



Algoritmos y Estructura de datos

Clase 10
Colas

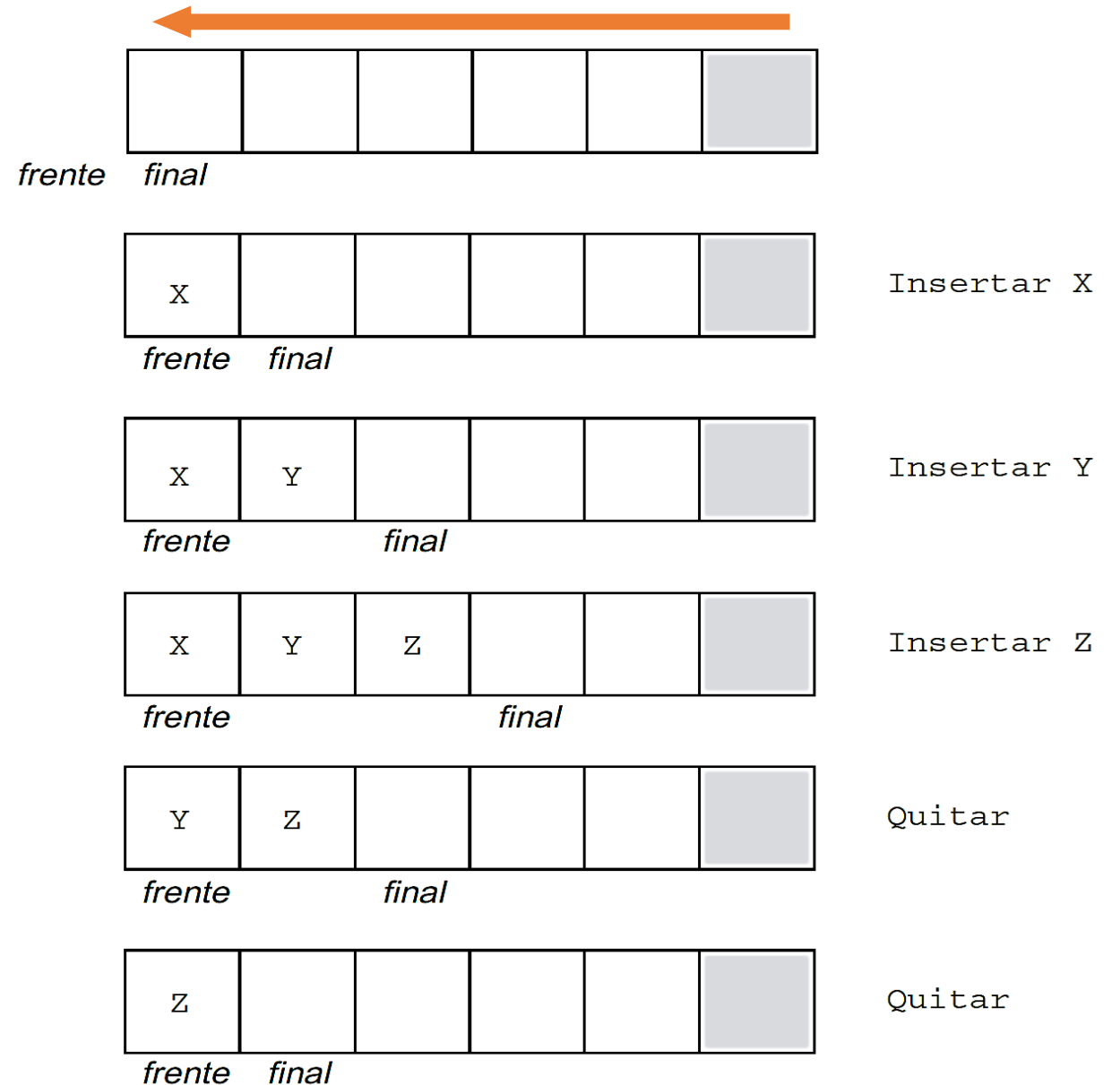
Profesor: Carlos Diaz

Contenido

- Concepto de Cola
- Implementación de una cola
- Operaciones en una cola
- Implementando una cola con lista enlazada
- Clase NodoCola
- Clase Cola
- Creando la cola y sus funciones
- Bicola
- Ejercicios

