Tecnológico Nacional de México

Campus Querétaro.

**Implementación de un árbol General**

*Alumno:*

*Arellano Ochoa Daniel Ignacio*

*Carrera:*

*Ingeniera en sistemas computacionales*

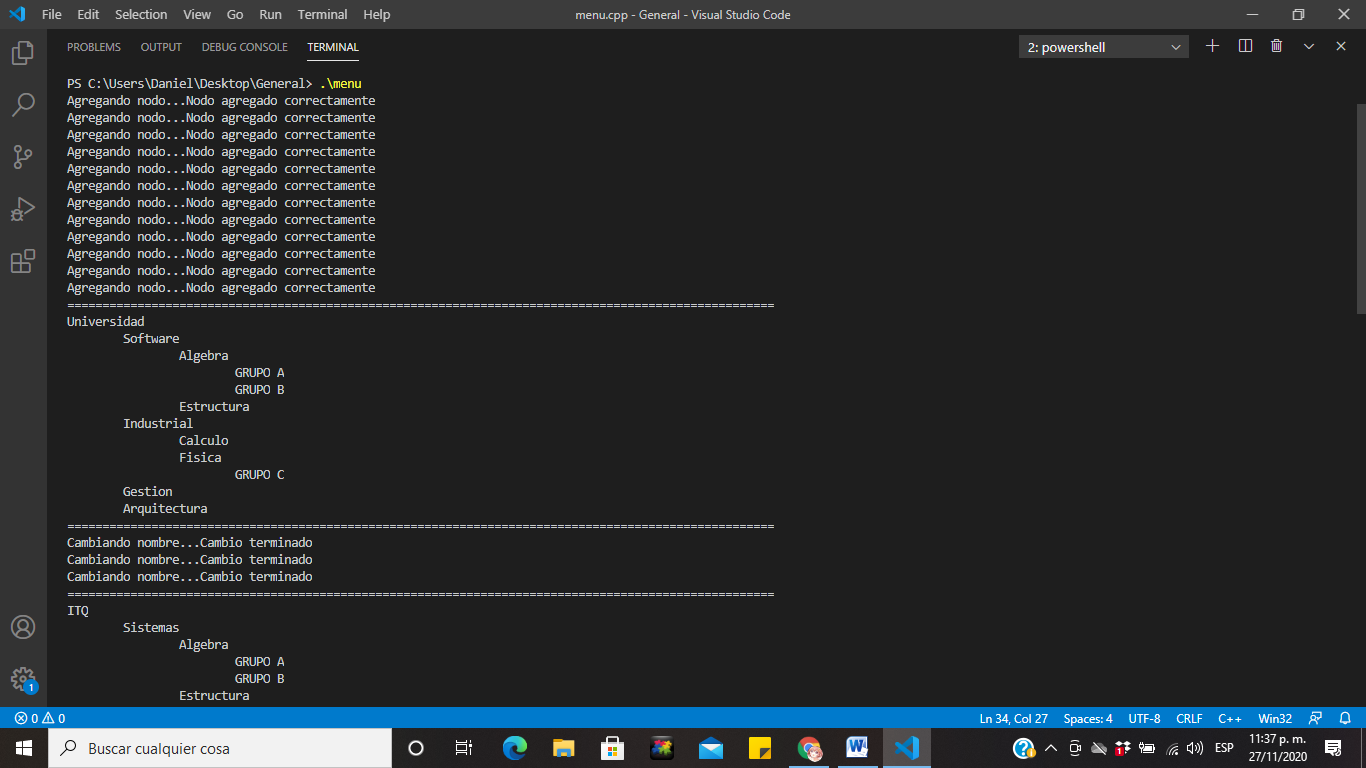
*Docente:*

*Felipe Estrada Rojas*

