Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Estructura de Datos, 1r. Semestre 2024.

PIXEL PRINT STUDIO

FASE 2

202201947– PABLO ANDRES RODRIGUEZ LIMA

Resumen

Los árboles en estructuras de datos son un tipo de estructura jerárquica que consta de nodos interconectados. Cada nodo tiene un valor y puede tener cero o más nodos hijos, organizados en una estructura de árbol. Estos árboles se utilizan para representar relaciones jerárquicas entre elementos, como estructuras de archivos en sistemas operativos o relaciones de parentesco en genealogías. Ofrecen eficiencia en la búsqueda, inserción y eliminación de elementos, siendo esenciales en la implementación de algoritmos y aplicaciones informáticas complejas.

Palabras clave

1. Jerárquico
2. Nodos
3. Relaciones padre-hijo
4. Búsqueda
5. Eficiencia

Abstract

Trees in data structures are hierarchical structures consisting of interconnected nodes. Each node holds a value and can have zero or more child nodes, organized in a tree-like structure. These trees are used to represent hierarchical relationships between elements, such as file structures in operating systems or parent-child relationships in genealogies. They offer efficiency in searching, inserting, and deleting elements, being essential in implementing complex algorithms and computer applications.

Keywords

1. Hierarchical
2. Nodes
3. Parent-child Relationships
4. Search
5. Efficiency

Introducción

En el mundo de la programación, las Estructuras de Datos Abstractas (EDA) son fundamentales para organizar y manipular datos de manera eficiente. Una EDA es una colección de datos y un conjunto de operaciones definidas en esos datos, independientes de cualquier implementación concreta. Estas estructuras proporcionan un marco conceptual para entender cómo los datos están organizados y cómo se puede acceder a ellos y manipularlos.

Desarrollo del tema

En el ámbito de la programación, las Estructuras de Datos Abstractas (EDA) son herramientas esenciales para organizar y manipular datos de manera eficiente. Estas estructuras proporcionan un enfoque sistemático para almacenar y acceder a datos, independientemente de la implementación subyacente. Entre las EDA más comunes se encuentran las listas enlazadas simples, las listas enlazadas dobles, las listas circulares, las estructuras tipo pila y las estructuras tipo cola.

Dato: Un dato es una unidad de información básica que puede ser procesada por un computador. Los tipos de datos más comunes son números, letras y símbolos.

- Los datos pueden ser clasificados en diferentes tipos, como:
    - **Tipos básicos:** enteros, reales, booleanos, caracteres.
    - **Tipos compuestos:** cadenas de texto, arrays, estructuras.

Estructura de Datos: Una estructura de datos es una forma específica de organizar y almacenar datos en la memoria de un computador para que puedan ser utilizados de manera eficiente. Cada estructura de datos tiene sus propias características y ventajas para diferentes tipos de operaciones.

Algunas de las operaciones más comunes que se pueden realizar sobre las estructuras de datos son:

- **Búsqueda:** encontrar un elemento específico dentro de la estructura.
- **Inserción:** agregar un nuevo elemento a la estructura.
- **Eliminación:** remover un elemento de la estructura.
- **Recorrido:** acceder a todos los elementos de la estructura en un orden específico

Árbol Binario: Un árbol binario es una estructura de datos no lineal en la que cada nodo tiene como máximo dos hijos. Los nodos se pueden organizar de forma recursiva, creando una estructura con subárboles.

- Los árboles binarios se utilizan para representar relaciones jerárquicas entre datos.
- Un ejemplo de un árbol binario es un árbol genealógico.

Árbol AVL: Un árbol AVL es un tipo de árbol binario auto-balanceado, lo que significa que la altura de cada subárbol siempre es similar. Esta propiedad permite realizar operaciones de búsqueda, inserción y eliminación de forma eficiente.

- Los árboles AVL son más eficientes que los árboles binarios normales para operaciones de búsqueda, inserción y eliminación.
- Se utilizan en aplicaciones donde la velocidad de acceso a los datos es importante.

