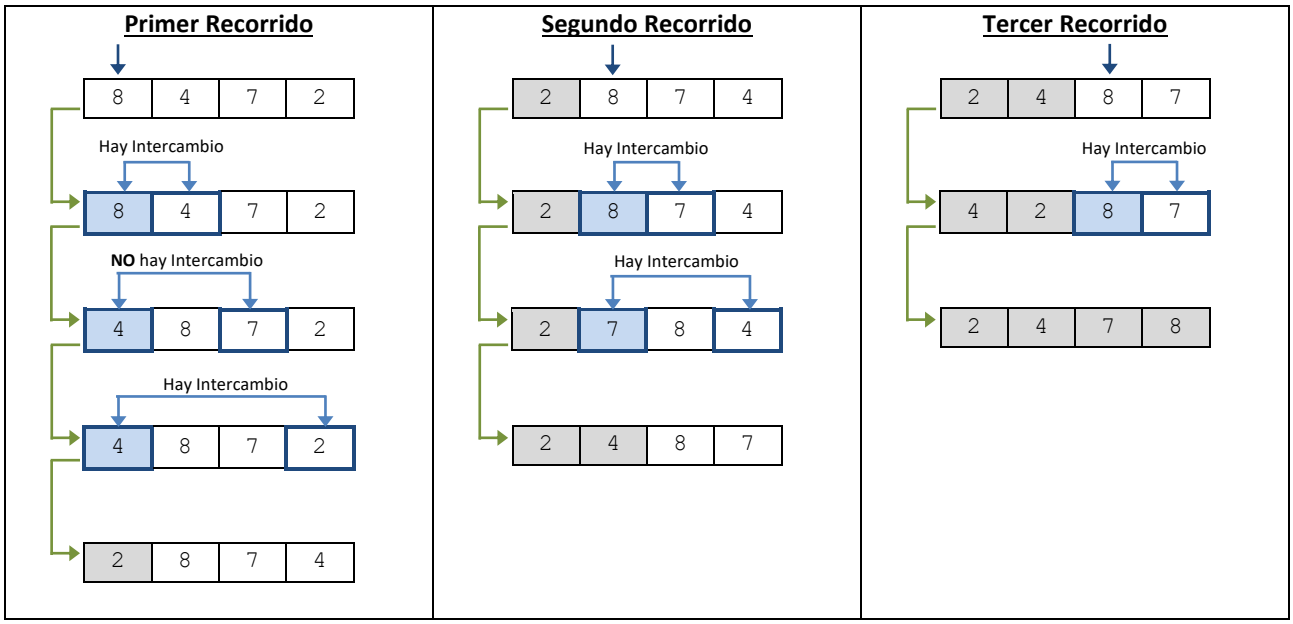
Note que, si el arreglo tiene `n` elementos, en el primer recorrido llegamos hasta el elemento `n`, pero en el segundo recorrido llegamos hasta el elemento `n-1` y así sucesivamente. El código para ordenar un arreglo usando el método de la burbuja es el siguiente:

```
# Este algoritmo requiere dos ciclos: uno para los recorridos el cual controlaremos con una
# variable denominada i y otro para las comparaciones, el cual controlaremos con una variable
# llamada j. Además, note que a medida que aumentan los recorridos disminuyen el número de
# comparaciones puesto que los elementos ya ordenados no es necesario volverlos a comparar.

# Declaramos las variables que requerimos.
Entero: n=4, x(4), i, j, aux

# Asignamos los valores al arreglo
x = {8, 4, 7, 2}

# La i controla el número de recorridos que son n-1
Para i=0 hasta n-1 incremento 1 Haga

  # La j la usamos para comparar los pares de elementos.
  # A medida que aumenta i, j disminuye su alcance
  Para j=0 hasta n-1-i incremento 1 Haga

    # Si el elemento de la izquierda es mayor que el de la derecha los intercambia
    Si (x[j] > x[j+1]) Entonces
      aux = x[j]
      x[j] = x[j+1]
      x[j+1] = aux
    Fin_Si
  Fin_Para
Fin_Para
```

3.2.2 Método de Selección

El método de selección consiste en buscar el elemento menor del arreglo y ubicarlo en la primera posición de este. Luego, se busca el segundo elemento menor y se ubica en la segunda posición y así sucesivamente hasta lograr que todos los elementos queden ordenados de menor a mayor.

