```
Algoritmo IsEmpty
        Si (primero = Nulo) Entonces
        Retornar Verdadero
        Sino
        Retornar Falso
        Fin Si
Fin Algoritmo

Algoritmo Last
        // Retorna el último nodo de la lista
        Si (No Está Vacía()) Entonces
        Definir aux como Node<T> y asignar primero a aux
        Mientras (aux.siguiente != Nulo) Hacer
        aux <- aux.siguiente
        Fin Mientras
        Retornar aux
        Fin Si
        Retornar Nulo
Fin Algoritmo

Algoritmo Push(siguiente: Node<T>)
        // Agrega un nuevo nodo a la lista
        Si (siguiente = Nulo o siguiente.val = Nulo) Entonces
        Retornar
        Fin Si
        Definir last como Node<T> y asignar Last()
        Si (last ≠ Nulo) Entonces
                last.siguiente <- siguiente
        Sino
                primero <- siguiente
        Fin Si
Fin Algoritmo


Algoritmo AddFirst(newFirst: Node<T>)
```

```
        // Agrega un nuevo nodo al principio de la lista
        Si (newFirst = Nulo) Entonces
        Retornar
        Fin Si
        Si (Está Vacía()) Entonces
                primero <- newFirst
                Retornar
        Fin Si

        Definir aux como Node<T> y asignar primero a aux
        primero <- newFirst
        newFirst.siguiente <- aux
Fin Algoritmo

Algoritmo find(findVal: T) : Node<T>
        // Busca un nodo en la lista que contenga un valor específico
        Si (Está Vacía()) Entonces
        Retornar Nulo
        Fin Si
        Para cada nodo aux en la lista, desde el primer nodo hasta el último
                Si (aux.val = findVal) Entonces
                        Retornar aux
                Fin Si
        Fin Para

        Retornar Nulo
Fin Algoritmo

Algoritmo Delete(item: Node<T>)
```

```
        // Elimina un nodo específico de la lista
        Si (Está Vacía()) Entonces
        Retornar
        Fin Si
        Si (primer nodo no es igual a item) Entonces
                // Se debe buscar el nodo anterior a item y modificar su enlace
        next
                Asignar a left el primer nodo
                Para cada nodo right en la lista, comenzando por el segundo
        nodo
                        Si (right es igual a item) Entonces
                                left.next = right.next
                                Retornar
                        Fin Si
                        Asignar a left el nodo right actual
                        Asignar a right el siguiente nodo de la lista
                Fin Para
        Sino
                // item es el primer nodo
                primer nodo = primer nodo.next
        Fin Si
Fin Algoritmo
```

Algoritmo InsertAt (entero index, Nodo<T> item)

```
    Si (index > -1) Entonces
        Si (index = 0) Entonces
            Llamar: AddFirst(item)
            Retornar
        Fin Si
        Si (No IsEmpty()) Entonces
            i <- 1
            Para cada Nodo<T> aux en first hasta que aux sea nulo hacer
                Si (i = index) Entonces
                    item.next <- aux.next
                    aux.next <- item
                    Retornar
                Fin Si
                i <- i + 1
            Fin Para
        Fin Si
    Fin Si
Fin Algoritmo
```

Algoritmo DeleteDuplicates()

```
    Si (IsEmpty()) Entonces
        Retornar
    Fin Si

    conocidos <- Crear un nuevo conjunto vacío
    previo <- null
    aux <- first
    Mientras (aux != null) Hacer
        Si (No conocidos.Contiene(aux.val)) Entonces
            previo <- aux
            conocidos.Agregar(aux.val)
        Sino
            Llamar: Delete(previo)
            previo <- aux.next
        Fin Si
        aux <- aux.next
    Fin Mientras
Fin Algoritmo

Algoritmo Print() Devolver cadena
    Si (IsEmpty()) Entonces
        Devolver null
    Fin Si

    aux = first.siguiente
    imprimir = Crear cadena con first.Print()
    Mientras (aux != null) Hacer
        imprimir = concatenar(imprimir, "->", aux.Print())
        aux = aux.siguiente
    Fin Mientras

    Devolver imprimir
Fin Funcion
```