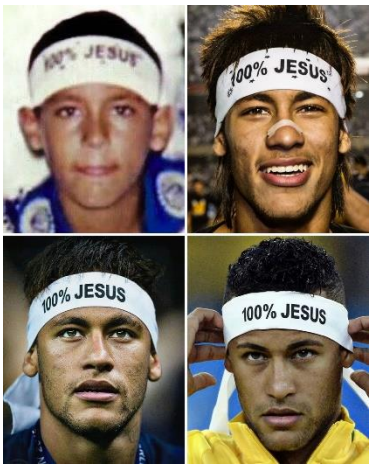
PILAS



ESTRUCTURAS BASICAS:

```
//NODO PARA COLAS Y PILAS
struct Nodo{
        int dato;
        Nodo *siguiente;
};

//AGREGAR UN DATO A LA PILA
void agregarPila(Nodo *&pila, int n);

//ELIMINAR UN DATO DE LA PILA
void sacarPila(Nodo *&pila, int &n);

//AGREGAR UN DATO A LA PILA
void agregarPila(Nodo *&pila, int n){
        Nodo *nuevo_nodo = new Nodo();
        nuevo_nodo-> dato = n;
        nuevo_nodo-> siguiente = pila;
        pila = nuevo_nodo;

        cout<<endl<<"Elemento "<<n<<" agregado a la pila correctamente"<<endl;
}

//ELIMINAR UN DATO DE LA PILA
void sacarPila(Nodo *&pila, int &n){
        Nodo *aux = pila;
        n = aux-> dato;
        pila = aux -> siguiente;
        delete aux;
}


int main(int argc, char *argv[]) {
        Nodo *pila = NULL;
        int dato;
```

```
        cout<<"Ingrese un numero: ";
        cin>>dato;
        agregarPila(pila,dato);

        cout<<endl<<"Ingrese un numero: ";
        cin>>dato;
        agregarPila(pila,dato);

        cout<<endl<<"Sacando elementos de la pila ";
        while(pila != NULL){//HASTA QUE LA PILA ESTE VACIA SE ELIMINA TODOS LOS
 DATOS
                sacarPila(pila,dato);
                if(pila!=NULL){
                        cout<<dato<<", ";
                }else{
                        cout<<dato<<".";
                }
        }

        return 0;
 }
```

COLAS

```
struct Nodo{
        int dato;
        Nodo *siguiente;
};

//checkear si la cola esta vacia
void insertarCola(Nodo *&frente, Nodo *&fin, int  n);

//Meter elementos en la cola
bool cola_vacia(Nodo *frente);

//Eliminar elementos de la cola
void suprimirCola(Nodo *&frente, Nodo *&fin, int &n);

// checkear si la cola esta vacia
bool cola_vacia(Nodo *frente){
        return (frente==nullptr)? true:false;
}

//Meter elementos en la cola
void insertarCola(Nodo *&frente, Nodo *&fin, int  n){
        Nodo *nuevo_nodo = new Nodo();
        nuevo_nodo->dato = n;
        nuevo_nodo->siguiente = nullptr;

        if(cola_vacia(frente)){
```

```
                frente = nuevo_nodo;
        }else{
                fin->siguiente = nuevo_nodo;
        }

        fin=nuevo_nodo;

        cout<<"\nElemento "<<n<<" insertado a cola correctamente";
}

//eliminar los elementos de la cola
void suprimirCola(Nodo *&frente, Nodo *&fin, int &n){
        n = frente -> dato;
        Nodo *aux = frente;

        if(frente == fin){
                frente = nullptr;
                fin = nullptr;
        }else{
                frente = frente->siguiente;
        }
        delete aux;
}

int main(int argc, char *argv[]) {
        Nodo *frente = NULL;
        Nodo *fin = NULL;
        int dato;

        cout<<"Digite un numero: ";
        cin>>dato;
        insertarCola(frente,fin,dato);

        cout<<"\nDigite un numero: ";
        cin>>dato;
        insertarCola(frente,fin,dato);

        cout<<"\nDigite un numero: ";
        cin>>dato;
        insertarCola(frente,fin,dato);

//Eliminar los elementos de la cola
        cout<<"\nQuitando los nodos de la cola: ";
        while(frente != nullptr){
                suprimirCola(frente,fin,dato);
                if(frente != nullptr){
                        cout<<dato<<", ";
                }else{
                        cout<<dato<<".";
                }
        }
```

```
        return 0;
}
```

