Manual Técnico

02 de abril del 2024

CONTACTS-MANAGER

AVLtree.h

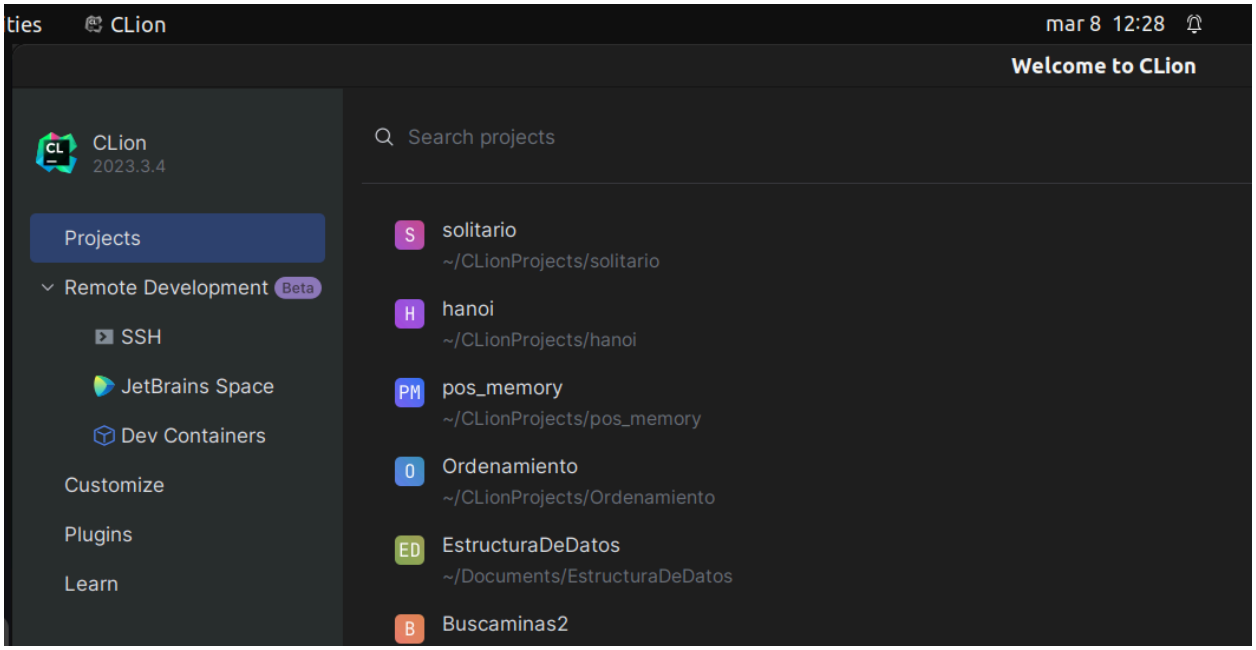
```
void balance(TreeNode<T>* &recentInserted){
    TreeNode<T>* currentNode = recentInserted;
    int oneStep = NOTHING;
    int twoStep = NOTHING;
    while (currentNode != nullptr){ //hasta llegar a la raiz
        currentNode->adjustWeight();
        if(!currentNode->isBalanced()){
            int weight = currentNode->getInclinationWeight();
            if(weight > 0 && (oneStep == RIGHT && twoStep == RIGHT)
                && currentNode->getRight() != nullptr && currentNode->getRight()->getRight() != nullptr){
                rotateLeftLeft( &currentNode);
            }else if(weight < 0 && (oneStep == LEFT && twoStep == LEFT)
                && currentNode->getLeft() != nullptr && currentNode->getLeft()->getLeft() != nullptr){
                rotateRightRight( &currentNode);
            }else if(weight < 0 && (oneStep == LEFT && twoStep == RIGHT)
                && currentNode->getLeft() != nullptr && currentNode->getLeft()->getRight() != nullptr){
                rotateLeftRight( &currentNode);
            }else if(weight > 0 && (oneStep == RIGHT && twoStep == LEFT)
                && currentNode->getRight() != nullptr && currentNode->getRight()->getLeft() != nullptr){
                rotateRightLeft( &currentNode);
            }else{
                std::cout<<"No se pudo balancear el arbol ERROR fatal"<<std::endl;
            }
        }
        twoStep = oneStep;
        oneStep = currentNode->isRightFromFather() ? RIGHT : LEFT;
        currentNode = currentNode->getFather();
    }
}
```