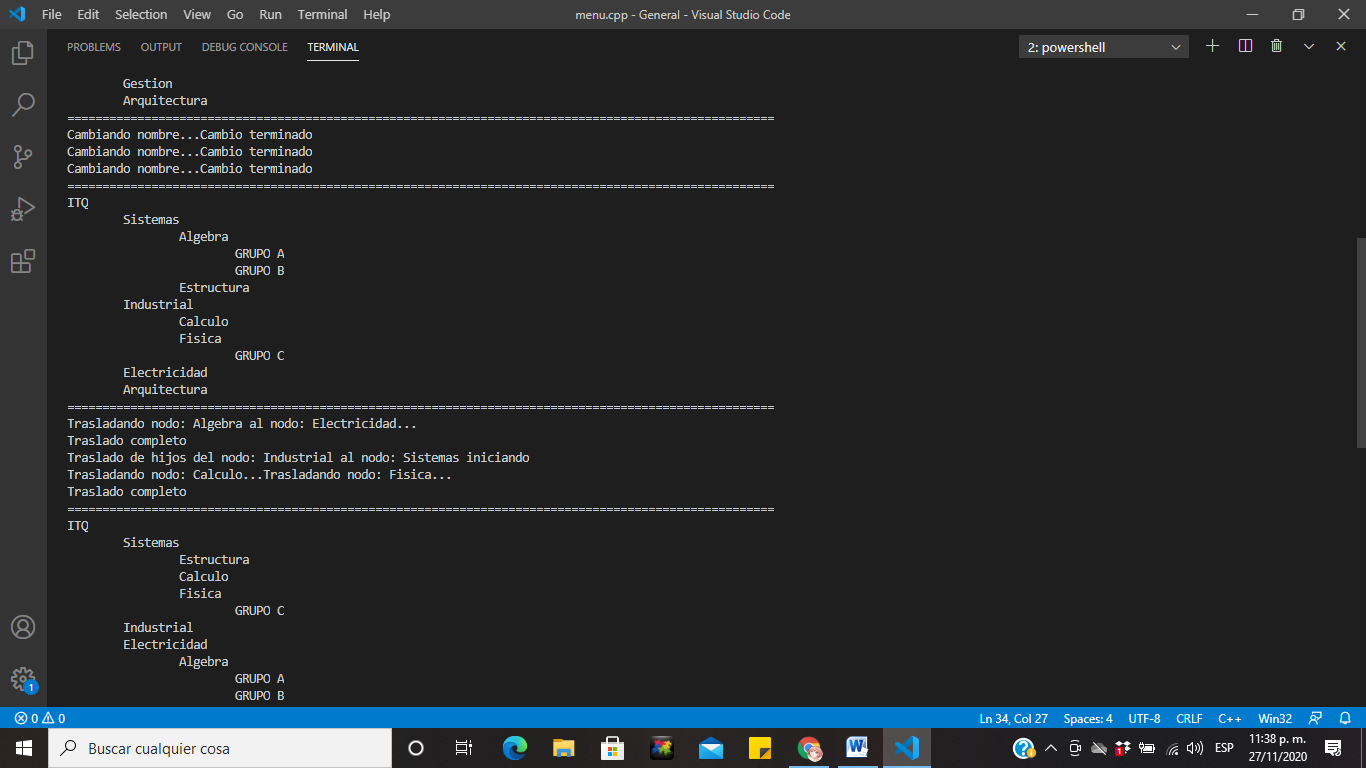
*Fecha: 27/11/2020*

**Ejecución:**

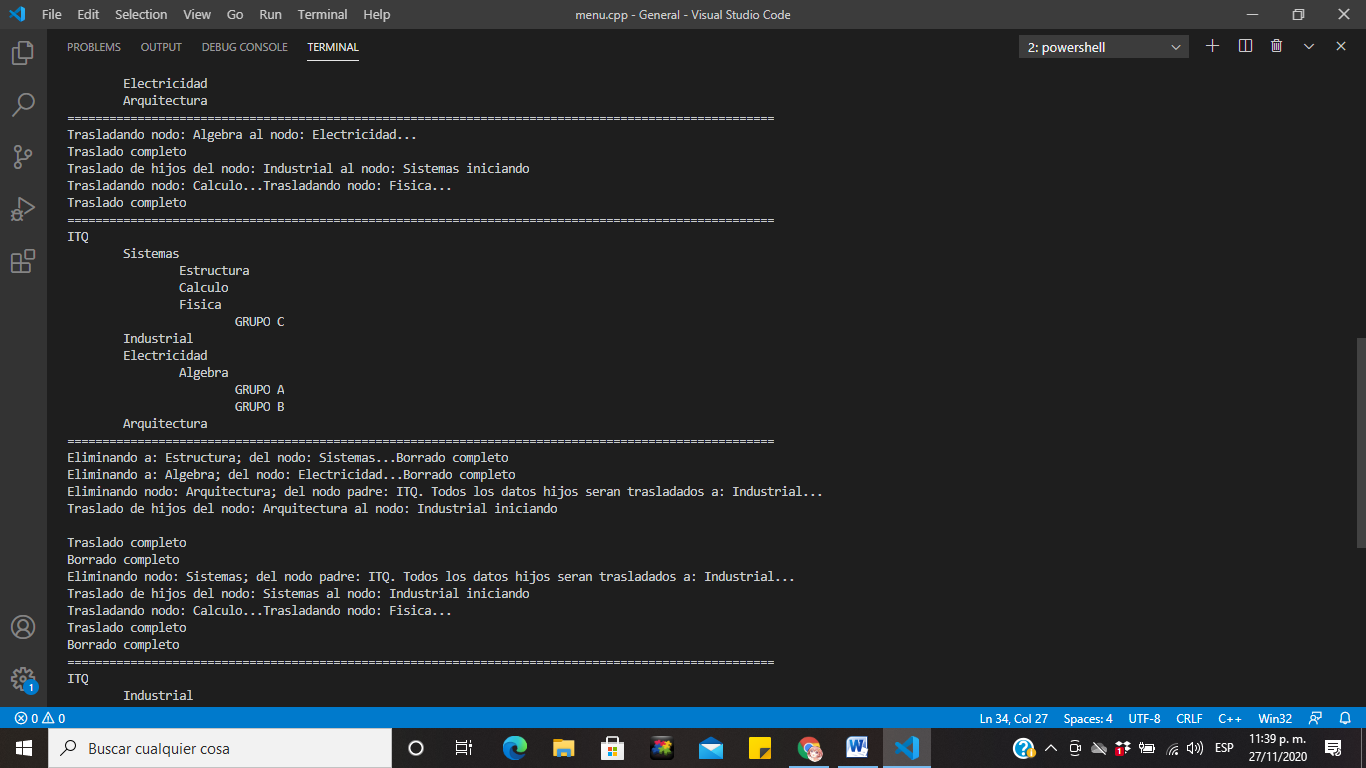
Se agregan los nodos:



