UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS

CATEDRÁTICO: ING. EDGAR RENE ORNELIS HOILS

TUTOR ACADÉMICO: ELIAN SAUL ESTRADA URBINA



**MANUAL TÉCNICO RENTA DE ACTIVOS**

ENNER ESAÍ MENDIZABAL CASTRO

CARNÉ: 202302220

SECCIÓN: A

GUATEMALA, 16 DE DICIEMBRE DEL 2,024

# ÍNDICE

[ÍNDICE 1](#_Toc165147344)

[INTRODUCCIÓN 2](#_Toc165147345)

[OBJETIVOS 2](#_Toc165147346)

[1. GENERAL 2](#_Toc165147347)

[2. ESPECÍFICOS 2](#_Toc165147348)

[ALCANCES DEL SISTEMA 2](#_Toc165147349)

[ESPECIFICACIÓN TÉCNICA 3](#_Toc165147350)

[● REQUISITOS DE HARDWARE 3](#_Toc165147351)

[● REQUISITOS DE SOFTWARE 3](#_Toc165147352)

[DESCRIPCIÓN DE LA SOLUCIÓN 4](#_Toc165147353)

[LÓGICA DEL PROGRAMA 7](#_Toc165147354)

# INTRODUCCIÓN

Este documento técnico tiene como objetivo proporcionar una descripción detallada de la arquitectura, diseño e implementación del programa Renta de Activos. Se presentarán las soluciones técnicas empleadas, los tipos de datos abstractos utilizados y la lógica de programación subyacente a cada una de sus funcionalidades. Con esto, se busca facilitar la comprensión del sistema a desarrolladores interesados en replicar o extender sus capacidades.

# OBJETIVOS

## GENERAL

* 1. Documentar de manera exhaustiva las soluciones técnicas adoptadas en la implementación del programa Renta de Activos.

## ESPECÍFICOS

* 1. Detallar los Tipos de Datos Abstractos (TDA) utilizados en la construcción del programa y su justificación.
  2. Describir la lógica de implementación de las estructuras de datos y algoritmos empleados, así como su relación con las funcionalidades del sistema.

# ALCANCES DEL SISTEMA

Este manual se centra en la arquitectura interna del programa Renta de Activos, proporcionando una visión detallada de su diseño y desarrollo. Se abordan aspectos como:

* Estructura: Organización general del código y relación entre los diferentes módulos.
* Algoritmos: Descripción de los algoritmos utilizados para resolver los problemas planteados.
* Tipos de datos: Definición y uso de los TDA empleados para representar la información.
* Tecnologías: Herramientas y lenguajes de programación utilizados en el desarrollo.

Este documento está dirigido a desarrolladores con conocimientos básicos de programación y un interés particular en la arquitectura de software y el diseño de algoritmos.

# ESPECIFICACIÓN TÉCNICA

## REQUISITOS DE HARDWARE

* + Resolución mínima de 1024x768
  + 8 GB de memoria RAM
  + Dispositivos de entrada y salida: Pantalla, Ratón y Teclado.
  + 3.5GB de almacenamiento libre en disco

## REQUISITOS DE SOFTWARE

* + Sistema operativo: Windows, macOS o Linux
  + Compilador de C++/C. (Si se tiene Clion, no es necesario)
  + Entorno de desarrollo. Se sugiere utilizar un entorno de desarrollo integrado (IDE) como Clion, aunque cualquier editor de texto con soporte para C++ puede ser utilizado.

# LOGICA Y DESCRIPCIÓN DE LA SOLUCIÓN

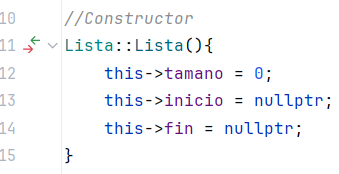
Para la creación del programa Renta Activos, se utilizaron varias estructuras utilizando las clases, usando convenciones para tener una mejor calidad de código creando *headers*, las cuales son:

