Manual técnico

Gestor de contactos

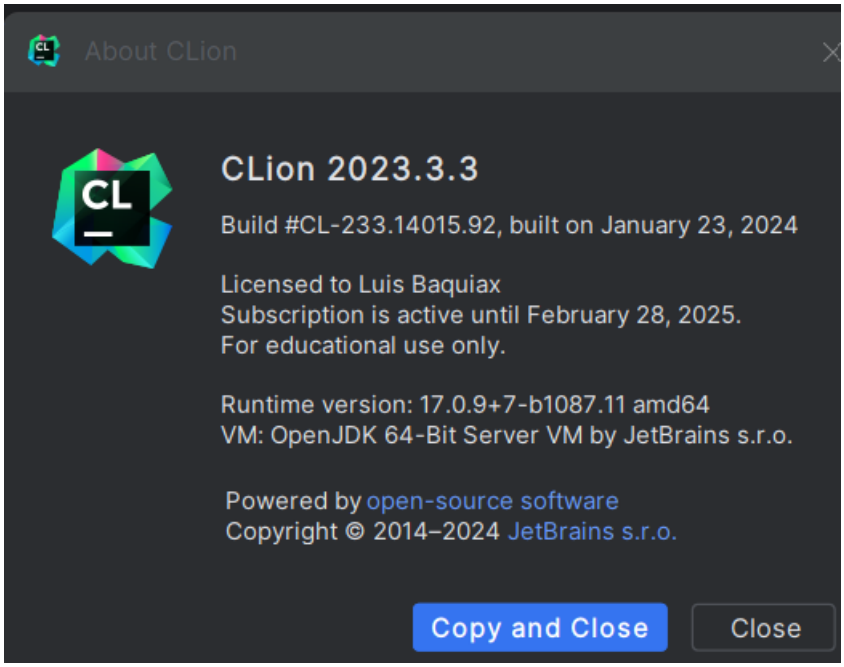
LUIS BASILIO BAQUIAX SIC
201430801
Curso: estructura de datos

Requerimientos del programa

1. Memoria RAM: al menos 2Gb
2. Sistema operativo recomendado: Windows 10

Tecnología y lenguaje usado

1. IDE: Clion



2. Lenguaje: c++

Métodos importantes

1. Creación de grupo:
   Este método se encarga de crear. Para crear un grupo el método busca en la tabla hashGruop si existe un grupo con el nombre que se le envía por parámetro creará el grupo y sino mostrará el mensaje que el grupo ya existe.

```
void ContanctManager::createGruop(std::string &nameGruop, Atributo **&listAtributos, int tam) {
    Atributo **list = listAtributos;

    TableHashAttributes *table = getTableAttributes(list, tam);

    hashGruoup.push(nameGruop, table);

    ItemHsGroup *itemHsGroup = hashGruoup.getItemGroup(nameGruop);
    if (itemHsGroup != nullptr) {
        auto **listadoNombres = new std::string *[tam];
        itemHsGroup->tamNamesAttributes = tam;
        //agregamos el nombre de los atributos
        for (int i = 0; i < tam; ++i) {
            listadoNombres[i] = &listAtributos[i]->valor;
        }
```

```
        hashGruoup.getItemGroup(nameGruop)->setListNames(listadoNombres);
        printf("Nombre de los atributos que se ingresan \n");
        for (int i = 0; i < hashGruoup.getItemGroup(nameGruop)->tamNamesAttributes; ++i) {
            std::cout << *hashGruoup.getItemGroup(nameGruop)->listNameAttributes[i] << std::endl;
        }
        printf("\n");
        Log *log = new Log(getFechaHora(), "Creacion del grupo " + nameGruop);
        listLog.insert(log);
    }
```