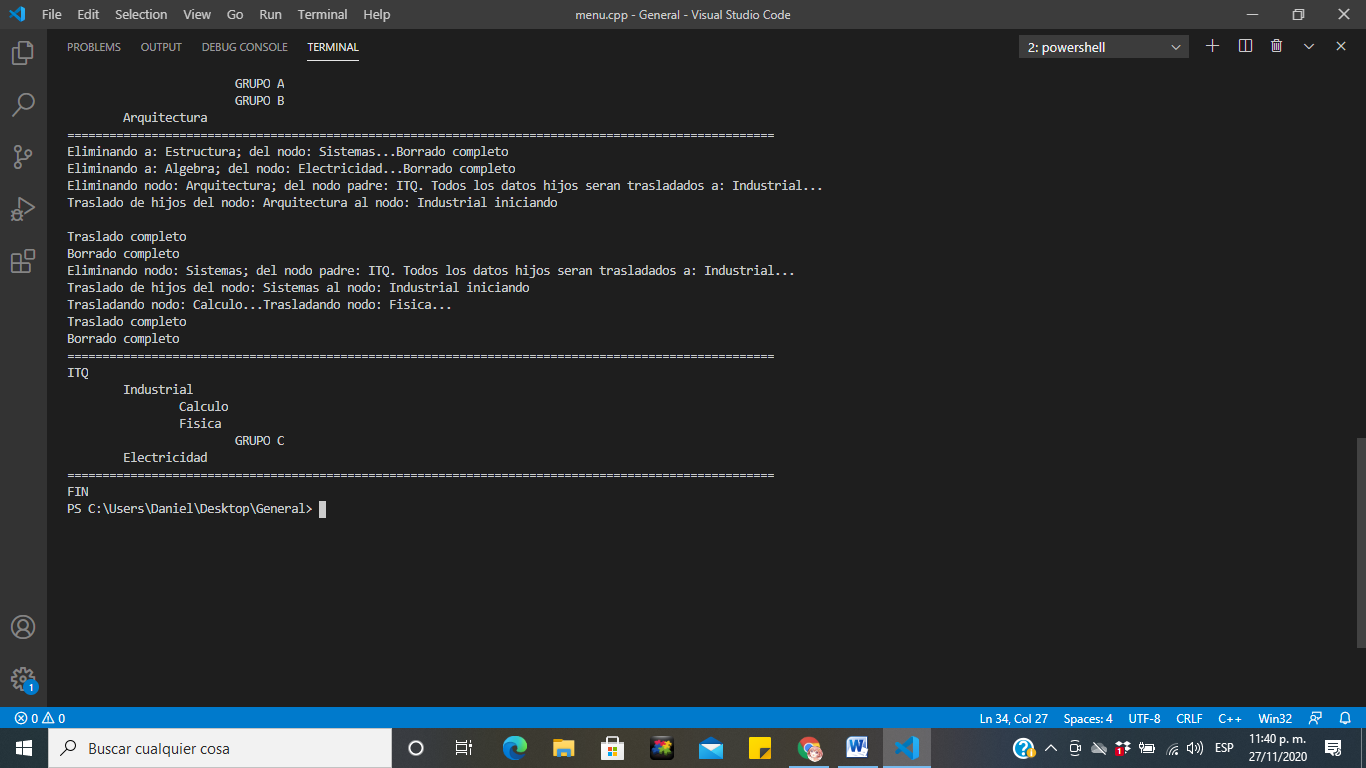
Se realiza cambios en los nombres de algunos nodos:



Se realizan algunos traslados de nodos:



Se realiza cuatro eliminación, las dos primeras eliminando al nodo con todo y sus hijos y las dos últimas realizando un traslado de hijos a otro nodo:



**Código:**

**Menu.cpp**

/\*Nombre: Arellano Ochoa Daniel Ignacio

No. de control: 19141118

Implementacion de un arbol general, incluyendo las bajas

Se tomo el ejemplo de como se jerarquiza una universidad

\*/

#include <string>

#include <iostream>

#include "Nodo.hpp"

#include "ArbolGeneral.hpp"

using namespace std;

//Menu de inicio del programa

int main(int argc, const char \* argv[]){

   ArbolGeneral\* arbol=new ArbolGeneral();

    //nivel 0

    arbol->agregarNodo("Universidad", "Universidad");

    //nivel 1

    arbol->agregarNodo("Software", "Universidad");

    arbol->agregarNodo("Industrial", "Universidad");

    arbol->agregarNodo("Gestion", "Universidad");

    arbol->agregarNodo("Arquitectura", "Universidad");

    //nivel 2

    arbol->agregarNodo("Algebra", "Software");

    arbol->agregarNodo("Estructura", "Software");

    arbol->agregarNodo("Calculo", "Industrial");

    arbol->agregarNodo("Fisica", "Industrial");

    //nivel 3

    arbol->agregarNodo("GRUPO A","Algebra");

    arbol->agregarNodo("GRUPO B","Algebra");

    arbol->agregarNodo("GRUPO C","Fisica");

    arbol->mostrar(arbol->getRaiz(), 1);

    //cambios: renombrar

    arbol->renombrarNodo("Software","Sistemas");//renombra software

    arbol->renombrarNodo("Universidad", "ITQ");//renombra Universisdad

    arbol->renombrarNodo("Gestion", "Electricidad");//renombre gestion

    arbol->mostrar(arbol->getRaiz(), 1);

    //cambios: traslados

    arbol->trasladarHijo("Algebra", "Electricidad");//el nodo algebra pasara a ingeneria en electricidad

    arbol->trasladarHijos("Industrial", "Sistemas");//todos los hijos de industrial pasaran a sistemas

    arbol->mostrar(arbol->getRaiz(), 1);

