UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE

ESTUDIANTE:
JORGE ANIBAL BRAVO RODRÍGUEZ

CARNÉ:
202131782

AUXILIAR:
YEFER RODRIGO ALVARADO TZUL

CURSO:
ESTRUCTURAS DE DATOS - LABORATORIO

PRACTICA 2

1. HERRAMIENTAS DE DESARROLLO

Ultramarine Linux
Ultramarine Linux es un sistema operativo de código abierto basado en Linux que ofrece una plataforma estable y segura para el desarrollo y la ejecución de aplicaciones. Se caracteriza por su equilibrio entre funcionalidades avanzadas y eficiencia, lo que lo convierte en una opción ideal para entornos de tamaño mediano que requieren un sistema operativo confiable y flexible.

Makefile
Un Makefile es un archivo utilizado en el desarrollo de software para automatizar el proceso de compilación y enlazado de programas. Contiene reglas que especifican cómo se deben compilar los archivos fuente y cómo se deben generar los ejecutables. En entornos de tamaño mediano, el uso de Makefiles facilita la gestión y la organización de proyectos de software de manera eficiente.

Make
El programa Make es una herramienta de construcción utilizada en entornos de desarrollo de software para automatizar el proceso de compilación y enlazado de programas. Make interpreta un archivo llamado Makefile que contiene reglas y dependencias para determinar cómo compilar y vincular los archivos fuente de un proyecto. Su función principal es reconstruir solo los componentes modificados de un proyecto, lo que agiliza el proceso de desarrollo y garantiza la coherencia en la generación de ejecutables.

Visual Studio Code
Visual Studio Code es un editor de código fuente ligero y altamente personalizable desarrollado por Microsoft. Se destaca por su amplia gama de extensiones y su integración con herramientas de desarrollo populares. En entornos de tamaño mediano, Visual Studio Code proporciona un entorno de desarrollo versátil y eficaz para programadores que buscan una solución ágil y potente.

C++
C++ es un lenguaje de programación de alto nivel y multiparadigma diseñado para el desarrollo de software de sistemas y aplicaciones de alto rendimiento. Creado como una extensión del lenguaje C, C++ incorpora características orientadas a objetos que permiten la programación orientada a objetos, así como características de programación genérica y funcional. Con una sintaxis flexible y potente, C++ es ampliamente utilizado en la industria del software para desarrollar aplicaciones de gran escala, sistemas embebidos, juegos, herramientas de desarrollo y más. Su capacidad para combinar eficiencia, flexibilidad y abstracción lo convierte en una herramienta poderosa para programadores que buscan un equilibrio entre rendimiento y facilidad de uso.

G++

G++ es un compilador de código abierto que forma parte del conjunto de herramientas GNU Compiler Collection (GCC) y se utiliza principalmente para compilar programas escritos en C++ y C. En entornos de tamaño mediano, G++ ofrece un rendimiento sólido y una amplia compatibilidad con los estándares de programación, lo que lo convierte en una opción confiable para el desarrollo de software en este tipo de entornos.

Graphviz

Graphviz es una herramienta de visualización de gráficos que permite representar de manera gráfica estructuras de datos y relaciones complejas. Con su lenguaje de descripción de gráficos DOT, Graphviz facilita la creación de diagramas y visualizaciones interactivas. En entornos de tamaño mediano, Graphviz es una herramienta útil para la representación visual de datos y procesos.

Git

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software para gestionar cambios en el código fuente de manera eficiente y colaborativa. Permite a los equipos de desarrollo trabajar de forma concurrente en proyectos, realizar seguimiento de versiones y fusionar cambios de manera controlada. En entornos de tamaño mediano, Git es fundamental para mantener la integridad y la trazabilidad del código en proyectos de software.

2. ARCHIVO MAKEFILE

El código c++ escrito para esta práctica fue hecho en un editor de código (Visual Studio Code) por lo que prescindí de herramientas que automatizaran la compilación desde un primer momento.

