

## Tema 2 'PR02' {

[Estructuras de datos Lineales]

< pilas, colas y listas >

}

# Tabla de 'Contenidos' {

## 01 Pilas

< Metodos, Aplicaciones >

## 02 Colas

< Metodos, Aplicaciones >

## 03 Listas

< Iteradores, Metodos,  
Aplicaciones>

}

01 {

[Pilas <stacks>]

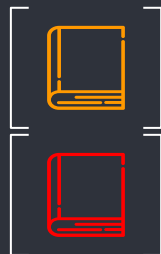
< Ejemplo, Métodos y  
Aplicaciones >

}

# Que son los 'stacks' {

< Simulan una torre o **PILA** de objetos >

< Siguen la regla del **LIFO** (Last In, First Out) >



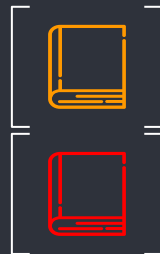
}

# Que son los 'stacks' {

< Simulan una torre o **PILA** de objetos >

< Siguen la regla del **LIFO** (Last In, First Out) >

**ENTRAR**



# Que son los 'stacks' {

< Simulan una torre o **PILA** de objetos >

< Siguen la regla del **LIFO** (Last In, First Out) >



<< TOP (Last in)

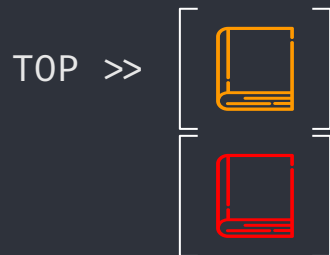
}

# Que son los 'stacks' {

< Simulan una torre o **PILA** de objetos >

< Siguen la regla del **LIFO** (Last In, First Out) >

**SALIR**



First Out

}

```
1 Concepto < /1 > LIFO {
```



< El ultimo que entra (Last In), será el primero en salir >

```
7  
8  
9  
10  
11  
12  
13  
14 }
```

```
1 Concepto < /2 > TOP {
```



< Para referirnos al elemento “cima” de la pila, usaremos la palabra TOP>

```
7  
8  
9  
10  
11  
12  
13  
14 }
```



# Como usamos los 'stacks' {

< Usamos la clase stack de la libreria estandar <stack> >

```
#include <stack>
```

```
int main() {
```

```
    stack<type> name;
```

```
}
```

```
}
```

# Metodos clase '`<stack>`' {

`.push()`

`stack.push("Renfe")`

< Inserta en la cima un  
elemento, y lo convierte  
en top >

`.top()`

`stack.top()`

< Devuelve el elemento  
top de la pila (no lo  
elimina) >

`.pop()`

`stack.pop()`

< Elimina el elemento  
top >

`.empty()`

`stack.empty()`

< Indica si la pila está  
o no vacia >

}

## Ejemplo visual 'stacks'

```
#include <stack>

int main() {

    stack<string> trenes;

}
```

}



## Ejemplo visual 'stacks'

```
#include <stack>

int main() {

    stack<string> trenes;

    trenes.push("Renfe");

}
```

**TOP**

Renfe

# Ejemplo visual 'stacks' {

```
#include <stack>

int main() {

    stack<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");

}
```

**TOP**

**TMB**

**Renfe**

## Ejemplo visual 'stacks' {

```
#include <stack>

int main() {

    stack<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");
    trenes.push("Rodalies");

}
```

**TOP**

Rodalies

TMB

Renfe

# Ejemplo visual 'stacks'

```
#include <stack>

int main() {

    stack<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");
    trenes.push("Rodalies");
    trenes.pop();

}
```

**TOP**

**TMB**

**Renfe**

# Ejemplo visual 'stacks' {

```
#include <stack>
```

```
int main() {
```

```
    stack<string> trenes;
```

```
    trenes.push("Renfe");
```

```
    trenes.push("TMB");
```

```
    trenes.push("Rodalies");
```

```
    trenes.top() = "Sagalés";
```

```
}
```

```
}
```

