Pila

struct nodoPila{

int dato;

nodoPila\* link = nullptr;

};

//Pila vacia

bool isEmpty(nodoPila\* pila){

if (pila == nullptr)

return true;

return false;

}

//Agregar nodos a la pila

void Pila\_push(nodoPila\* &pila, int dato){

nodoPila\* nuevoNodo = new nodoPila;

nuevoNodo->dato = dato;

if (pila == nullptr)

pila = nuevoNodo;

else{

nuevoNodo->link = pila;

pila = nuevoNodo;

}

}

// Obtiene un nodo de la pila

int Pila\_pop(nodoPila\* &pila){

if (isEmpty(pila))

return 0;

int dato = pila->dato;

nodoPila\* nodoAremover = pila;

pila = pila->link;

delete nodoAremover;

return dato;

}

Cola

using namespace std;

struct nodoCola{

nodoCola\* next;

int dato;

};

Cola vacia

bool ColaVacia(nodoCola\* frente){

if (frente == nullptr)

return true;

return false;

}

// Agrega nodos a la cola

void add(nodoCola\* &frente, nodoCola\* &fondo, int nuevoDato){

nodoCola\* nuevoNodo = new nodoCola;

nuevoNodo->dato = nuevoDato;

if (frente == NULL)

frente = fondo = nuevoNodo;

else{

fondo->next = nuevoNodo;

fondo = nuevoNodo;

}

}

// Obtiene un nodo de la cola

int get(nodoCola\* &frente, nodoCola\* &fondo){

nodoCola\* aux = frente;

frente = frente->next;

int dato = aux->dato;

if (aux == fondo)

fondo = NULL;\

delete aux;

return dato;

}

Lista SE

struct nodo\_listaSE{

int dato;

nodo\_listaSE\* link;

};

//Agregar dato

void agregar\_dato(nodo\_listaSE\* &inicio, int dato\_agregar){

nodo\_listaSE\* aux = new nodo\_listaSE;

aux->dato = dato\_agregar;

aux->link = inicio;

inicio = aux;

}

//Mostrar lista

void listaSE\_mostrar(nodo\_listaSE\* inicio){

while (inicio != NULL)

{

cout << inicio->dato << " -> ";

inicio = inicio->link;

}

cout << "NULL" << endl;

}

//Agregar dato al final

void agregar\_al\_final(nodo\_listaSE\* listase, int ndato){

nodo\_listaSE\* nuevo\_nodo = new nodo\_listaSE;

nuevo\_nodo->dato = ndato;

nuevo\_nodo->link = NULL;

if (listase == NULL) {

listase = nuevo\_nodo;

} else {

nodo\_listaSE\* ultimo = listase;

while (ultimo->link != NULL) {

ultimo = ultimo->link;

}

ultimo->link = nuevo\_nodo;

}

};

//Generar datos aleatorios

void datos\_aleatorios(nodo\_listaSE\* &listase){

int datos;

cout << "Ingrese la cantidad de datos que desea generar: ";

cin >> datos;

for (int i = 0; i < datos; i++)

{

int number = rand()%100 + 1;

agregar\_dato(listase,number);

cout << number << " ";

}

cout << endl;

}

//Eliminar ocurrencia

void eliminar\_ocurrencias(nodo\_listaSE\* &listase, int ndato){

nodo\_listaSE\* anterior = NULL;

nodo\_listaSE\* actual = listase;

int cant\_ocurrencias = 0;

while (actual != NULL){

if(actual->dato == ndato){

if(anterior == NULL){

listase = actual->link;

}

else anterior->link = actual->link;

nodo\_listaSE\* eliminar = actual;

actual = actual->link;

delete eliminar;

cant\_ocurrencias++;

}

else{

anterior = actual;

actual = actual->link;

}

}

cout << "Se han eliminado " << cant\_ocurrencias << " ocurrencias" << endl;

};

//Eliminar primer dato

bool eliminar\_primer\_dato(nodo\_listaSE\* &listase){

if (listase != NULL)

{

nodo\_listaSE\* aux = listase;

listase = listase->link;

delete aux;

return true;