Para generar el ejecutable final utilicé la herramienta make propuesta en clase junto al archivo de compilación Makefile, tomando como ejemplo el código alojado en github por parte del auxiliar. Dicho archivo es el siguiente:

```
CPP = g++
TARGET = ../Binario_compilado/contactos
OBJ_DIR = ./obj
ESTR = ./Estructuras
ARB = ./Estructuras/Arboles
HSH = ./Estructuras/Hash

# all
all: $(TARGET)

$(TARGET): $(OBJ_DIR)/Main.o
   $(CPP) $(OBJ_DIR)/Main.o -o $(TARGET)

#Las definiciones que necesita Main para funcionar estan en los archivos Nodo.h y Arbol.h
$(OBJ_DIR)/Main.o: Main.cpp $(ESTR)/Nodo.h $(ESTR)/Arbol.h  $(ESTR)/Date.h $(ESTR)/TiposCamposEnum.h
$(ESTR)/Logger.h $(ARB)/ArbolBase.h $(ARB)/ArbolInt.h $(ARB)/ArbolString.h $(ARB)/ArbolChar.h
$(ARB)/ArbolDate.h $(HSH)/Campos.h $(HSH)/Grupos.h
   $(CPP) -c Main.cpp -o $(OBJ_DIR)/Main.o

# clean
clean:
   rm -f $(OBJ_DIR)/*.o $(TARGET)
```

- Variables Definidas:
  - CPP: Define el compilador a utilizar, en este caso, g++.
  - TARGET: Especifica la ruta y nombre del archivo binario compilado, en este caso, ../Binario_compilado/contactos.
  - OBJ_DIR: Directorio donde se almacenarán los archivos objeto generados durante la compilación.
  - ESTR, ARB, HSH: Definen rutas a directorios relacionados con las estructuras de datos utilizadas en el proyecto.

- Objetivo "all":
  - Define la regla all, que depende del objetivo $(TARGET).
  - El objetivo all se encarga de compilar el archivo binario especificado en $(TARGET).

- Regla para Compilar el Target:
  - La regla para compilar el $(TARGET) depende del archivo objeto $(OBJ_DIR)/Main.o.
  - Utiliza el compilador definido en CPP para compilar Main.o y generar el archivo binario en $(TARGET).

- Regla para Compilar Main.o:
  - La regla para compilar Main.o especifica las dependencias necesarias para compilar correctamente el archivo Main.cpp.
  - Se incluyen los archivos de cabecera necesarios para Main.cpp como Nodo.h, Arbol.h, Date.h, TiposCamposEnum.h, Logger.h, entre otros.

- Utiliza el compilador g++ para compilar Main.cpp y generar el archivo objeto Main.o en el directorio $(OBJ_DIR).

- Objetivo "clean":
  - Define la regla clean que se encarga de eliminar los archivos objeto generados y el archivo binario compilado.

3. ÁRBOL BINARIO DE BÚSQUEDA - AVL

La práctica requería una implementación propia de un Árbol Binario AVL.

Clase Nodo:
La clase nodo es una clase plantilla que permite definir una clase sin un tipo predeterminado para aprovecharse con diferentes tipos en tiempo de compilación. Al utilizar plantillas, se puede escribir código que sea independiente del tipo de datos con el que se va a operar, lo que facilita la reutilización y la flexibilidad del código.

Lo hice de esta manera para no duplicar código, ya que se nos requirió que los árboles trabajen con **enteros, fechas, caracteres** y **cadenas**.

```
#ifndef NODO_H
#define NODO_H

template <typename T>
class Nodo{

   private:
        Nodo<T>* izquierdo;
        Nodo<T>* derecho;
        T dato;
        int id;

   public:
        Nodo(T dato, int id);

        Nodo<T>* obtIzquierdo();
        Nodo<T>* obtDerecho();

        void insertarIzquierdo(Nodo<T>* nodo);
        void insertarDerecho(Nodo<T>* nodo);

        T obtDato();
        int obtId();
};
#endif
```

Código Nodo.h

Clase Árbol.h

Implementa un árbol binario AVL de búsqueda que puede almacenar cualquier tipo de dato.
Se aprovecha de la clase Nodo descrita anteriormente, permitiendo ser utilizado con diferentes tipos de datos.

