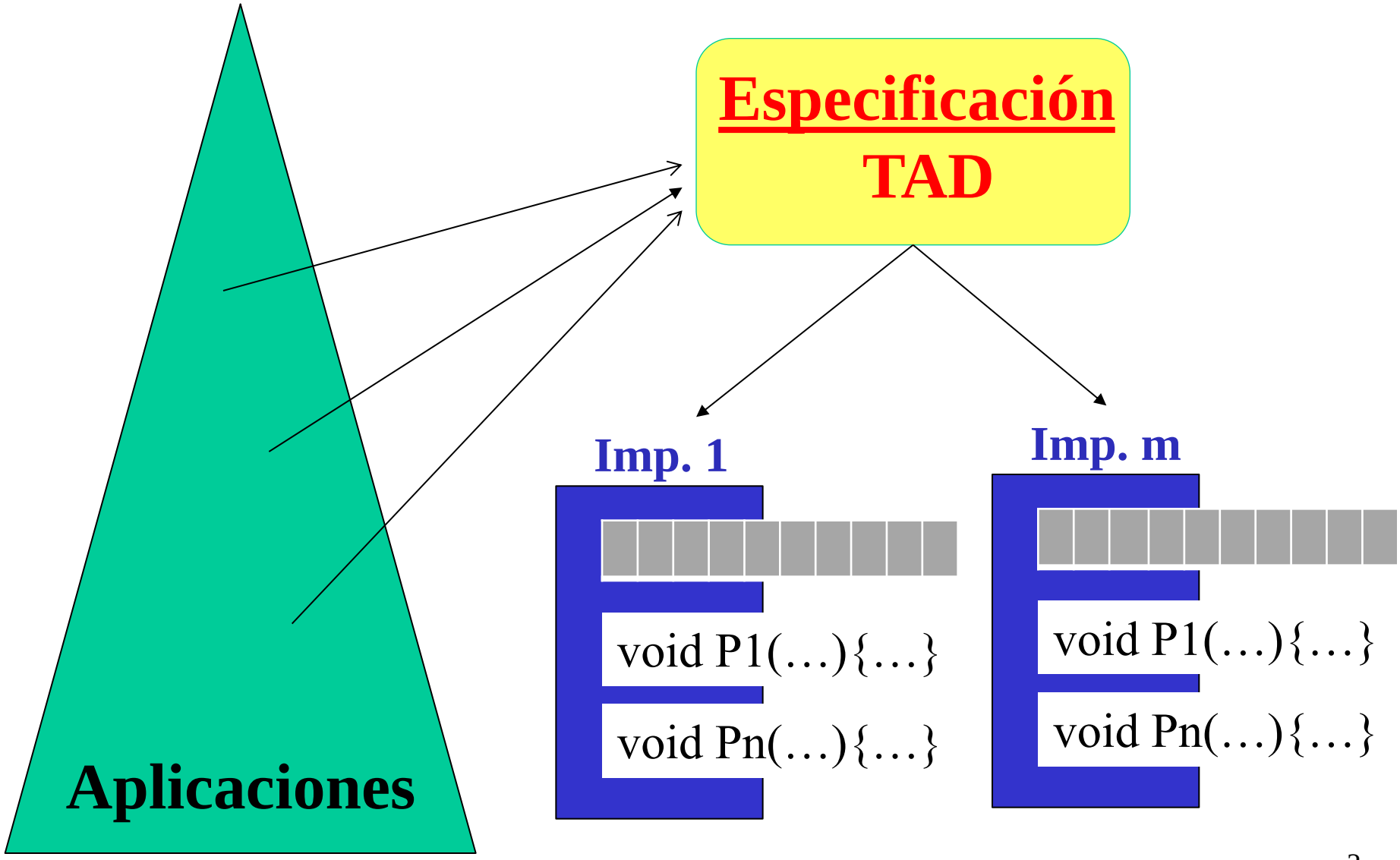
Programación 2

La Previa:
Introducción a TADs
Lista – Pila - Cola

Sobre TADs


Especificación
TAD
Aplicaciones
Imp. 1
void P1(…){…}
void Pn(…){…}
Imp. m
void P1(…){…}
void Pn(…){…}

TAD LISTA

Ejemplo de una especificación de Lista

Considere el TAD **Lista Indizada no acotada** de elementos de un tipo genérico, con las siguientes operaciones:

Constructoras:

Lista_Vacia: construye la lista vacía.

Insertar: dados una lista, un entero n y un elemento e, inserta e en la lista en la posición n. Si la lista tiene longitud m, con m menor a n-1, lo inserta en la posición m+1. Si la lista tiene longitud m, con m mayor o igual a n-1, inserta e en la posición n y desplaza en una posición los elementos que estuvieran en las posiciones siguientes.

Predicados:

Esta_Vacia: retorna true si, y solamente si, la lista es Vacía.

Esta_Definido: dados una lista y un entero n, retorna true si, y solamente si, la lista está definida en la posición n.

Ejemplo de una especificación de Lista (cont.)

Selectoras:

Elemento: dados una lista y un entero n, retorna el elemento en la posición n. Si la lista tiene longitud menor a n, la operación está indefinida.

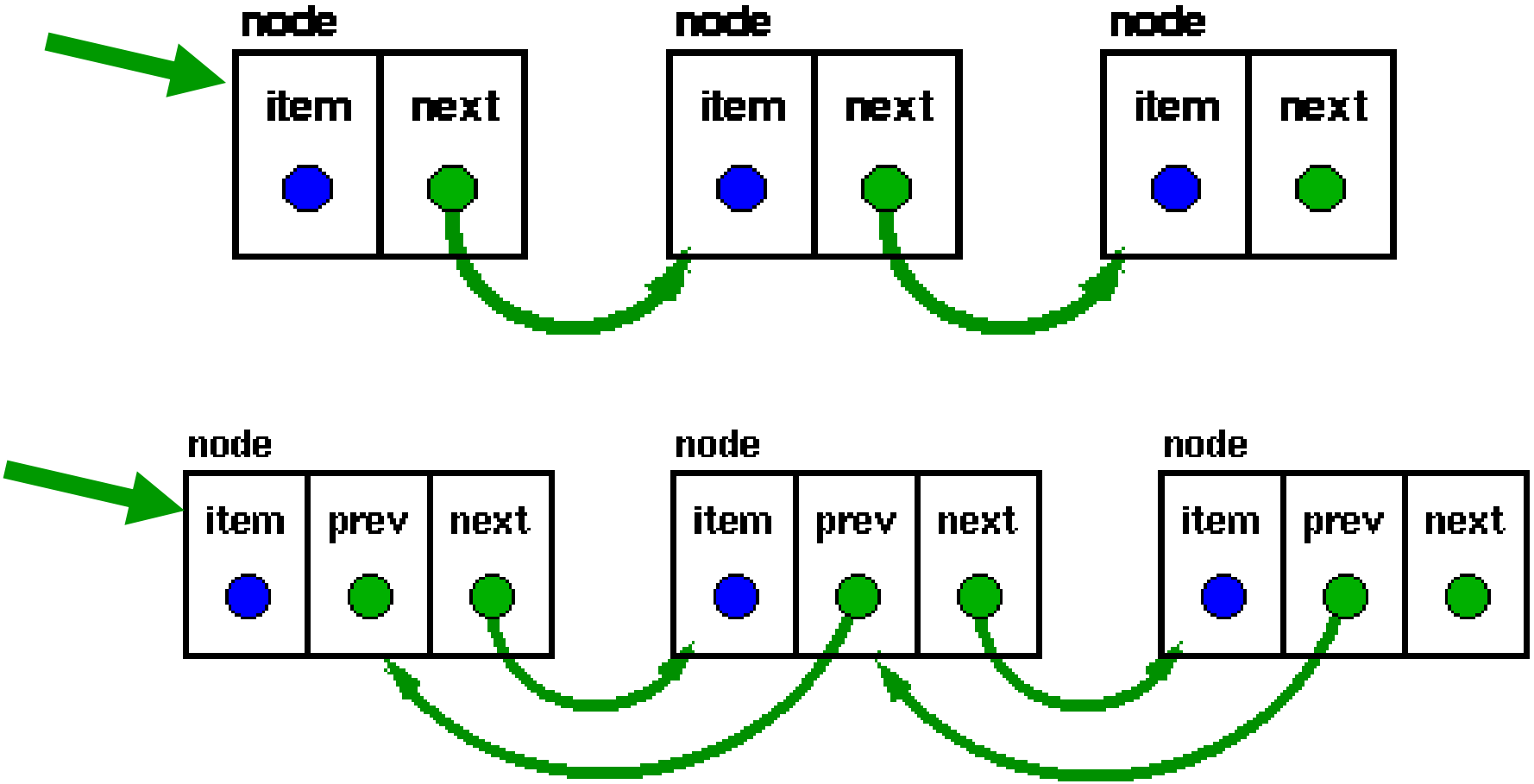
Borrar: dados una lista y un entero n, elimina de la lista el elemento en la posición n. Si la posición no está definida, la operación no hace nada. Si la posición está definida, elimina el elemento en dicha posición y desplaza en una posición los elementos que estuvieran en las posiciones siguientes (contrae la lista).

