

Phantone

www.wuolah.com/student/Phantone



ED-T3-3.pdf

Apuntes Teoría



2º Estructuras de Datos



Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



WUOLAH



El más PRO del lugar puedes ser Tú.



¿Quieres eliminar toda la publi de tus apuntes?



¡Fuera Publi!
Concéntrate al máximo



Apuntes a full.
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.



Quiero ser PRO

4,95 / mes

TEMA 3.3: LISTAS

Una lista es una ED lineal que contiene una secuencia de elementos, diseñada para hacer inserciones y borrados en cualquier posición.

La representaremos como $\langle a_1, a_2, \dots, a_n \rangle$.

Operaciones básicas

- **Set:** modifica el elemento de una posición.
- **Get:** devuelve el elemento de una posición.
- **Erase:** elimina el elemento de una posición.
- **Insert:** inserta un elemento en una posición.
- **Size:** devuelve el nº de elementos de la lista.

En una lista con n elementos consideramos $n+1$ posiciones, incluyendo la siguiente a la última, que llamaremos **fin de la lista**.

Esquema de la interfaz

```
#ifndef __LISTA_H__
#define __LISTA_H__

typedef char Tbase;

class Lista{
private:
    ... //La implementación que se elija
public:
    Lista();
    Lista(const Lista& l);
    ~Lista();
    Lista& operator=(const Lista& l);

    Tbase get(int pos) const;
    void set(int pos, Tbase e);
    void insert(int pos, Tbase e);
    void erase(int pos);
    int size() const;
};

#endif // __LISTA_H__
```

LISTAS: POSIBLES IMPLEMENTACIONES

- **Vectores:** parece sencilla: las posiciones que se pasan a los métodos son enteras y se traducen directamente en índices del vector. Inserciones y borrados ineficientes.
 - **Celdas enlazadas:** parece + eficiente: inserción y borrado no desplazan elementos. Los métodos `set`, `get`, `insertar` y `borrar` tienen orden lineal. Problema: pos. enteras.
- **Conclusión:** la implementación de las posiciones debe variar en función de la implementación de la lista. 😞😞

POSICIONES

Vamos a crear una abstracción de las posiciones, encapsulando el concepto de posición en una clase. Crearemos una clase **Posición**. Un objeto de la clase representará una posición en la lista.

- En el vector, se implementa como un entero.
- En las celdas enlazadas, se implementará como un puntero.

Observaciones

- Para una lista de tamaño n , habrá $n+1$ posiciones.
- El movimiento entre posiciones se hace de una en una.
- La comparación entre posiciones se limita a igualdad y desigualdad.

Esquema de la interfaz

```
#ifndef __LISTA_H__
#define __LISTA_H__

typedef char Tbase;

class Posicion{
private:
    ...      //La implementación que se elija

public:
    Posicion();
    Posicion(const Posicion& p);
    ~Posicion();
    Posicion& operator=(const Posicion& p);
    Posicion& operator++();
    Posicion operator++(int);
    Posicion& operator--();
    Posicion operator--(int);
    bool operator==(const Posicion& p) const;
    bool operator!=(const Posicion& p) const;
};

class Lista{
private:
    ...      //La implementación que se elija

public:
    Lista();
    Lista(const Lista& l);
    ~Lista();
    Lista& operator=(const Lista& l);

    Tbase get(const Posicion & p) const;
    void set(const Posicion & p, const Tbase & e);
    Posicion insert(const Posicion & p, const Tbase & e); ← Se modifican
    Posicion erase(const Posicion & p); ← Se modifican
    int size() const; ← size() no es fundamental
    Posicion begin() const; ← Necesitamos saber dónde
    Posicion end() const; ← empieza y acaba la lista
};

#endif // __LISTA_H__
```

begin() devuelve la posición del primer elemento
end() devuelve la posición posterior al último elemento (permite añadir al final)
En una lista vacía, begin() coincide con end()

Uso de una lista

```
#include <iostream>
#include "Lista.hpp"
using namespace std;
int main() {
    char dato;
    Lista l;

    cout << "Escriba una frase" << endl;
    while((dato=cin.get())!='\n')
        l.insert(l.end(), dato);
    cout << "La frase introducida es:" << endl;
    escribir(l);
    cout << "La frase en minúsculas:" << endl;
    escribir_minuscula(l);
    if(localizar(l, ' ')==l.end())
        cout << "La frase no tiene espacios" << endl;
    else{
        cout << "La frase sin espacios:" << endl;
        Lista aux(l);
        borrar_caracter(aux, ' ');
        escribir(aux);
    }
    cout << "La frase al revés: " << endl;
    escribir(al_reves(l));
    cout << (palindromo(l)? "Es " : "No es ") << "un palíndromo" << endl;
    return 0;
}
```



**El más PRO del lugar
puedes ser Tú.**

¿Quieres eliminar toda la publi de tus apuntes?