Código Arbol.h

```
template<typename T>
class Arbol{


   private:
       bool vacio;
       int cantidadNodos;
       Nodo<T>* raiz;
       Logger logger;

       void insertarRecursivo(Nodo<T>* nodo, T dato, int id);

       Nodo<T>* buscarPorIdRecursivo(int id, Nodo<T>* nodo);
       Nodo<T>* buscarPorContenidoRecursivo(T contenido, Nodo<T>* nodo);

       int altura(Nodo<T>* nodo);
       int calcularFactorBalanceo(Nodo<T>* nodo);
       void balancearRecursivo(Nodo<T>* nodo);

       void rotacionIzquierda(Nodo<T>* nodo);
       void rotacionDerecha(Nodo<T>* nodo);
       void rotacionIzquierdaDerecha(Nodo<T>* nodo);
       void rotacionDerechaIzquierda(Nodo<T>* nodo);

       void generarGraficoRecursivo(Nodo<T>* nodo, std::ofstream& archivo);

       string nombreCampo;

       string obtenerGrafoRecursivo(Nodo<T>* nodo);
       string obtenerListaElementosRecursivo(Nodo<T>* nodo);
   public:
       Arbol(string nombre);
       bool estaVacio();
       void insertar(T dato, int id);


       Nodo<T>* buscarPorId(int id);
       Nodo<T>* buscarPorContenido(T contenido);

      void balancear();
      void generarGrafico();

      string obtenerGrafo();
      string obtenerNombreCampo();

      string obtenerListaElementos();
```

```
        string to_string(Date* date);
        string to_string(int dato);
        string to_string(string dato);
        string to_string(char dato);
};
```

- **Atributos:**

vacio: Indica si el árbol está vacío.
cantidadNodos: Cantidad de nodos en el árbol.
raiz: Puntero al nodo raíz del árbol.
nombreCampo: Nombre del campo que representa el árbol (para generar el grafo).

- **Métodos:**

1. **Constructores:**

Arbol(string nombre): Crea un árbol vacío con el nombre de campo especificado.

2. **Modificadores:**

insertar(T dato, int id): Inserta un nuevo nodo en el árbol con el dato y el ID especificados.
balancear(): Balancea el árbol para mantenerlo con una altura lo más uniforme posible.

3. **Accesores:**

estaVacio(): Devuelve true si el árbol está vacío, false en caso contrario.
buscarPorId(int id): Busca un nodo en el árbol por su ID y devuelve un puntero al mismo.
buscarPorContenido(T contenido): Busca un nodo en el árbol por su contenido y devuelve un puntero al mismo.
obtenerGrafo(): Devuelve una cadena con la representación en formato DOT del árbol.
obtenerListaElementos(): Devuelve una cadena con una lista de los elementos del árbol.

4. **Métodos privados:**

insertarRecursivo(Nodo<T> nodo, T dato, int id):* Inserta un nuevo nodo en el árbol de forma recursiva.
buscarPorIdRecursivo(int id, Nodo<T> nodo):* Busca un nodo en el árbol por su ID de forma recursiva.
buscarPorContenidoRecursivo(T contenido, Nodo<T> nodo):* Busca un nodo en el árbol por su contenido de forma recursiva.
calcularFactorBalanceo(Nodo<T> nodo):* Calcula el factor de balanceo de un nodo.
altura(Nodo<T> nodo):* Calcula la altura de un nodo.
balancearRecursivo(Nodo<T> nodo):* Balancea el árbol de forma recursiva.
rotacionIzquierda(Nodo<T> nodo):* Realiza una rotación izquierda en un nodo.
rotacionDerecha(Nodo<T> nodo):* Realiza una rotación derecha en un nodo.
rotacionIzquierdaDerecha(Nodo<T> nodo):* Realiza una rotación izquierda-derecha en un nodo.
rotacionDerechaIzquierda(Nodo<T> nodo):* Realiza una rotación derecha-izquierda en un nodo.

generarGraficoRecursivo(Nodo<T> nodo, std::ofstream& archivo):* Genera la representación en formato DOT del árbol de forma recursiva.
obtenerGrafoRecursivo(Nodo<T> nodo):* Devuelve una cadena con la representación en formato DOT del árbol de forma recursiva.
obtenerListaElementosRecursivo(Nodo<T> nodo):* Devuelve una cadena con una lista de los elementos del árbol de forma recursiva.
to_string(T dato): Convierte un dato de tipo T a una cadena.