2. Creación de contactos:

Este método sirve para crear contactos, el cual busca el grupo correspondiente en la tabla hashGroup, después ingresa los atributos del contacto en el árbol correspondiente.

```
void ContanctManager::insertContact(std::string &nameGruop, Atributo **&listAtributos, int tam) {
    std::cout << "insertando en el grupo: " << nameGruop << std::endl;
    printf("\n");

    //enlazamos los nodos tanto su siguiente y su anterior
    for (int i = 0; i < tam - 1; ++i) {
        listAtributos[i]->next = listAtributos[i + 1];
        listAtributos[i + 1]->previous = listAtributos[i];
    }
    TableHashAttributes *tableAttribute = hashGruoup.getItemGroup(nameGruop)->tableAtributes;
    ItemHsGroup *itemHsGroup = hashGruoup.getItemGroup(nameGruop);

    if (tam == itemHsGroup->tamNamesAttributes) {

        printf("ingresando contacto... tamNamesAttributes: %d\n", itemHsGroup-
>tamNamesAttributes);
        for (int i = 0; i < tam; ++i) {
            //agregamos el grupo al que pertenece
            listAtributos[i]->nameGroup = nameGruop;
            //creamos un objeto Atributo para agregar al arbol correspondiente
            Atributo *nuevo = listAtributos[i];
            //imprimimos su informacion
            listAtributos[i]->printInfo();

            //agregamos los datos al arbol correspondiente
            controladorArbol.insertar(tableAttribute->getItemAttribute(listAtributos[i]->tipo)->tree,
nuevo);
            cantidadContactos++;
        }
        Log *log = new Log(getFechaHora(),
                           "Insercion de contacto en el grupo " + nameGruop);
        listLog.insert(log);
    } else {
        printf("No se puede ingresar el contacto.\n");
        printf("La lista de atributos nuevos no conicide con el tamanio del listado de atributos\n");
    }

}
```

3. Búsqueda de contactos

El método busca en el hashGroup el grupo y mediante el parámetro nameGruop. Recorre el árbol con el atributo buscado y los ingresa en una lista de contactos, por útlimo muestra toda la información de cada contacto.

```
void ContanctManager::searchContact(std::string &nameGruop, Atributo *&buscando) {
    List *listado = new List();

    ItemHsGroup *group = hashGruoup.getItemGroup(nameGruop);

    if (group != nullptr) {
        ItemHsAttributes *tabAttri = group->tableAtributes->getItemAttribute(buscando->tipo);
        if (tabAttri != nullptr) {
            std::cout << "Contactos encontrados con "
                      << buscando->tipo << " = "
                      << buscando->valor
                      << " del grupo: " << nameGruop << std::endl;
            Atributo *auxi = tabAttri->tree->raiz;
            if (auxi->valor == buscando->valor) {
                listado->addFinal(auxi);
            }
            searchContactRecursive(buscando, auxi, listado);
            Atributo *tem = listado->initial;
            while (tem != nullptr) {
                std::cout << tem->getInfoNextPrevious() << std::endl;
                tem = tem->nextList;
            }
            Log *log = new Log(getFechaHora(), "Busqueda de contacto en el grupo " + nameGruop);
            listLog.insert(log);
        } else {
            std::cout << "No existe datos con el parametro: "
                      << buscando->tipo << " = "
                      << buscando->valor << std::endl;
        }
    } else {
        std::cout << "No existe el grupo " << nameGruop << std::endl;
    }
}
```

4. Función hash:

Devuelve el valor número de la cadena que se le envía por parámetro, se hace uso de código ASCII.

```
int FuncionHash::valueHash(std::string &cadena) {
    int auxi = 0;

    const char *caracters = cadena.c_str();
    for (int i = 0; i < cadena.length(); ++i) {
        auxi += (int) (caracters[i]);
    }
    return auxi;
}
```

5. GentIndice:

Se encarga de regresar un entero mayor o igual que 0 el cual servirá para las tablasHash de los grupos y atributos de cada grupo.

```
int FuncionHash::getIndice(int value, int capacity) {
    return value % capacity;
}
```

Diagrama de clases

En la carpeta de este documento se adjunta una imagen donde se podrá observar mejor el diagrama de clases en formato png.