Destructora:

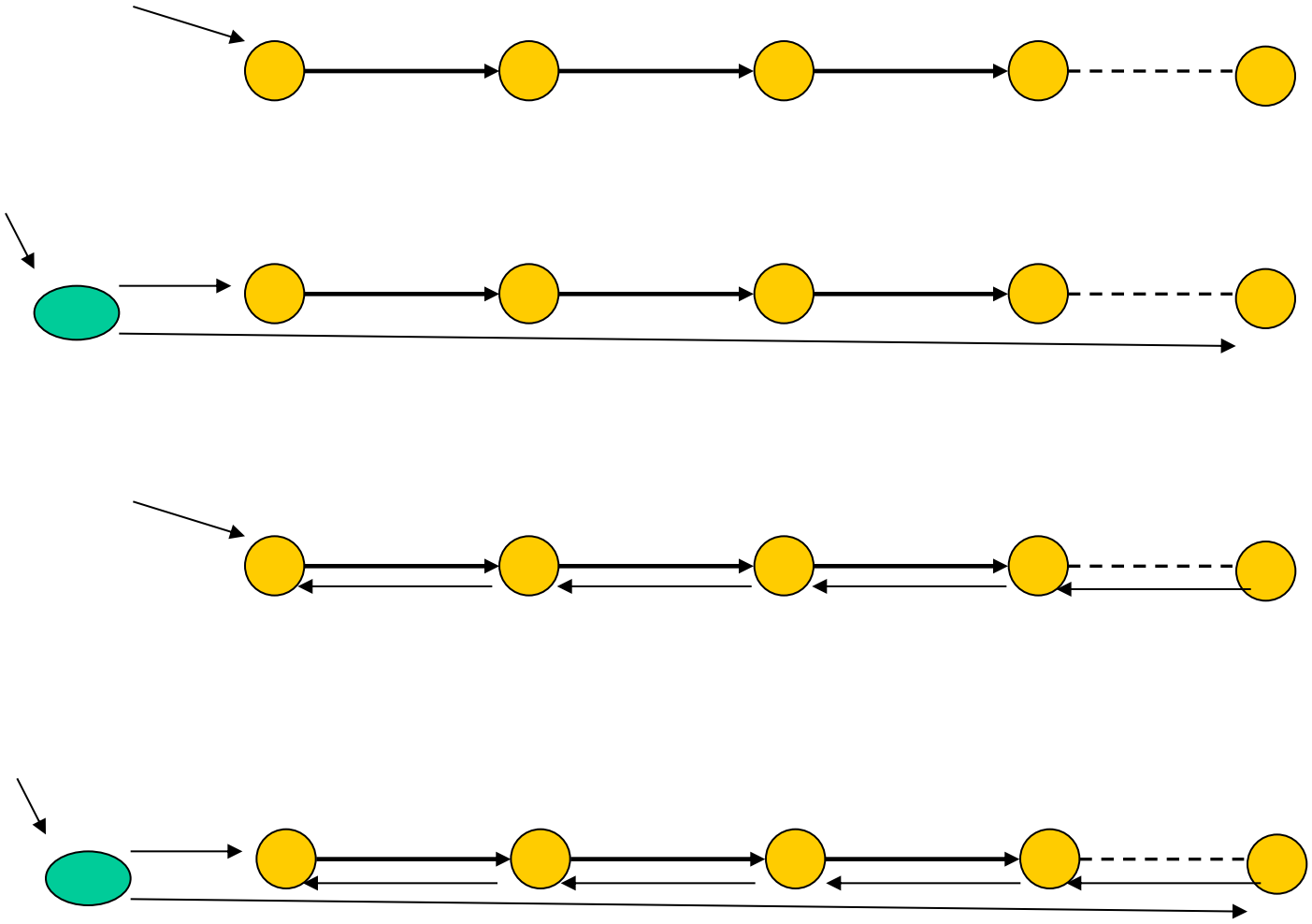
Destruir: destruye una lista, liberando la memoria que ésta ocupa.

Para estas operaciones, ¿cuáles serían las precondiciones y cuáles las postcondiciones?