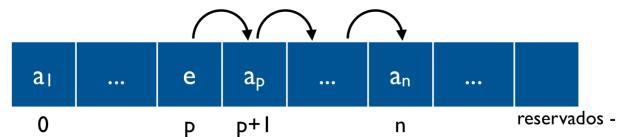
 Hazte PRO y elimina la publi de tus apuntes
4,95€ /mes

LISTAS: IMPLEMENTACIÓN CON VECTORES

Almacenamos la secuencia en un vector. Las posiciones son enteros.



- La posición `begin()` corresponde al 0.
- La posición `end()` corresponde a n .
- Las inserciones suponen desplazar elementos a la derecha y, los borrados, a la izquierda.



Lista.h

```
#ifndef __LISTA_H__
#define __LISTA_H__

typedef char Tbase;

class Posicion{
private:
    int i;
public:
    Posicion();
    Posicion(const Posicion& p) = default;
    ~Posicion() = default;
    Posicion& operator=(const Posicion& p) = default;
    Posicion& operator++();
    Posicion operator++(int);
    Posicion& operator--();
    Posicion operator--(int);
    bool operator==(const Posicion& p) const;
    bool operator!=(const Posicion& p) const;
    friend class Lista;
};

class Lista{
private:
    Tbase* datos;
    int nElementos;
    int reservados;
public:
    Lista();
    Lista(const Lista& l);
    ~Lista();
    Lista& operator=(const Lista& l);

    Tbase get(const Posicion & p) const;
    void set(const Posicion & p, const Tbase & e);
    Posicion insert(const Posicion & p, const Tbase & e);
    Posicion erase(const Posicion & p);
    int size() const;
    Posicion begin() const;
    Posicion end() const;
private:
    void liberar();
    void redimensionar(int n);
    void copiar(const Lista& l);
    void reservar(int n);
};

#endif // __LISTA_H__
```

W

WUOLAH

Lista.cpp

```
#include <cassert>
#include "Lista.hpp"

using namespace std;
//Clase Posicion

Posicion::Posicion(){
    i = 0;
}

//Operador ++ prefijo
Posicion& Posicion::operator++(){
    i++;
    return *this;
}

//Operador ++ postfijo
Posicion Posicion::operator++(int){
    Posicion aux(*this);
    i++;
    return aux;
}

//Operador -- prefijo
Posicion& Posicion::operator--(){
    i--;
    return *this;
}

//Operador -- postfijo
Posicion Posicion::operator--(int){
    Posicion aux(*this);
    i--;
    return aux;
}

bool Posicion::operator==(const Posicion& p) const{
    return i==p.i;
}

bool Posicion::operator!=(const Posicion& p) const{
    return i!=p.i;
}

//Métodos auxiliares privados

void Lista::liberar(){
    delete []datos;
    reservados = nelementos = 0;
}

void Lista::reservar(int n){
    assert(n>0);
    reservados = n;
    datos = new Tbase[reservados];
}

void Lista::copiar(const Lista& l){
    nelementos = l.nelementos;
    for(int i=0; i<nelementos; i++)
        datos[i] = l.datos[i];
}

void Lista::redimensionar(int n){
    assert(n>0 && n>=nelementos);
    Tbase* aux = datos;
    reservar(n);
    for(int i=0; i<nelementos; i++)
        datos[i] = aux[i];
    // memcpy(datos, aux, nelementos*sizeof(Tbase));
    delete[] aux;
}

//Clase Lista

Lista::Lista(){
    nelementos = 0;
    reservar(1);
}

Lista::Lista(const Lista& l){
    reservar(l.nelementos);
    copiar(l);
}

Lista::~Lista(){
    liberar();
}

Lista& Lista::operator=(const Lista &l){
    if (this != &l){
        liberar();
        reservar(l.nelementos);
        copiar(l);
    }
    return *this;
}

void Lista::set(const Posicion &p, const Tbase &e){
    assert(p.i>=0 && p.i<nelementos);
    datos[p.i] = e;
}

Tbase Lista::get(const Posicion &p) const{
    assert(p.i>=0 && p.i<nelementos);
    return datos[p.i];
}

Posicion Lista::insert(const Posicion &p, const Tbase &e){
    if(nelementos == reservados)
        redimensionar(reservados*2);
    for(int j=nelementos; j>p.i; j--)
        datos[j] = datos[j-1];
    datos[p.i] = e;
    nelementos++;
    return p;
}

Posicion Lista::erase(const Posicion &p){
    assert(p!=end());
    for(int j=p.i; j<nelementos-1; j++)
        datos[j] = datos[j+1];
    nelementos--;
    if (nelementos<reservados/4)
        redimensionar(reservados/2);
    return p;
}

int Lista::size() const{
    return nelementos;
}

Posicion Lista::begin() const{
    Posicion p;
    p.i = 0; //Innecesario
    return p;
}

Posicion Lista::end() const{
    Posicion p;
    p.i = nelementos;
    return p;
}
```