Concepto de Cola

- Una cola es una estructura de datos cuyos elementos sólo se pueden añadir por un extremo (**final** de la cola) y extraer o eliminar por el otro extremo (**frente** de la cola).
- Los elementos se eliminan (se quitan) de la cola en el mismo orden en que se almacenan y, por consiguiente, una cola es una estructura de tipo **FIFO** (*first-in, first-out / primero en entrar, primero en salir*).
- Veamos un ejemplo:



Implementación de una Cola

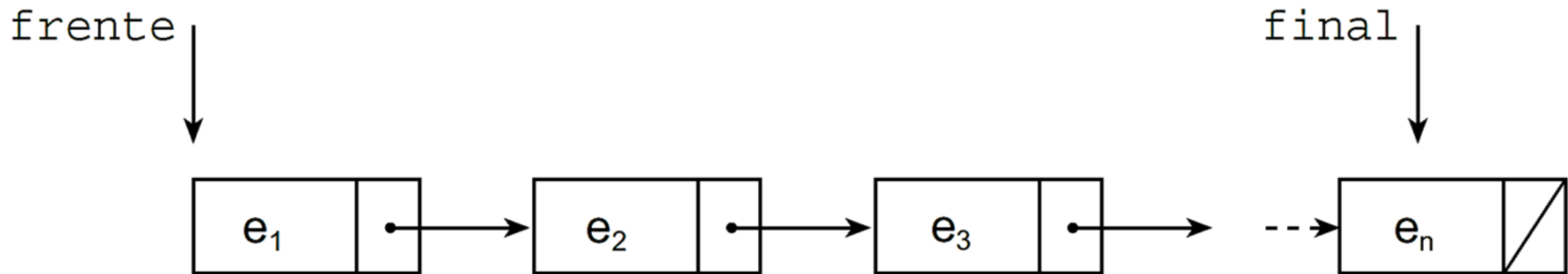
- Una **cola** es similar a una pila, pues los datos se almacenan de modo lineal y el acceso a los datos sólo está permitido en los extremos de la cola.
- La implementación de una **cola** depende de donde se almacenen los elementos (en un array o en una estructura dinámica).
- La utilización de arrays tiene el problema de que la **cola** no puede crecer indefinidamente, está limitada por el tamaño del array, como contrapartida el acceso a los extremos es muy eficiente.
- Utilizar una lista dinámica permite que el número de nodos se ajuste al de elementos de la **cola**, cada nodo necesita memoria extra para el enlace y también está el límite de memoria del computador.

Operaciones en una Cola

- Las operaciones que definen la estructura de una cola son las siguientes:
 - **CrearCola:** Inicia la cola como vacía.
 - **Insertar:** Añade un elemento por el final de la cola.
 - **Quitar:** Retira (extrae) el elemento frente de la cola.
 - **Cola vacía:** Comprobar si la cola no tiene elementos.
 - **Cola llena:** Comprobar si la cola está llena de elementos.
 - **Frente Cola:** Obtiene el elemento frente o primero de la cola.
 - **Final Cola:** Obtiene el elemento final o último de la cola.
 - **Devolver Frente:** Devuelve el nodo del frente.
 - **Devolver Final:** Devuelve el nodo del final

Implementando la Cola con lista enlazada

- La implementación del TAD Cola con una lista enlazada utiliza dos punteros de acceso a la lista: **frente** y **final**. Son los extremos por donde salen y por donde se ponen, respectivamente, los elementos de la cola.
- La lista enlazada crece y decrece, según se incorporen elementos o se retiren, y por esta razón en esta implementación no se considera la función de control de colallena().