    //cambios: eliminacion

    arbol->eliminarNodo("Estructura");//eliminacion de un nodo sin hijos sin traslado

    arbol->eliminarNodo("Algebra");//eliminacion de un nodo con hijos sin traslado

    arbol->elimnarNodo("Arquitectura","Industrial");//eliminacion de un nodo sin hijos con traslado

    arbol->elimnarNodo("Sistemas","Industrial");//eliminacion de un nodo con hijos con traslado

    arbol->mostrar(arbol->getRaiz(), 1);

    cout<<"FIN"<<endl;

}

**Nodo.hpp**

#ifndef Nodo\_hpp

#define Nodo\_hpp

#include <iostream>

#include <string>

#include <list>

using namespace std;

class Nodo

{

private:

    string dato; //el dato que alamecena el nodo

    list<Nodo\*> hijos; //lista de los hijos

public:

    Nodo(string);//constructor

    string getDato();//metodo para devolver el dato

    void setDato(string);//metodo para cambiar el dato

    list<Nodo\*>\* getHijos();//metodo que regresa un apuntador de la lista de sus hijos

    void agregarHijo(Nodo\*);//metodo para agregar un hijo a la lista

    void eliminarHijo(Nodo\*);//metodo para eliminar un hijo de la lista

};

#endif /\* Nodo\_hpp\*/

**Nodo.cpp**

#include "Nodo.hpp"

#include <iostream>

#include <string>

#include <list>

using namespace std;

//constructor, se inicializa el dato, la lista de hijos inicia con cero datos almacenados

Nodo::Nodo(string n){

    this->dato=n;

}

string Nodo::getDato(){return this->dato;};//metodo que regresa el dato

void Nodo::setDato(string n){this->dato=n;}//metodo que cambia el dato

//metodo que regresa un apuntador de la lista de los hijos del nodo

list<Nodo\*>\* Nodo::getHijos(){

    list<Nodo\*>\* auxH=&this->hijos; //apuntador que apunta a la lista de los hijos del nodo

    return auxH;

};

//metodo para agregar un nuevo hijo

void Nodo::agregarHijo(Nodo\* a){

    this->hijos.push\_back(a);//se agregara al final de la lista

}

//metodo que elimina un hijo de la lista de hijos

void Nodo::eliminarHijo(Nodo\* objetivo){

    list<Nodo\*>::iterator iterador;//iterador que ayudara a encontrar el nodo hijo para despues eliminarlo

    iterador=this->hijos.begin();

    while (iterador != this->hijos.end()){

        if(\*iterador==objetivo){

            this->hijos.erase(iterador);

            break;

        }

        iterador++;

    }

}

**ArbolGeneral.hpp**

#ifndef ArbolGeneral\_hpp

#define ArbolGeneral\_hpp

#include "Nodo.hpp"

#include <iostream>

#include <string>

using namespace std;

class ArbolGeneral

{

private:

    Nodo\* raiz;//nodo principal del arbol (raiz del arbol)

public:

    ArbolGeneral();//cosntructor

    Nodo\* getRaiz();//Regresa la raiz

    void setRaiz(Nodo\*);//(nueva raiz)Cambia la raiz

    void renombrarNodo(string, string);//(nombre actual, nombre nuevo)-->Renombra/cambia el dato del nodo

    void agregarNodo(string, string);//(hijo, padre)-->Agrega un nodo hijo a un nodo padre

    void elimnarNodo(string, string);//(nodo a eliminar, nuevo padre)-->Elimina un nodo, pasando sus hijos a otro nodo

    void eliminarNodo(string);//(nodo a eliminar) -->Elimina un nodo con todo y sus hijos

    void trasladarHijo(string, string);//(hijo a trasladar, nuevo padre)-->Traslada un solo hijo a otro nodo

    void trasladarHijos(string, string);//(padre actual, padre nuevo)-->Traslada todos los hijos de un nodo a otro

    void mostrar(Nodo\*, int);//(raiz, nivel)-->Imprime al árbol, a partir del nodo raiz, el nivel es para visualizar mejor el arbol

