LPC

Cola | **FIFO**: Primero que entra, primero que sale Pila | **LIFO**: El ultimo que entra, el primero que sale

> Los getters y los setters se escriben en la medida que sea necesario especificar alguna función con parámetros específicos o sólo se declaren métodos específicos debido a características específicas del problema/ejercicio. Si hay muchos se completan los getters/setters faltantes con "..." (tomar en cuenta que esto se hace en papel para conservar espacio, energía y tiempo).

Clase Nodo

```
Class Nodo<Element>
    Attributes:
        Private:
            Element: info
            pointer to Nodo<Element>: next
    Methods:
        Public:
            proc construir(Element:info, pointer to Nodo<Element>: next)
            //Getters
            ...
            //Setters
            ...
endClass
```

Clase Lista

```
Class Lista<Element>
    Attributes:
        Private:
            pointer to Nodo<Element>: head, tail
            int: long
    Methods:
        Public:
            proc construir()
            proc construir(ref Lista<Element>: target) //Constructor copia
            //Getters
            func getLong():int

            proc insertar(Element: e, int: pos)
            func consultar(int: pos):Element
            proc eliminar(int: pos)
            func buscar(Element: e): int
            proc invertir()
            func copiar(): Lista<Element> //O(n)
```

```
            proc vaciar() //O(n)
            func esVacia(): bool
            proc intercambiar(int: pos1, pos2)
            func concatenar(Lista<Element>: target): Lista<Element>
            proc destruir()
            proc modificar(int: pos, Element: e)
endClass
```

Clase Cola

```
Class Cola<Element>
    Attributes:
        Private:
            pointer to Nodo<Element>: frente, ultimo
            int: long
    Methods:
        Public:
            proc construir()
            proc construir(ref Cola<Element>: target) //Constructor copia
            func getFrente():Element
            func getUltimo():Element
            func getLong(): int
            proc encolar(Element: e)
            proc desencolar()
            func esVacia(): bool
            proc vaciar()
            proc destruir()
endClass
```

Clase Pila

```
Class Pila<Element>
    Attributes:
        Private:
            pointer to Nodo<Element>: tope
            int: long
    Methods:
        Public:
            proc construir()
            proc construir(ref Pila<Element>: target)
            func getTope(): Element
            proc apilar(Element: e)
            proc desapilar()
            func esVacia():bool
            proc vaciar()
            proc destruir()
endClass
```

Revisión rápida de la guía (errores y mejoras)

Errores o inconsistencias detectadas

- En la definición de clases genéricas, `Cola` y `Pila` deben declararse como `Class Cola<Element>` y `Class Pila<Element>`.
- Conviene mantener consistencia estricta en nombres (`getLong`, `construir`, `destruir`, etc.) en todos los ejercicios y parciales.
- En `Lista`, métodos como `insertar`, `consultar`, `eliminar` deben tratar explícitamente casos borde (`1` y `long`) para aprovechar `head`/`tail` en `O(1)`.

Métodos útiles para agregar (opcionales)

- `func contiene(Element: e): bool` //wrapper de `buscar`

Implementación de métodos (sin getters/setters)

> **Disclaimer:** Esta implementación sirve como base de estudio y referencia. **Falta revisión puntual método por método** (precondiciones, manejo de errores por posición inválida y decisiones de diseño de memoria) antes de considerarla versión final para evaluación.

Clase Nodo

```
proc Nodo<Element>::construir(Element: info, pointer to Nodo<Element>: next)
    Begin
        instance.info <- info
        instance.next <- next
endproc
```