**TOP**

Sagalés

TMB

Renfe



# Ejercicios JUTGE {

< /1 > \* [X96935](#) Palíndroms amb piles

< /2 > \* [X36902](#) Avaluacio d'una expressio amb  
parentesis

< /3 > \* [X80203](#) Indexar seqüències ben  
parentitzades

< /4 > \* [X68213](#) Biblioteca

}

02 {

[Colas <queues>]

< Ejemplo, Métodos y  
Aplicaciones >

}

# Que son las 'queues' {

< Simulan una **COLA** de objetos >

< Siguen la regla del **FIFO** (First In, First Out) >



**FRONT**



< **Trabajadora de Renfe** >  
< Sueldo: Lata de CocaCola  
y un Kit Kat >

}

# Que son las 'queues' {

< Simulan una **COLA** de objetos >

< Siguen la regla del **FIFO** (First In, First Out) >



**ENTRAR**



**FRONT**



< **Trabajadora de Renfe** >  
< Sueldo: Lata de CocaCola  
y un Kit Kat >

}

# Que son las 'queues' {

< Simulan una **COLA** de objetos >

< Siguen la regla del **FIFO** (First In, First Out) >



**FRONT**



< **Trabajadora de Renfe** >  
< Sueldo: Lata de CocaCola  
y un Kit Kat >

}

## Concepto < /1 > FIFO {



< El primero que entra (First In), será el primero en salir (First Out) >

}

## Concepto < /2 > FRONT {



< Para referirnos al elemento “del frente” de la cola, usamos la palabra FRONT >

}

# Como usamos los 'queue' {

< Usamos la clase queue de la librería estándar <queue> >

```
#include <queue>
```

```
int main() {
```

```
    queue<type> name;
```

```
}
```

```
}
```

# Metodos clase '`<queue>`' {

`.push()`

`queue.push("Renfe")`

< Inserta al final un  
elemento >

`.front()`

`queue.front()`

< Devuelve el elemento  
front de la cola (no lo  
elimina) >

`.pop()`

`queue.pop()`

< Elimina el elemento  
front >

`.empty()`

`queue.empty()`

< Indica si la cola está  
o no vacia >

}



## Ejemplo visual 'queue' {

```
#include <queue>

int main() {

    queue<string> trenes;

}
```

}

**SALIDA**

**ENTRADA**

## Ejemplo visual queue{

```
#include <queue>

int main() {

    queue<string> trenes;

    trenes.push("Renfe");

}
```

}

**SALIDA**

Renfe

**ENTRADA**

## Ejemplo visual 'queue' {

```
#include <queue>

int main() {

    queue<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");

}
```

}

**SALIDA**

Renfe

TMB

**ENTRADA**

## Ejemplo visual 'queue' {

```
#include <queue>

int main() {

    queue<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");
    trenes.push("Rodalies");

}
```

**SALIDA**

Renfe

TMB

Rodalies

**ENTRADA**

## Ejemplo visual 'queue' {

```
#include <queue>

int main() {

    queue<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");
    trenes.push("Rodalies");
    trenes.pop();

}
```

**SALIDA**

TMB

Rodalies

**ENTRADA**

## Ejemplo visual 'queue' {

```
#include <queue>

int main() {

    queue<string> trenes;

    trenes.push("Renfe");
    trenes.push("TMB");
    trenes.push("Rodalies");
    trenes.front() = "Sagalés";

}
```

**SALIDA**

Sagalés

Rodalies

**ENTRADA**

# Ejercicios JUTGE {

< /1 > \* X13425 Distribucio justa de cues

< /2 > \* P90861 Queues of a supermarket (1)

< /3 > \* X38371 Estadístiques d'una seqüència  
d'enters amb esborrat del més antic

}

03 {

[Listas <lists>]

