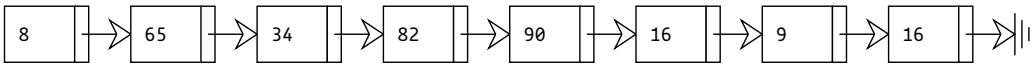
Listas

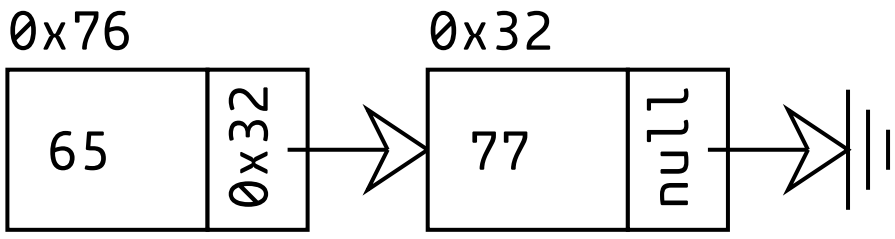
- Una lista es un conjunto de elementos llamados nodos que están conectados mediante un puntero.
- Es probablemente la segunda estructura de datos más usada.
- Una lista vacía es aquella que no contiene nodos.
- Las grandes ventajas de una lista frente a los arreglos son:
  - Los nodos no necesariamente son adyacentes en memoria.
  - Insertar o eliminar un elemento a la mitad de la lista NO implica mover el resto de elementos por lo que es de O(1).
  - Es dinámica por defecto.
- Las grandes desventajas de una lista frente a los arreglos son:
  - No es posible obtener un elemento por índice. Para obtener un elemento es necesario recorrer todos los elementos que se encuentren delante del mismo.

Enlaces

- En un alista cada nodo está conectado por un enlace.
- Lo que se obtiene es una especie de “cadena”, en la cual cada elemento “apunta” al siguiente.
- En el ultimo elemento, el puntero de siguiente apunta a `null` para indicar que no existen más elementos.



- Un nodo se aloja en la memoria dinámica (heap) en una dirección establecida por el sistema. Cada nodo contiene el elemento correspondiente, el cual puede ser de cualquier tipo, y la dirección del nodo siguiente.
- De éste modo, en la figura observamos que el primer nodo está ubicado en la dirección `0x76` mientras que el segundo nodo se encuentra en la dirección `0x32`.
- El primer nodo contiene la dirección del segundo nodo; mientras que el segundo nodo contiene la dirección `null`, indicando que es el último nodo de una lista simplemente enlazada.



Implementación de la clase lista

```
#pragma once

#include <functional>

using namespace std;

typedef unsigned int uint;



template <typename T, T NADA = 0>
class Lista {
    struct Nodo; // Prototipo de la clase nodo, la cual se implementa después
    typedef function<int(T, T)> Comp;

    Nodo*   ini;
    uint    lon; // número de elementos en la lista
    Comp    comparar; // lambda de criterio de comparación
public:
    Lista(): ini(nullptr), lon(0), comparar([](T a,T b) {return a - b;}) {}
    Lista(Comp comparar): ini(nullptr), lon(0), comparar(comparar) {}
    ~Lista();

    uint    longitud();
    bool    esVacia();
    void    agregaInicial(T elem);
    void    agregaPos(T elem, uint pos);
    void    agregaFinal(T elem);
    void    modificarInicial(T elem);
    void    modificarPos(T elem, uint pos);
    void    modificarFinal(T elem);
    void    eliminaInicial();
    void    eliminaPos(uint pos);
    void    eliminaFinal();
    T       obtenerInicial();
    T       obtenerPos(uint pos);
    T       obtenerFinal();
    T       buscar(T elem);
};
```

Implementación del Nodo

```
template <typename T, T NADA>
struct Lista<T, NADA>::Nodo {
    T       elem;
    Nodo*   sig; // puntero apunta al siguiente nodo

    Nodo(T elem = NADA, Nodo* sig = nullptr): elem(elem), sig(sig) {}
};
```