Clase NodoCola

- La clase **NodoCola** representa un nodo de la cola que contiene una variable **siguiente** que apuntará al siguiente elemento que se introduzca en la cola; y una variable **elemento** que contiene el valor almacenado en la cola.
- Inicialmente este nodo se crea conteniendo el valor del elemento y apuntando a **NULL**.

```
#include <iostream>
using namespace std;
class NodoCola
{
    public:
    NodoCola* siguiente;
    int elemento;
    NodoCola (int x)
    {
        elemento = x;
        siguiente = NULL;
    }
};
```

Clase Cola

- La clase **Cola** se encarga de crear la cola.
- Para ello, crea dos **NodoCola**, **frente** y **final**, al principio ambos se inicializan a **NULL**, indicando que la cola está vacía.
- También contiene la declaración de las funciones que permitirán manipular la cola.

```
class Cola
{
    private:
        NodoCola* frente;
        NodoCola* final;
    public:
        Cola() //Constructor que crea la cola vacía
        {
            frente = final = NULL;
        }
        void insertar(int); //Inserta un elemento al final de la cola
        int quitar(); //Retira un elemento del frente de la cola
        int frenteCola(); //Devuelve el elemento del frente de la cola
        int finalCola(); //Devuelve el elemento del final de la cola
        bool colaVacía(); //Verdadero si la cola esta vacía
        NodoCola* devolverFrente(); //Devuelve el nodo de el frente
        NodoCola* devolverFinal(); //Devuelve el nodo de el frente
};
```


La función insertar()

- En primer lugar la función `insertar()` crea un nodo `nuevo` y le asigna el `elemento`.
- Luego verifica si la cola esta vacía, si es así, el `nuevo` se convierte automáticamente en el `frente`.
- El caso que no esta vacía, coloca el nodo al final de la cola mediante:
`(final -> siguiente) = nuevo;`
- Finalmente, `nuevo` se convierte en el `final`.

- Retorna `Verdadero` si la cola esta vacía, es decir si el `frente` es igual a `NULL`.
- Cuando esta condición se cumpla el `final` se iguala a `frente` para que ambos apunten a `NULL`.

```
void Cola::insertar(int elemento)
{
    NodoCola* nuevo;
    nuevo = new NodoCola (elemento);
    if (colaVacia())
    {
        frente = nuevo;
    }
    else
    {
        (final -> siguiente) = nuevo;
    }
    final = nuevo;
}
```

```
bool Cola::colaVacia()
{
    if (frente==NULL)
        frente=final=NULL;
    return frente == NULL;
}
```

La función quitar()

- En primer lugar la función quitar() verifica si la pila está vacía.
- Si la pila esta vacía muestra un mensaje.
- Si no está vacía, la función quitar() obtiene el **elemento** del **frente**.
- Luego crea un nodo **a** y lo iguala al nodo **frente**.
- Cambia el **frente** al siguiente nodo mediante:

frente = (frente -> siguiente);

- Elimina de la memoria el nodo **frente** y retorna el elemento.

```
int Cola::quitar()
{
    if (colaVacia())
    {
        cout<<"Cola vacia, no se puede extraer.\n";
    }
    else
    {
        cout<<"Se elimino de la memoria el nodo: "<<frente;
        int x = (frente -> elemento);
        NodoCola* a = frente;
        frente = (frente -> siguiente);
        delete a;
        return x;
    }
}
```

Las función mostrarCola()

- La función `mostrarCola()` llama a la función `quitar()` para retirar elementos a la cola, desde el frente, y eliminarlos de la memoria de la pc. Opcionalmente se muestra como se mantiene la variable `final`, mientras la variable `frente` decrementa a medida que eliminamos más elementos.

```
void mostrarCola(Cola &cola)
{
    int x;
    cout << "\nElementos de la Pila:" << endl;
    while (!cola.colaVacia())
    {
        cout << "Direccion frente: " << cola.devolverFrente() << " . Elemento frente: " << cola.frenteCola() << endl;
        cout << "Direccion final: " << cola.devolverFinal() << " . Elemento final: " << cola.finalCola() << endl;
        x = cola.quitar();
        cout << " . " << x << endl;
    }
    cout << "Direccion frente: " << cola.devolverFrente() << " . Elemento frente: " << cola.frenteCola() << endl;
    cout << "Direccion final: " << cola.devolverFinal() << " . Elemento final: " << cola.finalCola() << endl;
}
```