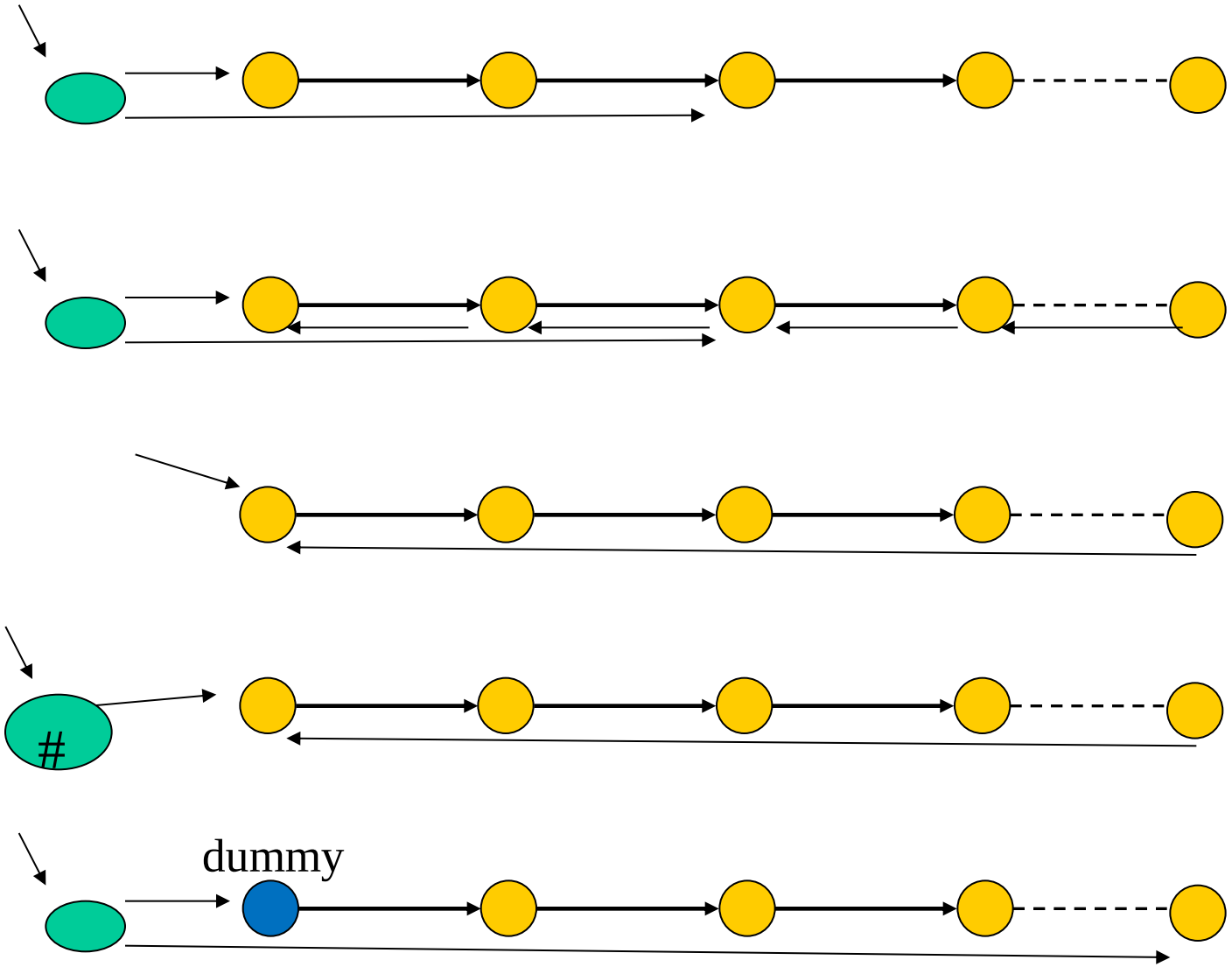
Implementaciones dinámicas de Listas



Algunas variantes de Listas



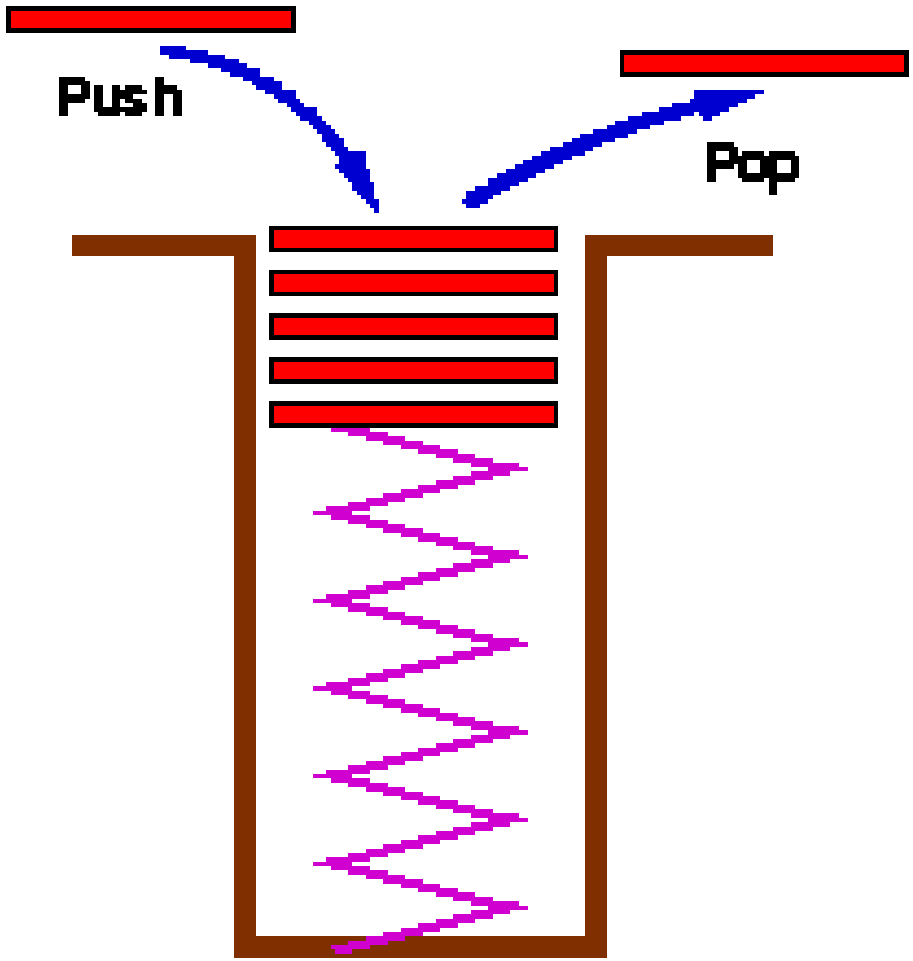
Algunas variantes de Listas



TAD PILA (STACK)

Definición

Otro nombre que se le da a este tipo de estructura es el de lista ***LIFO*** (last-in-first-out).



Operaciones

El TAD Stack (Pila) incluye básicamente las siguientes operaciones:

- la operación que construye un stack vacío, **`Empty`** (**`crearPila`**);
- **`Push`** (**`apilar`**), que inserta un elemento en el tope del stack;
- **`Top`** (**`cima`**), que retorna el elemento que se encuentra en el tope del stack;
- **`Pop`** (**`desapilar`**), remueve el elemento que se encuentra al tope (en la cima) del stack;
- **`IsEmpty`** (**`esVaciaPila`**), que testea si el stack es vacío o no;
- Si se trata de un stack acotado se incluye un predicado adicional, **`IsFull`** (**`esLlenaPila`**), que testea si el stack está lleno. Notar que en este caso la operación **`Push`** tendría precondición;
- Finalmente puede incluirse una operación para destruir un stack (**`destruirPila`**), liberando la memoria que éste ocupa.

Especificación del TAD Pila acotada en C/C++

```
#ifndef _PILA_H
#define _PILA_H

struct RepresentacionPila;
typedef RepresentacionPila * Pila;

// CONSTRUCTORAS
Pila crearPila (int cota);
// Devuelve la pila vacía que podrá contener hasta cota elementos.

void apilar (int i, Pila &p);
/* Si !esLlenaPila(p) inserta i en la cima de p,
   en otro caso no hace nada. */

// SELECTORAS
int cima (Pila p);
/* Devuelve la cima de p.
   Precondicion: ! esVaciaPila(p). */

void desapilar (Pila &p);
/* Remueve la cima de p.
   Precondicion: ! esVaciaPila(p). */
```

Especificación del TAD Pila acotada en C/C++

```
// PREDICADOS
bool esVaciaPila (Pila p);
/* Devuelve true si y sólo si p es vacia. */

bool esLlenaPila (Pila p);
/* Devuelve 'true' si y sólo si p tiene cota elementos, donde cota
   es el valor del parámetro pasado en crearPila. */

// DESTRUCTORA
void destruirPila (Pila &p);
/* Libera toda la memoria ocupada por p. */


#endif /* _PILA_H */
```

Implementación con un vector de una Pila acotada de enteros en C/C++

```
#include <stddef.h>
#include <assert.h>

#include "pila.h"

struct RepresentacionPila{
   int* array;
   int tope;
   int cota;
};

// CONSTRUCTORAS
Pila crearPila (int cota) {
   Pila p = new RepresentacionPila ();
   p->tope = 0;
   p->array = new int [cota];
   p->cota = cota;
   return p;
}
```

Implementación (cont.)

```
void apilar (int i, Pila &p) {
   if (! esLlenaPila (p)) {
        p->array [p->tope] = i;
        p->tope ++;
   }
}

// SELECTORAS
int cima (Pila p) {
   assert (p->tope > 0);
   return p->array [p->tope - 1];
}

void desapilar (Pila &p) {
   assert (p->tope > 0);
   p->tope--;
}
```

