# Estructuras dinámicas lineales (iii)

## *Introducción*

En las anteriores lecciones vimos la forma de implementar listas simplemente enlazadas mediante la notación algorítmica; se mostraron los algoritmos para realizar inserciones, recorridos, borrados, etc. tanto de forma iterativa como recursiva. En el presente capítulo se presentará la implementación de dichas acciones en el lenguaje de

## *Implementación de listas simplemente enlazadas en FORTRAN*

Como sabemos, para implementar cualquier estructura dinámica es necesario:

1. Determinar el tipo de datos que se va a almacenar.
2. Determinar la estructura del nodo que se va a utilizar.
3. Tener un `pointer`
   `pointer` inicialmente será nulo.

En estos ejemplos, como en los anteriores, se creará una lista de números enteros; así pues, la estructura del

```
type nodo
  integer numero
  type(nodo), pointer::siguiente
end type
```

Una vez se ha definido el tipo para los nodos de la lista ya es posible definir un `pointer` en el programa principal que apuntará a la cabeza de la futura lista:

```
type(nodo), pointer::cabeza
nullify(cabeza)
```

Por último, para manipular la lista se implementarán las operaciones habituales: recorrer, búsqueda, inserción, borrado y vaciado. Todas estas operaciones pueden implementarse tanto de forma iterativa como de forma recursiva; proponiéndose a continuación algoritmos para las mismas en ambas versiones.

### Operaciones sobre listas (simplemente enlazadas) implementadas iterativamente

#### *Inserción de un elemento en una lista simplemente enlazada*

##### Inserción en una lista vacía

Ya sabemos que la inserción del primer elemento de una lista, esto es una inserción en una lista vacía, es un caso especial:

```
subroutine insertarVacia (cabezaLista,dato)
  implicit none
  type(nodo), pointer::cabezaLista
  integer dato

  if (.not.associated(cabezaLista)) then
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato
  end if
end subroutine
```

##### Inserción “por delante” de un elemento en una lista simplemente enlazada

La siguiente subrutina FORTRAN implementa la inserción “por delante”:

```
subroutine insertarDelante (cabezaLista,dato_nuevo,dato_viejo)
  implicit none
  type(nodo), pointer::cabezaLista,cursor,nuevo
  integer dato_nuevo,dato_viejo

  cursor=>cabezaLista
```

```
  do while (associated(cursor).and.(cursor.numero/=dato_viejo))
    cursor=>cursor.siguiente
  end do

  if (associated(cursor)) then
    allocate(nuevo)
    nuevo.siguiente=>cursor.siguiente
    cursor.siguiente=>nuevo
    nuevo.numero=cursor.numero
    cursor.numero=dato_nuevo
  end if
end subroutine
```

**Inserción “por detrás” de un elemento en una lista simplemente enlazada**

Subrutina FORTRAN para la inserción “por detrás”:

```
subroutine insertarDetras (cabezaLista,dato_nuevo,dato_viejo)
  implicit none
  type(nodo), pointer::cabezaLista,cursor,nuevo
  integer dato_nuevo,dato_viejo

  cursor=>cabezaLista

  do while (associated(cursor).and.(cursor.numero/=dato_viejo))
    cursor=>cursor.siguiente
  end do

  if (associated(cursor)) then
    allocate(nuevo)
    nuevo.siguiente=>cursor.siguiente
    nuevo.numero=dato_nuevo
    cursor.siguiente=>nuevo
  end if
end subroutine
```

Obsérvese que la única diferencia entre esta subrutina y la anterior es la eliminación de la sentencia en que se asigna al nodo nuevo el `dato_viejo` y al `cursor` el `dato_nuevo`.

Sabemos que los tres algoritmos anteriores pueden “refundirse” para implementar las acciones para insertar en una cola, en una pila y en una lista ordenada.

**Inserción en una cola**

La inserción en una cola precisa buscar el último elemento de la lista e insertar el nuevo elemento detrás del mismo.

```
subroutine insertarCola (cabezaLista,dato_nuevo)
  implicit none
  type(nodo), pointer::cabezaLista,cursor,nuevo
  integer dato_nuevo

  if (associated(cabezaLista)) then
    cursor=>cabezaLista

    do while (associated(cursor.siguiente))
      cursor=>cursor.siguiente
    end do

    allocate(nuevo)
    nuevo.siguiente=>cursor.siguiente
    nuevo.numero=dato_nuevo
    cursor.siguiente=>nuevo
  else
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato_nuevo
  end if
end subroutine
```