5. **Definición de métodos:**

insertar(T dato, int id):

Inserta un nuevo nodo en el árbol con el dato y el ID especificados.
Si el árbol está vacío, se crea un nuevo nodo raíz.
De lo contrario, se llama a la función insertarRecursivo para insertar el nodo de forma recursiva.
Se incrementa la cantidad de nodos en el árbol.

estaVacio():

Devuelve true si el árbol está vacío, false en caso contrario.
Se verifica si el puntero a la raíz es nullptr.

buscarPorId(int id):

Busca un nodo en el árbol por su ID y devuelve un puntero al mismo.
Si el árbol está vacío, se devuelve nullptr.
De lo contrario, se llama a la función buscarPorIdRecursivo para buscar el nodo de forma recursiva.

buscarPorContenido(T contenido):

Busca un nodo en el árbol por su contenido y devuelve un puntero al mismo.
Si el árbol está vacío, se devuelve nullptr.
De lo contrario, se llama a la función buscarPorContenidoRecursivo para buscar el nodo de forma recursiva.

balancear():

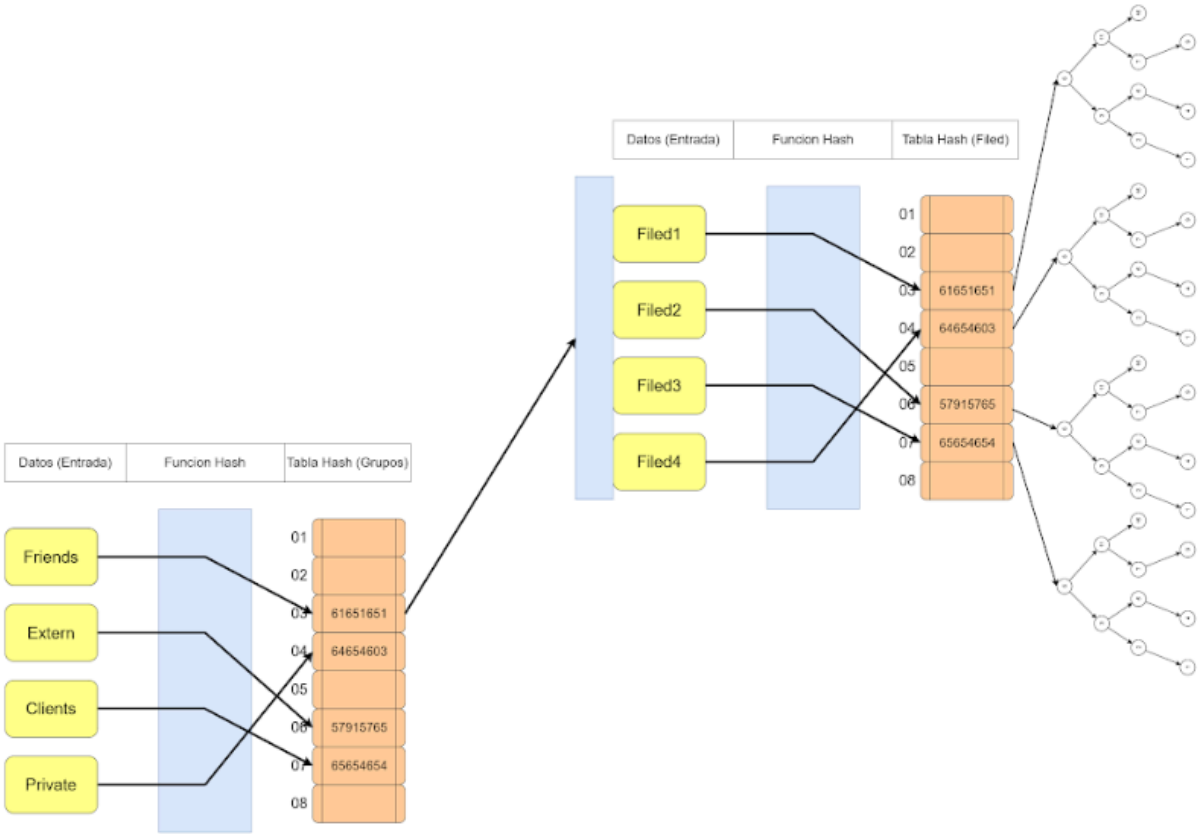
Balancea el árbol para mantenerlo con una altura lo más uniforme posible.
Se llama a la función balancearRecursivo.