Lista Doblemente Enlazada: Una lista doblemente enlazada es una estructura de datos lineal en la que cada nodo tiene dos punteros: uno al nodo anterior y otro al siguiente. Esta estructura permite acceder a los

datos en ambos sentidos, facilitando la eliminación e inserción de nodos en cualquier posición.

- Las listas doblemente enlazadas son más flexibles que las listas simples para realizar operaciones de inserción y eliminación de nodos.
- Se utilizan en aplicaciones donde es necesario modificar la lista con frecuencia.

Árbol B: Un árbol B es un árbol balanceado que puede tener más de dos hijos por nodo. Esta estructura se utiliza para almacenar grandes cantidades de datos de forma eficiente, ya que permite realizar búsquedas y operaciones de actualización en tiempo logarítmico.

- Los árboles B son más eficientes que los árboles binarios para almacenar grandes cantidades de datos.
- Se utilizan en bases de datos y sistemas de archivos.

Lista Simple: Una lista simple es una estructura de datos lineal en la que cada nodo tiene un solo puntero al siguiente nodo. Es la estructura de datos más simple y se utiliza para almacenar datos en orden secuencial.

- Las listas simples son la estructura de datos más simple y eficiente en términos de memoria.
- Se utilizan para almacenar datos en orden secuencial.

FORTRAN FPM para la Lectura de JSON: FPM (Flexible Parsing Model) es una extensión del lenguaje de programación FORTRAN que permite leer y escribir archivos JSON de forma sencilla. FPM proporciona una serie de herramientas para analizar la estructura del archivo JSON y acceder a sus datos.

- FPM es una herramienta poderosa para leer y escribir archivos JSON en FORTRAN.
- FPM facilita el acceso a los datos dentro del archivo JSON.

Matriz Dispersa: Una matriz dispersa es una matriz en la que la mayoría de los elementos son nulos. Se utiliza para almacenar datos que no están distribuidos uniformemente, lo que permite ahorrar espacio en memoria.

- Las matrices dispersas son una forma eficiente de almacenar matrices con muchos elementos nulos.
- Se utilizan en aplicaciones donde la mayoría de los elementos de la matriz son nulos.

Conclusiones

Las Estructuras de Datos Abstractas son herramientas esenciales en programación para organizar y manipular datos de manera eficiente.

Las listas enlazadas simples, dobles y circulares, junto con las estructuras tipo pila y cola, son ejemplos fundamentales de Estructuras de Datos Abstractas con aplicaciones variadas en el desarrollo de software.

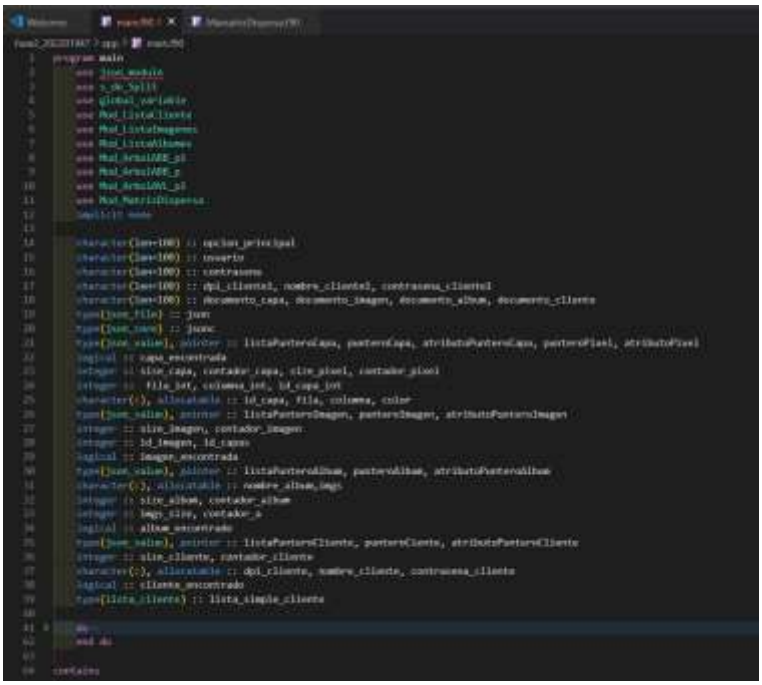
Dominar estas estructuras es crucial para mejorar la eficiencia y la calidad del código, así como para resolver una amplia gama de problemas de programación de manera efectiva.