LISTAS

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct Nodo{
        int dato;
        Nodo *siguiente;
};
Nodo *lista = NULL;

//INSERTA DATOS A LA LISTA
void insertarlista(Nodo *&lista,int n);

//MUESTRA LA LISTA
void mostarListas(Nodo *lista);

//BUSCA UN DATO EN LA LISTA
void buscarLista(Nodo *lista, int n);

//ELIMINA UN NODO DE LA LISTA
void eliminarNodo(Nodo *&lista, int n);

//ELIMINA LA LISTA ENTERA
void eliminarLista(Nodo *&ista, int &n);

//INSERTA DATOS A LA LISTA
void insertarlista(Nodo *&lista,int n){
        Nodo *nuevo_nodo = new Nodo();
        nuevo_nodo->dato = n;

        Nodo *aux1 = lista;
        Nodo *aux2;

        while((aux1!=NULL)&&(aux1-> dato < n)){
                aux2 = aux1;
                aux1 = aux1 -> siguiente;
        }

        if(lista == aux1){
                lista= nuevo_nodo;
        }else{
                aux2-> siguiente = nuevo_nodo;
        }
```

```
        nuevo_nodo -> siguiente = aux1;

        cout<<endl<<"ELEMENTO "<<n<<" INSERTADO A LA LISTA
CORRECTAMENTE."<<endl;
}

//MUESTRA LA LISTA
void mostarListas(Nodo *lista){
        Nodo *actual = new Nodo();
        actual = lista;

        while(actual != NULL){
                cout<<actual->dato<<" -> ";
                actual = actual->siguiente;
        }
}

//      BUSCA UN DATO EN LA LISTA
void buscarLista(Nodo *lista, int n){
        bool band = false;

        Nodo *actual = new Nodo();
        actual = lista;

        while((actual!=NULL)&&(actual->dato<=n)){
                if(actual->dato==n){
                        band=true;
                }
                actual = actual->siguiente;
        }

        if(band == true){
                cout<<"Elemento "<<n<<" SI ha sido encontrado en lista\n";
        }else{
                cout<<"Elemento "<<n<<" No ha sido encontrado\n";
        }
}

// ELIMINA UN NODO DE LA LISTA
void eliminarNodo(Nodo *&lista, int n){
//      Preguntar si la lista esta vacia
        if(lista != NULL){
                Nodo *aux_borrar;
                Nodo *anterior = NULL;

                aux_borrar = lista;
//              recorrer lista
                while((aux_borrar!=NULL)&&(aux_borrar->dato!=n)){
                        anterior = aux_borrar;
                        aux_borrar = aux_borrar->siguiente;
                }
//              EL elemento no ha sido encontrado
```

```
                if(aux_borrar == NULL){
                cout<<"El elemento no existe";
                }
//El primer elemento es el que vamos a eleminar
        else if (anterior == NULL){
                        lista = lista->siguiente;
                        delete aux_borrar;
                }
//El elemento esta en la lista, pero no es el primer nodo
                else{
                        anterior->siguiente = aux_borrar->siguiente;
                        delete aux_borrar;
                }
        }
}

//ELIMINA LA LISTA ENTERA
void eliminarLista(Nodo *&ista, int &n){
        Nodo *aux = lista;
        n = aux->dato;
        lista = aux->siguiente;
        delete aux;
}



void menu(){
        int opcion, dato;

        do{
                cout<<"\t.:MENU:.\n";
                cout<<"1. Insertar elemento a la lista\n";
                cout<<"2. Mostrar los elementos de la lista\n";
                cout<<"3. Buscar un elemento en lista\n";
                cout<<"4. Eliminar un nodo de la lista\n";
                cout<<"5. Eliminar la lista completa\n";
                cout<<"6. Salir\n";
                cout<<"Opcion:";
                cin>>opcion;

                switch(opcion){
                case 1:
                        cout<<"\nIngresar dato: ";
                        cin>>dato;
                        insertarlista(lista,dato);
                        cout<<"\n";
                        system("pause");
                        break;
                case 2:
                        mostarListas(lista);
                        cout<<"\n";
                        system("pause");
```

