

**Universidad de Costa Rica**  
**Facultad de Ingeniería**  
**Escuela de Computación e Informática**  
**Estructuras de Datos y Análisis de Algoritmos**  
**CI-1221**  
**Grupo 04**

## **II Etapa I TP**

**Tarea Programada 1 Análisis de algoritmos y estructuras de  
datos.**

**Profesora:**

**Sandra Kikut**

**Elaborado por:**

**Carrión Claeys Archibald C01736**

**Fernando Arce Castillo C10572**

**Saltachín Solano Javier C17632**

**Día 19 de mes 10 del año 2022**

## Tabla de Contenidos Mínimos

### 1. Introducción

En el siguiente trabajo práctico describiremos e implementaremos distintos modelos lógicos como lo son árbol n-ario, cola y lista indexada. Con el propósito de implementar dichos modelos lógicos se utilizan distintas estructuras de datos aprendidas en clase, como lo son lista simplemente enlazada, arreglo circular, lista de hijos, estructura hijo más izquierdo hermano derecho, etc.

También se incluyen cláusulas necesarias para el funcionamiento de este programa, como lo son el manual de usuario, cláusulas requiere y modifica de cada método e instrucciones para su compilación.

### 2. Objetivos

#### **Objetivo general:**

Comprender el funcionamiento e implementación de los modelos lógicos cola, lista y árbol n-ario.

#### **Objetivos específicos:**

Implementar el modelo lógico cola, mediante la estructura de datos arreglo circular.

Implementar el modelo lógico lista indexada, mediante la estructura de datos lista simplemente enlazada.

Implementar el modelo lógico árbol n-ario, mediante las estructuras de datos arreglo con señalador al padre, lista de hijos e Hijo Más Izquierdo-Hermano Derecho.

### 3. Enunciado (Descripción del Problema)

Para esta tarea debemos implementar los modelos lógicos, árbol n-ario, lista indexada y cola.

### 4. Desarrollo

#### **Modelo Cola.**

Cola: El modelo lógico Cola consiste en un grupo de datos, tal que ellos se encuentran ordenados linealmente (sin ramificaciones), de manera análoga a una fila de espera. En él solo se pueden insertar datos por un extremo, y extraer datos por el otro extremo.

Esta inserción-extracción corresponde al orden FIFO (First in First Out), es decir, que el Elemento que entró primero respecto a los demás es el primero en salir del modelo.

#### Definición y especificación de operadores básicos del modelo Cola.

##### • Crear

<b>Nombre</b>	Crear
<b>Parámetros</b>	Una Cola

<b>Efecto</b>	Inicializa una estructura de datos para representar al modelo lógico Cola
<b>Requiere</b>	Una Cola que no esté inicializada
<b>Modifica</b>	Crea una estructura de datos

• Destruir

<b>Nombre</b>	Destruir
<b>Parámetros</b>	Una Cola inicializada aún no destruida
<b>Efecto</b>	Borra la estructura Cola y los datos que contiene
<b>Requiere</b>	Una Cola inicializada
<b>Modifica</b>	Elimina la estructura de datos, libera el espacio utilizado por la estructura

• Encolar

<b>Nombre</b>	Encolar
<b>Parámetros</b>	Una Cola, un elemento
<b>Efecto</b>	Agrega al inicio de la Cola el elemento dado por parámetro
<b>Requiere</b>	Una Cola inicializada, un Elemento válido
<b>Modifica</b>	La cantidad total de datos en la Cola, agrega un elemento a la Cola

• Desencolar?

<b>Nombre</b>	Desencolar
<b>Parámetros</b>	Una Cola
<b>Efecto</b>	Devuelve un elemento almacenado en la Cola (por definición, el primero que haya entrado)
<b>Requiere</b>	Una Cola inicializada
<b>Modifica</b>	El tamaño de la Cola disminuye en una unidad y se extrae un elemento.

- NumElem

<b>Nombre</b>	NumElem
<b>Parámetros</b>	Una Cola
<b>Efecto</b>	Devuelve un valor entero igual a la cantidad de elementos en la Cola
<b>Requiere</b>	Una Cola inicializada
<b>Modifica</b>	No modifica la estructura de datos.