}

return false;

}

//Inserta ordenado

void insertar\_ordenado(nodo\_listaSE\* &listase, int dato\_insertar){

//supone que la lista esta ordenada en forma ascendente y agrega el dato

nodo\_listaSE\* actual = listase;

nodo\_listaSE\* anterior = NULL;

nodo\_listaSE\* nuevo\_dato = new nodo\_listaSE;

nuevo\_dato->link = NULL;

nuevo\_dato->dato = dato\_insertar;

if (listase == NULL || listase->dato > dato\_insertar) {

nuevo\_dato->link = listase;

listase = nuevo\_dato;

}

else{

while (actual != NULL && actual->dato < dato\_insertar)

{

anterior = actual;

actual = actual->link;

}

}

nuevo\_dato->link = actual;

anterior->link = nuevo\_dato;

}

void eliminar\_lista(nodo\_listaSE\* &listase){

nodo\_listaSE\* aux = listase;

while (listase != NULL)

{

listase = listase->link;

delete aux;

aux = listase;

}

cout << "Se ha eliminado la lista completa" << endl;

}

Lista DE

struct nodeDE{

// data

int data;

// pointers to move around

nodeDE\* next = NULL;

nodeDE\* back = NULL;

};

//Mostar lista

void MostrarLista(nodeDE\* lista){

while(lista != NULL){

cout << lista->data << " ";

lista = lista->next;

}

}

// Insertar sin orden

void insert(nodeDE\* &lista, int nuevoDato){

nodeDE\* newNode = new nodeDE;

newNode->data = nuevoDato;

if (lista == NULL)

lista = newNode;

else{

newNode->next = lista;

lista->back = newNode;

lista = newNode;

}

}

// Insertar con orden

void sorted\_insert(nodeDE\* &lista, int nuevoDato){

nodeDE\* newNode = new nodeDE;

newNode->data = nuevoDato;

if (lista == NULL || nuevoDato < lista->data){

newNode->next = lista;

if (lista != NULL)

lista->back = newNode;

lista = newNode;

} else {

nodeDE\* aux = lista;

while(aux->next != NULL && nuevoDato > aux->next->data)

aux = aux->next;

newNode->next = aux->next;

newNode->back = aux;

if (aux->next != NULL)

aux->next->back = newNode;

aux->next = newNode;

}

}

// Buscar un dato

bool lookfor(int data, nodeDE\* lista, nodeDE\* &node){

while(lista->next != NULL){

if (lista->data == data){

node = lista;

return true;

}

lista = lista->next;

}

return false;

}

// Eliminar un nodo

bool delete\_node(int data, nodeDE\* &lista){

nodeDE\* node2delete;

if (lookfor(data, lista, node2delete)){

// Si es el primer nodo

if (node2delete == lista)

lista = lista->next;

// Si es un nodo que esta de por medio

else if (node2delete->back != NULL && node2delete->next != NULL){

node2delete->back->next = node2delete->next;

node2delete->next->back = node2delete->back;

// Si es un nodo que esta al final

} else if (node2delete->back != NULL)

node2delete->back->next = node2delete->next;

delete node2delete;

return true;

}

return false;

}

// Eliminar la lista

void deleteList(nodeDE\* &lista){

nodeDE\* aux;

while (lista != NULL){

aux = lista;

lista = lista->next;

delete aux;

}

}

// Ordenar lista

void sortList(nodeDE\* &lista){

nodeDE\* aux = lista;

nodeDE\* newlist = NULL;

while(aux != NULL){

sorted\_insert(newlist, aux->data);

aux = aux->next;

}

deleteList(lista);

lista = newlist;

}

Arboles

struct nodeArbol{

int dato;

nodeArbol\* izq = nullptr;

nodeArbol\* der = nullptr;

};