```
                        break;
                case 3:
                        cout<<"\nDigite un numero a buscar: ";
                        cin>>dato;
                        buscarLista(lista,dato);
                        cout<<"\n";
                        system("pause");
                        break;
                case 4:
                        cout<<"\nDigite el elemento que desear eliminar: ";
                        cin>>dato;
                        eliminarNodo(lista, dato);
                        cout<<"\n";
                        system("pause");
                case 5:
                        while(lista != NULL){
                                eliminarLista(lista,dato);
                                cout<<dato<<" -> ";
                                if(lista==NULL){
                                        cout<<"NULL";
                                }
                        }
                        cout<<"\n";
                        system("pause");
                        break;
                }

                system("cls");
        }while(opcion != 6);
}

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

        menu();


        return 0;
}
```

ARBOLES

```
#include <iostream>
```

```
#include <stdlib.h>
using namespace std;

struct nodo{
        int dato;
        nodo *der;
        nodo *izq;
        nodo *padre;
};

nodo *crearnodo(int n, nodo *padre);
void insertarnodo(nodo *&arbol, int n, nodo *padre);
void mostrararbol(nodo *arbol, int contador);
bool busqueda(nodo *arbol, int n);
void preorden(nodo *arbol);
void inorden(nodo *arbol);
void postorden(nodo *arbol);
void eliminar(nodo *arbol, int n);
void eliminarnodo(nodo *nodoeliminar);
nodo *minimo(nodo *arbol);
void reemplazar(nodo *arbol, nodo *nuevonodo);
void destruirnodo(nodo *nodo);

int main(int argc, char *argv[]) {
        nodo *arbol = NULL;
        int dato, opcion, contador = 0;
        do {
                cout << "MENU" << endl;
                cout << "1)INSERTAR UN NUEVO NODO" << endl;
                cout << "2)MOSTRAR EL ARBOL COMPLETO" << endl;
                cout << "3)BUSCAR UN ELEMENTO EN EL ARBOL" << endl;
                cout << "4)RECORRER ARBOL EN PREORDEN" << endl;
                cout << "5)RECORRER EL ARBOL EN INORDEN" << endl;
                cout << "6)RECORRER EL ARBOL EN POSTORDEN" << endl;
                cout << "7)ELIMINAR UN NODO DEL ARBOL" << endl;
                cout << "8)SALIR" << endl << ">";
                cin >> opcion;
                switch (opcion){
                case 1: cout << "Digite un numero: " << endl << ">";
                        cin >> dato;
                        insertarnodo(arbol, dato, NULL);
                        cout << endl;
                        break;
                case 2: cout << "Mostrando el arbol completo: " << endl;
                        mostrararbol(arbol, contador);
                        cout << endl;
                        break;
                case 3: cout << "Ingrese el elemento a buscar: " << endl
<< ">";
                        cin >> dato;
                        if (busqueda(arbol, dato) == true){
                                cout << "ELEMENTO " << dato << " A SIDO
ENCONTRADO EN EL ARBOL" << endl;
```

```
                        } else {
                                cout << "ELEMENTO NO ENCONTRADO" << endl;
                        }
                case 4: cout << "RECORRIDO EN PREORDEN: " << endl;
                        preorden(arbol);
                        cout << endl;
                        break;
                case 5: cout << "RECORRIDO EN INORDEN: " << endl;
                        inorden(arbol);
                        cout << endl;
                        break;
                case 6: cout << "RECORRIDO EN POSTORDEN: " << endl;
                        postorden(arbol);
                        cout << endl;
                        break;
                case 7: cout << "Digite el nodo que quiere eliminar: " <<
endl << ">";
                        cin >> dato;
                        eliminar(arbol, dato);
                        cout << endl;
                        break;
                case 8: break;
                }
        } while (opcion != 8);

        return 0;
}

// FUNCION PARA CREAR NUEVO NODO
nodo *crearnodo(int n, nodo *padre){
        nodo *nuevo_nodo = new nodo();
        nuevo_nodo -> dato = n;
        nuevo_nodo -> der = NULL;
        nuevo_nodo -> izq = NULL;
        nuevo_nodo -> padre = padre;
        return nuevo_nodo;
}