### Modelo Lista Indexada

Lista Indexada: Es un modelo lógico que extiende a manera de especialización a la Lista Posicionada. Consiste en un contenedor de agrupación concreta, finita, ordenada y discreta de elementos (agnósticos al tipo de dato agrupado), donde se dice que cada elemento tiene una posición ordinal en la agrupación.

Las posiciones de un elemento en una Lista Indexada tienen un índice numérico discreto concreto conocido. Para esta Lista se puede hablar de la i-ésima posición de elementos, y es consenso que el índice de aquella posición esté contenido en un rango acotado por el tamaño de la Lista.

### Definición y especificación de operadores básicos del modelo Cola.

- Crear

<b>Nombre</b>	Crear
<b>Parámetros</b>	Una Lista Indexada
<b>Efecto</b>	Inicializa una estructura de datos subyacente para representar al modelo.
<b>Requiere</b>	Que la Lista Indexada alimentada como parámetro esté sin inicializar
<b>Modifica</b>	La Lista Indexada tiene su estructura de datos subyacente inicializada.

- Destruir

<b>Nombre</b>	Destruir
<b>Parámetros</b>	Una Lista Indexada inicializada

<b>Efecto</b>	Destruye a la estructura de datos subyacente de la Lista Indexada, liberando su espacio.
<b>Requiere</b>	Que la Lista Indexada alimentada como parámetro esté inicializada.
<b>Modifica</b>	La Lista Indexada ya no tiene a su estructura de datos subyacente inicializada, sino destruida

• Insertar

<b>Nombre</b>	Insertar
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Una Lista Indexada</li> <li>• Un Elemento</li> <li>• Una Posición (número entero como índice de inserción)</li> </ul>
<b>Efecto</b>	Inserta en la estructura de datos subyacente al elemento tal que aquel elemento toma aquella posición
<b>Requiere</b>	<p>Que:</p> <ul style="list-style-type: none"> <li>• La Lista Indexada alimentada como parámetro esté inicializada.</li> <li>• El Elemento alimentado como parámetro esté inicializado</li> <li>• La Posición de inserción alimentada se refiera a un índice válido para la inserción: entre el principio (índice 0) y la posición hipotética después de la última (índice igual al tamaño de la lista)</li> </ul>
<b>Modifica</b>	<ul style="list-style-type: none"> <li>• La Lista Indexada crece en tamaño, en 1 unidad su cantidad de elementos.</li> <li>• Si existe previamente un elemento en la posición de inserción, entonces aquel elemento preexistente y todos aquellos en posiciones siguientes (si existen) incrementan en 1 unidad su índice de posición.</li> <li>• Si no existe previamente un</li> </ul>

	<p>elemento en la posición de inserción, entonces el elemento adopta esa posición y no modifica la posición de ningún otro.</p>
--	---

• Borrar

<b>Nombre</b>	Borrar
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Una Lista Indexada</li> <li>• Una Posición (número entero como índice de borrado)</li> </ul>
<b>Efecto</b>	Elimina de la estructura de datos subyacente al elemento que posea aquella dirección.
<b>Requiere</b>	<p>Que:</p> <ul style="list-style-type: none"> <li>• La Lista Indexada alimentada como parámetro esté inicializada.</li> <li>• La Lista Indexada alimentada como parámetro no esté vacía.</li> <li>• La Posición de borrado alimentada se refiera a un índice válido para el borrado: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)</li> </ul>
<b>Modifica</b>	<ul style="list-style-type: none"> <li>• La Lista Indexada disminuye en tamaño, en 1 unidad su cantidad de elementos.</li> <li>• El elemento preexistente en la posición de borrado es destruido, y todos aquellos en posiciones siguientes (si existen) disminuyen en 1 unidad su índice de posición.</li> </ul>

• Recuperar?