Operaciones auxiliares

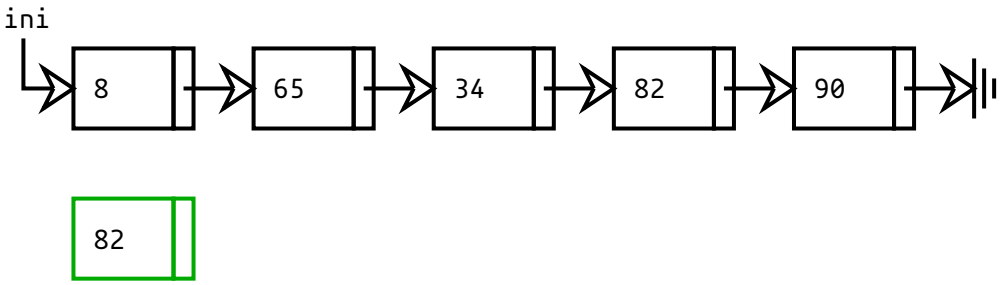
```
template <typename T, T NADA>
Lista<T, NADA>::~Lista() {
    Nodo* aux = ini;
    while (aux != nullptr) {
        aux = ini;
        ini = ini->sig;
        delete aux;
    }
}
```

```
template <typename T, T NADA>
uint Lista<T, NADA>::longitud() {
    return lon;
}
```

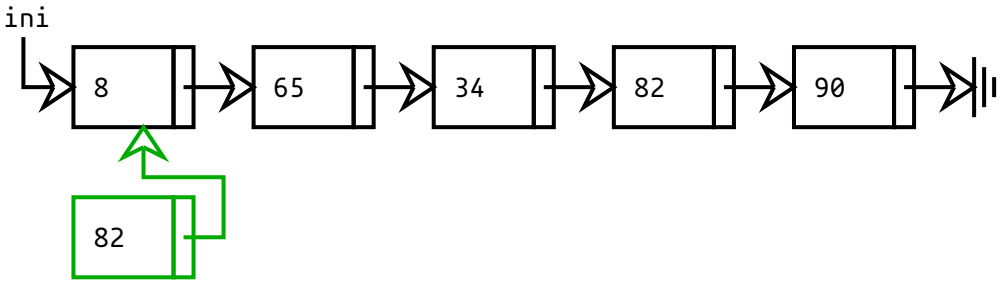
```
template <typename T, T NADA>
bool Lista<T, NADA>::esVacia() {
    return lon == 0;
}
```

Agregar un elemento al inicio de la lista

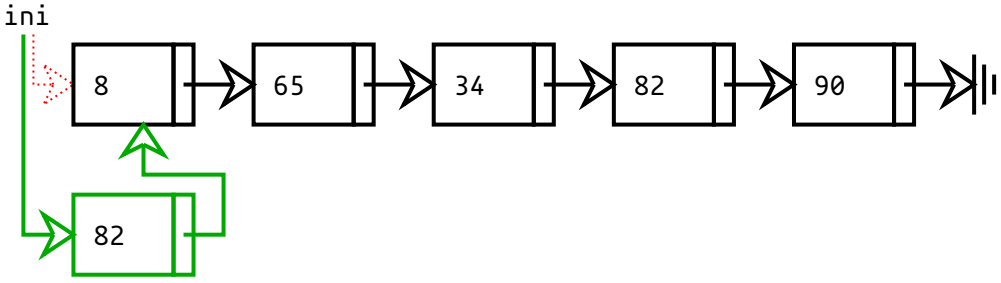
Crear nuevo nodo



Nuevo nodo apunta a primer nodo



Puntero al primer elemento **`ini`** apunta al nuevo nodo

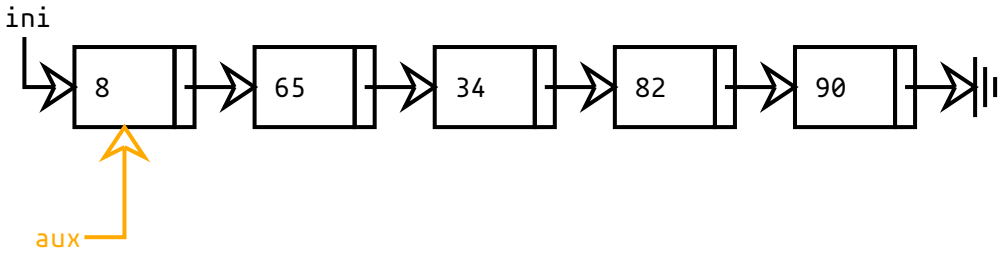


Código

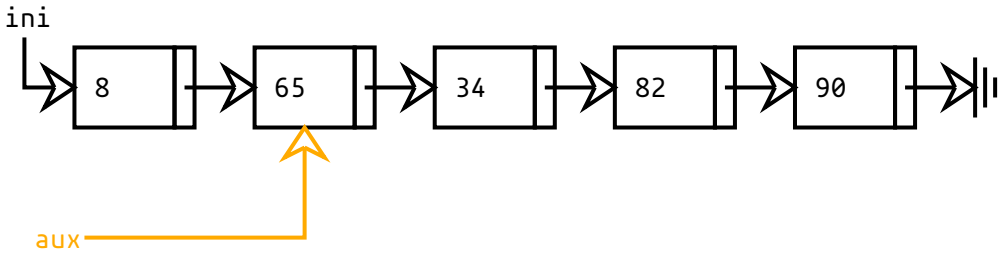
```
template <typename T, T NADA>
void Lista<T, NADA>::agregaInicial(T elem) {
    Nodo* nuevo = new Nodo(elem, ini);
    if (nuevo != nullptr) {
        ini = nuevo;
        lon++;
    }
}
```