// FUNCION PARA INSERTAS ELEMENTOS EN EL ARBOL
void insertarnodo(nodo *&arbol, int n, nodo *padre){
        if (arbol == NULL){ // SI EL ARBOL ESTA VACIO
                nodo *nuevo_nodo = crearnodo(n, padre);
                arbol = nuevo_nodo;
        } else { // SI EL ARBOL TIENE UN NODO O MAS
                int valoraiz = arbol -> dato; //OBTENEMOS VALOR DE LA RAIZ
                if (n < valoraiz){ // SI EL ELEMENTO ES MENOR A LA RAIZ A
LA IZQUIERDA PA
                        insertarnodo(arbol -> izq, n, arbol);
                } else {
                        insertarnodo(arbol -> der, n, arbol);
                }
        }
}
```

```
//MOSTRANDO ARBOL COMPLETO
void mostrararbol(nodo *arbol, int contador){
        if (arbol == NULL){ // ARBOL VACIO?
                return;
        } else {
                mostrararbol(arbol -> der, contador+1);
                for (int i = 0; i < contador; i++){
                        cout << "   ";
                }
                cout << arbol -> dato << endl;
                mostrararbol(arbol -> izq, contador+1);
        }
}

//FUNCION PARA BUSCAR ELEMENTO EN EL ARBOL
bool busqueda(nodo *arbol, int n){
        if(arbol == NULL){ //SI EL ARBOL ESTA VACIO
                return false;
        } else if(arbol -> dato == n){ // SI EL NODO ES IGUAL AL ELEMENTO
                return true;
        } else if(n < arbol -> dato){
                return busqueda(arbol -> izq, n);
        } else {
                return busqueda(arbol -> der, n);
        }
}


void preorden(nodo *arbol){
        if (arbol == NULL){ // ARBOL VACIO?
                return;
        } else{
                cout << arbol -> dato << " - "; //RAIZ
                preorden(arbol -> izq);
                preorden(arbol -> der);
        }
}

void inorden(nodo *arbol){
        if (arbol == NULL){
                return;
        } else {
                inorden(arbol -> izq);
                cout << arbol -> dato << " - ";
                inorden(arbol -> der);
        }
}

void postorden(nodo *arbol){
        if (arbol == NULL){
                return;
        } else {
```

```
                postorden(arbol -> izq);
                postorden(arbol -> der);
                cout << arbol -> dato << " - ";
        }
}

void eliminar(nodo *arbol, int n){
        if (arbol == NULL){ //ARBOL VACIO
                return;
        } else if (n < arbol -> dato){ // SI EL VALOR ES MENOR BUSCAR POR
LA IZQUIERDA
                eliminar(arbol -> izq, n);
        } else if (n > arbol -> dato){ // SI EL VALOR ES MAYOR BUSCA POR
LA DERECHA
                eliminar(arbol -> der, n);
        } else { // SI YA LO ENCONTRASTE
                eliminarnodo(arbol);
        }
}

// FUNCION ELIMINAR NODO
void eliminarnodo(nodo *nodoeliminar){
        if (nodoeliminar -> izq && nodoeliminar -> der){ //SI EL NODO
TIENE HIJO IZQ Y DER
                nodo *menor = minimo(nodoeliminar -> der);
                nodoeliminar -> dato = menor -> dato;
                eliminarnodo(menor);
        }else if (nodoeliminar -> izq){ // SI TIENE HIJO IZQUIERDO
                reemplazar(nodoeliminar, nodoeliminar -> izq);
                destruirnodo(nodoeliminar);
        } else if(nodoeliminar -> der){ // SI TIENE UN HIJO DER
                reemplazar(nodoeliminar, nodoeliminar -> der);
                destruirnodo(nodoeliminar);
        } else { //NO TIENE HIJOS
                reemplazar(nodoeliminar, NULL);
                destruirnodo(nodoeliminar);
        }