<b>Nombre</b>	Recuperar
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Una Lista Indexada</li> <li>• Una Posición (número entero)</li> </ul>

	como índice de recuperado)
<b>Efecto</b>	Obtiene los datos de un Elemento que tenga la posición especificada como parámetro de la Lista Indexada.
<b>Requiere</b>	Que: <ul style="list-style-type: none"> <li>• La Lista Indexada alimentada como parámetro esté inicializada.</li> <li>• La Lista Indexada alimentada como parámetro no esté vacía.</li> <li>• La Posición de recuperado alimentada se refiera a un índice válido: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)</li> </ul>
<b>Modifica</b>	La Lista Indexada y su estructura de datos no sufren ningún cambio. Se retorna una copia del Elemento citado.

• Modificar

<b>Nombre</b>	Modificar
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Una Lista Indexada</li> <li>• Un Elemento (para sustitución)</li> <li>• Una Posición (número entero como índice para modificación)</li> </ul>
<b>Efecto</b>	Reemplaza en la estructura de datos subyacente al elemento preexistente en aquella posición con el nuevo elemento alimentado.
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• La Lista Indexada alimentada como parámetro esté inicializada.</li> <li>• El Elemento alimentado como parámetro esté inicializado</li> <li>• La Posición de modificación alimentada se refiera a un índice válido para la inserción: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)</li> </ul>

<b>Modifica</b>	Únicamente el elemento referido por la posición (basado en un índice) es modificado: sus datos son reemplazados con los datos del elemento alimentado. El resto del modelo lógico y la estructura de datos subyacentes no sufren ningún cambio.
-----------------	---

- NumElem?

<b>Nombre</b>	NumElem
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Una Lista Indexada</li> </ul>
<b>Efecto</b>	Obtiene la cantidad de elementos existente en la Lista Indexada.
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• La Lista Indexada alimentada como parámetro esté inicializada.</li> </ul>
<b>Modifica</b>	La Lista Indexada y su estructura de datos no sufren ningún cambio. Se retorna la cantidad de elementos existente en la Lista indexada.

### Modelo Logico Arbol n-ario.

Árbol N-ario: El modelo lógico Árbol n-ario es un tipo de dato abstracto no lineal ya que está formado por una multitud de ramificaciones, de manera análoga a un árbol orgánico encontrado en la naturaleza.

Este modelo está conformado por Nodos y una jerarquía entre ellos. Se dice que cada Nodo, salvo la raíz (el Nodo más “arriba”), tiene solo un Nodo padre, pero cada Nodo puede tener n hijos (n-ario). Así, cada Nodo puede tener o una relación de preeminencia (padre-hijo), o de igualdad (hermanos).

- Crear

<b>Nombre</b>	Crear
<b>Parámetros</b>	Un Árbol
<b>Efecto</b>	Crea un Árbol y le asigna un espacio



<b>Requiere</b>	Un Árbol que no haya sido inicializado
<b>Modifica</b>	Se inicializa la estructura de datos subyacente.

• Destruir

<b>Nombre</b>	Destruir
<b>Parámetros</b>	Un Árbol
<b>Efecto</b>	Destruye la estructura de datos subyacentes del Árbol
<b>Requiere</b>	Un Árbol que haya sido inicializado
<b>Modifica</b>	La estructura del Árbol, libera el espacio usado

• PonerRaíz

<b>Nombre</b>	PonerRaíz
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Un Árbol</li> <li>• Un Nodo</li> </ul>
<b>Efecto</b>	Asigna al Nodo que se brinda como parámetro como la raíz del Árbol.
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado sin una raíz.</li> <li>• Un Nodo no nulo</li> </ul>
<b>Modifica</b>	Se asigna el Nodo brindado como la raíz del Árbol.

• AgregarHijo?