Funciones devolverFrente(), frenteCola(), devolverFinal() y finalCola()

```
NodoCola* Cola::devolverFrente()
{
    return frente;
}
```

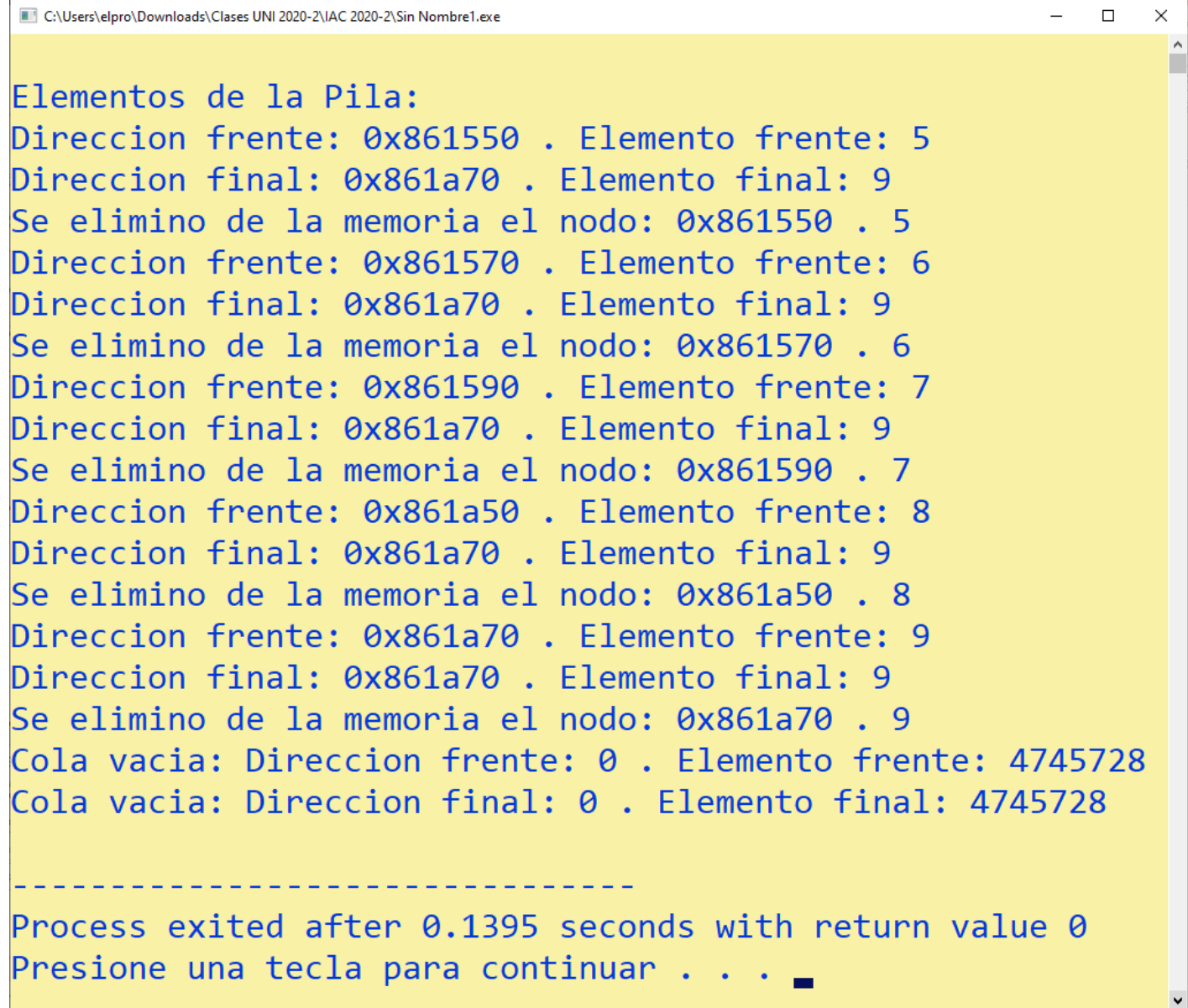
```
NodoCola* Cola::devolverFinal()
{
    return final;
}
```

```
int Cola::frenteCola()
{
    if (colaVacia())
    {
        cout<<"Cola vacia: ";
    }
    else
        return frente -> elemento;
}
```

```
int Cola::finalCola()
{
    if (colaVacia())
    {
        cout<<"Cola vacia: ";
    }
    else
        return final -> elemento;
}
```