}
// FUNCION PARA DETERMINAR EL NODO MAS IZQ POSIBLE
nodo *minimo(nodo *arbol){
        if (arbol == NULL){ // si el arbol esta vacio
                return NULL; //retorna nulo
        }
        if (arbol -> izq){ // SI TIENE HIJO IZQ
                return minimo(arbol -> izq); // BUSCAMOS LA PARTE MAS IZQ
POSIBLE
        } else { // SI NO TIENE HIJO IZQUIERDO
                return arbol; // RETORNAMOS EL MISMO NODO
        }
}

// FUNCION PARA REEMPLAZAR DOS NODOS
```

```
void reemplazar(nodo *arbol, nodo *nuevonodo){
        if (arbol -> padre){
                // arbol -> padre asignarle su nuevo hijo
                if (arbol -> dato == arbol -> padre -> izq -> dato){
                        arbol -> padre -> izq = nuevonodo;
                } else if (arbol -> dato == arbol -> padre -> der ->
dato){
                        arbol -> padre -> der = nuevonodo;
                }
        }
        if (nuevonodo){
                // asignarle su nuevo padre
                nuevonodo -> padre = arbol -> padre;
        }
}

// FUNCION PARA DESTRUIR UN NODO
void destruirnodo(nodo *nodo){
        nodo -> izq = NULL;
        nodo -> der = NULL;

        delete nodo;
}
```

PARCIAL DEL AÑO PASADO RESUELTO

```
struct nene{
    string nombre, nombre_tutor;
    int edad;
    float distancia;
    string salita;
};

struct nodoLDE{
    nodoLDE* sig =  NULL;
    nodoLDE* ant = NULL;
    nene* info;
};

struct nodoPila{
    nodoLDE* link =  NULL;
    nene* info;
};

void pilaInsertar(nodoPila* &pila, nene* nuevoDato);

void removerNodo(nodoLDE* &lista){
    nodoLDE* aux = lista;
    if (lista->ant == NULL){
        lista = lista->sig;
        delete aux;
```

```
    } else {
        aux->ant->sig = aux->sig;
        if (aux->sig != NULL)
            aux->sig->ant = aux->ant;
        delete aux;
    }
}

void funcionEj1(nodoLDE* &lista, nodoPila* &SalitaVerde, nodoPila*
&SalitaNegra){
    nodoLDE* aux = lista; nodoLDE* aux2;
    while (aux != NULL){
        aux2 = aux->sig;
        if ((aux->info->edad < 4) && (aux->info->edad > 2) && (aux->info-
>distancia > 2)){
            pilaInsertar(SalitaVerde, aux->info);
            if (lista == aux)
                removerNodo(lista);
            else removerNodo(aux);
        } else if ((aux->info->edad < 6) && (aux->info->edad >= 4) &&
(aux->info->distancia > 2)){
            pilaInsertar(SalitaNegra, aux->info);
            if (lista == aux)
                removerNodo(lista);
            else removerNodo(aux);
        }
        aux = aux2;
    }
}


struct nodoArbolTERNARIO{
    int dato;

    nodoArbolTERNARIO* izq;
    nodoArbolTERNARIO* der;
    nodoArbolTERNARIO* medio;
};

void BarridoRID(nodoArbolTERNARIO* arbol){
    if (arbol == nullptr)
        cout << "Arbol vacio";
    else{
        nodoPila* pila = nullptr; nodoArbolTERNARIO* aux;
        pilaInsertar(pila,arbol);
        while (!isEmpty(pila)){
            aux = pop(pila);