Implementación (cont.)

```
// PREDICADOS
bool esVaciaPila (Pila p) {
   return (p->tope == 0);
}

bool esLlenaPila (Pila p) {
   return (p->tope == p->cota);
}

// DESTRUCTORA
void destruirPila (Pila &p) {
   delete [] p->array;
   delete p;
}
```

Ejemplo de uso del TAD Pila

```
#include <stdio.h>
#include "pila.h"
void main{
   Pila p;
   p = crearPila(10);
   for (int i = 1, i<= 5, i++)
       apilar(i, p);
   while (!esVaciaPila(p)){
            printf("%d\n", cima(p))
            desapilar(p);
   }
   destruirPila(p);
}
```

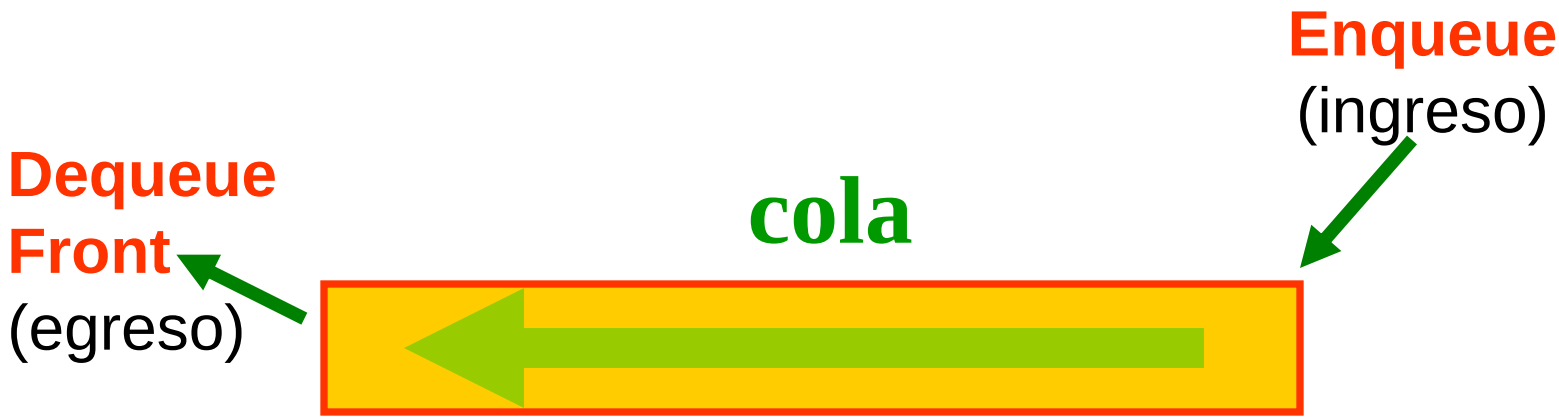
Notar que este código no depende de la implementación de la pila

TAD COLA (QUEUE)

Definición

Una ***Queue*** (cola) es otra clase especial de lista, donde los elementos son insertados en un extremo (el final de la cola) y borrados en el otro extremo (el frente de la cola).

Otro nombre usado para este tipo abstracto es el de lista FIFO (first-in-first-out).



Operaciones

El TAD Queue incluye las siguientes operaciones:

- la operación que construye una cola vacía, **Empty (crearCola)**.
- **Enqueue (encolar)**, que inserta un elemento al final de la cola.
- **Front (frente)**, que retorna el elemento que se encuentra en el comienzo de la cola.
- **Dequeue (desencolar)**, que borra el primer elemento de la cola.
- **IsEmpty (esVaciaCola)**, que testea si la cola es vacía.
- **IsFull (esLlenaCola)**, que testea si la cola está llena (si es una cola acotada).
- Una operación destructora, para eliminar una cola y liberar la memoria que ésta ocupa.

NOTAS:

- *Enqueue* podría no especificar donde se hacen las inserciones y en este caso las operaciones selectoras deberían referirse al primer elemento ingresado (el más antiguo).
- Si se trata de una cola acotada, *Enqueue* tendría precondición.

Algunas Aplicaciones

Prácticamente, toda fila real es (supuestamente) una cola: “se atiende al primero que llega”.

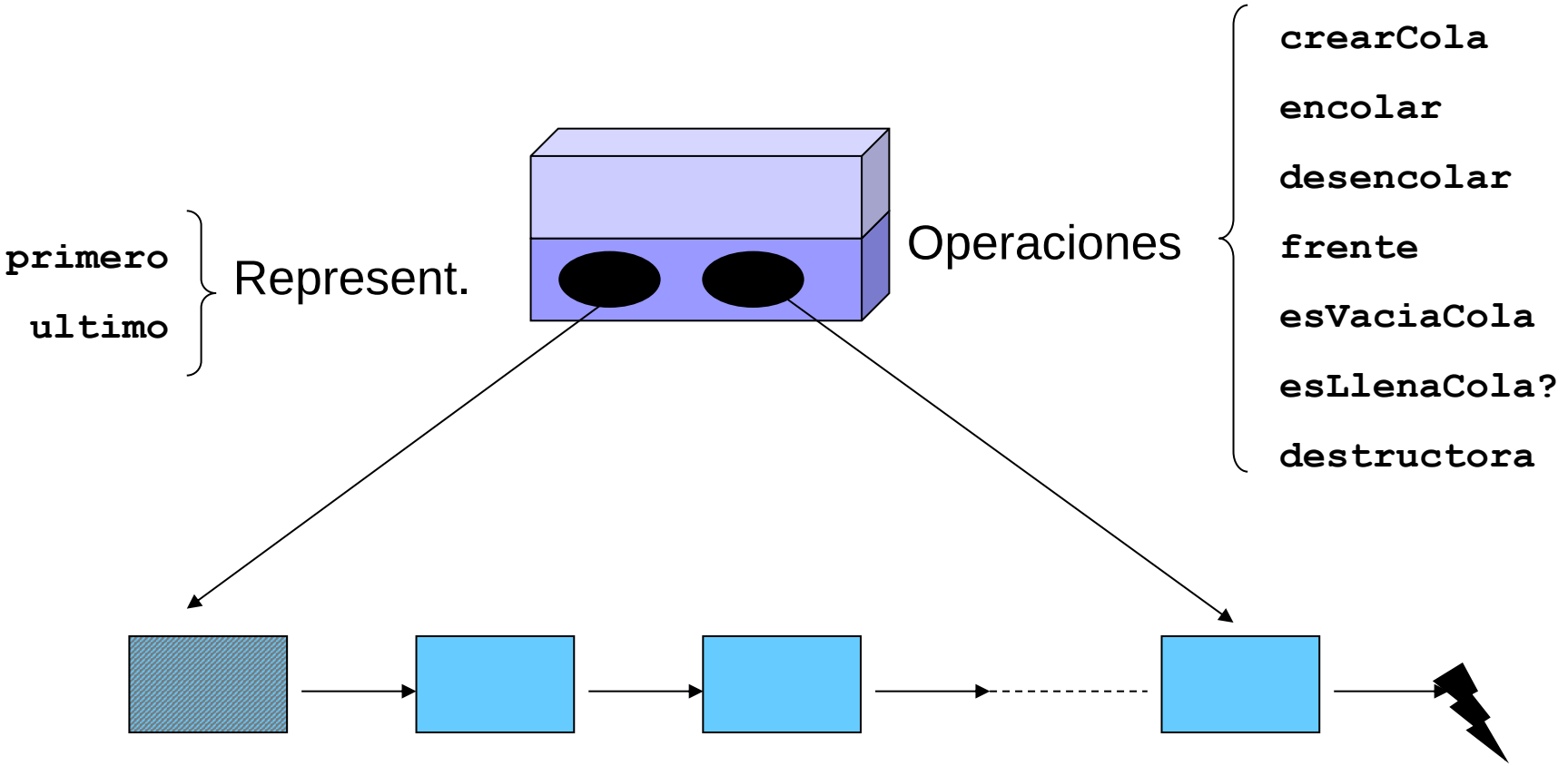
Las listas de espera son en general colas (por ejemplo, de llamadas telefónicas en una central) .