< Ejemplo, Iteradores,  
Métodos y Aplicaciones >

}



# Qué son las 'lists' {

< Simulan un vector o **LISTA** de elementos llamados NODOS >

}

# Qué son las 'lists' {

< Simulan un vector o **LISTA** de elementos llamados NODOS >



< NODO >

}

# Qué son las 'lists' {

< Simulan un vector o **LISTA** de elementos >



\*anterior



< NODO >



\*posterior

}

# Qué son las 'lists' {

< Simulan un vector o **LISTA** de elementos >



\*anterior



< NODO >



\*posterior


}


# Qué son las 'lists' {


< Simulan un vector o **LISTA** de elementos >



< NODO >

NODO item = 

NODO \*anterior = 

NODO \*posterior = 

}

# Qué son las 'lists' {

< Simulan un vector o **LISTA** de elementos >

DIFERENCIAS ENTRE <vector> Y <lists>:

<vector>:

- Acceso de elemento randomizado
- Recorrido rápido (elementos contiguos en memoria)
- Inserción y eliminación costosa  $O(n)$

<lists>:

- Acceso de elemento no randomizado (bidireccional)
- Recorrido lento (nodos repartidos dinámicamente)
- Inserción y eliminación rápida  $O(1)$

}

# Qué son los 'iteradores' {

< Son PUNTEROS inteligentes que permiten recorrer una estructura de manera abstracta y controlada >

DIFERENCIAS ENTRE <iterador> Y <puntero>:

<puntero>:

- Requiere de una posición de memoria explícita
- Necesita saber que recorrerá posteriormente

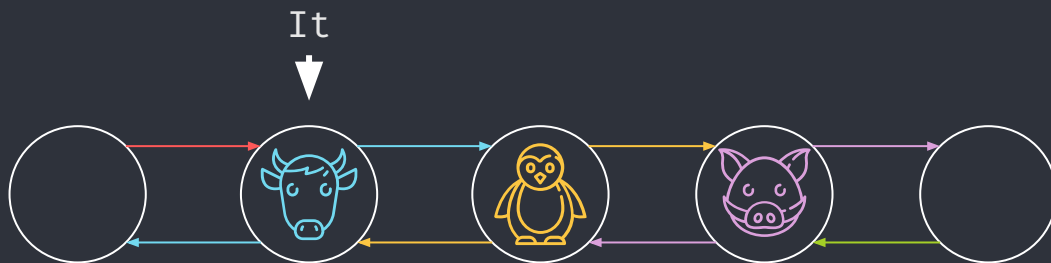
<iterador>:

- No requiere de una posición de memoria explícita
- La estructura se encarga de controlar el recorrido de un iterador