**Inserción en una pila**

La inserción en una pila consiste en insertar delante de la cabeza de la lista.

```
subroutine insertarPila (cabezaLista,dato_nuevo)
  implicit none
  type(nodo), pointer::cabezaLista,nuevo
  integer dato_nuevo

  if (associated(cabezaLista)) then
    allocate(nuevo)
    nuevo.siguiente=>cabezaLista.siguiente
    cabezaLista.siguiente=>nuevo
    nuevo.numero=cabezaLista.numero
    cabezaLista.numero=dato_nuevo
  else
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato_nuevo
  end if
end subroutine
```

**Inserción en una lista ordenada ascendentemente**

```
subroutine insertarOrdenado (cabezaLista,dato_nuevo)
  implicit none
  type(nodo), pointer::cabezaLista,nuevo,cursor
  integer dato_nuevo

  if (associated(cabezaLista)) then
    cursor=>cabezaLista

    do while (associated(cursor.siguiente).and.(cursor.numero<dato_nuevo))
      cursor=>cursor.siguiente
    end do

    if (cursor.numero>=dato_nuevo) then
      allocate(nuevo)
      nuevo.siguiente=>cursor.siguiente
      cursor.siguiente=>nuevo
      nuevo.numero=cursor.numero
      cursor.numero=dato_nuevo
    else
      allocate(nuevo)
      nuevo.siguiente=>cursor.siguiente
      nuevo.numero=dato_nuevo
      cursor.siguiente=>nuevo
    end if
  else
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato_nuevo
  end if
end subroutine
```

### *Recorrido de una lista simplemente enlazada*

```
subroutine recorrer (cabezaLista)
  implicit none
  type(nodo), pointer::cabezaLista, cursor

  cursor=>cabezaLista

  do while (associated(cursor))
    print *, cursor.numero
    cursor=>cursor.siguiente
  end do
end subroutine
```

La subrutina recibe como argumento un `pointer` a la cabeza de la lista; no se utiliza dicho puntero para recorrer la estructura puesto que si se hiciera de esa forma se perdería la cabeza de la lista y se “desmantelaría” la estructura.

Por esa razón se utiliza una variable auxiliar, `cursor`, que se utiliza para recorrer la lista. Inicialmente dicha variable se apunta hacia la cabeza de la lista y mientras esté asociado se va imprimiendo por pantalla el contenido del nodo apuntado y, una vez mostrado el dato, se le hace pasar al siguiente nodo.

### *Búsqueda de un elemento en una lista simplemente enlazada*

La operación de búsqueda se basa en la de recorrido; básicamente se trata de recorrer la lista hasta que se encuentre el elemento o llegar al final, retornando el cursor.

Si el elemento ha sido encontrado el retorno de la función apuntará a una posición de memoria y si no ha sido

```
function buscar (cabezaLista,elemento)
  implicit none
  type(nodo),pointer::buscar,cabezaLista,cursor
  integer elemento

  cursor=>cabezaLista

  do while (associated(cursor).and.(cursor.numero/=elemento))
    cursor=>cursor.siguiente
  end do

  buscar=>cursor
end function
```

### *Eliminar un elemento de una lista simplemente enlazada*

El algoritmo básico de eliminación simplemente debe recorrer la lista hasta encontrar el dato a eliminar, enlazar la lista de forma adecuada y destruir el nodo sobrante.

```
subroutine eliminarElemento (cabezaLista,dato)
  implicit none
  type(nodo),pointer::cabezaLista,anterior,cursor
  integer dato

  if (associated(cabezaLista)) then
    cursor=>cabezaLista

    do while (associated(cursor.siguiente).and.(cursor.numero/=dato))
      anterior=>cursor
      cursor=>cursor.siguiente
    end do

    if (cursor.numero==dato) then
      if (.not.associated(cursor,cabezaLista)) then
        anterior.siguiente=>cursor.siguiente
        deallocate(cursor)
      else
        cursor=>cabezaLista.siguiente
        cabezaLista.numero=cursor.numero
        cabezaLista.siguiente=>cursor.siguiente
        deallocate(cursor)
      end if
    end if
  end if
end subroutine
```

### *Extracción de elementos de pilas y colas*

La extracción siempre elimina el valor que se encuentra en la cabeza de la lista y, además, retorna su valor al programa principal; por esa razón se implementa como una función:

```
integer function extraerElemento (cabezaLista)
  implicit none
  type(nodo),pointer::cabezaLista,cursor

  if (associated(cabezaLista)) then
    extraerElemento=cabezaLista.numero

    cursor=>cabezaLista.siguiente

    cabezaLista.numero=cursor.numero
    cabezaLista.siguiente=>cursor.siguiente

    deallocate(cursor)
  end if
end function
```

