



Phantone

www.wuolah.com/student/Phantone



ED-T3-2.pdf

Apuntes Teoría



2º Estructuras de Datos



Grado en Ingeniería Informática



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



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

¿Quieres eliminar toda la publi
de tus apuntes?

Hazte PRO

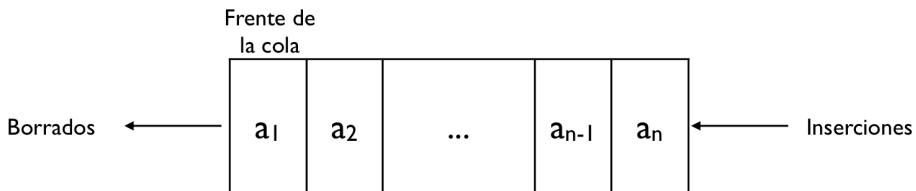
4,95€/mes

W

TEMA 3.2: COLAS

Una cola es una ED lineal en la que los elementos se insertan y borran en extremos opuestos.

Se caracteriza por su comportamiento FIFO.



Operaciones básicas

- **Frente:** devuelve el elemento del frente.
- **Poner:** añade un elemento al final de la cola.
- **Quitar:** elimina el elemento del frente.
- **Vacia:** indica si la cola está vacía.

Esquema de la interfaz

```
#ifndef __COLA_H__
#define __COLA_H__

typedef char Tbase;

class Cola{
private:
    ...
public:
    Cola();
    Cola(const Cola& c);
    ~Cola();
    Cola& operator=(const Cola& c);

    bool vacia() const;
    void poner(const Tbase valor);
    void quitar();
    Tbase frente() const;
};

#endif // __COLA_H__
```

Uso de una Cola

```
#include <iostream>
#include "Pila.hpp"
#include "Cola.hpp"
using namespace std;

int main() {
    Pila p;
    Cola c;
    char dato;
    cout << "Escriba una frase" << endl;
    while((dato=cin.get()) != '\n'){
        if (dato != ' '){
            p.poner(dato);
            c.poner(dato);
        }
    }
    bool palindromo = true;
    while(!p.vacia() && palindromo){
        if(c.frente() != p.tope())
            palindromo = false;
        p.quitar();
        c.quitar();
    }
    cout << "La frase "
        << (palindromo?"es":"no es")
        << " un palíndromo" << endl;
    return 0;
}
```

Tbase & frente();
const Tbase & frente() const;



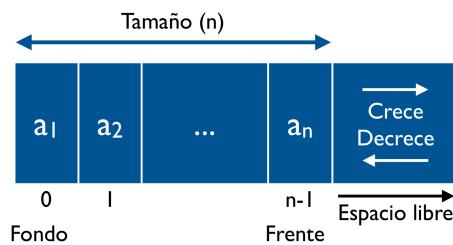
Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



COLAS: IMPLEMENTACIÓN CON VECTORES

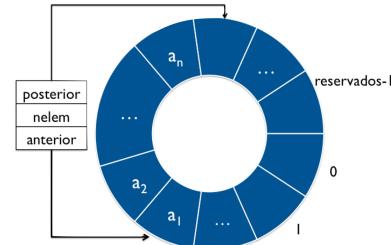
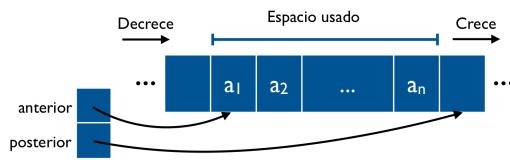
Almacenamos la secuencia de valores en un vector.



- El fondo está en la posición 0.
- El número de elementos varía (lo almacenamos).
- El vector puede agotarse cuando insertamos. Resolvemos con memoria dinámica.
- Problema: no garantiza $O(1)$ en inserciones y borrados.

COLAS: IMPLEMENTACIÓN CON VECTORES CIRCULARES

Almacenamos la secuencia de valores en un vector.



Cola.h (vect. circular)

```
#ifndef __COLA_H__
#define __COLA_H__

typedef char Tbase;

class Cola{
private:
    Tbase * datos;
    int reservados;
    int nelem;
    int anterior, posterior;
public:
    Cola();
    Cola(const Cola & c);
    ~Cola();
    Cola& operator=(const Cola & c);
    bool vacia() const;
    void poner(const Tbase & valor);
    Tbase frente() const;
    void reservar(const int n);
    void liberar();
    void copiar(const Cola & c);
    void redimensionar(const int n);
};
```

#endif // __COLA_H__

Cola.cpp (vect. circular)

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

Cola::Cola(){
    reservar(10);
    anterior = posterior = nelem = 0;
}

Cola::Cola(const Cola & c){
    reservar(c.reservados);
    copiar(c);
}

Cola& Cola::operator=(const Cola & c){
    if(this!=&c){
        liberar();
        reservar(c.reservados);
        copiar(c);
    }
    return(*this);
}