Los trabajos enviados a una impresora se manejan generalmente siguiendo una política de cola (suponiendo que los trabajos no son cancelados).

Dado un servidor de archivos en una red de computadoras, los usuarios pueden obtener acceso a los archivos sobre la base de que el primero en llegar es el primero en ser atendido, así que la estructura es una cola (no siempre se usa este esquema). ¿Cómo se procesan los procesos en espera de un procesador?

Existe toda una rama de las matemáticas, denominada *teoría de colas*, que se ocupa de hacer cálculos probabilistas de cuánto tiempo deben esperar los usuarios en una fila, cuán larga es la fila.....

Implementación del TAD Queue (cont.)



Especificación e Implementación de un TAD Cola no acotada de elementos de un tipo T en C/C++

Especificación en C/C++ de un TAD
Cola no acotada de elementos de tipo T

```
#ifndef _COLA_H
#define _COLA_H

struct RepresentacionCola;
typedef RepresentacionCola* Cola;

Cola crearCola ();
/* Devuelve en c la cola vacia. */

void encolar (T t, Cola &c);
/* Agrega el elemento t al final de c. */

bool esVaciaCola (Cola c);
/* Devuelve 'true' si c es vacia, 'false' en otro caso. */
```

Especificación en C/C++ de un TAD Cola no acotada de elementos de tipo T

```
T frente (Cola c);
/* Devuelve el primer elemento de c.
   Precondicion: ! esVaciaCola(c). */

void desencolar (Cola &c);
/* Remueve el primer elemento de c.
   Precondicion: ! esVaciaCola(c). */

void destruirCola (Cola &c);
/* Libera toda la memoria ocupada por c. */

#endif /* _COLA_H */
```

Implementación en C/C++ de un TAD Cola no acotada de elementos de tipo T

```
#include "Cola.h"
#include <stddef.h>
#include <assert.h>

struct Nodo
{
    T valor;
    Nodo * sig;
};

struct RepresentacionCola
{
    Nodo *primero, *ultimo;
};

Cola crearCola ()
{
    Cola c = new RepresentacionCola;
    c->primero = c->ultimo = NULL;
    return c;
}
```

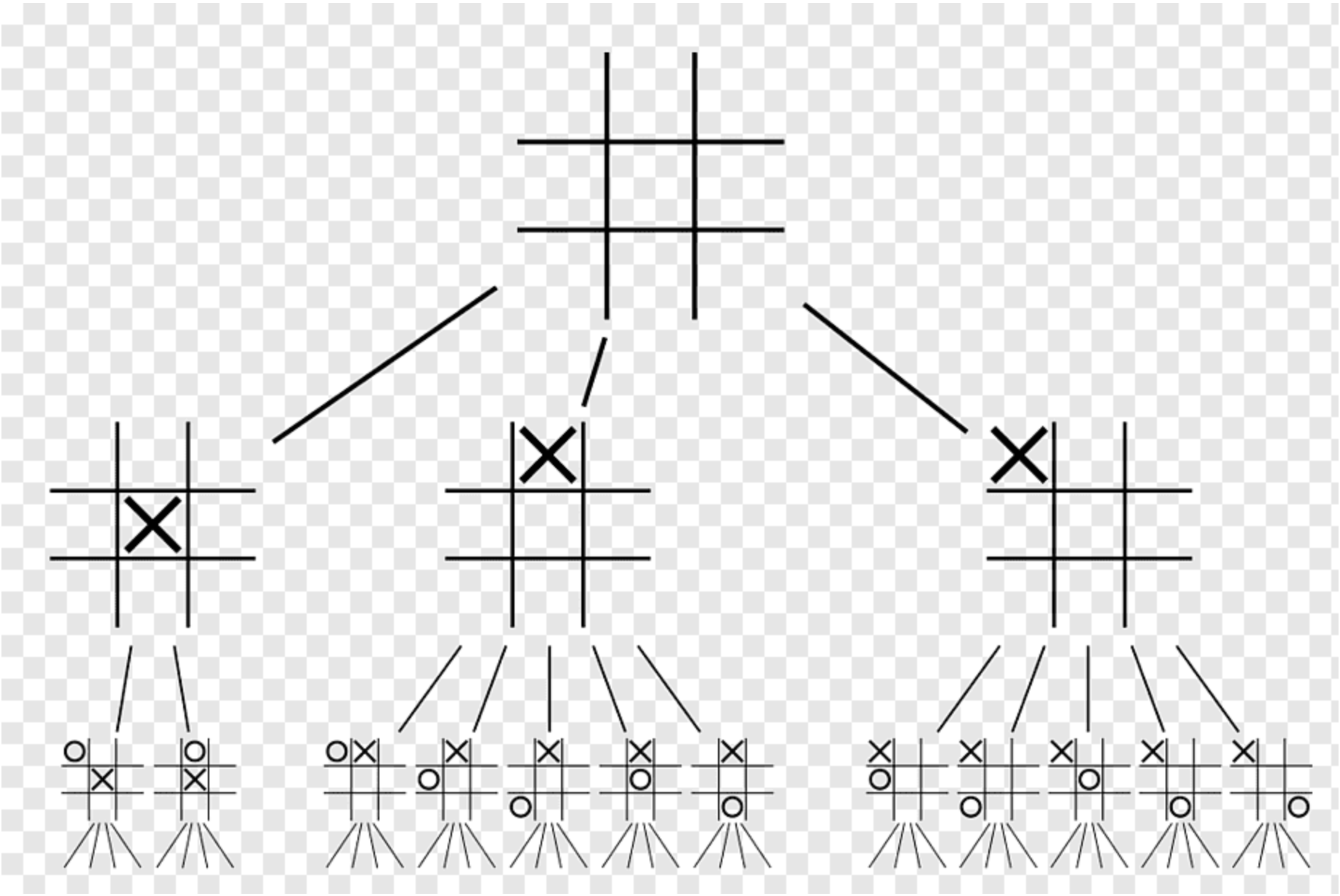
Implementación en C/C++ de un TAD Cola no acotada de elementos de tipo T

```
void encolar (T t, Cola &c)
{
    Nodo *nuevo = new Nodo;
    nuevo->valor = t;
    nuevo->sig = NULL;
    if (c->primero == NULL) c->primero = nuevo;
       else c->ultimo->sig = nuevo;
    c->ultimo = nuevo;
}

bool esVaciaCola (Cola c)
{
    return (c->primero == NULL);
}

T frente (Cola c)
{
    assert(c->primero != NULL);
    return c->primero->valor;
}
```