    Nodo\* buscar(string, Nodo\*);//(objetivo a buscar, raiz)-->Busca un nodo desde la raiz

    Nodo\* buscarPadre(string, Nodo\*);//(hijo, raiz)-->Busca el nodo padre del hijo

};

#endif /\*ArbolGeneral\_hpp\*/

**ArbolGeneral.cpp**

#include "ArbolGeneral.hpp"

#include "Nodo.hpp"

#include <iostream>

#include <string>

using namespace std;

//constructor

ArbolGeneral::ArbolGeneral(){

    this->raiz=NULL;//se inicializa la raiz con NULL

}

Nodo\* ArbolGeneral::getRaiz(){return this->raiz;}//Regresa la raiz

void ArbolGeneral::setRaiz(Nodo\* a){this->raiz=a;}//(nueva raiz)Cambia la raiz

//(nombre actual, nombre nuevo)-->Renombra/cambia el dato del nodo

void ArbolGeneral::renombrarNodo(string actual, string nuevo){

    cout<<"Cambiando nombre...";

    Nodo\* aux;//nodo auxiliar

    aux=buscar(actual, this->raiz);

    if(aux != NULL){

        aux->setDato(nuevo);

        cout<<"Cambio terminado"<<endl;

    }else{

        cout<<"No existe el nodo "<<actual<<" por lo tanto no es posible cambiarle de nombre"<<endl;

    }

}

//(hijo, padre)-->Agrega un nodo hijo a un nodo padre

void ArbolGeneral::agregarNodo(string hijo, string padre){

    Nodo\* nuevoNodo;//nuevo nodo

    if(this->raiz==NULL){

        cout<<"Agregando nodo...";

        nuevoNodo= new Nodo(hijo);

        this->raiz=nuevoNodo;

        cout<<"Nodo agregado correctamente"<<endl;

    }else{

        Nodo\* padreNodo= buscar(padre, this->raiz);//nodo auxiliar del nodo padre

        if(padreNodo != NULL){

            cout<<"Agregando nodo...";

            nuevoNodo=new Nodo(hijo);

            padreNodo->agregarHijo(nuevoNodo);

            cout<<"Nodo agregado correctamente"<<endl;

        }else{

            cout<<"No existe el padre: "<<padre<<". Por lo que no puede dar de alta a: "<<hijo<<endl;

        }

    }

}

//(nodo a eliminar, nuevo padre)-->Elimina un nodo, pasando sus hijos a otro nodo

void ArbolGeneral::elimnarNodo(string objetivo, string nuevoPadre){

    Nodo\* nodoEliminar;//nodo auxiliar del nodo que sera eliminado

    nodoEliminar=buscar(objetivo, this->raiz);

    Nodo\* nodoPadre;//nodo auxiliar del padre del nodo a eliminar

    nodoPadre=buscarPadre(objetivo, this->raiz);

    Nodo\* padreNuevo;//nodo auxiliar del padre nuevo de los hijos del nodo a eliminar

    padreNuevo=buscar(nuevoPadre, this->raiz);

    if(nodoEliminar != NULL && padreNuevo != NULL && nodoEliminar != this->raiz){

        cout<<"Eliminando nodo: "<< nodoEliminar->getDato()<<"; del nodo padre: "<<nodoPadre->getDato();

        cout<<". Todos los datos hijos seran trasladados a: "<<padreNuevo->getDato()<<"..."<<endl;

        trasladarHijos(nodoEliminar->getDato(), padreNuevo->getDato());

        nodoPadre->eliminarHijo(nodoEliminar);

        cout<<"Borrado completo"<<endl;

    }else{

        if(nodoEliminar == NULL)

            cout<<"El nodo "<<objetivo<<" no existe, por lo tanto no puede eliminarse"<<endl;

        else if(padreNuevo==NULL)

            cout<<"El nodo "<<nuevoPadre<<" no existe, por lo tanto no se pueden trasladar los hijos de "<<objetivo<<endl;

        else

            cout<<"El nodo "<<objetivo<<" es la raiz principal del arbol, por lo tanto no puede eliminarse"<<endl;