// ## FUNCIONES PARA INSERTAR NODOS ##

// Inserta nodos y es capaz de reacomodar el arbol en caso de organizarlo mal (no me gusta como quedo)

void BTinsert\_V2(nodeArbol\* &arbol, int nuevoDato){

nodeArbol\* newnode = new nodeArbol;

newnode->dato = nuevoDato;

if (arbol == nullptr)

arbol = newnode;

else if (arbol->izq != nullptr && (arbol->izq->dato < nuevoDato && arbol->dato > nuevoDato)){

nodeArbol\* aux = arbol;

arbol = newnode;

arbol->izq = aux->izq;

arbol->der = aux;

aux->izq = nullptr;

}

else if (nuevoDato < arbol->dato)

BTinsert\_V2(arbol->izq, nuevoDato);

else if (nuevoDato > arbol->dato)

BTinsert\_V2(arbol->der, nuevoDato);

}

// Inserta nodos en el arbol

void BTinsert(nodeArbol\* &arbol, int data){

if (arbol == nullptr){

nodeArbol\* node = new nodeArbol;

node->dato = data;

arbol = node;

} else if (data < arbol->dato)

BTinsert(arbol->izq, data);

else if (data > arbol->dato)

BTinsert(arbol->der, data);

}

// ## FUNCIONES DE BUSQUEDA ##

// Busca un dato en el arbol binario y avisa si lo encontro, ademas de devolver el nodo

bool BTlook(nodeArbol\* arbol, int data, nodeArbol\* &node){

if (arbol != nullptr){

node = arbol;

while (node != nullptr){

if (node->dato == data)

return true;

if (data < node->dato)

node = node->izq;

else node = node->der;

}

}

return false;

}

// Lo mismo que la anterior pero sin recursividad, y ademas devuelve el nodo y su padre

bool BTlook(nodeArbol\* arbol, int data, nodeArbol\* &node, nodeArbol\* &father){

node = father = nullptr;

if (arbol != nullptr){

node = arbol;

while (node != nullptr){

if (node->dato == data)

return true;

father = node;

if (data < node->dato)

node = node->izq;

else node = node->der;

}

}

return false;

}

// ### BARRIDOS ITERATIVOS ###

// Hace un barrido RID y muestra las cosas, pero donde dice procesamiento se pueden implementar mas cosas

void BarridoRID(nodeArbol\* arbol){

if (arbol == nullptr)

cout << "Arbol vacio";

else{

nodoPila\* pila = nullptr; nodeArbol\* aux;

push(pila,arbol);

while (!isEmpty(pila)){

aux = pop(pila);

// Procesamiento

cout << aux->dato << " ";

if (aux->der != nullptr)

push(pila,aux->der);

if (aux->izq != nullptr)

push(pila,aux->izq);

}

}

}

void BarridoPorNiveles(nodeArbol\* arbol){

if (arbol == nullptr)

cout << "Arbol vacio";

else{

nodoCola\* frente = nullptr; nodoCola\* fondo = nullptr;

nodeArbol\* aux;

insert(frente,fondo,arbol);

while (!ColaVacia(frente)){

aux = get(frente,fondo);

// Procesamiento

cout << aux->dato << " ";

// Entre estas 2 lineas deberia ir cualquier cosa que quieran procesar

if (aux->izq != nullptr)

insert(frente,fondo,aux->izq);

if (aux->der != nullptr)

insert(frente,fondo,aux->der);

}

}

}

void BPNv2(nodeArbol\* arbol){ // Barrido por niveles, pero indicando el nivel.

if (arbol == nullptr)

cout << "Arbol vacio";

else{

int nivel = 0, nivelActual;

nodoCola\* frente = nullptr; nodoCola\* fondo = nullptr;

nodeArbol\* aux;

insert(frente,fondo,arbol,nivel);

cout << "Nodo rey: ";

while (!ColaVacia(frente)){

aux = get(frente,fondo, nivelActual);

// Procesamiento

if (nivel != nivelActual)

cout << "\nNivel " << nivel+1 << ": ";

cout << aux->dato << " ";

// Obtener altura actual

if (nivel < nivelActual)

nivel = nivelActual;

if (aux->izq != nullptr)

insert(frente,fondo,aux->izq,nivel+1);

if (aux->der != nullptr)

insert(frente,fondo,aux->der,nivel+1);