- **Envolturas para la clase Arbol**

La práctica se basa en arrays para manejar las tablas hash. Para poder usar diferentes instancias de un arbol de diferentes tipos en un mismo array usé herencia.
De esta manera, creé la clase **ArbolBase**, de la cual, heredan **ArbolInt, ArbolChar, ArbolString** y **ArbolDate,** especializandola con el tipo de dato indicado en su nombre.

4. IMPLEMENTACION DEL SISTEMA DE CONTACTOS

El sistema de gestión de contactos hace uso de dos clases con el fin de lograr la estructura descrita en el documento de definición de la práctica. Dicha estructura es la siguiente:

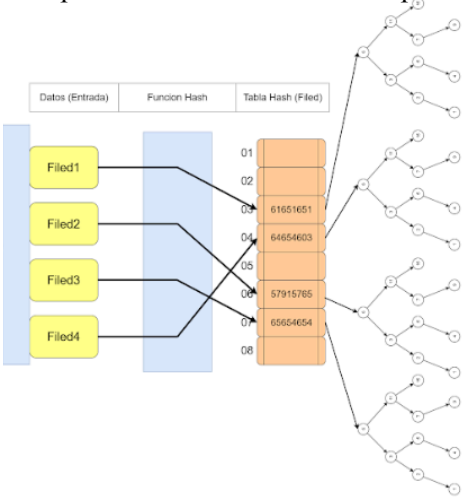


(Ilustración tomada del documento de especificación de la práctica)

Clase Campos

Tiene como finalidad albergar y administrar un array de punteros de árboles binarios de búsqueda correspondientes a un campo con un tipo de dato por posición. Para ello, se vale de una función hash que evalúa el nombre del campo y retorna una posición del array. Esta función hash es útil al crear un nuevo campo para el grupo y buscar una tupla en base a un dato y un campo, además de servir en funciones para crear un grafo de un grupo y de todo el sistema.

Representación de la clase Campos



Clase Grupos

La clase Grupos, de manera similar a la clase Campos, posee un array de punteros de tipo Campos, administrando todos los grupos del sistema. Para ello , al igual que lo descrito anteriormente, se vale de métodos de hash y rehash para asignar posiciones del array en base a una cadena con el nombre del grupo.

Función hash

La función hash es la misma tanto en la clase Grupos como en la clase Campos. Se basan en la suma del valor ascii de la cadena de entrada para luego verificar la cantidad de desplazamientos dentro del tamaño del array.

```
Funcion hash
int Grupos::hash(string nombreGrupo){
       int sumaCaracteres = 0;
       int caracter = 0;
       int resultado = 0;

       if(!nombreGrupo.empty()){
           //Sumar los valores ascii del nombre del campo y desplazazar
esa cantidad de veces dentro del tamanio
           for(char caracter : nombreGrupo){
               caracter = static_cast<int>(caracter);
               sumaCaracteres += caracter;
           }
           resultado = sumaCaracteres % tamanio;

       }
       return resultado;
}
```

El desplazamiento fue planteado originalmente de esta manera:

```
for (int i = 0; i < sumaCaracteres; i++){
   resultado ++;
   if(resultado > tamanio){
        resultado = 0;
   }
}
```

Pero realiza la misma función que el módulo de sumaCaracteres entre tamaño.

