Sergio Alejandro Rodriguez López 202131443
Laboratorio de Estructura de Datos

Practica No. 2
Manual Técnico

Descripción General del Programa:

El programa busca ser un gestor de contactos eficiente para el uso cotidiano del cliente. Permite crear grupos de contactos que el cliente puede agregar a su conveniencia, así como guardar la información que vea necesaria y buscar contactos con parámetros específicos de cada contacto.
Se busca un sistema eficiente de gestión por lo que se utilizaron árboles binarios de búsqueda para guardar los datos de manera fácil y ordenada y estos mismos nos facilitan el poder agregar o buscar datos específicos.

Descripción General de las Clases:

Árbol:

La clase Arbol implementa las operaciones básicas de un árbol binario de búsqueda, como la inserción y búsqueda de nodos. Además, proporciona funcionalidades para obtener información asociada a un dato específico. Utiliza la clase Nodo para representar los nodos del árbol y probablemente utiliza la clase Log para registrar eventos o mensajes importantes durante la ejecución del programa.
Tiene sus respectivos derivados árboles que manejan los distintos tipos de datos que se trabajan en el proyecto, es decir, Char, Int, Date, String.

Métodos Privados:

- insertRecursive(): Un método recursivo para insertar un nuevo nodo en el árbol de manera ordenada.
- findRecursive(): Un método recursivo para buscar un nodo con un dato específico en el árbol.
- buscarPorContenidoRecursivo(): Un método recursivo para buscar un nodo por su contenido (dato) en el árbol.
- height(): Un método que calcula la altura de un nodo en el árbol.
- balanceFactor(): Un método que calcula el factor de equilibrio de un nodo en el árbol.
- rotateRight(): Un método que realiza una rotación hacia la derecha en el árbol.
- rotateLeft(): Un método que realiza una rotación hacia la izquierda en el árbol.
- balance(): Un método que balancea el árbol después de la inserción de un nuevo nodo.

Métodos públicos:

- insert(): Un método público para insertar un nuevo nodo en el árbol.
- find(): Un método público para buscar un nodo con un dato específico en el árbol.
- obtenerInfoDato(): Un método público que devuelve la información asociada a un dato específico del árbol.

Nodo:

La clase Nodo proporciona una estructura básica para almacenar datos genéricos junto con información adicional sobre un contacto. Se puede utilizar en conjunción con otras clases para construir diferentes tipos de estructuras de datos, como listas enlazadas. Además, utiliza un objeto de la clase Log para registrar eventos o errores importantes durante la ejecución del programa.

Métodos Públicos:

- definirSiguiente(): Establece el puntero al siguiente nodo.
- definirAnterior(): Establece el puntero al nodo anterior.
- irSiguiente(): Devuelve el puntero al siguiente nodo.
- irAnterior(): Devuelve el puntero al nodo anterior.
- insertar(): Inserta un nuevo dato en el nodo junto con información de contacto.
- obtenerDato(): Devuelve el dato almacenado en el nodo.

Grupo:

En resumen, esta clase Grupo se utiliza para administrar los contactos de un grupo específico, almacenando su información de manera organizada en árboles de diferentes tipos de datos. Permite agregar nuevos contactos, buscar información de contactos existentes y agregar diferentes tipos de datos a los contactos. Además, hace uso de un objeto de la clase Log para registrar eventos o mensajes importantes durante la ejecución del programa.

Métodos privados:

- obtenerTipo(): Un método para determinar el tipo de datos asociado con un campo de información específico.
- agregarArbol(): Un método para agregar un árbol correspondiente al tipo de datos de un campo específico.
- hash(): Un método para calcular un valor hash utilizado para asignar un campo a un índice en los arreglos de árboles.

Métodos públicos:

- Grupo(): El constructor de la clase, que inicializa los arreglos de árboles y procesa la lista de tipos de datos para construir la lista de campos.
- agregarContacto(): Un método para agregar un nuevo contacto al grupo, procesando y almacenando su información en los árboles correspondientes.
- busqueda(): Un método para buscar la información de un contacto basado en un campo específico.
- agregarNombre(), agregarChar(), agregarInt(), agregarStrn(), agregarDate(): Métodos para agregar diferentes tipos de datos a los contactos en el grupo.