Para realizar este ordenamiento fijamos una de las posiciones del arreglo (la que se resalta en azul) y buscamos el menor desde esa posición hacia adelante. Veámoslo con un ejemplo.



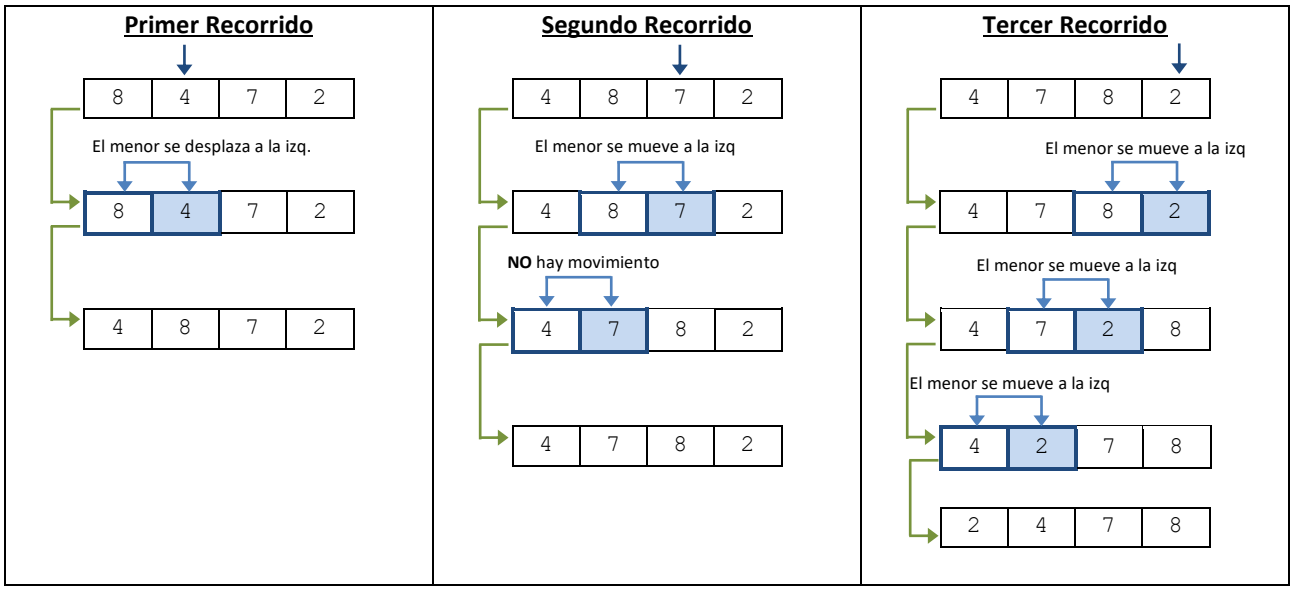
El código para ordenar un arreglo usando el método de selección es el siguiente:

```
# Este algoritmo requiere dos ciclos: uno en el que se fija la casilla que se está ordenando
# (la que se resalta en azul claro) y para la que usamos una variable denominada i. El otro
# ciclo lo usamos para especificar la casilla con la que se hace la comparación buscando el
# menor. Ese ciclo lo controlamos con la variable j.

# Declaramos las variables que requerimos.
Entero n=4, x(4), i, j, aux

# Asignamos los valores al arreglo
x = {8, 4, 7, 2}

# La i controla la casilla que se está ordenando
Para i=0 hasta n-1 incremento 1 Haga

  # La j la usamos para comparar las otras casillas con la casilla i.
  # Note que la j siempre empieza en la casilla siguiente a la que se tiene fija
  Para j=i+1 hasta n incremento 1 Haga

    # Si el elemento en j es menor que el elemento i, los intercambiamos
    Si (x[j] < x[i]) Entonces
      aux = x[j]
      x[j] = x[i]
      x[i] = aux
    Fin_Si
  Fin_Para
Fin_Para
```