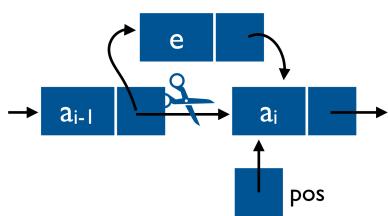
LISTAS: IMPLEMENTACIÓN CON CELDAS ENLAZADAS

Almacenamos la secuencia de valores en celdas enlazadas.

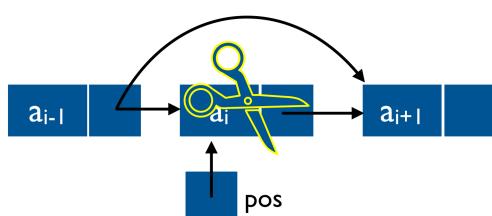


- Una lista es un puntero a la primera celda (si no está vacía).
- Una posición son dos punteros. El 2º se necesita para algunos operadores.
- Inserción / borrado en la primera posición son casos especiales.

Inserción

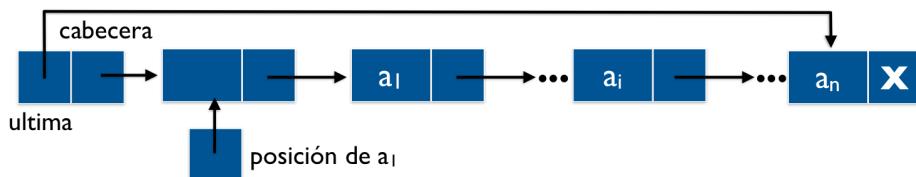


Borrado



LISTAS: IMPLEMENTACIÓN CON CELDAS ENLAZADAS CON CABECERA

Almacenamos la secuencia de valores en celdas enlazadas.



- Una lista es un puntero a la primera celda (si vacía, sólo tiene una celda).
- Una posición son dos punteros. El 2º se necesita para algunos operadores.
- Inserción / borrado en la primera posición **NO** son casos especiales.

Lista.h

```
#ifndef __LISTA_H__
#define __LISTA_H__

typedef char Tbase;

struct CeldaLista{
    Tbase elemento;
    CeldaLista* siguiente;
};

class Lista;

class Posicion{
private:
    CeldaLista* puntero;
    CeldaLista* primera;
public:
    Posicion();
    Posicion(const Posicion& p) = default;
    ~Posicion() = default;
    Posicion& operator=(const Posicion& p) = default;
    Posicion& operator++();
    Posicion& operator++(int);
    Posicion& operator--();
    Posicion& operator--(int);
    bool operator==(const Posicion& p) const;
    bool operator!=(const Posicion& p) const;
    friend class Lista;
};
```

```
class Lista{
private:
    CeldaLista* cabecera;
    CeldaLista* ultima;
public:
    Lista();
    Lista(const Lista& l);
    ~Lista();
    Lista& operator=(const Lista& l);

    Tbase get(Posicion p) const;
    void set(Posicion p, Tbase e);
    Posicion insert(Posicion p, Tbase e);
    Posicion erase(Posicion p);
    int size() const;
    Posicion begin() const;
    Posicion end() const;
};

#endif // __LISTA_H__
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

Lista.cpp