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

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

void Cola::liberar(){
    delete[] datos;
    datos = 0;
    anterior = posterior = nelem = reservados = 0;
}

void Cola::copiar(const Cola & c){
    for(int i=c.anterior; i!=c.posterior; i=(i+1)%reservados)
        datos[i] = c.datos[i];
    anterior = c.anterior;
    posterior = c.posterior;
    nelem = c.nelem;
}

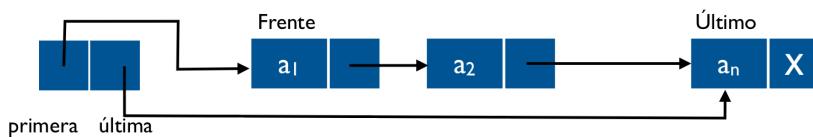
void Cola::redimensionar(const int n){
    assert(n>0 && n>=nelem);
    Tbase* aux = datos;
    int tam_aux = reservados;
    reservar(n);
    for(int i=0; i<nelem; i++)
        datos[i] = aux[(anterior+i)%tam_aux];
    anterior = 0;
    posterior = nelem;
    delete[] aux;
}
```



WUOLAH

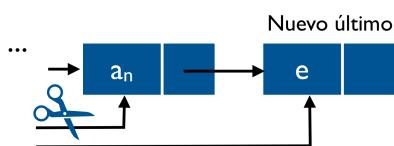
COLAS: IMPLEMENTACIÓN CON CELDAS ENLAZADAS

Almacenamos la secuencia en celdas enlazadas.

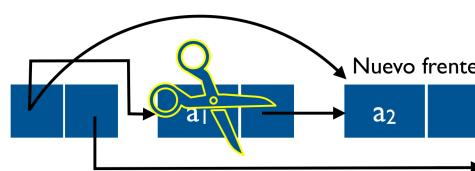


- Una cola vacía tiene dos punteros nulos.
- El frente está en la primera celda (eficiente).
- En la inserción se añade una celda al final y en el borrado se elimina la primera celda.

Inserción



Borrado



Cola.h

```
#ifndef __COLA_H__
#define __COLA_H__

typedef char Tbase;

struct CeldaCola{
    Tbase elemento;
    CeldaCola* sig;
};

class Cola{
private:
    CeldaCola* primera, *ultima;
public:
    Cola();
    Cola(const Cola& c);
    ~Cola();
    Cola& operator=(const Cola& c);
    bool vacia() const;
    void poner(const Tbase & c);
    void quitar();
    Tbase frente() const;
private:
    void copiar(const Cola& c);
    void liberar();
};

#endif // __COLA_H__
```

Cola.cpp

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

Cola::Cola(){
    primera = ultima = 0;
}

Cola::Cola(const Cola& c){
    copiar(c);
}

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

Cola& Cola::operator=(const Cola &c){
    if(this!=&c){
        liberar();
        copiar(c);
    }
    return *this;
}

bool Cola::vacia() const{
    return (primera == 0);
}

void Cola::poner(const Tbase & c){
    //Creamos una nueva celda
    CeldaCola* nueva = new CeldaCola;
    nueva->elemento = c;
    nueva->sig = 0;
    //Conectamos la celda
    if (primera==0) //Cola vacía
        primera = ultima = nueva;
    else{ //Cola no vacía
        ultima->sig = nueva;
        ultima = nueva;
    }
}

void Cola::quitar(){
    //Comprobamos que la cola no está vacía
    assert(primera!=0);
    //Hacemos que primera apunte
    //a la siguiente celda
    CeldaCola* aux = primera;
    primera = primera->sig;
    //Borramos la celda
    delete aux;
    //Si la cola queda vacía, tenemos que ajustar ultima
    if (primera==0)
        ultima = 0;
}

Tbase Cola::frente() const{
    //Comprobamos que no está vacía
    assert(primera!=0);
    return primera->elemento;
}
```

```
void Cola::copiar(const Cola& c){
    if (c.primera == 0) //Si la cola está vacía
        primera = ultima = 0;
    else{ //Caso general. No está vacía
        //Creamos la primera celda
        primera = new CeldaCola;
        primera->elemento = c.primera->elemento;
        ultima = primera;
        //Recorremos y copiamos el resto de la cola
        CeldaCola* orig = c.primera;
        while(orig->sig != 0){
            orig = orig->sig;
            ultima->sig = new CeldaCola;
            ultima = ultima->sig;
            ultima->elemento = orig->elemento;
        }
        ultima->sig = 0;
    }
}

void Cola::liberar(){
    CeldaCola* aux;
    while(primera!=0){
        aux = primera;
        primera = primera->sig;
        delete aux;
    }
    ultima = 0;
}
```