Clase Lista

```
proc Lista<Element>::construir()
    Begin
        instance.head <- NULL
        instance.tail <- NULL
        instance.long <- 0
endproc

proc Lista<Element>::construir(ref Lista<Element>: target)
    Var
        pointer to Nodo<Element>: p
    Begin
        instance.construir()
        p <- target.head
```

```
        while p do
            instance.insertar(p->getInfo(), instance.long + 1)
            p <— p->getNext()
        endwhile
endproc

proc Lista<Element>::insertar(Element: e, int: pos)
    Var
        pointer to Nodo<Element>: pNew, pPrev, pAct
        int: i
    Begin
        pNew <— new(Nodo<Element>)
        pNew->construir(e, NULL)

        if instance.long = 0 then
            instance.head <— pNew
            instance.tail <— pNew
            instance.long <— 1
            return
        endif

        if pos = 1 then
            pNew->setNext(instance.head)
            instance.head <— pNew
            instance.long <— instance.long + 1
            return
        endif

        if pos = instance.long + 1 then
            instance.tail->setNext(pNew)
            instance.tail <— pNew
            instance.long <— instance.long + 1
            return
        endif

        pPrev <— instance.head
        for i <— 1 to pos - 2 do
            pPrev <— pPrev->getNext()
        endfor
        pAct <— pPrev->getNext()
        pPrev->setNext(pNew)
        pNew->setNext(pAct)
        instance.long <— instance.long + 1
endproc

func Lista<Element>::consultar(int: pos): Element
    Var
        pointer to Nodo<Element>: p
        int: i
        Element: e
```

```
    Begin
        if pos = 1 then
            return instance.head->getInfo()
        endif

        if pos = instance.long then
            return instance.tail->getInfo()
        endif

        p <- instance.head
        for i <- 1 to pos - 1 do
            p <- p->getNext()
        endfor
        return p->getInfo()
endfunc

proc Lista<Element>::eliminar(int: pos)
    Var
        pointer to Nodo<Element>: pDel, pPrev
        int: i
    Begin
        if instance.long = 1 then
            delete(instance.head)
            instance.head <- NULL
            instance.tail <- NULL
            instance.long <- 0
            return
        endif

        if pos = 1 then
            pDel <- instance.head
            instance.head <- instance.head->getNext()
            delete(pDel)
            instance.long <- instance.long - 1
            return
        endif

        if pos = instance.long then
            pPrev <- instance.head
            for i <- 1 to instance.long - 2 do
                pPrev <- pPrev->getNext()
            endfor
            delete(instance.tail)
            instance.tail <- pPrev
            instance.tail->setNext(NULL)
            instance.long <- instance.long - 1
            return
        endif

        pPrev <- instance.head
```

```
        for i <— 1 to pos - 2 do
            pPrev <— pPrev->getNext()
        endfor
        pDel <— pPrev->getNext()
        pPrev->setNext(pDel->getNext())
        delete(pDel)
        instance.long <— instance.long - 1
endproc

func Lista<Element>::buscar(Element: e): int
    Var
        pointer to Nodo<Element>: p
        int: pos
    Begin
        p <— instance.head
        pos <— 1
        while p do
            if p->getInfo() = e then
                return pos
            endif
            p <— p->getNext()
            pos <— pos + 1
        endwhile
        return -1
endfunc

proc Lista<Element>::invertir()
    Var
        pointer to Nodo<Element>: pPrev, pAct, pNext
    Begin
        pPrev <— NULL
        pAct <— instance.head
        instance.tail <— instance.head

        while pAct do
            pNext <— pAct->getNext()
            pAct->setNext(pPrev)
            pPrev <— pAct
            pAct <— pNext
        endwhile

        instance.head <— pPrev
endproc

func Lista<Element>::copiar(): Lista<Element>
    Var
        Lista<Element>: result
    Begin
        result.construir(instance)
        return result
```

```
endfunc

proc Lista<Element>::vaciar()
    Var
        pointer to Nodo<Element>: p
    Begin
        while instance.head do
            p <— instance.head
            instance.head <— instance.head->getNext()
            delete(p)
        endwhile
        instance.tail <— NULL
        instance.long <— 0
endproc

func Lista<Element>::esVacia(): bool
    Begin
        return instance.long = 0
endfunc

proc Lista<Element>::intercambiar(int: pos1, pos2)
    Var
        Element: e1, e2
    Begin
        if pos1 < 1 v pos1 > instance.long v pos2 < 1 v pos2 > instance.long then
            return
        endif
        if pos1 = pos2 then
            return
        endif

        e1 <— instance.consultar(pos1)
        e2 <— instance.consultar(pos2)
        instance.modificar(pos1, e2)
        instance.modificar(pos2, e1)
endproc

func Lista<Element>::concatenar(Lista<Element>: target): Lista<Element>
    Var
        Lista<Element>: result
        pointer to Nodo<Element>: p
    Begin
        result.construir(instance)
        p <— target.head
        while p do
            result.insertar(p->getInfo(), result.getLong() + 1)
            p <— p->getNext()
        endwhile
        return result
endfunc
```

```
proc Lista<Element>::destruir()
    Begin
        instance.vaciar()
endproc

proc Lista<Element>::modificar(int: pos, Element: e)
    Var
        pointer to Nodo<Element>: p
        int: i
    Begin
        if pos < 1 v pos > instance.long then
            return
        endif

        if pos = 1 then
            instance.head->setInfo(e)
            return
        endif

        if pos = instance.long then
            instance.tail->setInfo(e)
            return
        endif

        p <— instance.head
        for i <— 1 to pos - 1 do
            p <— p->getNext()
        endfor
        p->setInfo(e)
endproc
```