```
#include <cassert>
#include <utility>
#include "Lista.hpp"
using namespace std;

//Clase Posicion

Posicion::Posicion(){
    primera = puntero = 0;
}

//Operador ++ prefijo
Posicion& Posicion::operator++(){
    puntero = puntero->siguiente;
    return *this;
}

//Operador ++ postfijo
Posicion Posicion::operator++(int){
    Posicion p(*this);
    //operator++();
    ++(*this);
    return p;
}

//Operador -- prefijo
Posicion& Posicion::operator--(){
    assert(puntero!=primera);
    CeldaLista* aux = primera;
    while(aux->siguiente!=puntero){
        aux = aux->siguiente;
    }
    puntero = aux;
    return *this;
}

//Operador -- postfijo
Posicion Posicion::operator--(int){
    Posicion p(*this);
    //operator--();
    --(*this);
    return p;
}

bool Posicion::operator==(const Posicion & p) const{
    return(puntero==p.puntero);
}

bool Posicion::operator!=(const Posicion & p) const{
    return(puntero!=p.puntero);
}

//Clase Lista

Lista::Lista(){
    ultima = cabecera = new CeldaLista;
    cabecera->siguiente = 0;
}

Lista::Lista(const Lista& l){
    ultima = cabecera = new CeldaLista;
    CeldaLista* orig = l.cabecera;
    while(orig->siguiente!=0){
        ultima->siguiente = new CeldaLista;
        ultima = ultima->siguiente;
        orig = orig->siguiente;
        ultima->elemento = orig->elemento;
    }
    ultima->siguiente = 0;
}

Lista::~Lista(){
    CeldaLista* aux;
    while(cabecera!=0){
        aux = cabecera;
        cabecera = cabecera->siguiente;
        delete aux;
    }
}
```

```
Lista& Lista::operator=(const Lista& l){
    Lista aux(l);
    swap(cabecera, aux.cabecera);
    swap(ultima, aux.ultima);
    return *this;
}

void Lista::set(Posicion p, Tbase e){
    p.puntero->siguiente->elemento = e;
}

Tbase Lista::get(Posicion p) const{
    return p.puntero->siguiente->elemento;
}

Posicion Lista::insert(Posicion p, Tbase e){
    CeldaLista* nueva = new CeldaLista;
    nueva->elemento = e;
    nueva->siguiente = p.puntero->siguiente;
    p.puntero->siguiente = nueva;
    if(p.puntero == ultima)
        ultima = nueva;
    return p;
}

Posicion Lista::erase(Posicion p){
    assert(p!=end());
    CeldaLista* aux = p.puntero->siguiente;
    p.puntero->siguiente = aux->siguiente;
    if(aux==ultima)
        ultima = p.puntero;
    delete aux;
    return p;
}

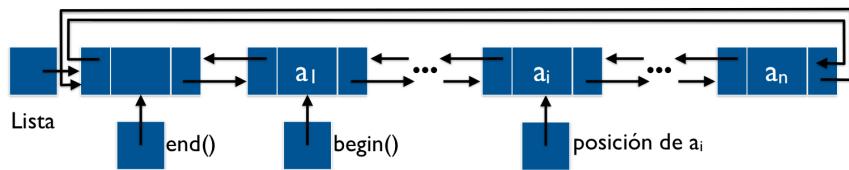
int Lista::size() const{
    int n=0;
    for(Posicion p=begin(); p!=end(); ++p)
        n++;
    return n;
}

Posicion Lista::begin() const{
    Posicion p;
    p.puntero = p.primera = cabecera;
    return p;
}

Posicion Lista::end() const{
    Posicion p;
    p.puntero = ultima;
    p.primera = cabecera;
    return p;
}
```

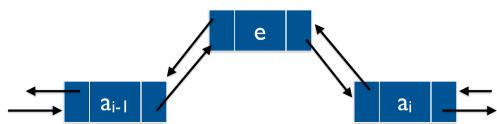
LISTAS: IMP. CON CELDAS DOBLEMENTE ENLAZADAS CIRCULARES

Almacenamos la secuencia en celdas doblemente enlazadas.



- Una lista es un puntero a la cabecera (si vacía, sólo tiene una celda).
- Una posición es un único puntero a la celda.
- Inserciones / borrados son independientes de la posición.

Inserción



Borrado



Lista.h