}

}

}

void BarridoPorAlturas(nodeArbol\* arbol){ // Barrido por ALTURAS.

if (arbol == nullptr)

cout << "Arbol vacio";

else{

int altura = 0, alturaActual;

// Cola para organizar los nodos por niveles

nodoCola\* frente = nullptr; nodoCola\* fondo = nullptr;

// Pila donde guardaremos los nodos

nodoPila\* pila = nullptr;

insert(frente,fondo,arbol,altura);

while (!ColaVacia(frente)){

arbol = get(frente,fondo, alturaActual);

// Guardamos por niveles los datos en la pila

push(pila,arbol,alturaActual);

// Obtener altura actual

if (altura < alturaActual)

altura = alturaActual;

if (arbol->der != nullptr)

insert(frente,fondo,arbol->der,altura+1);

if (arbol->izq != nullptr)

insert(frente,fondo,arbol->izq,altura+1);

}

// Con la pila cargada, con los nodos ordenados, procedemos a mostrar los nodos por altura.

int ValorCentinela = 1;

arbol = pop(pila,altura);

cout << "\nAltura 0: ";

cout << arbol->dato << " ";

while(!isEmpty(pila)){

arbol = pop(pila,alturaActual);

if (alturaActual != altura){

altura = alturaActual;

cout << "\nAltura " << ValorCentinela++ << ": ";

}

cout << arbol->dato << " ";

}

}

}

// ## BARRIDOS RECURSIVOS ##

void BarridoRID\_Rec(nodeArbol\* arbol){ // Barrido Preorden Recursivo

if (arbol != nullptr){

cout << arbol->dato << " ";

BarridoRID\_Rec(arbol->izq);

BarridoRID\_Rec(arbol->der);

}

}

void BarridoIRD\_Rec(nodeArbol\* arbol){ // Barrido InOrden Recursivo

if (arbol != nullptr){

BarridoIRD\_Rec(arbol->izq);

cout << arbol->dato << " ";

BarridoIRD\_Rec(arbol->der);

}

}

void BarridoIDR\_Rec(nodeArbol\* arbol){ // Barrido PostOrden Recursivo

if (arbol != nullptr){

BarridoIDR\_Rec(arbol->izq);

BarridoIDR\_Rec(arbol->der);

cout << arbol->dato << " ";

}

}

// ## ELIMINAR UN NODO ##

void insertarRecursivo(nodeArbol\* &arbol, nodeArbol\* nodo){

if (arbol == nullptr){

arbol = nodo;

} else if (nodo->dato < arbol->dato)

insertarRecursivo(arbol->izq, nodo);

else if (nodo->dato > arbol->dato)

insertarRecursivo(arbol->der, nodo);

}

void BarridoDeNodos(nodeArbol\* arbol, nodoCola\* &frente, nodoCola\* &fondo){

nodoPila\* pila = nullptr;

push(pila, arbol);

while(!isEmpty(pila)){

arbol = pop(pila);

// Insertamos los nodos que traiga en la cola.

insert(frente,fondo,arbol);

if (arbol->der != nullptr)

push(pila, arbol->der);

if (arbol->izq != nullptr)

push(pila, arbol->izq);

// Eliminamos los punteros, porque estos nodos cambiaran de posicion.

arbol->izq = arbol->der = nullptr;

}

}

bool eliminarNodo(nodeArbol\* &arbol,int dato){

// Tenemos 4 casos, graficados aca: https://prnt.sc/RhbabCLpXbCY

nodeArbol\* padre; nodeArbol\* nodo2delete;

if (BTlook(arbol, dato, nodo2delete, padre)){

// Caso 0: Eliminar el nodo raiz o rey

// Caso 1: Eliminar nodo hoja

if (nodo2delete->izq == nullptr && nodo2delete->der == nullptr){

if (padre != nullptr){

if (padre->izq == nodo2delete)

padre->izq = nullptr;

else padre->der = nullptr;

}

else arbol = nullptr;

delete nodo2delete;

return true;