}

# Cómo funcionan los 'iteradores' {

< Permiten el movimiento entre nodos enlazados sin  
exponer la memoria >

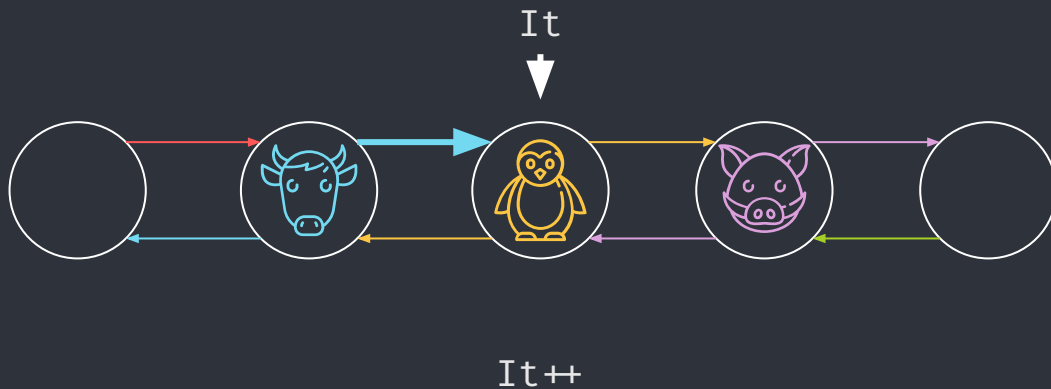


}



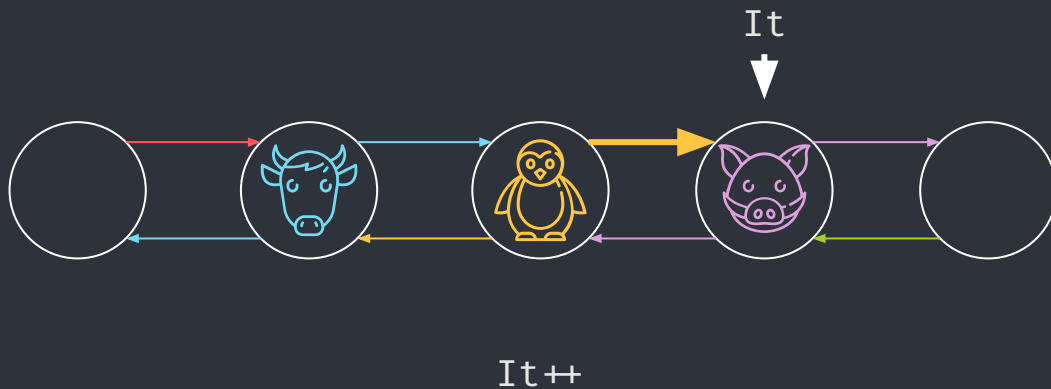
# Cómo funcionan los 'iteradores' {

< Permiten el movimiento entre nodos enlazados sin exponer la memoria >



# Cómo funcionan los 'iteradores' {

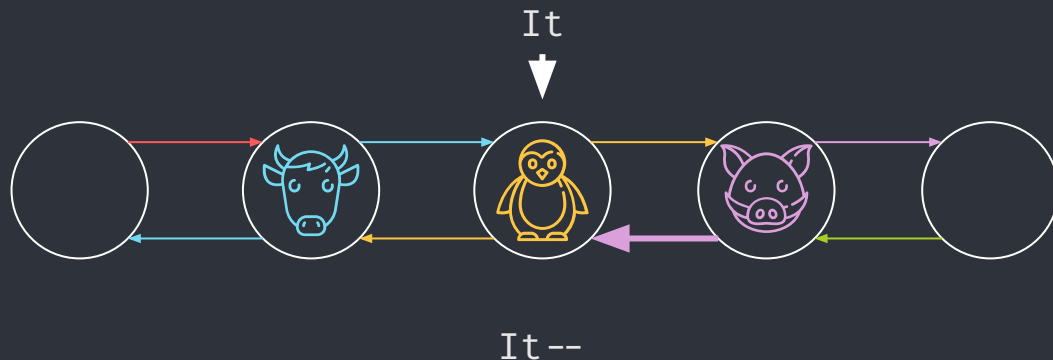
< Permiten el movimiento entre nodos enlazados sin exponer la memoria >