    }

}

//(nodo a eliminar) -->Elimina un nodo con todo y sus hijos

void ArbolGeneral::eliminarNodo(string objetivo){

    Nodo\* nodoEliminar;//nodo auxiliar del nodo que sera eliminado

    nodoEliminar=buscar(objetivo, this->raiz);

    Nodo\* nodoPadre;//nodo auxiliar del padre del nodo a eliminar

    nodoPadre=buscarPadre(objetivo, this->raiz);

    if(nodoEliminar != NULL && nodoEliminar != this->raiz){

        cout<<"Eliminando a: "<<nodoEliminar->getDato()<<"; del nodo: "<<nodoPadre->getDato()<<"...";

        nodoPadre->eliminarHijo(nodoEliminar);

        cout<<"Borrado completo"<<endl;

    }else{

        if(nodoEliminar==NULL)

            cout<<"El nodo "<<objetivo<<" no existe, por lo tanto no puede eliminarse"<<endl;

        else

            cout<<"El nodo "<<objetivo<<" es la raiz principal del arbol, por lo tanto no puede eliminarse"<<endl;

    }

}

//(hijo a trasladar, nuevo padre)-->Traslada un solo hijo a otro nodo

void ArbolGeneral::trasladarHijo(string hijo, string nuevoPadre){

    Nodo\* auxHijo;//nodo auxiliar del nodo a trasladar

    auxHijo=buscar(hijo, this->raiz);

    Nodo\* auxActualPadre;//nodo auxiliar del padre actual del nodo a trasladar

    auxActualPadre=buscarPadre(hijo, this->raiz);

    Nodo\* auxNuevoPadre;//nodo auxiliar del padre nuevo

    auxNuevoPadre=buscar(nuevoPadre, this->raiz);

    if(auxHijo != NULL || auxNuevoPadre!=NULL || auxActualPadre!=NULL){

        cout<<"Trasladando nodo: "<<auxHijo->getDato()<<" al nodo: "<<auxNuevoPadre->getDato()<<"...";

        auxNuevoPadre->agregarHijo(auxHijo);

        auxActualPadre->eliminarHijo(auxHijo);

        cout<<endl<<"Traslado completo"<<endl;

    }else{

        if(auxHijo==NULL)

            cout<<"El nodo "<<hijo<<" no existe, por lo tanto no puede trasladarse"<<endl;

        else if(auxNuevoPadre==NULL)

            cout<<"El nodo "<<nuevoPadre<<" no existe, por lo tanto no se puede trasladar el nodo "<<hijo<<endl;

        else

            cout<<"Intenta trasladar el nodo raiz principal del arbol, eso no es posible"<<endl;

    }

}

//(padre actual, padre nuevo)-->Traslada todos los hijos de un nodo a otro

void ArbolGeneral::trasladarHijos(string actualPadre, string nuevoPadre){

    Nodo\* auxActualPadre;//nodo auxiliar del padre actual

    auxActualPadre=buscar(actualPadre, this->raiz);

    Nodo\* auxNuevoPadre;//nodo auxiliar del nuevo padre

    auxNuevoPadre=buscar(nuevoPadre, this->raiz);

    if(auxActualPadre!=NULL || auxNuevoPadre!=NULL){

        list<Nodo\*>\* listahuerfana;//apuntador auxiliar que apunta a la lista de los hijos que seran trasladados

        listahuerfana=auxActualPadre->getHijos();

        list<Nodo\*>::iterator iterador;//iterador que ayudara a recorrer la lista

        iterador=listahuerfana->begin();

        Nodo\* aux;//nodo auxiliar para pasar cada nodo

        cout<<"Traslado de hijos del nodo: "<<auxActualPadre->getDato()<<" al nodo: "<<auxNuevoPadre->getDato()<<" iniciando"<<endl;

        while (iterador != listahuerfana->end()){

            aux=\*iterador;

            cout<<"Trasladando nodo: "<<aux->getDato()<<"...";

            auxNuevoPadre->agregarHijo(aux);

            iterador++;

        }

        auxActualPadre->getHijos()->clear();

        cout<<endl<<"Traslado completo"<<endl;