3.2.3 Método de Inserción

El método de inserción ordena de manera progresiva el arreglo de tal manera que recorre el arreglo buscando en qué posición debe ir un elemento y desplazando los elementos siguientes hacia derecha del arreglo. Así, el método inicia comparando los dos primeros elementos y si el segundo es menor que el primero, entonces el segundo se desplaza a la primera posición y el que estaba en la primera posición se mueve a la siguiente posición, es decir a la segunda posición. De este modo, los dos primeros elementos quedan organizados. Lo siguiente que se hace es tomar el tercer elemento del arreglo y compararlo con los anteriores para identificar en qué posición debe ir. Los elementos que sean mayores

se mueven hacia la derecha y se ubica el elemento en la posición correspondiente. Esto se repite hasta lograr que el arreglo esté organizado. A continuación, se ilustra cómo funciona el algoritmo.



El código para ordenar un arreglo usando el método de inserción es el siguiente:

```
# Este algoritmo requiere dos ciclos: uno que señala al número que se está ordenando
# (que se resalta en azul claro) y para el usamos una variable denominada i. El otro
# ciclo lo usamos para especificar la casilla con la que se hace la comparación y determinar
# si se debe mover a la izquierda o no. Ese ciclo lo controlamos con la variable j.

# Declaramos las variables que requerimos.
Entero: n=4, x(4), i, j, aux

# Asignamos los valores al arreglo
x = {8, 4, 7, 2}

# La i controla el elemento que se va a ordenar. Note que empezamos por la posición 1.
Para i=1 hasta n incremento 1 Haga
  aux = x[i]

  # La j la usamos para comparar los elementos que están hacia la izq e inicia en el elemento
  # que está a la izq de la i. Además, el índice decrece para ir moviendo el menor a la izq.
  j=i-1
  Mientras (j >= 0 and x[j]>aux) Haga
    x[j+1] = x[j]
    j=j-1
  Fin_Mientras
  x[j+1] = aux
Fin_Para
```

> **Extiende tu conocimiento ...**
> Si quieres ver una explicación más en detalle y una animación de cómo funcionan los métodos de ordenamiento te invito a que visites los siguientes enlaces:
> - Ordenamiento por Selección: https:# bit.ly/2YMobXk
> - Ordenamiento de Inserción: https:# bit.ly/3e8mIRf

3.3 Búsqueda en un Arreglo

Buscar información en un arreglo requiere recorrer el arreglo para determinar si un elemento específico está o no almacenado en el arreglo. La estrategia más simple para realizar una búsqueda es la búsqueda secuencial. En la búsqueda secuencial se compra el valor que se está buscando con cada uno de los elementos del arreglo, de manera

secuencial. Por esto la búsqueda empieza en el primero elemento del arreglo y uno a uno se van comparando los elementos con el valor buscado hasta que éste se encuentra o hasta que se recorre el arreglo por completo. Normalmente, la búsqueda secuencial muestra o almacena la posición en el arreglo en la que se encontró el elemento buscado.

El código de una búsqueda secuencial es el siguiente:

```
# Este algoritmo requiere de un ciclo para recorrer el arreglo por completo. Además, usaremos
# una variable (llamada pos) para almacenar la posición del arreglo en la que está el elemento
# que buscamos.

# Declaramos las variables que requerimos.
# Note que pos inicia en -1. Si su valor no cambia significa que no se encontró el elemento
Entero: n=4, x(4), i, pos=-1, val

# Asignamos los valores al arreglo
x = {8, 4, 7, 2}

# Pedimos al usuario un valor a buscar en el arreglo
Escribir("Ingrese el valor que desea buscar en el arreglo: ")
Leer(val)

# Usando un ciclo para recorrer secuencialmente el arreglo
Para i=0 hasta n incremento 1 Haga
  Si (x[i] == val) Entonces
    pos = i # Almacenamos en pos la posición del elemento buscado
    Escribir("El valor buscado se encontró en la posición ", i)
  Fin_si
Fin_Para

Si (pos == -1) Entonces
  Escribir("El valor buscado no está en el arreglo")
Fin_si
```