}

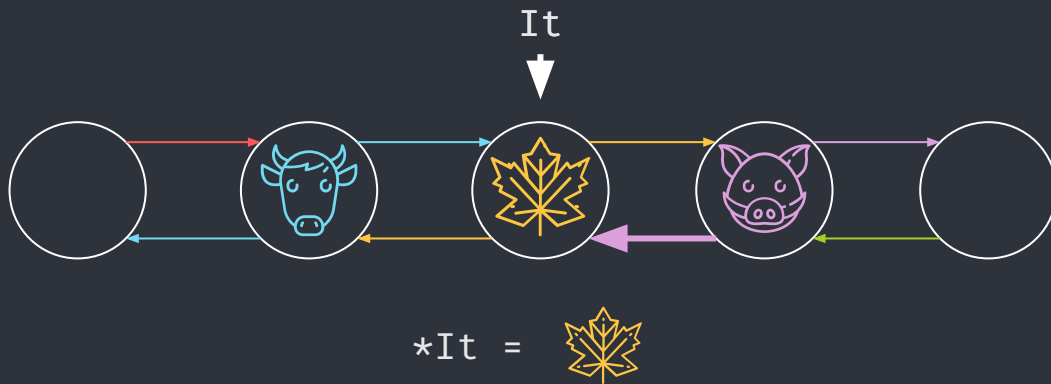
# Cómo funcionan los 'iteradores' {

< Permiten el movimiento entre nodos enlazados sin exponer la memoria >



# Cómo funcionan los 'iteradores' {

< Permiten el movimiento entre nodos enlazados sin exponer la memoria >



}

# Como usamos las 'lists' {

< Usamos la clase list de la librería estándar <list> >

```
#include <list>
```

```
int main() {
```

```
    list<type> name;
```

```
}
```

```
}
```

# Metodos clase '`<list>`' {

`.begin()`

`list.begin()`

< Iterador que marca el  
inicio de la lista >

`.end()`

`list.end()`

< Iterador del elemento  
ficticio final >

`.empty()`

`list.empty()`

< Elimina el elemento  
front >

`.insert()`

`list.insert(it, val)`

< Inserta en la  
posición del iterador  
un valor >

`.erase()`

`list.erase(it)`

< Elimina el valor  
referenciado de la  
lista >

`.splice()`

`list.splice(it, list2)`

< Inserta una sublista  
en la posición del  
iterador >

}

# Como declaramos un 'iterator' {

```
#include <list>

int main() {

    list<type> name;
    list<type>::iterator it;
    list<type>::const_iterator it2;

}
```

```
}
```

```
1 Iterador < /1 > CONSTANTE {
```



```
4 | < No podemos modificar el contenido de un  
5 | elemento referenciado >
```

```
6 }  
7
```

```
8 Iterador < /2 > NO CONSTANTE {
```



```
11 | < Podemos modificar el contenido del elemento  
12 | referenciado >
```

```
13 }  
14
```



# Cómo accedemos a un 'iterator' {

```
#include <list>
```

```
int main() {
```

```
    list<type> name;
```

```
    list<type>::iterator it;
```

```
    it→item; # Clases o Structs
```

```
    *it; # Cualquier otra estructura de datos
```

```
}
```

```
}
```

# Ejercicio Previo {

```
#include <vector>
```

```
Struct Player {
```

```
    String name;
```

```
    Int score;
```

```
}
```

```
vector<string> winners(vector<Player>::iterator ini, vector<Player>::iterator end);
```

< Dado un vector de Players, devuelve todos los nombres de los Players con mayor puntuación (puede haber empate) >

```
}
```

```
1  #include <vector>
2
3  Struct Player {
4      String name;
5      Int score;
6  }
7
8  vector<string> winners(vector<Player>::iterator ini, vector<Player>::iterator end) {
9      vector<string> win_p;
10     if (ini == end) return win_p;
11
12 }
13
14
```

```
1  #include <vector>
2
3  Struct Player {
4      String name;
5      Int score;
6  }
7
8  vector<string> winners(vector<Player>::iterator ini, vector<Player>::iterator end) {
9      vector<string> win_p;
10     if (begin == end) return win_p;
11
12     int max = ini->score
13
14 }
```

```
1  #include <vector>
2
3  Struct Player {
4      String name;
5      Int score;
6  }
7
8  vector<string> winners(vector<Player>::iterator ini, vector<Player>::iterator end) {
9      vector<string> win_p;
10     if (begin == end) return win_p;
11
12     int max = ini->score
13     while (ini != end) {
14         if (ini->score >= max) {
15             if (ini->score > max) win_p.clear();
16             max = ini->score;
17             win.push_back(ini->name)
18         }
19         ini++;
20     }
21     return win_p;
22 }
```