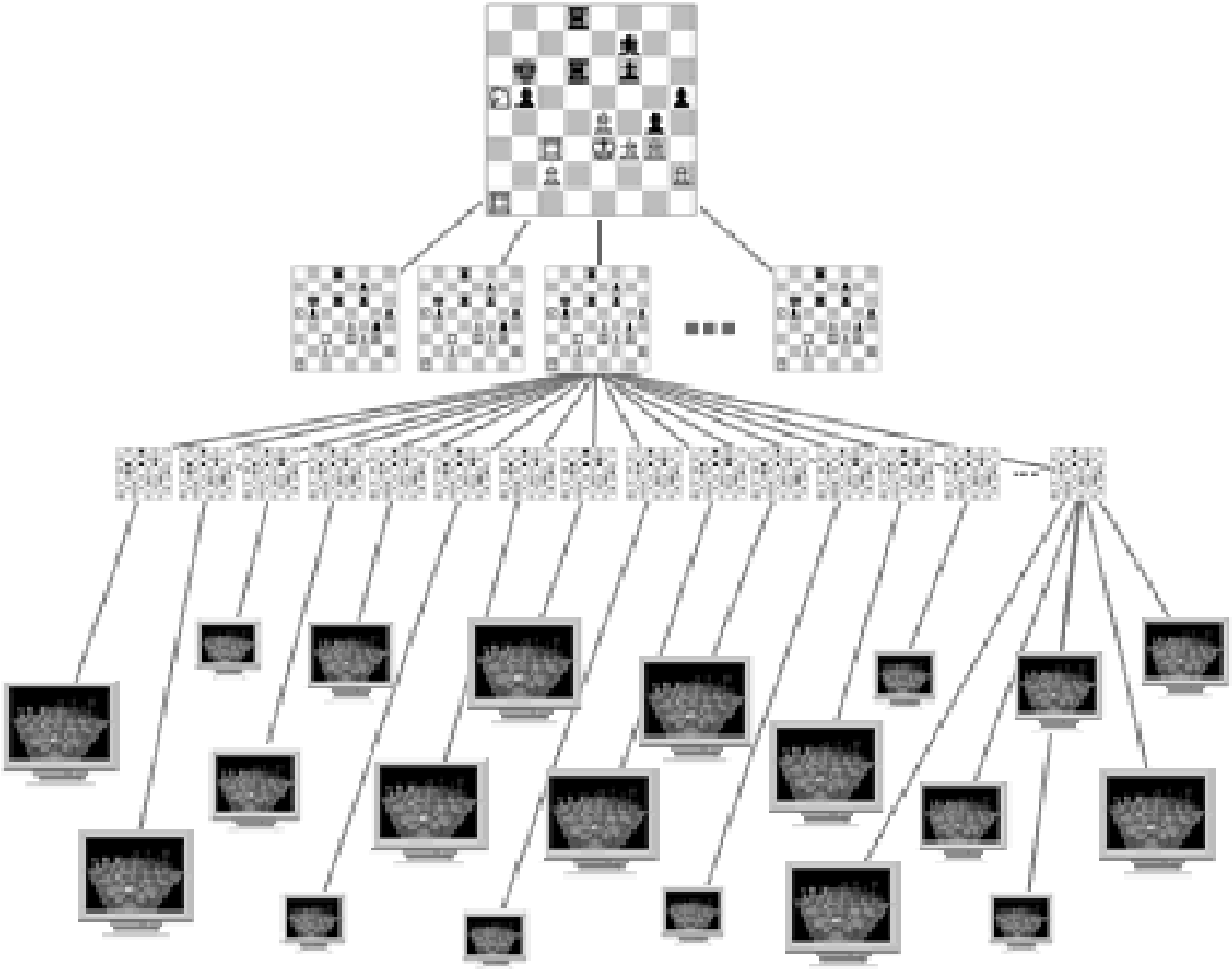
Implementación en C/C++ de un TAD Cola no acotada de elementos de tipo T

```
void desencolar (Cola &c)
{
    assert(c->primero != NULL);
    Nodo *temp = c->primero;
    c->primero = c->primero->sig;
    if (c->primero == NULL) c->ultimo = NULL;
    delete temp;
}

void destruirCola (Cola &c)
{
    Nodo * temp;
    while (c->primero != NULL)
    {
        temp = c->primero;
        c->primero = c->primero->sig;
        delete temp;
    }
    delete c;
}
```

Árboles de juegos (Ta-Te-Ti)



Árboles de juegos (Ajedrez)

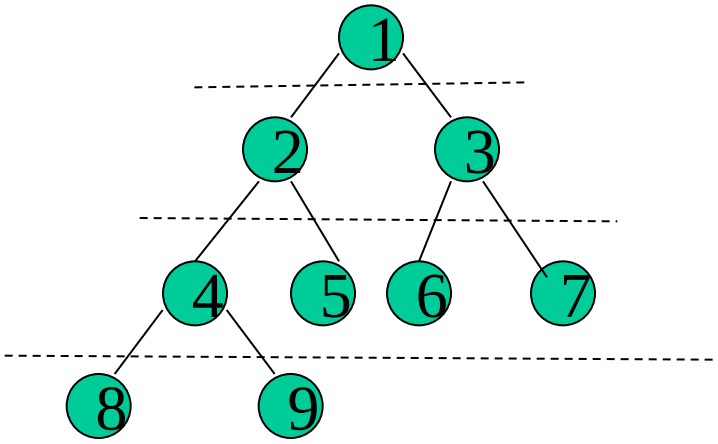


Aplicación del TAD Cola (Queue): Recorrido por niveles

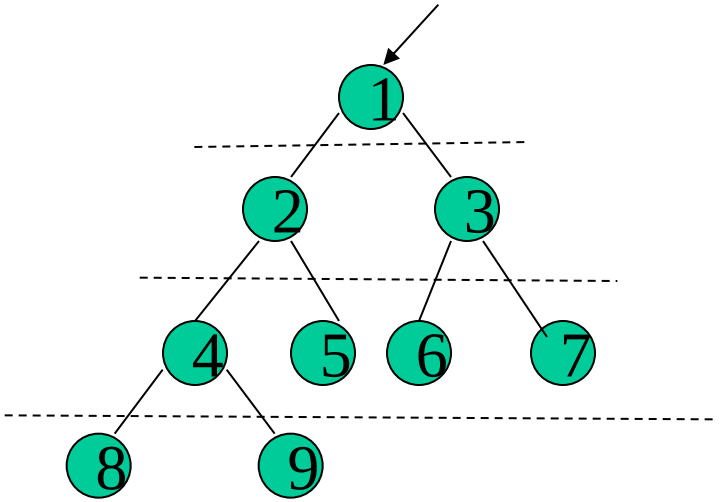
¿Qué es un recorrido por niveles de un árbol? ¿Qué aplicaciones tiene?

¿Cómo se puede recorrer un árbol binario por niveles?

¿Es possible resolver este problema en O(*n*), siendo *n* la cantidad de nodos del árbol?

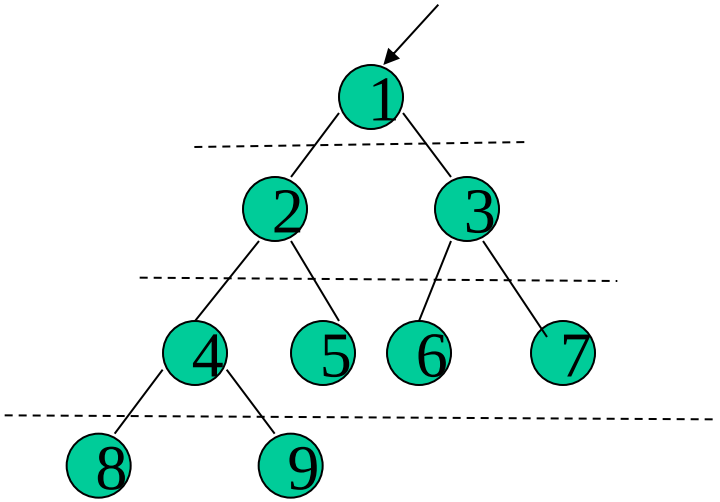


Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

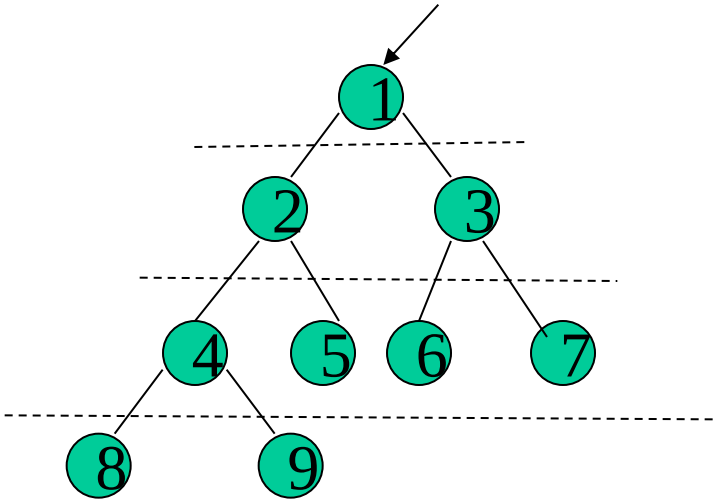
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