3.4 Arreglos de Objetos

Inicialmente mencionamos que un arreglo es una estructura de datos que contiene un número fijo y finito de valores, todos del **mismo tipo de dato**. Entonces, así como podemos crear arreglos que contienen valores de los tipos de datos primitivos (Entero, Real, Carácter), también podemos crear arreglos que almacenen objetos de una misma clase, con sus atributos y métodos. Es decir, así como podemos tener un arreglo que almacene 100 valores enteros, también se pueden crear un arreglo que almacenen a 100 Estudiantes. A continuación, veremos cómo crear y manipular este tipo de arreglos y esto lo haremos con la clase Persona, cuyo se seudocódigo es el siguiente:

```
# Esta clase almacena la información básica de una persona, es decir, su nombre completo,
# edad y la fruta preferida.

clase Persona
  Texto nombre_completo, fruta_preferida
  Entero edad

  # Este método constructor crea una Persona recibiendo como parámetros todos sus datos
  Persona(Texto nombre, Texto fruta, Entero edadP)
    self.nombre_completo = nombre
    self.fruta_preferida = fruta
    self.edad = edadP
  Fin_Metodo

  # Este método indica si la persona es mayor de edad o no
  publico Logico es_mayor_de_edad()
    Si (self.edad >= 18) Entonces
      Retornar True
    Sino
      Retornar False
    Fin_si
  Fin_Metodo

  # Este método retorna una cadena de texto con la información completa de la Persona
```

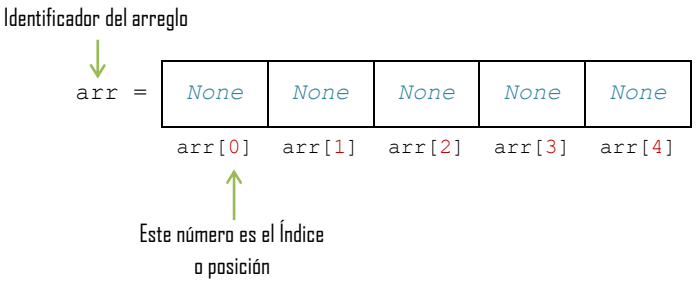
```
  publico Texto obtener_informacion()
    Texto str = ""
    str += "Nombre: " + self.nombre_completo
    str += "\nEdad: " + self.edad
    str += "\nFruta preferida: " + self.fruta_preferida
    Retornar str
  Fin_Metodo
Fin_clase
```

3.4.1 DECLARACIÓN DEL ARREGLO

De acuerdo con lo indico antes, para declarar un arreglo es necesario definir la variable de referencia, especificar el tipo y el tamaño de este. En caso de declarar arreglos de objetos, el tipo de la variable es la clase a la que pertenecen los objetos que vamos a almacenar. Por ejemplo, si queremos un arreglo que almacene cinco objetos de tipo Persona la declaración del arreglo es la siguiente:

```
# Declaración de una variable que es un arreglo que almacenará 5 objetos de la clase Persona
Persona arr(5)
```

Al declarar un arreglo de esta forma se crea una estructura de datos que gráficamente se puede representar así:



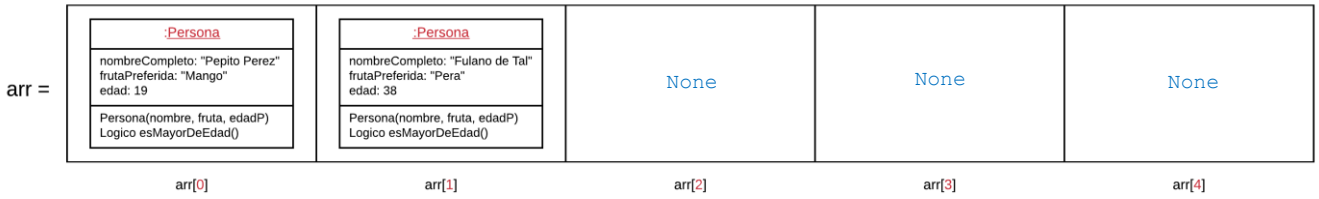
La diferencia con la creación de un arreglo de un tipo de dato primitivo es que cuando creamos un arreglo de objetos, por en el arreglo no hay objetos y, por tanto, el valor que se almacena inicialmente en cada posición es el valor *None*.