ÍNDICE

TECNOLOGÍA USADA

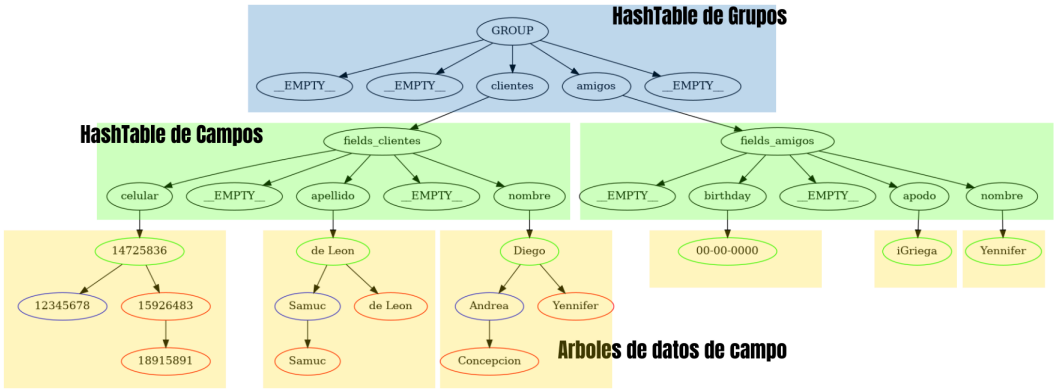
- Lenguaje de programación: C++ 17

- Ide: CLion2023.3.4



VISTA GENERAL DE ESTRUCTURAS DEL PROGRAMA

La estructura del programa de almacenamiento de datos es la siguiente:



Otra forma de visualización:

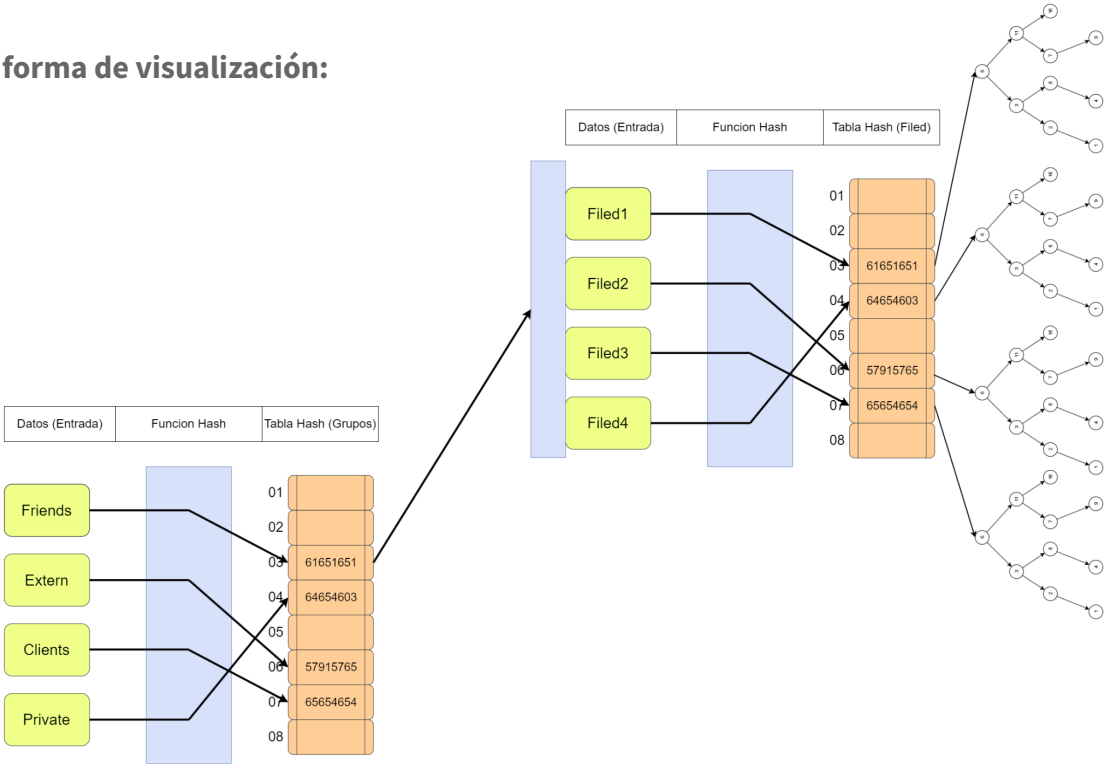
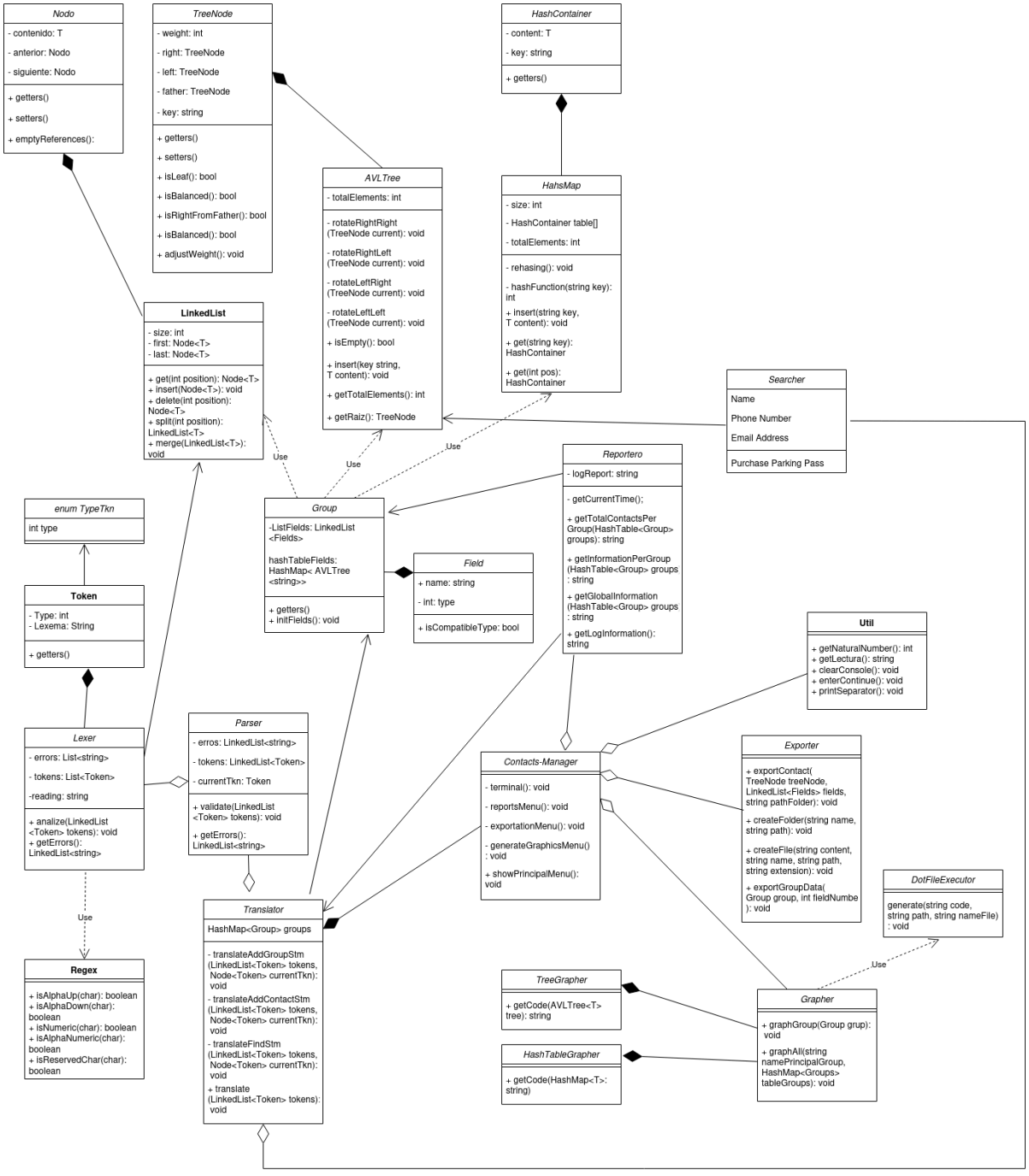