}

// Caso 2: Eliminar nodo intermedio sin hijo derecho

if (nodo2delete->der == nullptr){

if (padre == nullptr){

arbol = nodo2delete->izq;

delete nodo2delete;

} else

padre->izq = nodo2delete->izq;

} else {

// Caso 3: Eliminar nodo intermedio con hijo derecho

nodoCola\* frente = nullptr; nodoCola\* fondo = nullptr;

BarridoDeNodos(nodo2delete->der, frente, fondo);

nodeArbol\* aux = get(frente,fondo);

aux->izq = aux->der = nullptr;

if (padre == nullptr){

//Caso 3b o 4: Reemplazar raiz

aux->izq = arbol->izq;

delete arbol;

arbol = aux;

while(!ColaVacia(frente))

insertarRecursivo(aux, get(frente,fondo));

} else {

if (padre->izq == nodo2delete)

padre->izq = aux;

else padre->der = aux;

aux->izq = nodo2delete->izq;

delete nodo2delete;

while(!ColaVacia(frente))

insertarRecursivo(aux, get(frente,fondo));

}

}

return true;

}

return false;

}

// ## ELIMINAR NODOS V2 ##

// Inspirado en un video q me vi de programacionATS..

void AcomodarNodo(nodeArbol\* padre, nodeArbol\* comparacion, nodeArbol\* resultado){

if (padre->izq == comparacion)

padre->izq = resultado;

else padre->der = resultado;

}

nodeArbol\* EncontrarMenorHijoIZQ(nodeArbol\* &arbol){

if (arbol->izq == nullptr){

nodeArbol\* aux = arbol;

arbol = arbol->der;

return aux;

}

return EncontrarMenorHijoIZQ(arbol->izq);

}

nodeArbol\* AcomodarArbol(nodeArbol\* node2delete){

nodeArbol\* aux = node2delete->der;

// Caso 3: No tiene subarbol derecho

if (aux == nullptr)

return node2delete->izq;

// Caso 4: Si tiene subarbol derecho

if (aux->izq != nullptr)

aux = EncontrarMenorHijoIZQ(aux);

aux->izq = node2delete->izq;

if (aux != node2delete->der)

aux->der = node2delete->der;

return aux;

}

bool eliminar2(nodeArbol\* &arbol, int dato){

nodeArbol\* padre; nodeArbol\* node2delete; nodeArbol\* arbolAcomodado;

if (BTlook(arbol, dato, node2delete, padre)){

// Caso 1: Nodo a eliminar es hoja

if (node2delete->izq==nullptr && node2delete->der==nullptr)

arbolAcomodado = nullptr;

else

// Caso 2: Nodo a eliminar es interno (No es hoja)

arbolAcomodado = AcomodarArbol(node2delete);

// Tomamos la decision de si el nodo a eliminar es raiz o no.

if (padre != nullptr)

AcomodarNodo(padre, node2delete, arbolAcomodado);

else

arbol = arbolAcomodado;

delete node2delete;

return true;

}

return false;

}

// ## CONTAR NODOS HOJA ##

int contarNodosHoja(nodeArbol\* arbol){

int hojas = 0;

if (arbol != nullptr){

nodoPila\* pila = nullptr;

push(pila,arbol);

while(!isEmpty(pila)){

arbol = pop(pila);

if (arbol->izq == nullptr && arbol->der == nullptr)

hojas++;

if (arbol->izq != nullptr)

push(pila,arbol->izq);

if (arbol->der != nullptr)

push(pila,arbol->der);

}

}

return hojas;

}

void mostrarNodosHoja(nodeArbol\* arbol){

if (arbol != nullptr){

nodoPila\* pila = nullptr;

push(pila,arbol);

while(!isEmpty(pila)){

arbol = pop(pila);

if (arbol->izq == nullptr && arbol->der == nullptr)

cout << arbol->dato << " ";

if (arbol->der != nullptr)

push(pila,arbol->der);

if (arbol->izq != nullptr)

push(pila,arbol->izq);