3.4.2 GUARDANDO OBJETOS EN UN ARREGLO

Ahora, para almacenar (o guardar) objetos en el arreglo debemos entonces primero instanciar los objetos de la clase. En el código siguiente usamos un ciclo para crear objetos de la clase Persona y almacenarlos en el arreglo.

```
# Declaramos las variables que requerimos.
Entero n=5, i, edad
Texto nombre, fruta
Persona arr(n)

# Usamos un arreglo para pedir los datos de la persona, crear el objeto y guardarlo en el
arreglo
Para i=0 hasta n incremento 1 Haga

  # Primero pedimos los datos de la persona
  Escribir("Ingrese el nombre completo de la persona # ", i+1)
  Leer(nombre)
  Escribir("Ingrese la edad de la persona")
  Leer(edad)
  Escribir("Ingrese la fruta preferida de la persona")
  Leer(fruta)

  # Ahora creamos el objeto con los datos ingresados
  Persona p = nuevo Persona(nombre, fruta, edad)

  # Ya que tenemos el objeto lo agregamos al arreglo
  arr[i] = p

Fin_Para
```

Cuando se almacena un objeto de una clase en un arreglo, el objeto se almacena como un todo, es decir, en cada posición del arreglo se almacena una instancia de la clase que como tal tiene sus atributos y métodos. Gráficamente, por ejemplo, cuando se han ingresado 2 Personas el arreglo se vería así:



En esta representación, en cada posición hay un objeto de la clase Persona, y por tanto, ese objeto tiene un `nombre_completo`, una `edad` y una `fruta_preferida`. Además, el objeto cuenta con un constructor y un método que retorna un valor lógico y que se llama `es_mayor_de_edad`().

Importante: Note que el objeto se declara y se instancia dentro del ciclo. Por esta razón, con cada iteración de este se crea una nueva variable `p`, la cual reserva un nuevo espacio en memoria, en la que se almacena el objeto en la posición `i` del arreglo.

3.4.3 Ordenando los Objetos de un Arreglo

Hay muchas aplicaciones en las que es necesario ordenar los objetos que están almacenados en un arreglo. Por ejemplo, ordenemos nuestro arreglo de personas con base en su edad (de mayor a menor). Para hacer esto, en el seudocódigo siguiente usaremos el método de ordenamiento por selección (aunque tú puedes hacer los mismo con los otros métodos de ordenamiento).

```
# Declaramos las variables que requerimos y asumimos que el arreglo ya está lleno
Entero n=5, i, j,

# La variable aux nos servirá para intercambiar de posición los objetos
Persona arr(n), aux

# Usamos dos ciclos para ordenar el arreglo
Para i=0 hasta n-1 incremento 1 Haga
  Para j=i+1 hasta n incremento 1 Haga
    # Comparamos la edad de los objetos en las posiciones i y j del arreglo
    Si (arr[j].edad > arr[i].edad) Entonces
      # Si se cumple la condición hacemos el intercambio
      aux = arr[j]
      arr[j] = arr[i]
      arr[i] = aux
    Fin_si
  Fin_Para
Fin_Para
```