    }else{

        if(auxActualPadre==NULL)

            cout<<"El nodo "<<actualPadre<<" no existe, por lo tanto no tiene hijos que trasladar"<<endl;

        else

            cout<<"El nodo "<<nuevoPadre<<" no existe, por lo tanto no hay donde trasladar los hijos de "<<actualPadre<<endl;

    }

}

//(raiz, nivel)-->Imprime al árbol, a partir del nodo raiz, el nivel es para dar n tabulaciones y se observe mejor los niveles del arbol

void ArbolGeneral::mostrar(Nodo\* auxN, int nivel){

    if(nivel==1){

        cout<<"====================================================================================================="<<endl;

    }

    if(auxN != NULL){

        cout<<auxN->getDato()<<endl;

        list<Nodo\*>::iterator ite;//iterador que ayudara a recorrer el arbol

        ite= auxN->getHijos()->begin();

        Nodo\* aux;//nodo auxiliar

        while (ite != auxN->getHijos()->end()){

            for (size\_t i = 0; i < nivel; i++)

            {

                cout<<"\t";

            }

            mostrar(\*ite, nivel+1);

            ite++;

        }

    }

    if(nivel==1){

        cout<<"====================================================================================================="<<endl;

    }

}

//(objetivo a buscar, raiz)-->Busca un nodo desde la raiz

Nodo\* ArbolGeneral::buscar(string bus, Nodo\* n){

    if(n !=NULL){//verifica la existencia del nodo n

        if(n->getDato()==bus){//verifica si no es el que se busca

            return n;

        }else{

            if(n->getHijos()->size()==0){//verifica si el nodo n tiene hijos

                return NULL;

            }else{

                list<Nodo\*>::iterator iter;//iterador que ayudara a recorrer la lista de los hijos de n

                iter=n->getHijos()->begin();

                Nodo\* aux;//nodo auxiliar para tomar los datos de cada hijo

                //analiza cada hijo de n

                while (iter != n->getHijos()->end()){

                    aux=\*iter;

                    if(aux->getDato()==bus){//verifica si un hijo de n es el que se busca

                        return aux;

                    }else{

                        iter++;

                    }

                }

                Nodo\* aux2;//nodo auxiliar

                iter=n->getHijos()->begin();

                //analiza los hijos de cada hijo de n (nietos de n)

                while (iter != n->getHijos()->end()){

                    aux=\*iter;

                    if(aux->getHijos()->size()==0){//verifica si el hijo de n tiene hijos

                        iter++;

                    }else{

                        aux2=buscar(bus, aux);//caso recursivo

                        if(aux2==NULL){//verifica si aux2 es un dato NULL

                            iter++;

                        }else{

                            return aux2;

                        }

                    }

                }

                return NULL;

            }

        }

    }else{

        return NULL;

    }

}

//(hijo, raiz)-->Busca el nodo padre del hijo

//es el mismo codigo del metodo "buscar",

//con la unica diferencia que en vez de regresar el nodo que contenga el valor a buscar, regresa a su padre

Nodo\* ArbolGeneral::buscarPadre(string bus, Nodo\* n){

    if(n !=NULL){

        if(n->getDato()==bus){

            return n;

        }else{

            if(n->getHijos()->size()==0){

                return NULL;

            }else{

                list<Nodo\*>::iterator iter;//iterador que ayudara a recorrer la lista de los hijos de n

                iter=n->getHijos()->begin();

                Nodo\* aux;//nodo auxiliar

                //analiza cada hijo de n

                while (iter != n->getHijos()->end()){

                    aux=\*iter;

                    if(aux->getDato()==bus){

                        return n;

                    }else{

                        iter++;

                    }

                }

                Nodo\* aux2;//segundo nodo auxiliar

                iter=n->getHijos()->begin();

                //analiza los nietos de n

                while (iter != n->getHijos()->end()){

                    aux=\*iter;

                    if(aux->getHijos()->size()==0){

                        iter++;

                    }else{

                        aux2=buscarPadre(bus, aux);//caso recursivo

                        if(aux2==NULL){

                            iter++;

                        }else{

                            return aux2;

                        }

                    }

                }

                return NULL;

            }

        }

    }else{

        return NULL;

    }

}