Referencias bibliográficas

Programación II - Tipo de Dato Abstracto. (n.d.). https://sites.google.com/site/programacioniiuno/temario/unidad-2---tipo-abstracto-de-dato/tipo-de-dato-abstracto

Estructuras de datos: listas enlazadas, pilas y colas. (n.d.). https://calcifer.org/documentos/librognome/glib-lists-queues.html

MENU PRINCIPAL



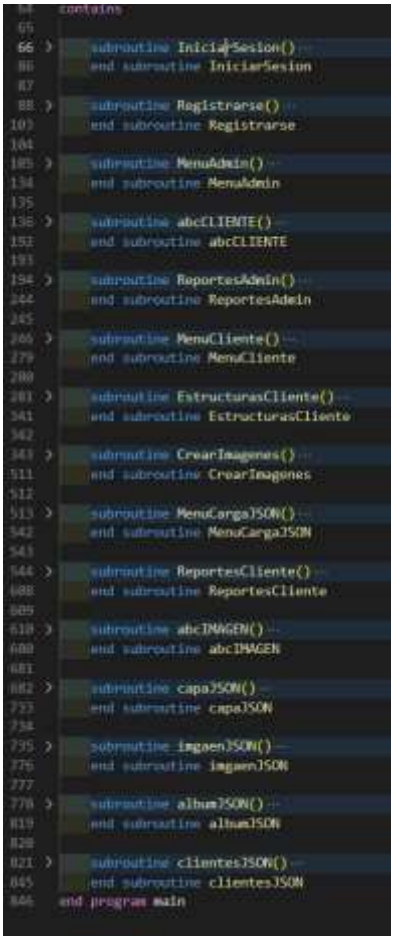
Uso de las importaciones de las estructuras:

En esta sección, se detalla el uso adecuado de las importaciones de las estructuras necesarias para el funcionamiento del proyecto. Las importaciones proporcionan acceso a las bibliotecas y módulos externos requeridos para implementar funcionalidades específicas. A continuación, se describen las principales estructuras que deben importarse y su propósito en el proyecto. Además, se explican las buenas prácticas para organizar las importaciones y garantizar la modularidad y legibilidad del código.

Declaración de variables para el menú y para la lectura de los JSON:

Esta sección aborda la declaración apropiada de variables necesarias para la implementación del menú y la lectura de archivos JSON en el proyecto. Las variables desempeñan un papel fundamental en el almacenamiento temporal de datos y en la configuración de opciones disponibles para el usuario. Se detallan las convenciones de nomenclatura y los tipos de datos recomendados para cada variable, así como su ámbito de aplicación dentro del código.

Subrutinas del menú principal:



Esta sección detalla las subrutinas que componen el menú principal del proyecto en Fortran. El menú principal incluye opciones para iniciar sesión como cliente o administrador, así como las funcionalidades específicas asociadas con cada rol. Se describen las subrutinas relevantes para el proceso de inicio de sesión, la navegación del menú y la ejecución de las acciones correspondientes a cada rol.

Inicio de sesión:

Se describe la subrutina encargada de gestionar el proceso de inicio de sesión, donde los usuarios ingresan sus credenciales para acceder al sistema como cliente o administrador. Se explican los pasos necesarios para validar las credenciales proporcionadas por el usuario y dirigirlos al menú correspondiente según su rol.

Menú del cliente:

Se detallan las subrutinas que forman parte del menú destinado a los clientes una vez que han iniciado sesión. Este menú proporciona opciones específicas para que los clientes gestionen sus álbumes, carguen capas de imágenes y realicen otras acciones relacionadas con su cuenta. Se describen las funcionalidades disponibles y cómo los clientes pueden interactuar con el sistema para llevar a cabo sus tareas.