Probando el código

```
int main()
{
    Cola cola;
    cola.insertar(5);
    cola.insertar(6);
    cola.insertar(7);
    cola.insertar(8);
    cola.insertar(9);
    mostrarCola(cola);
    return 0;
}
```



```
C:\Users\elpro\Downloads\Clases UNI 2020-2\IAC 2020-2\Sin Nombre1.exe

Elementos de la Pila:
Direccion frente: 0x861550 . Elemento frente: 5
Direccion final: 0x861a70 . Elemento final: 9
Se elimino de la memoria el nodo: 0x861550 . 5
Direccion frente: 0x861570 . Elemento frente: 6
Direccion final: 0x861a70 . Elemento final: 9
Se elimino de la memoria el nodo: 0x861570 . 6
Direccion frente: 0x861590 . Elemento frente: 7
Direccion final: 0x861a70 . Elemento final: 9
Se elimino de la memoria el nodo: 0x861590 . 7
Direccion frente: 0x861a50 . Elemento frente: 8
Direccion final: 0x861a70 . Elemento final: 9
Se elimino de la memoria el nodo: 0x861a50 . 8
Direccion frente: 0x861a70 . Elemento frente: 9
Direccion final: 0x861a70 . Elemento final: 9
Se elimino de la memoria el nodo: 0x861a70 . 9
Cola vacia: Direccion frente: 0 . Elemento frente: 4745728
Cola vacia: Direccion final: 0 . Elemento final: 4745728

-----
Process exited after 0.1395 seconds with return value 0
Presione una tecla para continuar . . . █
```

La función completarCola()

- La función `completarCola()` llama a la función `insertar()` para insertar elementos a la cola, desde la parte final. Opcionalmente se muestra como se mantiene la variable frente, mientras la variable final se incrementa a medida que ingresamos más elementos.

```
void completarCola(Cola &cola)
{
    int x, n;
    cout<<"Cuantos elementos ingresara a la cola: ";
    cin>>n;
    cout<<"Digite el elemento y luego pulse Enter\n";
    for (int i = 0; i < n ; i++)
    {
        cout<<"Direccion frente: "<<cola.devolverFrente()<<" . Elemento frente: "<<cola.frenteCola()<<endl;
        cout<<"Direccion final: "<<cola.devolverFinal()<<" . Elemento final: "<<cola.finalCola()<<endl;
        cout<<"indice "<<i<<": ";
        cin >> x;
        cola.insertar(x);
    }
    cout<<"Direccion frente: "<<cola.devolverFrente()<<" . Elemento frente: "<<cola.frenteCola()<<endl;
    cout<<"Direccion final: "<<cola.devolverFinal()<<" . Elemento final: "<<cola.finalCola()<<endl;
}
```