DIAGRAMA DE CLASES



USO DE ESTRUCTURAS

A continuación se especifican los métodos de las estructuras que usa el programa para funcionar.

HASH TABLE (HASH MAP)

ATRIBUTOS

Modificador de Acceso	Nombre	Descripción
private int	size	Tamaño del array de la tabla actual
private HashContainer<T>**	table	Puntero a un array de punteros que representa la tala actual
private int	totalElements	Número total de elementos ingresados en la tabla hash
private const int	INCREMENT	incremento a la tabla hash cuando ocurre el rehasing.

MÉTODOS PRIVADOS

Tipo de retorno	Nombre	Parámetros	Descripción
int	hashFunction	string* &key = *clave del valor* int sizeArray = *tamaño del array actual*	Devuelve un entero que representa un índice en una tabla hash, dependiendo de la llave ingresada y en el rango de 0 y sizeArray.
void	insertInternal	HashContainer<T>** &currentTable = *tabla en la cual se insertará el elemento* HashContainer<T>*	Hace una inserción en una tabla hash comprobando si no hay colisiones en el índice, si hay colisiones corre un espacio más.

		content = *contenido a insertar* int sizeTable = *tamaño de la tabla actual*	
void	rehashing		Muda los elementos del array actual a uno más grande, aplica el método insertInternal.

MÉTODOS PÚBLICOS

Tipo de retorno	**Nombre**	**Parámetros**	**Descripción**
Constructor	HashMap		Crea una nueva HashTable o HashMap
void	insert	string* key = *clave del valor a insertar* T* content = *contenido de tipo arbitrario a insertar*	Inserta un elemento en la tabla
HashContainer <T>*	get	string* key = *clave del valor a obtener*	Busca en la tabla hash un elemento y devuelve un contenedor, que contiene la clave y el contenido almacenado en la tabla. *Lanza invalid_argument si no se encuentra el elemento.*
HashContainer <T>*	get	int pos = *posicion en la tabla*	Devuelve la un contenedor (que tiene clave y contenido) de la posición especificada de la tabla *Lanza invalid_argument si la posición no está dentro de los límites permitidos.*

HashContainer <T>*	getFirstNonNull Data		Busca en la tabla hash el primer dato y lo devuelve como un contenedor. Lanza invalid_argument si la tabla está vacía.
int	getSize		Devuelve el tamaño actual de la tabla hash usada.
int	getTotalElements		Devuelve el total de elementos almacenados en la tabla hash.
void	showKeys		Muestra en consola todas las claves guardadas en la tabla hash.
void	showExistentKeys		Muestra en consola las claves guardadas en la tabla hash excluyendo las que no contienen nada

AVL TREE

ATRIBUTOS

Modificador de Acceso	Nombre	Descripción
private TreeNode<T>*	raíz	Es un puntero a la raíz del árbol.
private int	totalElements	Número total de elementos ingresados en el árbol.
private const int	NOTHING	Clave que indica que no se ha recorrido nada.
private const int	RIGHT	Clave que indica que se recorrió a la derecha.
private const int	LEFT	Clave que indica que se recorrió a la izquierda.