Si imprime: 1

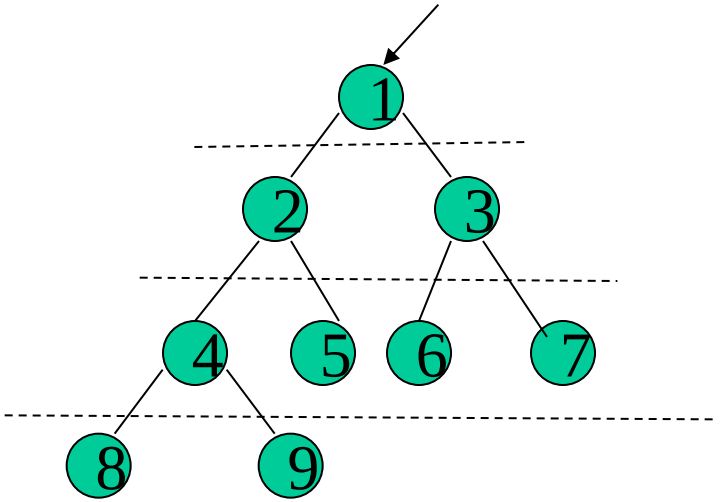
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

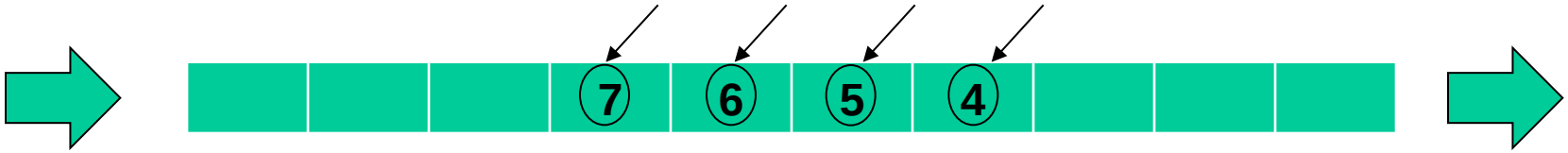


Cola de árboles (de punteros)

Si imprime: 1, 2

Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

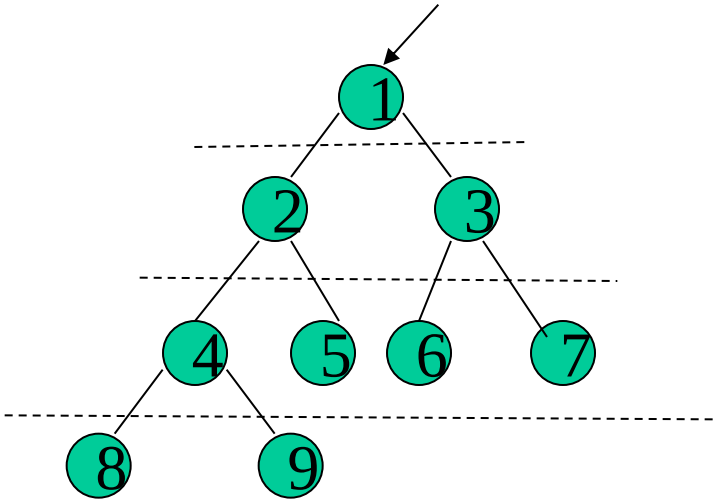


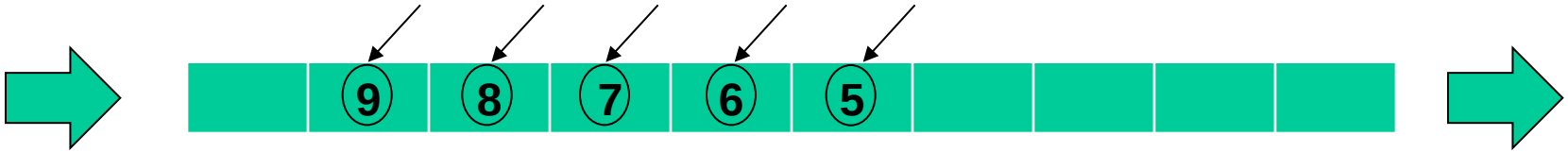


Cola de árboles (de punteros)