### *Vaciado de una lista simplemente enlazada*

```
subroutine vaciarLista (cabezaLista)
  implicit none
  type(nodo),pointer::cabezaLista,cursor

  do while (associated(cabezaLista))
    cursor=>cabezaLista.siguiente
    deallocate(cabezaLista)
    cabezaLista=>cursor
  end do
end subroutine
```

Para vaciar una lista simplemente enlazada basta con eliminar el nodo que se encuentra en la cabeza de la lista de forma repetida hasta que la lista quede vacía.

## Operaciones sobre listas (simplemente enlazadas) implementadas recursivamente

### *Recorrido de una lista simplemente enlazada (de forma recursiva)*

```
recursive subroutine recorrerRecursivo (cabezaLista)
  implicit none
  type(nodo),pointer::cabezaLista

  if (associated(cabezaLista)) then
    print *,cabezaLista.numero
    call recorrerRecursivo(cabezaLista.siguiente)
  end if
end subroutine
```

### *Búsqueda de un elemento en una lista simplemente enlazada (de forma recursiva)*

```
recursive function buscarRecursivo (cabezaLista,elemento) result (r)
  implicit none
  type(nodo),pointer::r,cabezaLista
  integer elemento

  if (associated(cabezaLista)) then
    if (cabezaLista.numero==elemento) then
      r=>cabezaLista
    else
      r=>buscarRecursivo(cabezaLista.siguiente,elemento)
    end if
  else
    nullify(r)
  end if
end function
```

### *Inserción de un elemento en una lista simplemente enlazada*

#### Inserción en una cola (de forma recursiva)

La inserción en una cola precisa llegar al último elemento de la lista e insertar el nuevo elemento detrás del mismo.

```
recursive subroutine insertarColaRecursivo (cabezaLista,dato_nuevo)
  implicit none
  type(nodo), pointer::cabezaLista
  integer dato_nuevo

  if (associated(cabezaLista)) then
    call insertarColaRecursivo (cabezaLista.siguiente,dato_nuevo)
  else
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato_nuevo
  end if
end subroutine
```

**Inserción en una lista ordenada ascendentemente (de forma recursiva)**

```
recursive subroutine insertarOrdenadoRecursivo (cabezaLista,dato_nuevo)
  implicit none
  type(nodo),pointer::cabezaLista,nuevo
  integer dato_nuevo

  if (associated(cabezaLista)) then
    if (cabezaLista.numero>=dato_nuevo) then
      allocate(nuevo)
      nuevo.siguiente=>cabezaLista.siguiente
      cabezaLista.siguiente=>nuevo
      nuevo.numero=cabezaLista.numero
      cabezaLista.numero=dato_nuevo
    else
      call insertarOrdenadoRecursivo (cabezaLista.siguiente,dato_nuevo)
    end if
  else
    allocate(cabezaLista)
    nullify(cabezaLista.siguiente)
    cabezaLista.numero=dato_nuevo
  end if
end subroutine
```

***Eliminar un elemento de una lista simplemente enlazada (de forma recursiva)***

```
recursive subroutine eliminarElementoRecursivo (cabezaLista,dato)
  implicit none
  type(nodo),pointer::cabezaLista,cursor
  integer dato

  if (associated(cabezaLista)) then
    if (cabezaLista.numero==dato) then
      if (associated(cabezaLista.siguiente)) then
        cursor=>cabezaLista.siguiente
        cabezaLista.numero=cursor.numero
        cabezaLista.siguiente=>cursor.siguiente
        deallocate(cursor)
      else
        deallocate(cabezaLista)
      end if
    else
      call eliminarElementoRecursivo (cabezaLista.siguiente,dato)
    end if
  end if
end subroutine
```

***Vaciado de una lista simplemente enlazada (de forma recursiva)***

```
recursive subroutine vaciarListaRecursivo (cabezaLista)
  implicit none
  type(nodo),pointer::cabezaLista

  if (associated(cabezaLista)) then
    call vaciarListaRecursivo(cabezaLista.siguiente)
    deallocate(cabezaLista)
  end if
end subroutine
```

Para vaciar una lista simplemente enlazada de forma recursiva basta con vaciar el resto de la lista, si es que existe, y después eliminar el nodo que se encuentra en la cabeza.