Menú del administrador:

Se describen las subrutinas que componen el menú diseñado para los administradores después de iniciar sesión. Este menú ofrece opciones para que los administradores gestionen clientes, carguen información en la base de datos y realicen otras tareas administrativas relacionadas con el sistema. Se explican las restricciones y permisos específicos asociados con el rol de administrador, como la capacidad exclusiva de cargar clientes en la base de datos.

Funcionalidades específicas:

Se detallan las funcionalidades exclusivas de cada rol, como la capacidad de los administradores para cargar clientes en el sistema y la capacidad de los clientes para cargar capas de imágenes y álbumes. Se explican las restricciones y validaciones asociadas con estas funcionalidades para garantizar la integridad y seguridad de los datos en el sistema.

Matriz Dispersa

Fase2_202201947 > src > MamatrizDispersa.f90

```fortran
module Mod_MatrizDispersa
    implicit none
    private
    type, public :: nodo_valor
        logical :: existe = .false.
        character(len=30) :: color
    end type nodo_valor

    type :: nodo_pixel
        private
        integer :: columna, fila
        character(len=:), allocatable :: color
        type(nodo_pixel), pointer :: arriba => null()
        type(nodo_pixel), pointer :: abajo => null()
        type(nodo_pixel), pointer :: derecha => null()
        type(nodo_pixel), pointer :: izquierda => null()
    end type nodo_pixel

    type, public :: matriz
        private
        type(nodo_pixel), pointer :: raiz => null()
        integer, public :: ancho = 0
        integer, public :: alto = 0
        contains
        procedure :: insertar_nodo
        procedure :: insertar_matriz
        procedure :: insertar_cabecera_fila
        procedure :: insertar_cabecera_columna
        procedure :: buscar_fila
        procedure :: buscar_columna
        procedure :: existe_nodo
        procedure :: graficar_matriz
    end type matriz

contains
```

Una matriz dispersa en Fortran es una estructura de datos eficiente que almacena solo los elementos no nulos junto con sus ubicaciones (fila, columna). Enumerando sus propiedades:

1. **Eficiencia de almacenamiento:** Almacena solo elementos no nulos, reduciendo el uso de memoria en comparación con matrices densas.

Subrutina para insertar un elemento: Esta subrutina permite insertar un nuevo elemento en la matriz dispersa, especificando su valor, fila y columna. Si el elemento ya existe en la matriz, se actualiza su valor.

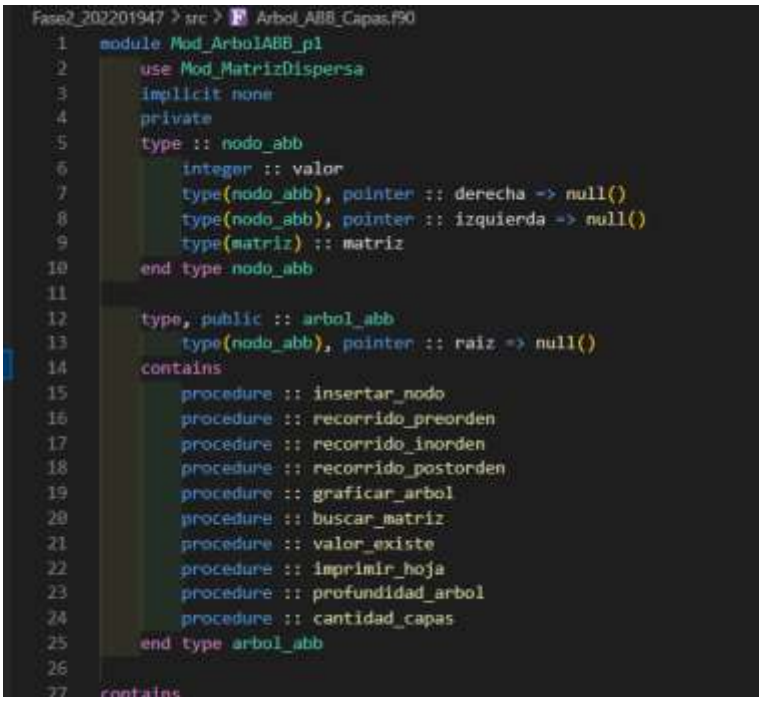
Subrutinas para gestionar la cabecera de filas y columnas: Estas subrutinas ayudan a mantener la estructura de la matriz dispersa, incluyendo la gestión de los punteros que apuntan al inicio de cada fila y columna.