Clase Cola

```
proc Cola<Element>::construir()
    Begin
        instance.frente <— NULL
        instance.ultimo <— NULL
        instance.long <— 0
endproc

proc Cola<Element>::construir(ref Cola<Element>: target)
    Var
        pointer to Nodo<Element>: p
    Begin
        instance.construir()
        p <— target.frente
        while p do
```

```
            instance.encolar(p->getInfo())
            p <- p->getNext()
        endwhile
endproc

proc Cola<Element>::encolar(Element: e)
    Var
        pointer to Nodo<Element>: pNew
    Begin
        pNew <- new(Nodo<Element>)
        pNew->construir(e, NULL)

        if instance.long = 0 then
            instance.frente <- pNew
            instance.ultimo <- pNew
        else
            instance.ultimo->setNext(pNew)
            instance.ultimo <- pNew
        endif
        instance.long <- instance.long + 1
endproc

proc Cola<Element>::desencolar()
    Var
        pointer to Nodo<Element>: pDel
    Begin
        if instance.long = 0 then
            return
        endif

        pDel <- instance.frente
        instance.frente <- instance.frente->getNext()
        delete(pDel)
        instance.long <- instance.long - 1

        if instance.long = 0 then
            instance.ultimo <- NULL
        endif
endproc

func Cola<Element>::esVacia(): bool
    Begin
        return instance.long = 0
endfunc

proc Cola<Element>::vaciar()
    Begin
        while ¬instance.esVacia() do
            instance.desencolar()
        endwhile
```

```
endproc

proc Cola<Element>::destruir()
    Begin
        instance.vaciar()
endproc
```

Clase Pila

```
proc Pila<Element>::construir()
    Begin
        instance.tope <— NULL
        instance.long <— 0
endproc

proc Pila<Element>::construir(ref Pila<Element>: target)
    Var
        Lista<Element>: aux
        pointer to Nodo<Element>: p
    Begin
        instance.construir()
        aux.construir()

        p <— target.tope
        while p do
            aux.insertar(p->getInfo(), 1)
            p <— p->getNext()
        endwhile

        while ¬aux.esVacia() do
            instance.apilar(aux.consultar(1))
            aux.eliminar(1)
        endwhile
endproc

proc Pila<Element>::apilar(Element: e)
    Var
        pointer to Nodo<Element>: pNew
    Begin
        pNew <— new(Nodo<Element>)
        pNew->construir(e, instance.tope)
        instance.tope <— pNew
        instance.long <— instance.long + 1
endproc

proc Pila<Element>::desapilar()
    Var
        pointer to Nodo<Element>: pDel
```

```
    Begin
        if instance.long = 0 then
            return
        endif

        pDel <- instance.tope
        instance.tope <- instance.tope->getNext()
        delete(pDel)
        instance.long <- instance.long - 1
endproc

func Pila<Element>::esVacia(): bool
    Begin
        return instance.long = 0
endfunc

proc Pila<Element>::vaciar()
    Begin
        while ¬instance.esVacia() do
            instance.desapilar()
        endwhile
endproc

proc Pila<Element>::destruir()
    Begin
        instance.vaciar()
endproc
```