Ahora, después de ordenar el arreglo mostremos en la pantalla el nombre y la fruta preferida de las tres personas con la mayor edad, en caso de que alguna de ellas sea menor de edad se debe indicar (esto lo haremos accediendo a los atributos y métodos de los objetos en el arreglo).

```
# Aunque lo podemos hacer directamente con las posiciones 0, 1 y 2, usaremos un ciclo
# la persona más vieja
Escribir("De mayor a menor, las 3 personas con más edad en el arreglo son: ")
Para i=0 hasta 3 incremento 1 Haga
  Escribir("\nPersona # ", i+1)
```

```
  Escribir("\nNombre: ", arr[i].nombre_completo)
  Escribir("\nFruta preferida: ", arr[i].fruta_preferida)

  Si (arr[i].es_mayor_de_edad() == True) Entonces
    Escribir("\n", arr[i].nombre, " es menor de edad",
  Fin_Si
Fin_Para
```

Importante:

Para acceder a los atributos y métodos de los objetos en el arreglo usamos el operador punto (.). Por lo tanto, si queremos acceder al nombre de la Persona en la posición 2 del arreglo usamos la instrucción

```
arr[2].nombre_completo
```

Otra opción es sacar el objeto del arreglo en una variable independiente. Por ejemplo, si queremos manipular el objeto de la posición 2, podemos hacer lo siguiente:

```
Persona una_persona = arr[2]
```

Cuando hacemos esto, el objeto queda guardado a la variable `una_persona` y para acceder al nombre usaríamos la instrucción

```
Una_persona.nombre_completo
```

4. EJERCICIO PROPUESTO

La ferretería "***La Puntilla***" es una pequeña ferretería de barrio en la que se venden desde tornillos hasta máquinas podadoras. Su dueño, don José, quiere implementar un pequeño sistema de ventas para su negocio. De acuerdo con don José, el sistema que él quiere debe tener un catálogo de productos en el que debe aparecer el nombre del producto, un código de identificación, el valor de venta sin IVA, el tipo de producto y la cantidad de existencias de éste en la ferretería. Un producto puede ser de tipo electrónico, de construcción o consumible. El IVA de cada producto depende de su tipo, por lo tanto, los productos de tipo electrónico pagan un 19% de IVA, los de construcción un 5% y los consumibles un 12%. Para el manejo de los productos, don José desea que el sistema le permita agregar (a demanda) un producto al listado de productos, eso sí, se debe tener en cuenta que en la ferretería no habrán más de 1000 productos. Como don José es medio despistado, cuando se va a ingresar un producto el sistema debe asegurarse (por el nombre o por el código) que el producto no exista. A veces, los productos cambian de nombre o precio, por eso el sistema también debe proveer una forma de buscar y actualizar los datos de un producto determinado. Adicionalmente, don José está interesado en que el sistema le permita generar algunos listados, entre ellos, un listado de los productos por tipo (el cual se define a la hora de generar el listado) en el que se muestre el nombre del producto, el valor sin IVA y el valor de IVA que se pagaría por dicho producto. Este listado debe mostrar los productos ordenados por precio. Otro listado que se requiere es que se muestren el nombre y el tipo de todos los productos que tengan un valor mayor o igual a un valor ingresado por don José.

Analice el enunciado y con base en él:

- Defina el cliente, el usuario y las entidades del mundo del problema que intervienen en la solución
- Defina los requerimientos que deben ser implementados para construir una solución al problema
- Describa los atributos y métodos que deben tener las clases que representan las entidades del mundo del problema
- Con base en la descripción anterior, bosqueje el diagrama de clases de la solución
- Implemente la solución en seudocódigo o algún lenguaje de programación orientado por objetos

5. BIBLIOGRAFÍA

- Herrera, A., Ebratt, R., & Capacho, J. (2016). Diseño y construcción de algoritmos. Barranquilla: Universidad del Norte.
- Aguilar, L. (2020). Fundamentos de programación: algoritmos, estructuras de datos y objetos (5a.ed.). México: Mc Graw Hill.
- Thomas Mailund. Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More, Apress, 2021.
- Steven S. Skiena. The Algorithm Design Manual. Third Edition, Springer, 2020.
- Samuel Tomi Klein. Basic Concepts in Algorithms, World Scientific Publishing, 2021.

Elaborado Por:	Carlos Andrés Mera Banguero
Versión:	1.0
Fecha	Marzo de 2024