Función reHash

El re-hashing es una operación que se reajusta para otorgar nuevas posiciones a las anteriormente proporcionadas por la función hash. Para este proyecto, se implementa una función de re-hashing en la clase Grupos, la cual no posee un tamaño estático para el array de punteros de Campos. Esta función redimensiona el array ya descrito al alcanzar el 60% de la capacidad establecida reubicando todos los elementos del array original en su nueva posición. Una vez aplicado el re-hash, la función hash retorna posiciones ajustadas puesto que se incrementa la variable tamaño.

```
void Grupos::reHash(){
   // Decidi aumentar un 60% el tamaño del array
   int tamanioTemporal = tamanio;
   tamanio *= 1.6;
   Campos** gruposTemporal = new Campos*[tamanio];
   for (int i = 0; i < tamanio; i++)
   {
       gruposTemporal[i] = nullptr;
   }

   Campos* grupoActual;
   string nombreGrupoActual;
   int nuevaPosicion;

   // Copiar la informacion del array en orden segun la funcion hash
   for (int i = 0; i < tamanioTemporal; i++)
   {
       grupoActual = grupos[i];
       if(grupoActual != nullptr){
           nombreGrupoActual = grupoActual->obtenerGrupo();
           nuevaPosicion = hash(nombreGrupoActual);
           if(gruposTemporal[nuevaPosicion] == nullptr){
               gruposTemporal[nuevaPosicion] = grupoActual;
               logger.log(nombreGrupoActual+" fue movido de la posicion " +
to_string(i) + " a la posicion" + to_string(nuevaPosicion) + " por la funcion reHash en
Grupos.h.");

           }else{
               logger.log("Hubo una colision en Grupos.h: en la funcion reHash.\nSe
intento asignar el grupo: "+ nombreGrupoActual + " en la posicion: " +
to_string(nuevaPosicion) + " otorgada por la funcion hash, pero " +
gruposTemporal[nuevaPosicion]->obtenerGrupo() + " ya se encontraba en esa posicion.");
           }
       }
   }
   grupos = gruposTemporal;
}
```

5. CLASE LOGGER

En todo momento se guardan registros de lo sucedido en el programa en un archivo .log indicando la acción ejecutada en la clase y función donde ocurre y si se completó satisfactoriamente o si hubo algún problema. A cada registro le acompaña la identificación de la fecha y hora en la que tuvo lugar.

Para ello decidí crear una clase que ejecute las acciones de registro sobre un archivo ubicado en ./Reportes/Contactos.log. La carpeta Reportes se encuentra en el mismo directorio en donde se almacena el ejecutable del programa.

El método que hace posible la funcionalidad de log es el siguiente:

```
void log(const std::string& message) {
       std::cout << message << std::endl;
       std::ofstream file("./Reportes/Contactos.log", std::ios::app);
       if (file.is_open()) {
           std::time_t now = std::time(nullptr);
           char timestamp[20];
                       std::strftime(timestamp,  sizeof(timestamp),  "%d-%m-%H:%M:%S",
std::localtime(&now));
```

```
            file << timestamp << std::endl;
            file << message << std::endl;
            file << std::endl;
            file.close();
        }
    }
```

6. LENGUAJE

Para poder interactuar con el programa se cuenta con un sencillo CLI (Interfaz de Linea de Comandos) el cual es capaz de detectar comandos como agregar nuevo grupo, agregar contacto en grupo y buscar contacto en grupo.

Los comandos requeridos por el documento de la práctica son los siguientes:

- Agregar nuevo grupo

```
ADD NEW-GROUP [NOMBRE DEL GRUPO] FIELDS ([campo TipoDato], ...);
```

Agrega un nuevo grupo (una instancia de la clase campos) al array de punteros de la clase Campos, agregando a su vez árboles binarios de tipos especificados en la entrada como [campo TipoDato].