MÉTODOS PRIVADOS

Tipo de retorno	Nombre	Parámetros	Descripción
void	internal Insertion	TreeNode<T>* &newNode = *nodo a insertar*	Inserta en el lugar correspondiente según la clave del nodo dicho nodo en el arbol.
void	rotate RightRight	TreeNode<T>* &current = *nodo recien insertado en el arbol*	Balancea el árbol con un giro derecha-derecha
void	rotate RightLeft	TreeNode<T>* &current = *nodo recien insertado en el arbol*	Balancea el árbol con un giro derecha-izquierda
void	rotate LeftRight	TreeNode<T>* &current = *nodo recien insertado en el arbol*	Balancea el árbol con un giro derecha-izquierda
void	rotate LeftLeft	TreeNode<T>* &current = *nodo recien insertado en el arbol*	Balancea el árbol con un giro izquierda-izquierda
void	fixFather References	TreeNode<T>* &current = *nodo actual* TreeNode<T>* &newCurrent = *nuevo nodo que pasara en lugar del actual*	Arregla las referencias del padre de dos nodos
void	balance	TreeNode<T>* &recentInserted = *nodo recien insertado*	Balancea el árbol para que mantenga las características de árbol AVL

MÉTODOS PÚBLICOS

Tipo de retorno	Nombre	Parámetros	Descripción
Constructor	iAVLtree		Crea un nuevo arbolAVL vacío
bool	isEmpty		Devuelve *true* si el árbol está vacío *false* si no está vacío
void	insert	string* &key = *clave del árbol* T* &content = *contenido en el árbol*	Inserta un nuevo contenido en el árbol.
int	getTotal Elements		Devuelve el total de elementos contenidos en el árbol.
void	rotate LeftLeft	TreeNode<T>* &current = *nodo recien insertado en el arbol*	Balancea el árbol con un giro izquierda-izquierda
TreeNode <T>*	getRaiz		devuelve la raiz del arbol,

ACLARACIÓN CON EL USO DE PLANTILLAS

Todos los métodos que usan algún tipo genérico, se escribió todo el archivo .h esto es debido a que no se puede adjuntar en al archivo .cpp

Fuentes:

- https://es.stackoverflow.com/questions/47912/problema-al-compilar-plantillas-en-archivos-separados-en-c
- https://codingornot.com/cc-plantillas-templates-en-c

LENGUAJE DE LA TERMINAL

ESPECIFICACIONES PARA EL LEXER

Los tokens aceptados por el lexer son los siguiente:

- Palabras reservadas
    - ADD
    - NEW
    - GROUP
    - FIELD
    - FIELDS
    - STRING
    - INTEGER
    - DATE
    - CHAR
    - CONTACT
    - FIND
    - IN
- Símbolos
    - IGUAL =
    - GUION -
    - PARENTESIS_L (
    - PARENTESIS_R)
    - COMA ,
    - FIN_INSTRUCCION ;
- ID = [a-ZA-Z_]+
- ENTERO = [0-9]+
- CADENA = "."
- CARACTER = '.'

ESPECIFICACIONES PARA EL PARSER

La gramática para el lenguaje que usa el usuario para comunicarse con el programa es el siguiente:

```
/*símbolo inicial = instructions*/
type_name ::= STRING | INTEGER | DATE | CHAR
type ::= CADENA
           | ENTERO numeric_aux
           | CARACTER
numeric_aux ::= GUION INTEGER GUION INTEGER
           | /* empty */

data ::= types
           | IDENTIFICADOR

*instructions ::= ADD add_stm more_instructions
           | FIND CONTACT IN IDENTIFICADOR CONTACT GUION FIELD
          IDENTIFICADOR  IGUAL data FIN_INSTRUCCION more_instructions
more_instructions ::= instructions
           | /* empty */

add_stm ::= NEW GUION GROUP IDENTIFICADOR FIELDS PARENTESIS_L fields
             PARENTESIS_R FIN_INSTRUCCION
            |CONTACT IN IDENTIFICADOR FIELDS PARENTESIS_L insertable
             PARENTESIS_R FIN_INSTRUCCION

fields ::= IDENTIFICADOR type_name more_fields
more_fields ::= COMA fields
            | /*empty*/

insertable ::= data more_insertable
more_insertable ::= COMA insertable
            | /*empty*/
```

ALGORITMO DE FUNCIONES IMPORTANTES

MÉTODOS EN EL ÁRBOL AVL

INSERTAR()

```
si esta vacio
   hacer el nuevo nodo raiz
si no esta vacio
   INSERCION_INTERNA();
   VERIFICAR A PARTIR DEL ULTIMO NODO INGRESADO EL BALANCE();
```