Subrutina para buscar una fila: Esta subrutina busca una fila específica en la matriz dispersa y devuelve un puntero al primer elemento de esa fila.

Función para verificar la existencia de un nodo: Esta función verifica si un nodo específico existe en la matriz dispersa, dada su fila y columna. Devuelve un valor lógico que indica si el nodo está presente o no.

Función para graficar la matriz dispersa: Esta función grafica la matriz dispersa, mostrando sus elementos no nulos y su ubicación en relación con las filas y columnas. Esta función puede utilizar bibliotecas gráficas externas o generar una representación visual simple dentro de la consola.

ARBOL ABB

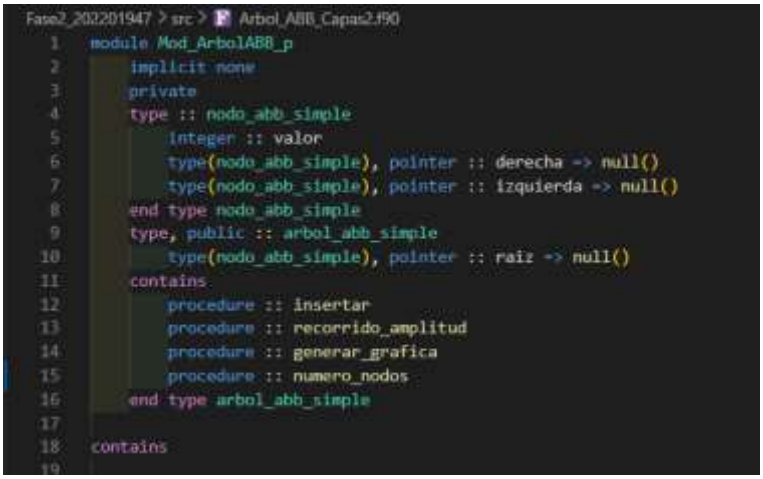


Para incorporar un Árbol Binario de Búsqueda (ABB) en tu proyecto de Fortran, que contiene una matriz dispersa en cada hoja, puedes implementar las siguientes subrutinas y funciones:

1. **Subrutinas de recorrido:** Recorrido preorden: Esta subrutina recorre el árbol ABB en orden preorden, visitando primero el nodo raíz, luego el subárbol izquierdo y finalmente el subárbol derecho.

Recorrido inorden: Esta subrutina recorre el árbol ABB en orden inorden, visitando primero el subárbol izquierdo, luego el nodo raíz y finalmente el subárbol derecho.

Recorrido postorden: Esta subrutina recorre el árbol ABB en orden postorden, visitando primero el subárbol izquierdo, luego el subárbol derecho y finalmente el nodo raíz.

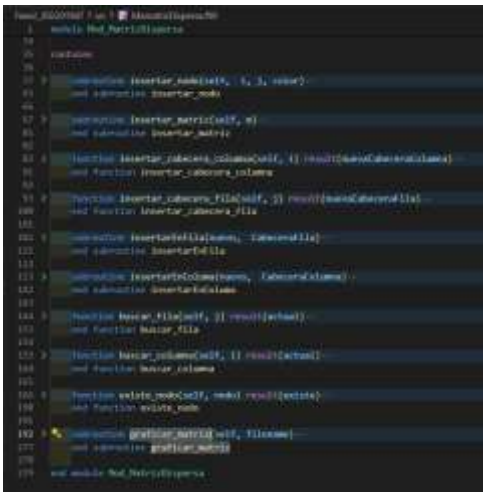


2. **Subrutina para graficar el árbol:**

Esta subrutina genera una representación gráfica del árbol ABB, mostrando sus nodos y conexiones entre ellos. Puedes utilizar bibliotecas gráficas externas o generar una representación visual simple dentro de la consola.