- Insertar contacto

```
ADD CONTACT IN [NOMBRE DEL GRUPO] FIELDS ([dato Del Campo], ...);
```

Accede a un grupo pasando [NOMBRE DEL GRUPO] por la función hash y delega la inserción del contacto en la clase Campos y Árbol.

- Buscar contacto

```
FIND CONTACT IN [NOMBRE DEL GRUPO] CONTACT-FIELD [campo]=[DatoQueBusca];
```

Accede a un grupo pasando [NOMBRE DEL GRUPO] por la función hash, para luego acceder al campo (arbol de un tipo de dato) pasando [campo] por la función hash y retornando un valor junto a un id. Este id sirve para buscar el resto de la tupla en los demás campos.

Comandos adicionales: Además de los comandos especificados en el documento de la práctica, se solicitó que se hicieran otras operaciones sobre el sistema de gestión de contactos. Introduje los siguientes comandos para ello:

- EXIT;

Cierra el programa de gestión de contactos.

- GENERATE GRAPH GROUP [grupo];

Genera un archivo con código .dot interpretable por graphviz para generar el grafo del estado de un grupo especificado en [grupo]. Para ello, se aprovecha de funciones de recorrido recursivo dentro de la clase Arbol.

- GENERATE CONTACTS GRAPH;

Genera un archivo con código .dot referente a un grafo con la información de todo el sistema y su disposición a través de las diferentes estructuras existentes. Se aprovecha del código implementado para generar el grafo de un grupo individual.

- EXPORT GROUP [grupo];

Crea un directorio para el grupo especificado donde se vuelcan archivos .ct con el contenido de cada contacto almacenado en ese grupo.

- IMPORT FROM SCRIPT [path/del/script];

Lee un archivo que puede contener las instrucciones descritas en este apartado y las ejecuta.

El CLI del programa está basado en el reconocimiento de órdenes evaluando longitudes de cadena y comparando subcadenas con órdenes modelo ya establecidas.

Por otro lado, el reconocimiento de pares clave-valor como campo TIPO se hacen por medio de acumuladores de caracteres conjuntos. Cada una de las implementaciones de estos acumuladores están escritas en base a las necesidades de cada acción, sin embargo, presento aquí un código modelo perteneciente al proyecto para estos acumuladores conjuntos:

```
Ejemplo modelo de cómo se tratan las entradas conjuntas clave-valor

string acumulador;
string campo = "";
string tipo = "";
for (int i = 0; i < campos.length(); i++)
{
   while(campo == "" | tipo == ""){
       if(i < campos.length() && campos[i] != ' '){
           acumulador += campos[i];
           if(acumulador.length() > 0){
               if(acumulador[acumulador.length()-1] == ','){
                   acumulador.erase(acumulador.length()-1, 1);
               }
           }
       }else{
           // Establecer par clave valor
           if(campo.empty()){
               campo = acumulador;
               acumulador = "";
           }else if(tipo.empty()){
               tipo = acumulador;
               acumulador = "";
           }
       }
       i++;
   }
```

```
   // Verificar que haya par clave valor para agregar el tipo
   if(campo.length() > 0 && tipo.length() > 0){
       if(tipo == "INTEGER"){
           grupos[posicionGrupo]->agregarCampo(campo, 1);
       }else if(tipo == "STRING"){
           grupos[posicionGrupo]->agregarCampo(campo, 2);
       }else if(tipo == "CHAR"){
           grupos[posicionGrupo]->agregarCampo(campo, 3);
       }else if(tipo == "DATE"){
           grupos[posicionGrupo]->agregarCampo(campo, 4);
       }else{
            logger.log("Error en Grupos.h en funcion AgregarGrupo. \nSe intento agregar
un grupo pero no se reconocio el tipo deseado. \nTipo que se quiso agregar: " + tipo);
       }
       campo = "";
       tipo = "";
       i--;
   }
}
```