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

            if (aux->der != nullptr)
                push(pila,aux->der);
```

```
            if (aux->medio != nullptr)
                push(pila,aux->medio);
            if (aux->izq != nullptr)
                push(pila,aux->izq);
        }
    }
}

void BARRIDORID(nodoArbolTERNARIO* arbol){
    if (arbol != NULL){
        cout << arbol->dato << " ";
        BARRIDORID(arbol->izq);
        BARRIDORID(arbol->medio);
        BARRIDORID(arbol->der);
    }
}
```

ESTRUCTURAS DINAMICAS DE ARBOL

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>


using namespace std;


// CONSTANTES

#define MAX 1000

// DEFINICION DE TIPOS.

struct nodo_arbol_binario
{
        int dato;
        struct nodo_arbol_binario* iz;
        struct nodo_arbol_binario* de;
};
typedef struct nodo_arbol_binario NABinario;

// "tope" se corresponde con la posici□n donde realizar□ la pr□xima
inserci□n.
// Cuando la pila esta vacia tope = 0.
struct pila_estatica
{
        NABinario* dato[MAX];
        int tamanio;
        int tope;
};
```

```
typedef struct pila_estatica PilaE;


// DECLARACION DE FUNCIONES.

bool pila_vacia (PilaE);
void pila_agregar (PilaE &, NABinario*);
NABinario* pila_sacar (PilaE &);

void menu_opcion1 (NABinario* arbol);
void menu_opcion2 (NABinario* &arbol);
void menu_opcion3 (NABinario* arbol);
void menu_opcion4 (NABinario* arbol);

void abinario_mostrar_recursivo (NABinario* arbol, int tabulado = 0);
void abinariob_alta_recursivo (NABinario* &arbol, int nuevo_dato);
void abinario_preorden_recursivo (NABinario* arbol);
void abinario_preorden_iterativo (NABinario* arbol);
void abinario_mostrar_recursivo2(NABinario* arbol, int n=0);


// DEFINICION DE FUNCIONES.

int main (void)
{
        NABinario* arbol = NULL;

        int opcion = 0;
        do {
                cout << "************Menu de Opciones************\n";
                cout << endl;
                cout << "****** Lista Simplemente Enlazada ******\n";
                cout << endl;
                cout << "1- Mostrar.\n";
                cout << "2- Insertar N elementos.\n";
                cout << "3- Preorden recursivo.\n";
                cout << "4- Preorden iterativo.\n";
                cout << endl;
                cout << "    0- Salir\n";
                cout << endl;
                cout << "                          Ingrese opcion: ";
                cin >> opcion;
                cout << endl;
                cout << endl;

                switch(opcion)
                {
                case 1:
                        menu_opcion1 (arbol);
                        break;
                case 2:
                        menu_opcion2 (arbol);
                        break;
```

```
                case 3:
                        menu_opcion3 (arbol);
                        break;
                case 4:
                        menu_opcion4 (arbol);
                        break;
                }
        } while ( opcion != 0);

        return 0;
}


void menu_opcion1 (NABinario* arbol)
{
        cout << "Arbol:" << endl << endl;
//      abinario_mostrar_recursivo (arbol);
        abinario_mostrar_recursivo2 (arbol);
        cout << endl << endl << endl;
}

void menu_opcion2 (NABinario* &arbol)
{
        int nuevo_dato, cantidad;

        cout << "Cuantos datos aleatorios desea cargar?: ";
        cin >> cantidad;
        cout << endl;
        cout << "Lista de datos cargados:\n";
        cout << endl;
        for (int i=0; i<cantidad; i++)
        {
                nuevo_dato = (rand () % 100) + 1;
                abinariob_alta_recursivo (arbol, nuevo_dato);
                cout << nuevo_dato << "  ";
        }
        cout << endl << endl << endl;
}

void menu_opcion3 (NABinario* arbol)
{
        cout << "Recorrido en PRE-Orden Recursivo:" << endl << endl;
        abinario_preorden_recursivo (arbol);
        cout << endl << endl << endl;
}

void menu_opcion4 (NABinario* arbol)
{
        cout << "Recorrido en PRE-Orden Iterativo:" << endl << endl;
        abinario_preorden_iterativo (arbol);
        cout << endl << endl << endl;
}
```

```
bool pila_vacia (PilaE pila)
{
// Agregue aqui su codigo
        return false;
}

void pila_agregar (PilaE &pila, NABinario* nodo)
{
        // Agregue aqui su codigo
}

NABinario* pila_sacar (PilaE &pila)
{
        // Agregue aqui su codigo
        return NULL;
}

void abinario_mostrar_recursivo (NABinario* arbol, int tabulado)
{
        if (arbol != NULL)
        {
                cout << string (tabulado, '\t');

                cout << "Nodo: " << arbol->dato << " | " << "Iz-> ";
                if (arbol->iz != NULL)
                        cout << arbol->iz->dato;
                else
                        cout << "NULL";
                cout << "  " << "De-> ";
                if (arbol->de != NULL)
                        cout << arbol->de->dato;
                else
                        cout << "NULL";
                cout << endl;

                tabulado++;
                abinario_mostrar_recursivo (arbol->iz, tabulado);
                abinario_mostrar_recursivo (arbol->de, tabulado);
        }
}

void abinario_mostrar_recursivo2 (NABinario* arbol, int n)
{
        if (arbol == NULL)
                return;

        abinario_mostrar_recursivo2 (arbol->de, n+1);
        cout << string (n, '\t') << arbol->dato << endl;
        abinario_mostrar_recursivo2(arbol->iz, n+1);
}

void abinariob_alta_recursivo (NABinario* &arbol, int nuevo_dato)
{
```

```
        if (arbol == NULL)
        {
                arbol = new (NABinario);
                arbol->iz = NULL; arbol->de = NULL;
                arbol->dato = nuevo_dato;
        }
        else if (nuevo_dato < arbol->dato)
                abinariob_alta_recursivo (arbol->iz, nuevo_dato);
        else if (nuevo_dato > arbol->dato)
                abinariob_alta_recursivo (arbol->de, nuevo_dato);
}

void abinario_preorden_recursivo (NABinario* arbol)
{
        if (arbol != NULL)
        {
                cout << arbol->dato << "  ";
                abinario_preorden_recursivo (arbol->iz);
                abinario_preorden_recursivo (arbol->de);
        }
}

void abinario_preorden_iterativo (NABinario* arbol)
{
        NABinario* aux;
        PilaE pila; pila.tamanio = MAX; pila.tope = 0;

        if (arbol != NULL)
                pila_agregar (pila, arbol);

        while (!pila_vacia (pila))
        {
                aux = pila_sacar (pila);
                cout << aux->dato << "  ";

                if (aux->de != NULL)
                        pila_agregar (pila, aux->de);
                if (aux->iz != NULL)
                        pila_agregar (pila, aux->iz);
        }
}
```