}

}

}

int contarHojasRecursivo(nodeArbol\* arbol){

if (arbol == nullptr)

return 0;

if (arbol->izq == nullptr && arbol->der == nullptr)

return 1;

return contarNodosHoja(arbol->izq) + contarNodosHoja(arbol->der);

}

void mostrarNodosHojaRecursivo(nodeArbol\* arbol){

if (arbol != nullptr){

if (arbol->izq == nullptr && arbol->der == nullptr)

cout << arbol->dato << " ";

mostrarNodosHojaRecursivo(arbol->izq);

mostrarNodosHojaRecursivo(arbol->der);

}

}

Cola/Pila2Arbol

// ## Codigo de pilas ##

struct nodoPila{

nodeArbol\* dato;

int altura;

nodoPila\* link = nullptr;

};

bool isEmpty(nodoPila\* pila){

return (pila == nullptr);

}

void push(nodoPila\* &pila, nodeArbol\* dato){

nodoPila\* nuevoNodo = new nodoPila;

nuevoNodo->dato = dato;

if (pila == nullptr)

pila = nuevoNodo;

else{

nuevoNodo->link = pila;

pila = nuevoNodo;

}

}

nodeArbol\* pop(nodoPila\* &pila){

if (isEmpty(pila))

return 0;

nodeArbol\* dato = pila->dato;

nodoPila\* nodoAremover = pila;

pila = pila->link;

delete nodoAremover;

return dato;

}

// Sobrecarga de las mismas funciones anteriores, pero aca pueden contener alturas tambien.

void push(nodoPila\* &pila, nodeArbol\* dato, int altura){

nodoPila\* nuevoNodo = new nodoPila;

nuevoNodo->dato = dato;

nuevoNodo->altura = altura;

if (pila == nullptr)

pila = nuevoNodo;

else{

nuevoNodo->link = pila;

pila = nuevoNodo;

}

}

nodeArbol\* pop(nodoPila\* &pila, int &altura){

if (isEmpty(pila))

return 0;

nodeArbol\* dato = pila->dato;

altura = pila->altura;

nodoPila\* nodoAremover = pila;

pila = pila->link;

delete nodoAremover;

return dato;

}

// ## Codigo para COLAS ##

struct nodoCola{

nodeArbol\* nodo; // Informacion de la pila

int altura; // Altura del nodo que guarda, lo uso para un ejercicio

nodoCola\* link = nullptr;

};

bool ColaVacia(nodoCola\* frente){

return (frente == nullptr);

}

void insert(nodoCola\* &frente, nodoCola\* &fondo, nodeArbol\* nodo){

nodoCola\* nuevo\_nodo = new nodoCola;

nuevo\_nodo->nodo = nodo;

if (ColaVacia(frente))

frente = fondo = nuevo\_nodo;

else {

fondo->link = nuevo\_nodo;

fondo = nuevo\_nodo;

}

}

nodeArbol\* get(nodoCola\* &frente, nodoCola\* &fondo){

if (!ColaVacia(frente)){

nodeArbol\* nodo = frente->nodo;

nodoCola\* aux = frente;

if (frente == fondo)

fondo = nullptr;

frente = frente->link;

delete aux;

return nodo;

}

return nullptr;

}

// Sobrecarga de las mismas funciones anteriores, pero aca pueden contener alturas tambien.

// Inserta tambien la altura

void insert(nodoCola\* &frente, nodoCola\* &fondo, nodeArbol\* nodo, int altura){

nodoCola\* nuevo\_nodo = new nodoCola;

nuevo\_nodo->nodo = nodo;

nuevo\_nodo->altura = altura;

if (ColaVacia(frente))

frente = fondo = nuevo\_nodo;

else {

fondo->link = nuevo\_nodo;

fondo = nuevo\_nodo;

}

}

// Obtiene tambien la altura

nodeArbol\* get(nodoCola\* &frente, nodoCola\* &fondo, int &altura){

if (!ColaVacia(frente)){

nodeArbol\* nodo = frente->nodo;

altura = frente->altura;

nodoCola\* aux = frente;

if (frente == fondo)

fondo = nullptr;

frente = frente->link;

delete aux;

return nodo;

}

return nullptr;

}