Creación de una cola

- Para crear una cola, el usuario debe usar las funciones definidas en la clase Cola.
- En la función main() creamos un objeto cola y llámanos a dos funciones:
- `completarCola()`: Se encargará de insertar elementos a la cola.
- `mostrarCola()`: Se encargará de mostrar los elementos de la cola.

```
int main()
{
    Cola cola;
    completarCola(cola);
    mostrarCola(cola);
    return 0;
}
```

Concepto de Bicola

- Una *bicola* o *cola de doble entrada* es un conjunto *ordenado* de elementos al que se puede *añadir* o *quitar* desde cualquier extremo del mismo.
- El acceso a la bicola está permitido desde cualquier extremo, por lo que se considera que es una *cola bidireccional*. La estructura *bicola* es una extensión del *TAD Cola*.
- Los dos extremos de una bicola se identifican con los apuntadores *frente* y *final* (mismos nombres que en una cola). Las operaciones básicas que definen una bicola son una ampliación de la operaciones de una cola:
 - *CrearBicola* : Inicializa una bicola sin elementos.
 - *PonerFrente* : Añade un elemento por extremo frente.
 - *PonerFinal* : Añade un elemento por extremo final.
 - *QuitarFrente* : Devuelve el elemento frente y lo retira de la bicola.
 - *QuitarFinal* : Devuelve el elemento final y lo retira de la bicola.
 - *Frente* : Devuelve el elemento frente de la bicola.
 - *Final* : Devuelve el elemento final de la bicola.
 - *Numero de elementos*: Devuelve la cantidad de elementos de la bicola,

Clase Bicola

```
class Bicola
{
    private:
        NodoCola* frente;
        NodoCola* final;
    public:
        Bicola()
        {
            frente = final = NULL;
        }
        void insertar(int elemento);
        int quitar();
        int frenteCola();
        int finalCola();
        bool colaVacia();
        NodoCola* devolverFrente();
        NodoCola* devolverFinal();
};
```

```
//Funciones para la Bicola
void ponerFinal(int); //Equivalente a insertar
void ponerFrente(int);
int quitarFrente(); // Equivalente a quitar
int quitarFinal();
int frenteBicola(); // Equivalente a frenteCola
int finalBicola();
bool bicolaVacia(); // Equivalente a colaVacia
int numElemsBicola();
};
```

Las funciones iguales a la cola

```
//Devuelve el nodo frente
NodoCola* Bicola::devolverFrente()
{
    return frente;
}
```

```
//Devuelve el nodo final
NodoCola* Bicola::devolverFinal()
{
    return final;
}
```

```
bool Bicola::colaVacia()
{
    if(frente == 0)
        frente=final=0;
    return frente == NULL;
}
```

```
//Inserta un elemento a la bicola por el final
void Bicola::insertar(int elemento)
{
    NodoCola* nuevo;
    nuevo = new NodoCola (elemento);
    if (colaVacia())
    {
        frente = nuevo;
    }
    else
    {
        (final -> siguiente) = nuevo;
    }
    final = nuevo;
}
```

Las funciones iguales a la cola

//Retirar un elemento de la bicola por el frente

```
int Bicola::quitar()
{
    if (colaVacia())
    {
        cout<<"Cola vacia, no se puede extraer.\n";
    }
    else
    {
        cout<<"Se elimino de la memoria el nodo: "<<frente;
        int x = frente -> elemento;
        NodoCola* a = frente;
        frente = frente -> siguiente;
        delete a;
        return x;
    }
}
```

//Devuelve el elemento frente de la cola

```
int Bicola::frenteCola()
{
    if (colaVacia())
    {
        cout<<"Cola vacia: ";
    }
    else
        return frente -> elemento;
}
```

//Devuelve el elemento final de la cola

```
int Bicola::finalCola()
{
    if (colaVacia())
    {
        cout<<"Cola vacia: ";
    }
    else
        return final -> elemento;
}
```

Funciones de la Bicola

//Poner elemento al frente

```
void Bicola::ponerFrente(int elemento)
{
    NodoCola* nuevo;
    nuevo = new NodoCola(elemento);
    if (bicolaVacia())
    {
        final = nuevo;
    }
    else
    {
        (nuevo -> siguiente)= frente;
    }
    frente = nuevo;
}
```