## Lista Circular Doblemente enlazada

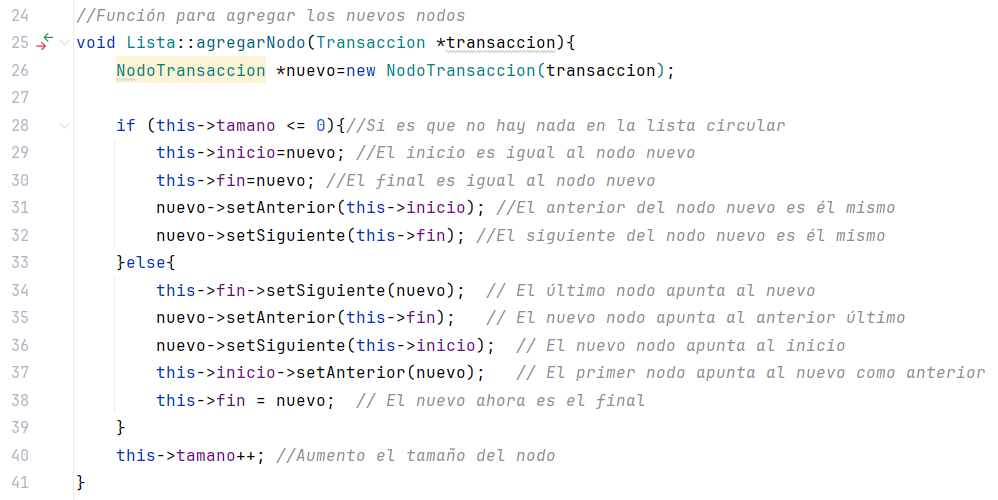
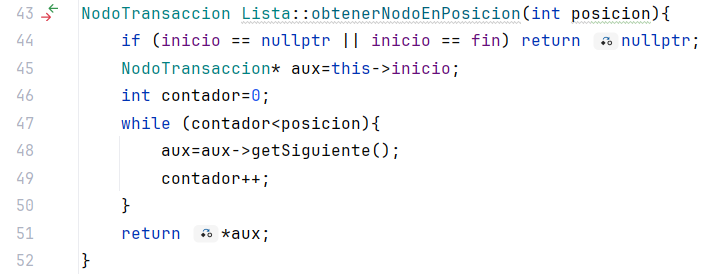
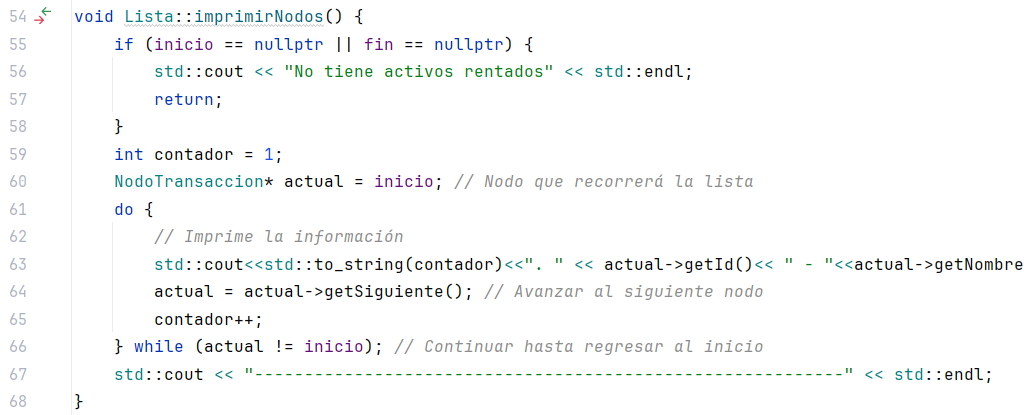
Una lista circular doblemente enlazada es una estructura de datos lineal en la que cada elemento, que se llama nodo, contiene, además de los datos, dos punteros: uno que referencia al nodo siguiente y otro que apunta al nodo anterior. En esta estructura, a diferencia de las listas normales, el último nodo apunta al primero, y el primero al último, formando así un círculo.

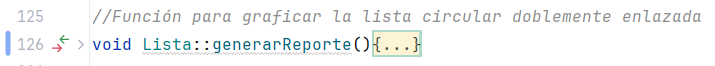
Para este proyecto, se usó este tipo de lista para guardar las transacciones que se harían durante la ejecución del programa Renta Activos y sus métodos principales son estos:

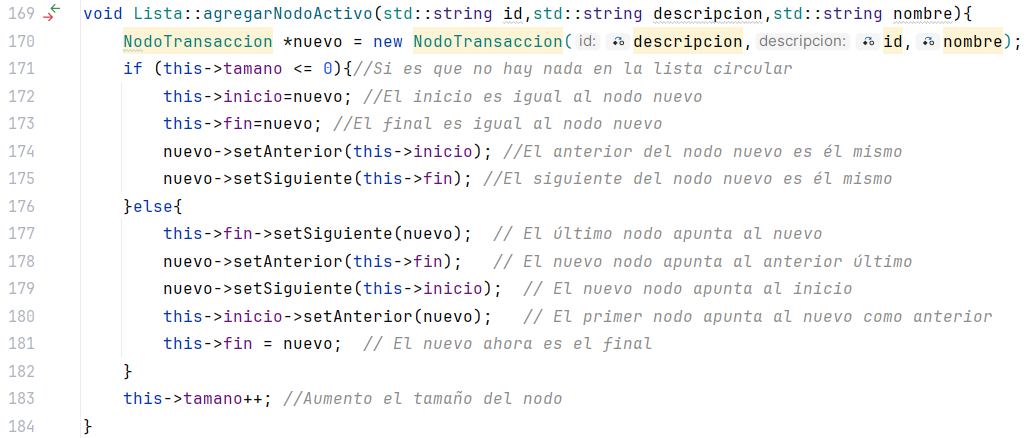
### Constructor

* **Lista()**: Inicializa una lista circular doblemente enlazada vacía, estableciendo los punteros inicio y fin en nullptr y el tamaño en 0. 