<b>Nombre</b>	AgregarHijo
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Un Árbol</li> <li>• Una Etiqueta.</li> <li>• Un Nodo “padre”</li> </ul>
<b>Efecto</b>	Crea un Nodo que contiene la Etiqueta dada y lo agrega como hijo más izquierdo del Nodo “padre”
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado</li> <li>• Un Nodo “padre” inicializado</li> </ul>

	que hace parte del Árbol dado <ul style="list-style-type: none"> <li>• Una Etiqueta válida (que mantenga relación de tipos con los elementos del Árbol).</li> </ul>
<b>Modifica</b>	Modifica la estructura del Árbol, agrega un hijo un Nodo del Árbol

• AgregarHijoMásDerecho?

<b>Nombre</b>	AgregarHijoMásDerecho
<b>Parámetros</b>	<ul style="list-style-type: none"> <li>• Un Árbol</li> <li>• Una Etiqueta.</li> <li>• Un Nodo “padre”.</li> </ul>
<b>Efecto</b>	Crea un Nodo que contiene la Etiqueta dada y lo agrega como hijo más derecho del Nodo “padre”
<b>Requiere</b>	Una Árbol inicializado, una Etiqueta válida, un Nodo “padre” inicializado que hace parte del Árbol dado
<b>Modifica</b>	Modifica la estructura del Árbol, pues agrega a un Nodo nuevo como hijo del Nodo alimentado.

• BorrarHoja

<b>Nombre</b>	BorrarHoja
<b>Parámetros</b>	Un Árbol, un Nodo
<b>Efecto</b>	Elimina una hoja de la estructura de datos
<b>Requiere</b>	Un Árbol inicializado, con al menos un Nodo, y que el Nodo que se brinda como parámetro sea una hoja.
<b>Modifica</b>	Disminuye en uno el tamaño del Árbol, elimina al Nodo que se brinda como parámetro.

• Raíz?

<b>Nombre</b>	Raiz
<b>Parámetros</b>	Un Árbol
<b>Efecto</b>	Devuelve el Nodo correspondiente a la raíz del Árbol. Si no hay raíz devuelve un Nodo nulo.
<b>Requiere</b>	Un Árbol inicializado.
<b>Modifica</b>	No modifica la estructura de datos.

• Padre?

<b>Nombre</b>	Padre
<b>Parámetros</b>	Un Árbol, un Nodo
<b>Efecto</b>	Devuelve un Nodo referente al padre de otro Nodo que nos brindan como parámetro, en caso de ser la raíz devuelve un Nodo nulo
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado, con al menos un Nodo.</li> <li>• El Nodo que se pide como parámetro pertenece al Árbol</li> </ul>
<b>Modifica</b>	No modifica la estructura de datos.

• HijoMásIzquierdo?

<b>Nombre</b>	HijoMasIzquierdo
<b>Parámetros</b>	Un Árbol, un Nodo
<b>Efecto</b>	Devuelve el hijo más izquierdo de un Nodo que se pide como parámetro. Si el Nodo dado no tiene hijo, entonces devuelve un Nodo nulo
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado con al menos un elemento.</li> <li>• Que el Nodo que se pide de parámetro pertenezca al Árbol.</li> </ul>
<b>Modifica</b>	No modifica la estructura de datos.

• Hermano Derecho?

<b>Nombre</b>	HermanoDerecho
<b>Parámetros</b>	Un Árbol, un Nodo
<b>Efecto</b>	Devuelve el hermano derecho del Nodo que se pide como parámetro. Si no existe un hermano derecho se devuelve un Nodo nulo
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado con al menos un Nodo</li> <li>• Que el Nodo que se pide de parámetro pertenezca al Árbol</li> </ul>
<b>Modifica</b>	No modifica la estructura de datos.

• Etiqueta?

<b>Nombre</b>	Etiqueta
<b>Parámetros</b>	Un Árbol, un Nodo
<b>Efecto</b>	Devuelve la Etiqueta almacenada en un Nodo
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado.</li> <li>• Un Nodo inicializado que existe en el Árbol dado.</li> </ul>
<b>Modifica</b>	No modifica la estructura de datos.