//Quitar elemento del final

```
int Bicola::quitarFinal()
{
    int x;
    if (!bicolaVacia())
    {
        if (frente == final) // Bicola dispone de un solo nodo
        {
            x = quitar();
        }
        else
        {
            cout<<"Se elimino de la memoria el nodo: "<<final;
            NodoCola* a = frente; //recorrerá la cola
            while (a -> siguiente != final)
                a = a -> siguiente;
            x = final -> elemento;
            final = a;
            delete (a -> siguiente);
            final->siguiente=NULL;
        }
    }
    else
    {
        cout<<"Bicola vacia\n";
        return 0; //NULL
    }
}
```

Funciones de la Bicola

//Devuelve el elemento final

```
int Bicola::finalBicola()
{
    if (bicolaVacia())
    {
        cout<<"Error: bicola vacía\n";
    }
    return (final -> elemento);
}
```

//Devuelve la cantidad de elementos de la Bicola

```
int Bicola::numElemsBicola()
{
    int n = 0;
    NodoCola* a = frente;
    if (!bicolaVacia())
    {
        n = 1;
        while (a != final)
        {
            n++;
            a = a -> siguiente;
        }
    }
    return n;
}
```

Funciones equivalentes a la Cola

```
void Bicola::ponerFinal(int elemento)
{
    insertar(elemento);
}
```

```
int Bicola::frenteBicola()
{
    return frenteCola();
}
```

```
int Bicola::quitarFrente()
{
    return quitar();
}
```

```
bool Bicola::bicolaVacia()
{
    return colaVacia();
}
```

Probando el código

```
//Antes debe programar la funcion
//completarCola y mostrarCola()
//igual como en programa para la Cola
int main()
{
    Bicola cola;
    completarCola(col);
    cola.ponerFrente(5);
    cola.ponerFinal(500);
    mostrarCola(col);
    return 0;
}
```

```
C:\Users\elpro\Downloads\Clases UNI 2020-2\UAC 2020-2\Sin Nombre2.exe
Cuantos elementos ingresara a la cola: 5
Digite el elemento y luego pulse Enter
Cola vacia: Direccion frente: 0 . Elemento frente: 4749824
Cola vacia: Direccion final: 0 . Elemento final: 4749824
indice 0: 10
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1550 . Elemento final: 10
indice 1: 20
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1570 . Elemento final: 20
indice 2: 30
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1590 . Elemento final: 30
indice 3: 40
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1a50 . Elemento final: 40
indice 4: 50
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1a70 . Elemento final: 50

Elementos de la Pila:
Direccion frente: 0xbb1a90 . Elemento frente: 5
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1a90 . 5
Direccion frente: 0xbb1550 . Elemento frente: 10
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1550 . 10
Direccion frente: 0xbb1570 . Elemento frente: 20
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1570 . 20
Direccion frente: 0xbb1590 . Elemento frente: 30
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1590 . 30
Direccion frente: 0xbb1a50 . Elemento frente: 40
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1a50 . 40
Direccion frente: 0xbb1a70 . Elemento frente: 50
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1a70 . 50
Direccion frente: 0xbb1ab0 . Elemento frente: 500
Direccion final: 0xbb1ab0 . Elemento final: 500
Se elimino de la memoria el nodo: 0xbb1ab0 . 500
Cola vacia: Direccion frente: 0 . Elemento frente: 4749824
Cola vacia: Direccion final: 0 . Elemento final: 4749824

-----
Process exited after 35.49 seconds with return value 0
Presione una tecla para continuar . . . █
```

Ejercicio 1

- Implemente una estructura de datos Cola utilizando un array con todas las operaciones que faciliten su manejo.

Ejercicio 2

- Implemente una estructura de datos Bicola utilizando un array con todas las operaciones que faciliten su manejo.

FIN