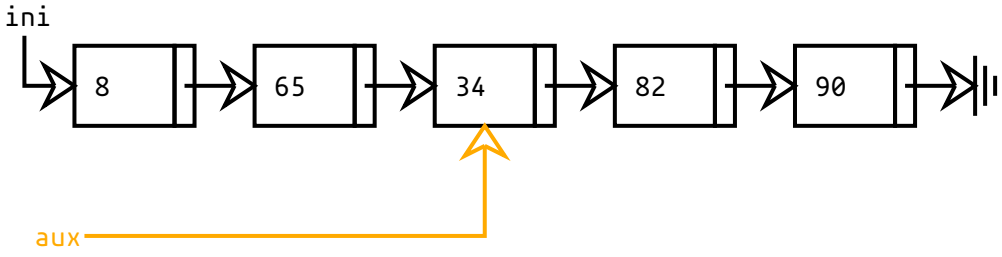
Buscar elemento

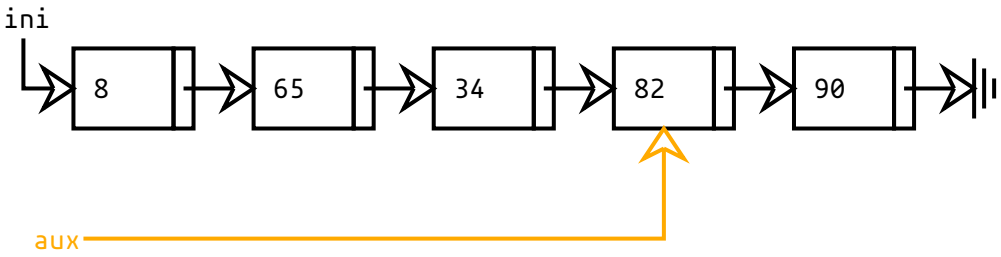
Asumiendo que buscamos el elemento **100**, iniciar un auxiliar en el primer elemento