• ModificaEtiqueta

<b>Nombre</b>	ModificarEtiqueta
<b>Parámetros</b>	Un Árbol, un Nodo, una Etiqueta
<b>Efecto</b>	Cambia el valor de la Etiqueta de un Nodo por la Etiqueta dada
<b>Requiere</b>	<ul style="list-style-type: none"> <li>• Un Árbol inicializado con al menos un Nodo.</li> <li>• Que el Nodo tenga una Etiqueta y exista en el Árbol</li> </ul>
<b>Modifica</b>	Al Nodo que se brinda como parámetro

- NumNodos?

<b>Nombre</b>	NumNodos
<b>Parámetros</b>	Un Árbol, un Nodo válido
<b>Efecto</b>	Devuelve la cantidad de Nodos de un Árbol o subárbol. Se considera que la raíz de ese Árbol es el Nodo dado como parámetro. Si el Árbol solo tiene una raíz y ningún otro Nodo entonces devuelve 1.
<b>Requiere</b>	Un Árbol (o sub árbol) validado e inicializado
<b>Modifica</b>	No modifica la estructura de datos.

## 5. Manual del Usuario

### 5.1. Requerimientos de Hardware

Este programa fue desarrollado en una computadora con las siguientes especificaciones: CPU Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz, con 8GB de memoria RAM.

### 5.2. Requerimientos de Software

Para la ejecución de este programa se requiere tener instalada una versión de c++ de la 11.1.0 en adelante y un medio o ambiente para la ejecución del programa en c++.

### 5.3. Arquitectura del programa

El programa consta de un archivo “main.cpp”, el cual está encargado de mostrar la interfaz al usuario.

Para la implementación de la estructura cola se utilizan los siguientes archivos:

- “Cola.cpp”: Está encargado de implementar la cola mediante arreglo circular.

Para la implementación de la estructura lista Indexada se utilizan los siguientes archivos:

- “Celda.h”: Está encargado de implementar los métodos de la clase celda, estos serán necesarios para la creación de una lista simplemente enlazada.
- “Lista.h”: Está encargado de implementar los métodos de la clase lista, que en conjunto con “Celda.h” crean una lista simplemente enlazada.

- “ListaIndexada.h”: Header file de la lista indexada, implementa los métodos de la clase.
- “ListaIndexada.cpp”: Está encargado de implementar los operadores básicos de la clase lista indexada, que en conjunto con “Celda.h” crean una lista simplemente enlazada.

Para la implementación del árbol con arreglo con señalador al padre se usan los siguientes archivos:

- “3.1.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol\_1.cpp”: Clase utilizada para implementar los operadores basicos del arbol.

Para la implementación del arbol lista de hijos:

- “3.2.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “ArbolLH.hpp”: Está encargado de implementar los operadores básicos de la clase árbol, que en conjunto con “Celda.h” y “Lista.h”, crean una estructura lista de hijos.
- “Celda.h”: Está encargado de implementar los métodos de la clase celda, estos serán necesarios para la creación de una lista simplemente enlazada.
- “Lista.h”: Está encargado de implementar los métodos de la clase lista, que en conjunto con “Celda.h” crean una lista simplemente enlazada.
- “NodoConcreto.hpp”: Clase utilizada como Nodo del arbol n-ario.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho con contador:

- “3.3.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase posee contador.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho con puntero al padre y al hermano izquierdo sin contador:

- “3.4.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase no posee contador y posee un señalador al padre y al hermano izquierdo.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho tal que el último hijo de un nodo apunta al padre sin contador:

- “3.5.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase no posee contador y el ultimo hijo de un nodo apunta al padre.

#### 5.4. Compilación

Para todos nuestros códigos usamos el mismo compilador: G++.

##### **Cola**

Note: en nuestro caso la función main esta metida dentro del mismo archivo Cola

Para compilar la Cola, usamos el siguiente comando:

```
g++ Cola.cpp -o code.out
```

##### **Lista indexada**

Todos los archivos se encuentran en el mismo repositorio:

```
g++ *.cpp
```

##### **Árboles**

Para cada prueba se usa el mismo main, solo se descomenta uno de los header que contiene el árbol que queremos usar, y después se compila con el siguiente comando:

```
g++ main.cpp
```