### Operaciones de la lista

* **agregarNodo(Transaccion\* transaccion):** Añade un nuevo nodo al final de la lista creando un nuevo NodoTransaccion con la transacción proporcionada. Si la lista está vacía, el nuevo nodo se convierte tanto en el inicio como en el fin de la lista, apuntándose a sí mismo como anterior y siguiente. Si la lista no está vacía, se actualizan los punteros del último nodo y del nuevo nodo para mantener la circularidad y el enlace doble. 
* **obtenerNodoEnPosicion(int posicion)**: Retorna un puntero al nodo que se encuentra en la posición especificada. Realiza una busqueda iterativa. 
* **imprimirNodos():** Imprime la información contenida en cada nodo de la lista. Recorre la lista circular y muestra el ID, nombre y descripción de cada nodo. Maneja el caso de una lista vacía
* **ordenarAscendente()**: Ordena la lista de forma ascendente según el idTransaccion de los objetos Transaccion almacenados en cada nodo. Utiliza un algoritmo de burbuja modificado para intercambiar punteros a los nodos. 
* **ordenarDescendente()**: Igual a ordenarAscendente(), pero ordena la lista de forma descendente según el idTransaccion (básicamente solo cambio un símbolo) 
* **generarReporte()**: La función recorre la lista y crea nodos en el grafo .dot con la información de cada transacción, conectándolos con flechas bidireccionales para representar el doble enlace.



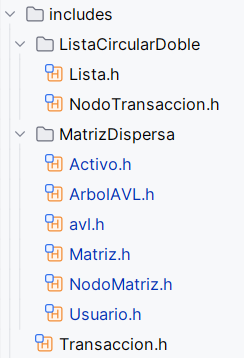
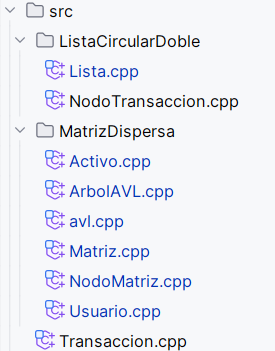
* **agregarNodoActivo(std::string id, std::string descripcion, std::string nombre):** Añade un nuevo nodo a la lista, pero en este caso, el nodo se crea directamente con un id, una descripcion y un nombre, en lugar de un objeto Transaccion completo. Esto debido a que se utilizará para guardar la información de los activos de los usuarios (reutilizando la estructura). 
* **eliminarNodoPorId(std::string id):** Elimina un nodo de la lista buscando por su ID.
* g**enerarReporteActivosRentados(std::string user):** Genera un reporte gráfico DOT similar a generarReporte(), pero hecho para los activos rentados.

## Matriz Dispersa

## Arbol AVL

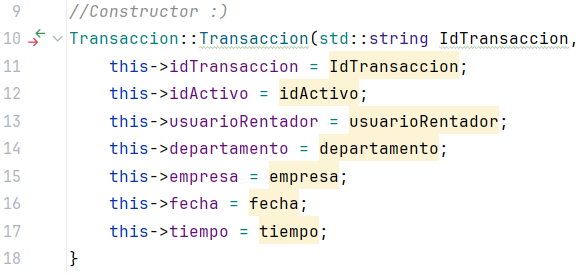
## Distintas Clases utilizadas

Como se mencionó previamente, se utilizaron *headers* para permitir un código de mejor calidad, por tal motivo, todas las clases, por consiguiente, las estructuras abstractas ya descritas, lo usan:

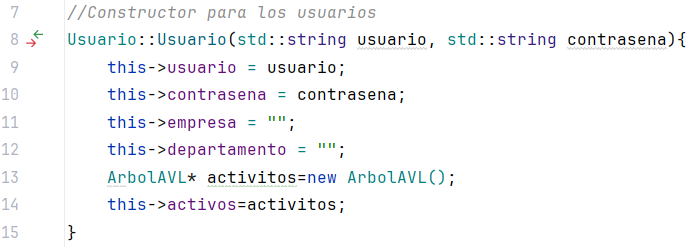
Las clases que se utilizaron a lo largo del proyecto para almacenar y representar de mejor manera la información son las siguiente:

### Transaccion



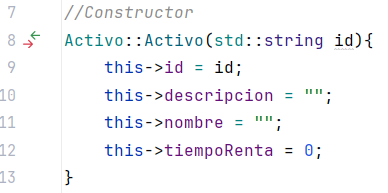
Esta clase se utilizó para almacenar la información de todas las transacciones dentro del nodo que iría dentro de la lista circular doblemente enlazada.

### Usuario



Dentro del programa Renta Activos, es esencial el manejo de usuario para que estos manejen los activos de manera óptima, por tal razón, esta clase permite representar cada uno de los usuarios dentro de la plataforma.

### Activo



La principal función de la aplicación es la renta de activos, por tal motivo se creó una clase que permitiría almacenar la información de cada activo que los usuario publiquen o renten.

## main.cpp

Esta es la función principal dentro del código, desde esta es donde todo comienza, por tal motivo, esta contiene la mayoría de funciones y ciclos que dan el flujo del programa.