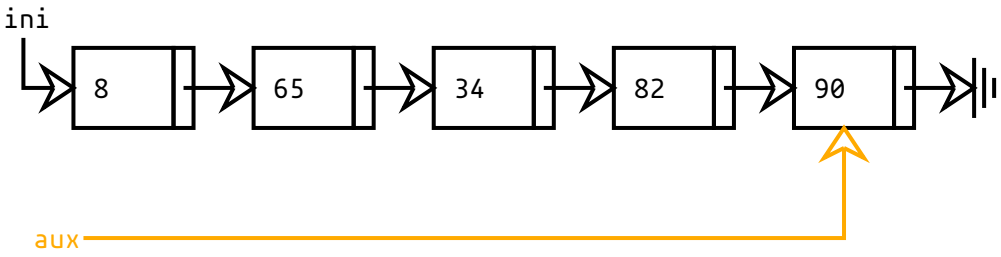


Apuntar al siguiente comparando con el elemento buscado

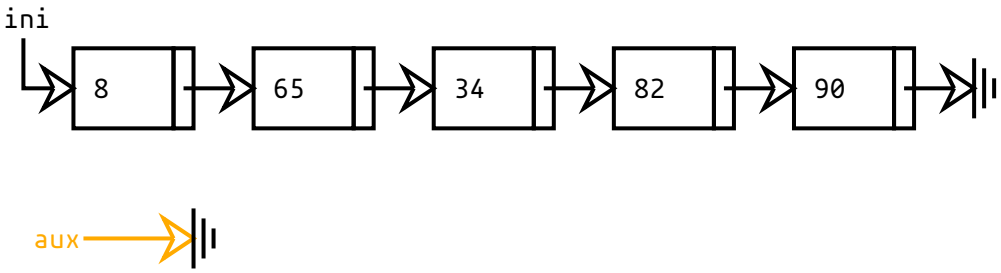








Si el auxiliar llega a tener valor **null**, no se encontró el elemento buscado

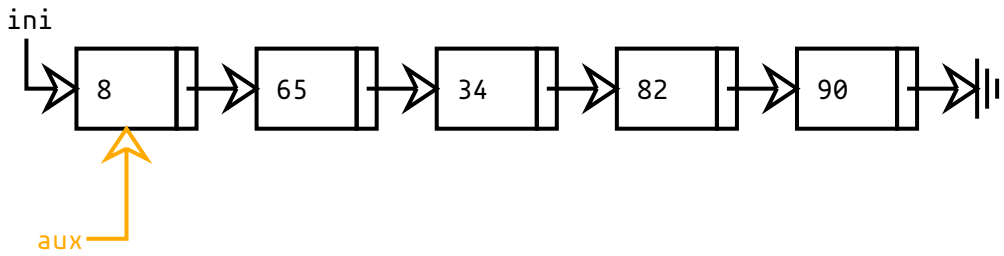


Código

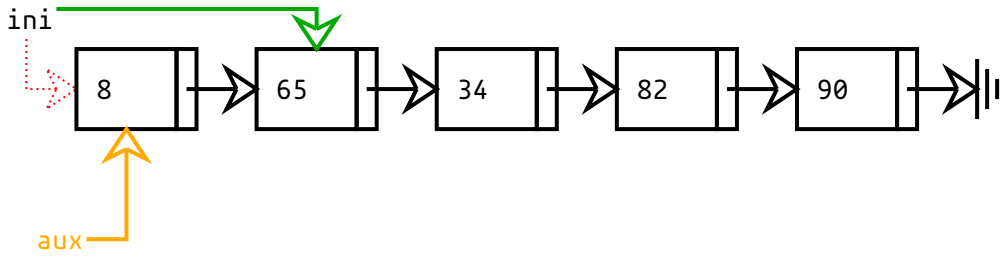
```
template <typename T, T NADA>
T Lista<T, NADA>::buscar(T elem) {
    Nodo* aux = ini;
    while (aux != nullptr) {
        if (comparar(aux->elem, elem) == 0) {
            return aux->elem;
        }
        aux = aux->sig;
    }
    return NADA;
}
```

Eliminar el primer elemento

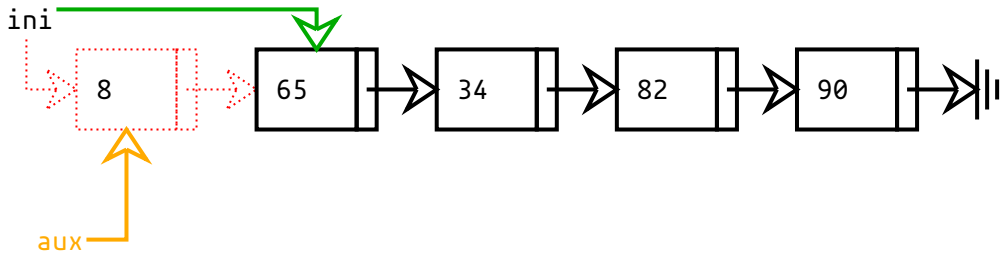
Si la lista no está vacía, crear un puntero **`aux`** que apunte al primer elemento.



El puntero al primer nodo **`ini`** debe apuntar al siguiente elemento.



Eliminar el primer elemento a través del puntero **`aux`**.

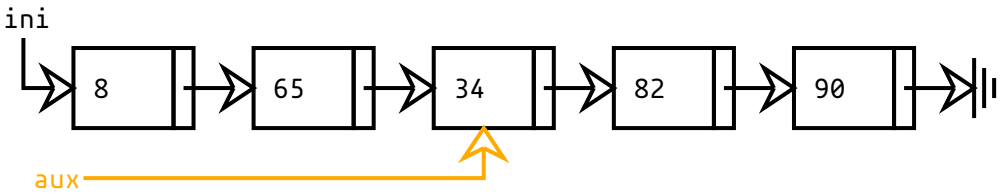


Código

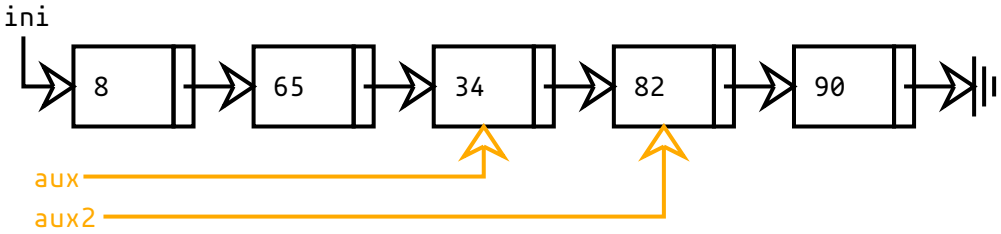
```
template <typename T, T NADA>
void Lista<T, NADA>::eliminaInicial() {
    if (lon > 0) {
        Nodo* aux = ini;
        ini = ini->sig;
        delete aux;
        lon--;
    }
}
```