Si imprime: 1, 2, 3

Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

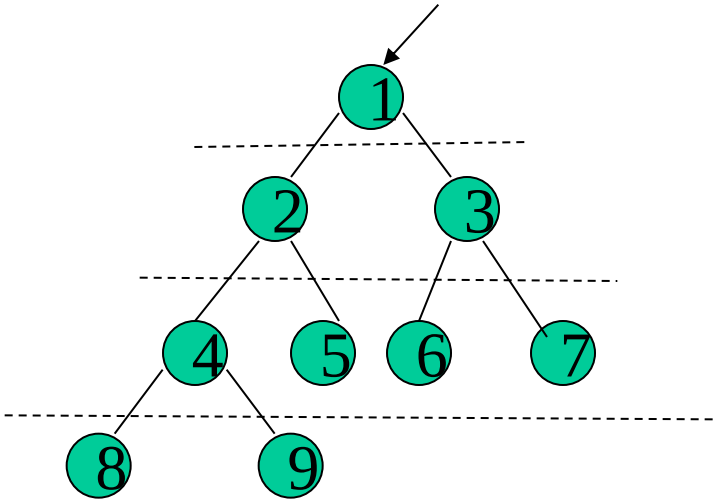


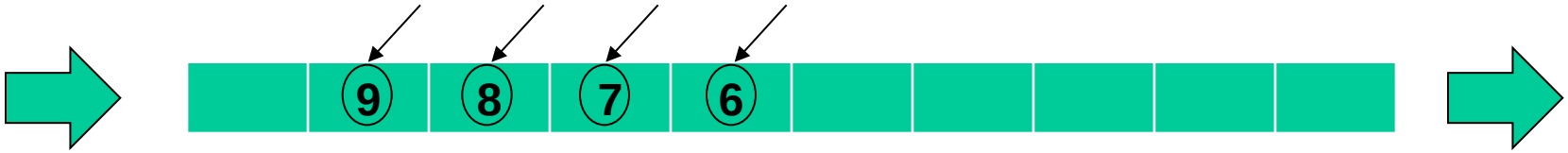


Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4

Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

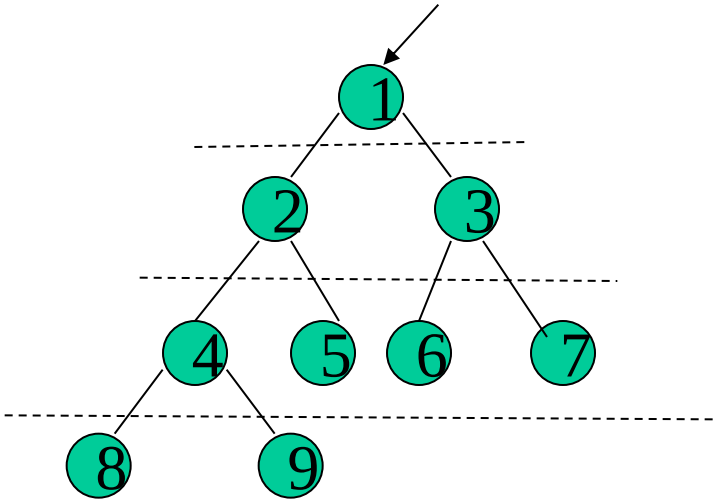




Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4, 5

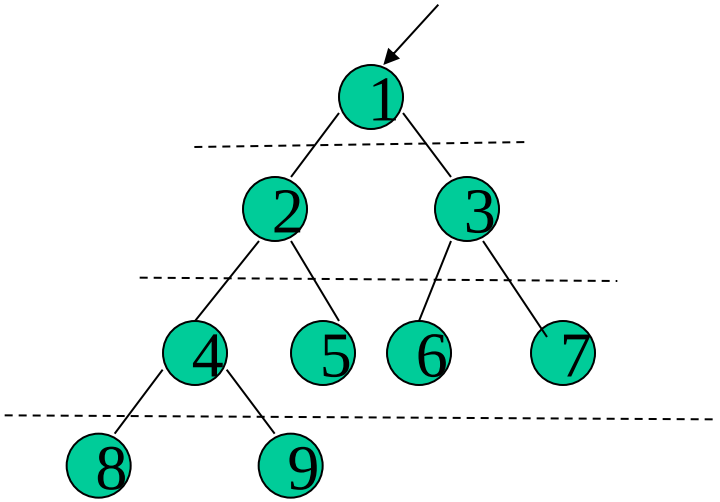
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4, 5, 6

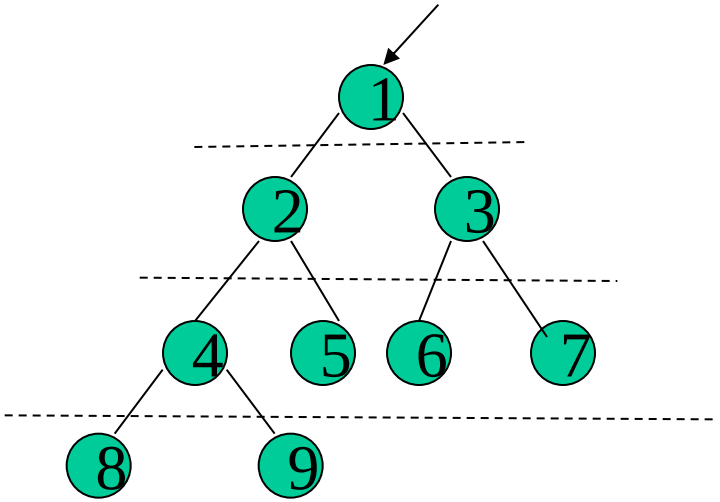
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4, 5, 6, 7

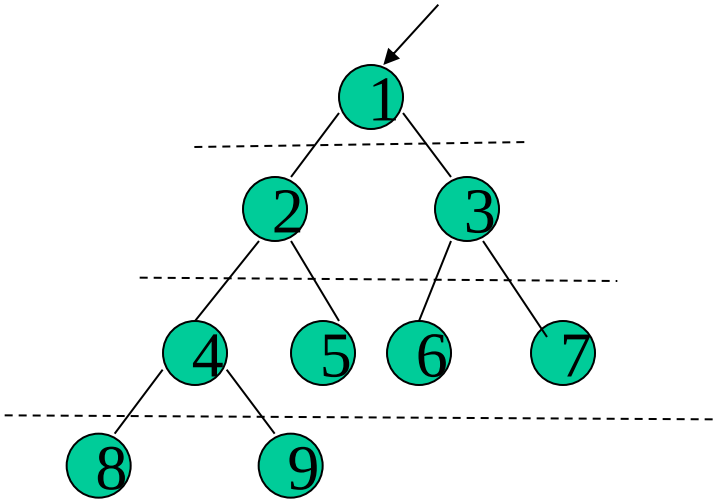
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4, 5, 6, 7, 8

Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario



Cola de árboles (de punteros)

Si imprime: 1, 2, 3, 4, 5, 6, 7, 8, 9

FIN

Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

Imprimir un árbol binario *a* por niveles:

- Si *a* no es vacío, encolar *a* en una cola *c* inicialmente vacía de árboles binarios.
- Mientras la cola *c* no sea vacía:
  - Obtener el primer (en orden *fifo*) árbol de *c;*
  - Imprimir el dato de su ráiz;
  - Para sus subárboles izquierdo y derecho, si no son vacíos entonces encolarlos en *c;*
  - Desencolar de *c;*

Escribir el código en C++ para un árbol de enteros.

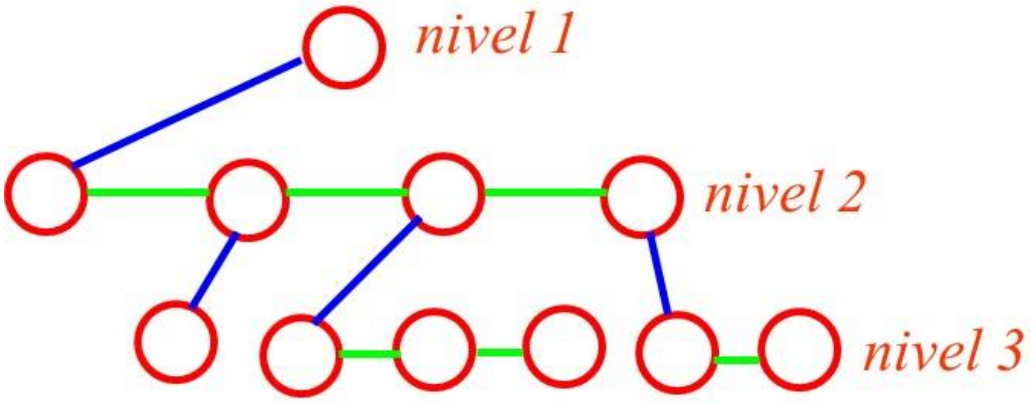
Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol binario

```
void imprimirABxNiveles(NodoAB* t){
\\ usamos una cola de elementos de tipo NodoAB* (dato,izq,der)
    ColaAB c = crearColaAB();
    If (t != NULL) encolarColaAB(t, c);
    while (!esVaciaColaAB(c)){
            t = frenteColaAB(c);
            ìmprimir(t->dato);
            If (t->izq != NULL) encolarColaAB(t->izq, c);
            If (t->der != NULL) encolarColaAB(t->der, c);
            desencolarColaAB(c);
    }
    destruirColaAB(c);
}
```

Aplicación del TAD Cola (Queue): Recorrido por niveles de árboles generales

¿Cómo se puede recorrer un árbol general (finitario) por niveles? ¿Es posible hacerlo en O(n), siendo n la cantidad de nodos del árbol?

Generalice la solución realizada para árboles binarios, considerando árboles generales representados como binarios con la semántica: primer hijo – siguiente hermano.



Aplicación del TAD Cola (Queue): Recorrido por niveles de un árbol general

```
void imprimirAGxNiveles(NodoAG* t){
    \\ usamos una cola de elementos de tipo NodoAG* (dato,pH,sH)
    ColaAG c = crearColaAG();
    while (t != NULL) { enconlarColaAG(t); t = t->sH; }
    // Contempla el caso en que t es un bosque y no solo un árbol.
    while (!esVaciaColaAG(c)){
        t = frenteColaAG(c);
        imprimir(t->dato);
        t = t->pH;
        while (t != NULL){
            encolarColaAG(t,c);
            t = t->sH;
        }
        desencolarColaAG(c);
    }
    destruirColaAG(c);
}
```

Se encolan los hijos del nodo cuyo dato se imprimió.