INSERCION_INTERNA():

```
tener un auxiliar
hacer el auxiliar igual a raiz
hacer:
        verificar de que lado va el nuevo nodo a partir de una comprobacion con auxiliar
        si va al izquierdo
                VERIFICAR_NODO_A_INSERTAR(ladoIzquierdo)
        si va al derecho
                VERIFICAR_NODO_A_INSERTAR(ladoDerecho)
mientras(no encontrado el lugar de nuevo nodo)
```

VERIFICAR_NODO_A_INSERTAR(TreeNode* auxiliar, TreeNode* nuevo)

```
verificar si izq/der de auxiliar es nulo
  si es nulo
        //ingresando el nuevo nodo
        el padre del nuevo nodo es auxiliar
        el lado izq/der de auxiliar es el nuevo nodo
  si no es nulo
        auxiliar es der/izq de auxiliar //moviendo la referencia
        es necesario volver a verificar
```

MÉTODOS PARA BUSCAR EN EL ÁRBOL AVL

```
TreeNode FIND(TreeNode) //al principio sera la raiz
  si el nodo es diferente de null
        verificar si el elemento actual es menor que el nodo actual
        si es menor
              dirigirse al nodo izquierdo
              si el nodo izquierdo no es null
                    return FIND(nodo izquierdo);
              de lo contrario
                    no hacer nada, ahi no esta el elemento
        si es mayor
              dirigirse al nodo derecho
              si el nodo no es null
                    return FIND(nodo derecho)
              de lo contrario
                    no hacer nada, ahi no esta el elemento
        si es igual
              return elemento actual

        lanzar una excepción("elemento no encontrado");
  de lo contrario
        lanzar una excepción

/*devolvera todas las coincidencia en un arbol*/
string ENCONTRAR COINCIDENCIA RECURSIVA(TreeNode nodo)
  intentar:
        encontrar el primer nodo que coincida con la clave
        agregar la informacion del primer nodo a la informacion global
        si primer_nodo.izquierda != null
              informacion += ENCONTRAR_COINCIDENCIA_RECURSIVA();

        si el primer nodo.derecha != null
              informacion += ENCONTRAR_COINCIDENCIA_RECURSIVA();
  si no funciona:
        retornar "";
```

MÉTODOS EN EL TRADUCTOR

ADD_CONTACTS():
buscar el nombre del grupo y obtenerlo
si el nombre del grupo no existe
  lanzar una excepcion
de lo contrario
  otener la lista de campos que necesita dicho grupo

Inicializar una nueva lista enlazada de tokens
iniciar un cliclo que pare hasta que encuentre el parentesi de cierre
  guardar en la lista enlzada los tokens de campos que se encuentren por ahi
fin del ciclo

verificar si la lista de campos de contactos tiene el mismo size de la lista que se quiere insertar
si no concuerdan
  lanzar una excepcion
verificar si cada uno de los datos de la lista concuerda en tipo
si alguno no concuerda
  lanzar una excepcion

si todo esta bien
  obtener la hashTable con la key del grupo actual
  pasar cada lexema de la sublista de tokens a su arbol correspondiente, dependiendo del nombre del campo en orden

MÉTODOS PARA GRAFICAR

GENERAR_GRAFICA_COMPLETA()
Generar la grafica de grupos
para cada grupo i=1
  obtener el grupo i-1
  si el grupo no es vacio
    obtener del grupo, su lista de campos
    generar la grafica de campos

del nombre del grupo adjuntarle su lista de campos
de lo contrario
no hacer nada
fin para

GRAFICAR_HASH_TABLE()
crear el nodo padre
por cada hijo
crear el nodo hijos
conectar el nodo hijo actual con el nodo padre
fin para

CREAR_NODO(string nombre_label)
agregar al texto = nombre_objeto_actual + numbero_nodo
agregar al texto = [label]
fin

CONECTAR_NODOS(string nombre_padre, string nombre_hijo)
agregar al codigo = nombre_padre -> nombre_hijo;