3. **Función para calcular la cantidad de capas:**

- Esta función determina la cantidad total de capas en el árbol ABB, contando desde la raíz hasta la hoja más profunda.



4. **Función para calcular la profundidad del árbol:**

Esta función calcula la profundidad del árbol ABB, es decir, la longitud del camino más largo desde la raíz hasta una hoja.

Estas subrutinas y funciones te permitirán trabajar eficazmente con un Árbol Binario de Búsqueda que contiene matrices dispersas en cada hoja. Además, podrás realizar operaciones de recorrido para explorar la estructura del árbol, generar representaciones visuales y obtener información estadística sobre su composición y profundidad.

ARBOL AVL

Para integrar un Árbol AVL en tu proyecto Fortran, que almacena imágenes en sus nodos, puedes adaptar las subrutinas y funciones anteriores para que funcionen con este tipo específico de árbol. Aquí te proporciono una descripción de cómo podrían ser estas subrutinas y funciones:

1. **Subrutinas de recorrido:**

   - Recorrido preorden: Recorre el árbol AVL en orden preorden, visitando primero el nodo raíz, luego el subárbol izquierdo y finalmente el subárbol derecho.
   - Recorrido inorden: Recorre el árbol AVL en orden inorden, visitando primero el subárbol izquierdo, luego el nodo raíz y finalmente el subárbol derecho.
   - Recorrido postorden: Recorre el árbol AVL en orden postorden, visitando primero el subárbol izquierdo, luego el subárbol derecho y finalmente el nodo raíz.

2. **Subrutina para graficar el árbol:**

- Genera una representación visual del árbol AVL, mostrando sus nodos y conexiones entre ellos. Puedes utilizar bibliotecas gráficas externas o generar una representación visual simple dentro de la consola.

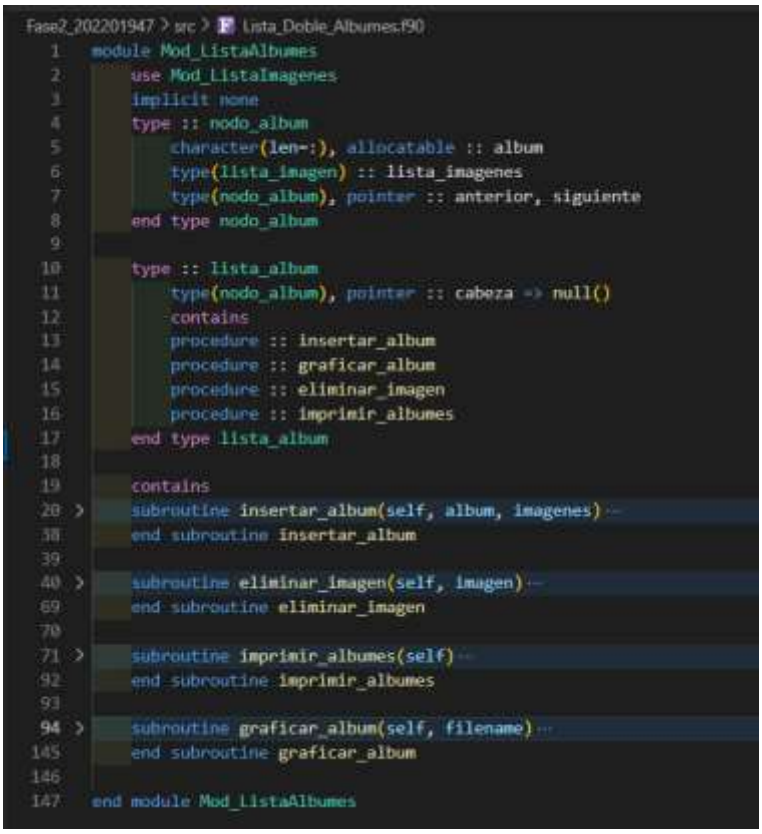
3. **Función para calcular la cantidad de imágenes:**

    - Determina la cantidad total de imágenes almacenadas en el árbol AVL, contando los nodos que contienen imágenes.