7. IMPORTAR DESDE SCRIPT

Esta funcionalidad redirecciona la lectura de instrucciones desde la consola hacia un archivo de texto especificado por el usuario. Si no se puede acceder al archivo, continua en modo lectura por consola. Estos archivos de script deben contener instrucciones en el propio lenguaje de la aplicación. Si hay instrucciones irreconocibles no se ejecutará ninguna acción.

```
Ejemplo de un archivo script de entrada
ADD NEW-GROUP escola FIELDS(tipo CHAR, estudiante STRING, fechaDeInscripcion DATE, clave
INTEGER);
ADD CONTACT IN escola FIELDS(Z, Pamela, 01-01-2003, 12);
ADD CONTACT IN escola FIELDS(P, Diana, 22-03-2010, 81);
ADD CONTACT IN escola FIELDS(M, Carlos, 13-12-1991, 11);
ADD CONTACT IN escola FIELDS(X, Andrea, 30-12-2004, 13);
ADD CONTACT IN escola FIELDS(A, Mario, 02-11-1994, 15);
ADD CONTACT IN escola FIELDS(B, Fernanda, 14-02-1992, 14);
ADD CONTACT IN escola FIELDS(O, Luis, 12-05-2005, 10);
ADD CONTACT IN escola FIELDS(V, Alejandro, 12-12-1997, 18);
ADD CONTACT IN escola FIELDS(W, Carolina, 28-05-1996, 24);
ADD CONTACT IN escola FIELDS(Y, Sofia, 07-04-2000, 21);
ADD NEW-GROUP clientes FIELDS (nombre STRING, telefono INTEGER);
ADD CONTACT IN clientes FIELDS (Pedro, 12453225);
ADD CONTACT IN clientes FIELDS (Juan, 45123568);
ADD CONTACT IN clientes FIELDS (Bartolo, 1001205);
ADD CONTACT IN clientes FIELDS (Maria, 55845543);
ADD CONTACT IN clientes FIELDS (Judas, 22323232);
ADD NEW-GROUP frens FIELDS (nombre STRING, telefono INTEGER, cumpleanoz DATE);
ADD CONTACT IN clientes FIELDS (Juan, 45123568);
ADD CONTACT IN clientes FIELDS (Bartolo, 1001205);
ADD CONTACT IN clientes FIELDS (Maria, 55845543);
ADD CONTACT IN clientes FIELDS (Isabel, 55554444);
ADD CONTACT IN clientes FIELDS (Andres, 12348765);
ADD CONTACT IN clientes FIELDS (Daniela, 13579246);
ADD CONTACT IN clientes FIELDS (Hugo, 24681357);
ADD CONTACT IN clientes FIELDS (Mariana, 78901234);
ADD CONTACT IN clientes FIELDS (Javier, 10293847);
ADD CONTACT IN clientes FIELDS (Valeria, 67584930);
```

```
GENERATE GRAPH GROUP clientes;
FIND CONTACT IN clientes CONTACT-FIELD nombre=Pedro;
FIND CONTACT IN clientes CONTACT-FIELD nombre=Juan;
FIND CONTACT IN clientes CONTACT-FIELD nombre=Bartolo;
FIND CONTACT IN clientes CONTACT-FIELD telefono=12453225;
FIND CONTACT IN clientes CONTACT-FIELD telefono=001205;
FIND CONTACT IN clientes CONTACT-FIELD telefono=55555555;
GENERATE GRAPH GROUP escola;
GENERATE CONTACTS GRAPH;
```

8. COMPILACIÓN

Este programa fue escrito en C++, lo que significa que no es exclusivo de un sistema operativo.
El código fuente no utiliza librerías que se encuentren para un solo sistema operativo en específico, sino que son multiplataforma, lo que hace que, aunque fue escrito y compilado en y para una computadora ejecutando Linux, puede ser fácilmente recompilado para otros sistemas como Mac o Windows.

En este caso, la construcción del binario final fue realizada utilizando g++ y la herramienta make a través de un makefile, por lo que, para su compilación también es necesario contar con estas herramientas.

Recomiendo realizar un make clean en la carpeta donde se encuentra el makefile, luego, ejecutar make debería ser suficiente para generar el binario. La ruta de salida para el ejecutable es /Binario_compilado/contactos (desde la raíz del proyecto).