Grupos:

La clase Grupos es parte de un sistema que gestiona grupos de contactos y permite realizar operaciones como agregar grupos, agregar contactos a esos grupos, buscar contactos, entre otras. La clase utiliza una estructura de datos de tabla hash para almacenar los grupos de contactos, lo que permite un acceso eficiente a los mismos. Además, hace uso de un objeto de la clase Log para registrar eventos o errores importantes durante la ejecución del programa.

Métodos Públicos:

- getName(): Devuelve el nombre del grupo.
- definirComando(): Define el comando de inserción y su cantidad.
- ingresarContacto(): Inserta un nuevo contacto en el grupo.
- hash(): Implementa una función de hash simple para mapear el nombre de los grupos a índices del arreglo de contactos.
- agregarGrupo(): Agrega un nuevo grupo al contenedor.
- agregarContacto(): Agrega un contacto a un grupo existente.
- buscar(): Busca un contacto dentro de un grupo y devuelve el resultado.

Main:

En resumen, esta clase principal actúa como un programa interactivo de gestión de contactos que permite al usuario crear grupos, agregar contactos a estos grupos y buscar contactos dentro de ellos, utilizando la clase Grupos para manejar la lógica de la gestión de contactos y la clase Log para registrar eventos importantes.

Funciones de Identificación:

- identificarCreacion(): Analiza el comando de entrada para identificar la creación de nuevos grupos de contactos, extrayendo el nombre del grupo y sus campos.
- identificarInsercion(): Analiza el comando de entrada para identificar la inserción de un nuevo contacto en un grupo específico.
- identificarBusqueda(): Analiza el comando de entrada para identificar la búsqueda de un contacto en un grupo específico.

Función Principal:

- Presenta un menú de opciones al usuario para realizar diferentes acciones: ingresar nuevos grupos, agregar contactos o buscar contactos.
- Lee la entrada del usuario y ejecuta la función correspondiente para identificar y procesar el comando ingresado.
- Utiliza un bucle while para continuar mostrando el menú y procesando los comandos hasta que el usuario decida salir.

MakeFile:

Variables definidas:

- CPP = g++: Define la variable CPP como el compilador de C++ a utilizar, en este caso, g++.
- TARGET = ../Practica_2: Define la variable TARGET como la ruta y el nombre del archivo de salida del programa compilado.

Objetivo all:

- all: $(TARGET): Define el objetivo principal all, que depende del archivo de destino $(TARGET). Esto significa que al ejecutar make all, se compilará el programa.

Regla para compilar el programa ($(TARGET)):

- $(TARGET): ./main.o: Indica que el archivo de destino $(TARGET) depende del archivo ./main.o.
- $(CPP) $(objetos_construccion)/main.o -o $(TARGET): Utiliza el compilador g++ para enlazar el archivo objeto main.o y generar el archivo ejecutable $(TARGET).

Regla para compilar el archivo objeto (./main.o):

- ./main.o: ./main.cpp Log.h Nodo.h Arbol.h ArbolDate.h ArbolInt.h ArbolString.h ArbolChar.h Grupo.h Grupos.h: Indica que el archivo objeto ./main.o depende de los archivos de código fuente y cabecera listados.
- $(CPP) -c main.cpp -o $(objetos_construccion)/main.o: Compila el archivo main.cpp en un archivo objeto main.o y lo guarda en la carpeta de objetos de construcción (se supone que $(objetos_construccion) es una variable no definida aquí).

Objetivo clean:

- clean: Define el objetivo clean, que se utiliza para limpiar los archivos generados durante la compilación.
- rm -f $(objetos_construccion)/*.o $(TARGET): Borra todos los archivos objeto (*.o) en la carpeta de objetos de construcción y el archivo ejecutable $(TARGET).

Para compilar el programa, simplemente ejecute el comando make en el directorio donde se encuentra este Makefile. Esto iniciará el proceso de compilación y generará el ejecutable Practica_2 en el directorio superior (../). Si desea limpiar los archivos generados, puede ejecutar make clean.