4. **Función para calcular la profundidad del árbol:**

    - Calcula la profundidad del árbol AVL, es decir, la longitud del camino más largo desde la raíz hasta una hoja.

LISTA DOBLEMENTE ENLAZADA



Para manejar álbumes utilizando una lista doblemente enlazada en tu proyecto Fortran, puedes implementar las siguientes subrutinas y funciones:

1. **Subrutina para insertar un álbum:**

- Esta subrutina permite insertar un nuevo álbum en la lista doblemente enlazada, especificando su nombre u otras características relevantes.

2. **Subrutina para eliminar un álbum:**

- Elimina un álbum específico de la lista doblemente enlazada, utilizando su nombre u otro identificador único.

3. **Subrutina para graficar la lista de álbumes:**

- Genera una representación visual de la lista doblemente enlazada de álbumes, mostrando los enlaces entre ellos.

4. **Subrutina para imprimir la lista de álbumes:**

- Imprime en la consola la información de todos los álbumes presentes en la lista doblemente enlazada, mostrando sus nombres u otras características relevantes.

5. **Subrutina para buscar un álbum:**

    - Busca un álbum específico en la lista doblemente enlazada, utilizando su nombre u otro identificador único, y devuelve un puntero al nodo que lo contiene.

LISTA CLIENTES

Para administrar una lista de clientes en tu proyecto Fortran, puedes utilizar una estructura de datos como una lista enlazada simple. Aquí te muestro cómo podrías implementar las subrutinas necesarias para insertar, modificar, eliminar y graficar clientes en esta lista:

1. **Subrutina para insertar un cliente:**

    - Esta subrutina permite agregar un nuevo cliente a la lista enlazada, proporcionando información como el nombre, ID, dirección de correo electrónico, etc.

2. **Subrutina para modificar un cliente:**

    - Permite actualizar la información de un cliente existente en la lista, identificándolo por su ID u otro identificador único, y proporcionando los nuevos datos.

3. **Subrutina para eliminar un cliente:**

    - Elimina un cliente específico de la lista enlazada, utilizando su ID u otro identificador único.

4. **Subrutina para graficar la lista de clientes:**

Fase2_202201947 > src > Lista_Simple_Clientes.f90

```
module Mod_ListaCliente
    use Mod_ArbolABB_p1
    use Mod_ArbolAVL_p1
    use Mod_ListaAlbumes
    implicit none
    type :: nodo_cliente
        character(len=:), allocatable :: dpi
        character(len=:), allocatable :: nombre
        character(len=:), allocatable :: contrasena
        type(arbol_abb) :: arbol_abb_capa
        type(arbol_avl) :: arbol_avl_imagen
        type(lista_album) :: lista_doble_album
        type(nodo_cliente), pointer :: siguiente => null()
    end type nodo_cliente

    type :: lista_cliente
        type(nodo_cliente), pointer :: cabeza => null()
    contains
        procedure :: insertar_cliente
        procedure :: eliminar_cliente
        procedure :: modificar_cliente
        procedure :: grafica_cliente
        procedure :: cliente_existe
        procedure :: iniciar_sesion_c
        procedure :: obtener_cliente
        procedure :: mostrar_cliente
        procedure :: reporte_albumes_cliente
        procedure :: reporte_imagenes_cliente
        procedure :: reporte_capas_cliente
        procedure :: reporte_listar_cliente
    end type lista_cliente

contains
```

- Genera una representación visual de la lista de clientes, mostrando los enlaces entre ellos y posiblemente alguna información relevante, como sus nombres y detalles.