Eliminar elemento en posición

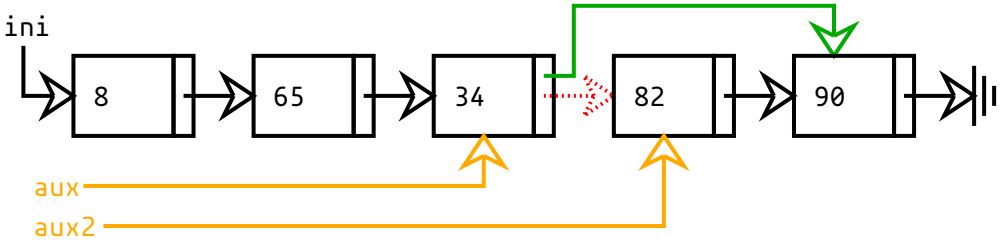
Avanzar un puntero **aux** hasta el elemento anterior al que se desea eliminar.



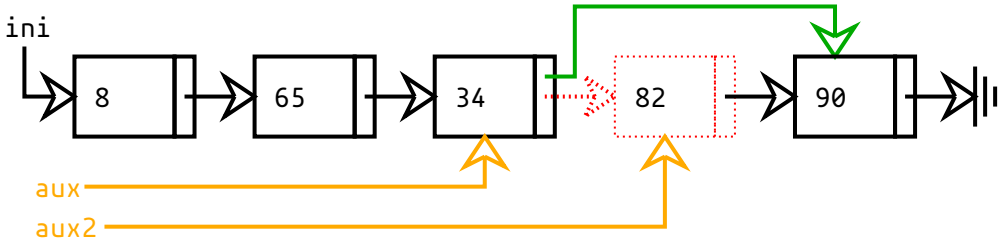
Posicionar un nuevo auxiliar **aux2** al elemento siguiente.



Enlazar al nodo apuntado por **aux** con el nodo posterior al elemento apuntado por **aux2**.



Eliminar el nodo deseado a través de **aux2**.



Ejercicio: Implementar el código de la función eliminar en posición y eliminar final.

```
template <typename T, T NADA>
void Lista<T, NADA>::eliminaPos(uint pos) {

}



template <typename T, T NADA>
void Lista<T, NADA>::eliminaFinal() {

}
```

Otras operaciones

```
template <typename T, T NADA>
void Lista<T, NADA>::agregaPos(T elem, uint pos) {
    if (pos > lon) return;
    if (pos == 0) {
        agregaInicial(elem);
    } else {
        Nodo* aux = ini;
        for (int i = 1; i < pos; i++) {
            aux = aux->sig;
        }
        Nodo* nuevo = new Nodo(elem, aux->sig);
        if (nuevo != nullptr) {
            aux->sig = nuevo;
            lon++;
        }
    }
}
```

```
template <typename T, T NADA>
void Lista<T, NADA>::agregaFinal(T elem) {
    agregarPos(elem, lon); // ;)
}



template <typename T, T NADA>
void Lista<T, NADA>::modificarInicial(T elem) {
    if (lon > 0) {
        ini->elem = elem;
    }
}



template <typename T, T NADA>
void Lista<T, NADA>::modificarPos(T elem, uint pos) {
    if (pos >= 0 && pos < lon) {
        Nodo* aux = ini;
        for (int i = 0; i < pos; i++) {
            aux = aux->sig;
        }
        aux->elem = elem;
    }
}



template <typename T, T NADA>
void Lista<T, NADA>::modificarFinal(T elem) {
    modificar(elem, lon - 1);
}



template <typename T, T NADA>
T Lista<T, NADA>::obtenerInicial() {
    return obtenerPos(0);
}
```

```
template <typename T, T NADA>
T Lista<T, NADA>::obtenerPos(uint pos) {
    if (pos >= 0 && pos < lon) {
        Nodo* aux = ini;
        for (int i = 0; i < pos; i++) {
            aux = aux->sig;
        }
        return aux->elem;
    } else {
        return NADA;
    }
}
```

```
template <typename T, T NADA>
T Lista<T, NADA>::obtenerFinal() {
    return obtenerPos(lon - 1);
}
```