```
#ifndef __LISTA_H__
#define __LISTA_H__

typedef char Tbase;

struct CeldaLista{
    Tbase elemento;
    Celdalista* anterior;
    Celdalista* siguiente;
};

class Lista;

class Posicion{
private:
    Celdalista* puntero;
public:
    Posicion();
    Posicion(const Posicion& p) = default;
    ~Posicion() = default;
    Posicion& operator=(const Posicion& p) = default;
    Posicion& operator++();
    Posicion operator++(int);
    Posicion& operator--();
    Posicion operator--(int);
    bool operator==(const Posicion& p) const;
    bool operator!=(const Posicion& p) const;
    friend class Lista;
};

class Lista{
private:
    Celdalista* cabecera;
public:
    Lista();
    Lista(const Lista& l);
    ~Lista();
    Lista& operator=(const Lista& l);
    void set(Posicion p, Tbase e);
    Tbase get(Posicion p) const;
    Posicion insert(Posicion p, Tbase e);
    Posicion erase(Posicion p);
    int size() const;
    Posicion begin() const;
    Posicion end() const;
};
#endif //__LISTA_H__
```

Lista.cpp

```
#include <cassert>
#include "Lista.hpp"
using namespace std;

//Clase Posicion

Posicion::Posicion(){
    puntero = 0;
}

//Operador ++ prefijo
Posicion& Posicion::operator++(){
    puntero = puntero->siguiente;
    return *this;
}

//Operador ++ postfixo
Posicion Posicion::operator++(int){
    Posicion p(*this);
    ++(*this);
    return p;
}

//Operador -- prefijo
Posicion& Posicion::operator--(){
    puntero = puntero->anterior;
    return *this;
}

//Operador -- postfixo
Posicion Posicion::operator--(int){
    Posicion p(*this);
    --(*this);
    return p;
}

bool Posicion::operator==(const Posicion& p) const{
    return (puntero==p.puntero);
}

bool Posicion::operator!=(const Posicion& p) const{
    return (puntero!=p.puntero);
}

Lista::Lista(){
    cabecera = new CeldaLista;
    cabecera->siguiente = cabecera;
    cabecera->anterior = cabecera;
}

Lista::Lista(const Lista& l){
    cabecera = new CeldaLista;
    cabecera->siguiente = cabecera;
    cabecera->anterior = cabecera;

    CeldaLista* p = l.cabecera->siguiente;
    while(p!=l.cabecera){
        CeldaLista* q = new CeldaLista;
        q->elemento = p->elemento;
        q->anterior = cabecera->anterior;
        cabecera->anterior->siguiente = q;
        cabecera->anterior = q;
        q->siguiente = cabecera;
        p = p->siguiente;
    }
}

Lista::~Lista(){
    while(begin()!=end())
        erase(begin());
    delete cabecera;
}

Lista& Lista::operator=(const Lista &l){
    Lista aux(l);
    swap(this->cabecera, aux.cabecera);
    return *this;
}

void Lista::set(Posicion p, Tbase e){
    p.puntero->elemento = e;
}

Tbase Lista::get(Posicion p) const{
    return p.puntero->elemento;
}

Posicion Lista::insert(Posicion p, Tbase e){
    CeldaLista* q = new CeldaLista;
    q->elemento = e;
    q->anterior = p.puntero->anterior;
    q->siguiente = p.puntero;
    p.puntero->anterior = q;
    q->anterior->siguiente = q;
    p.puntero = q;
    return p;
}

Posicion Lista::erase(Posicion p){
    assert(p!=end());
    CeldaLista* q = p.puntero;
    q->anterior->siguiente = q->siguiente;
    q->siguiente->anterior = q->anterior;
    p.puntero = q->siguiente;
    delete q;
    return p;
}

int Lista::size() const{
    int n = 0;
    for(Posicion p=begin(); p!= end(); p++)
        n++;
    return n;
}

Posicion Lista::begin() const{
    Posicion p;
    p.puntero = cabecera->siguiente;
    return p;
}

Posicion Lista::end() const{
    Posicion p;
    p.puntero = cabecera;
    return p;
}
```