ALGORITMO Y ESTRUCTURAS DE DATOS DINAMICAS LINEALES.

```
#include <iostream>
#include <stdlib.h>

using namespace std;


// DEFINICION DE TIPOS.

struct nodo_pila
{
        int dato;
        struct nodo_pila* link;
};
typedef struct nodo_pila NPila;

struct nodo_cola
{
        int dato;
        struct nodo_cola* link;
};
typedef struct nodo_pila NCola;

struct nodo_listase
{
        int dato;
        struct nodo_listase* link;
};
typedef struct nodo_listase NListaSE;


// DECLARACION DE FUNCIONES.

void pila_agregar (NPila* &pila, int ndato);
int pila_obtener (NPila* &pila);
bool pila_vacia (NPila* pila);

void cola_agregar (NCola* &frete, NCola* &fondo, int ndato);
int cola_obtener (NCola* &frete, NCola* &fondo);
bool cola_vacia (NCola* frete, NCola* fondo);

void listase_mostrar (NListaSE* listase);
void listase_agregar_final (NListaSE* &listase, int ndato);
void listase_agregar_ordenado (NListaSE* &listase, int ndato);
bool listase_eliminar_ocurrencia (NListaSE* &listase, int datoe);
void listase_eliminar_ocurrencias (NListaSE* &listase, int datoe);

void menu_opcion1 (NListaSE* listase);
```

```
void menu_opcion2 (NListaSE* &listase);
void menu_opcion3 (NListaSE* &listase);
void menu_opcion4 (NListaSE* &listase);
void menu_opcion5 (NListaSE* &listase);


// DEFINICION DE FUNCIONES.

int main (void)
{
//      NPila* pila = NULL;
//      NCola* cola_frente = NULL, cola_fondo = NULL;
        NListaSE* listase = NULL;

        int opcion = 0;
    do {
                cout << "************Menu de Opciones************\n";
                cout << endl;
                cout << "****** Lista Simplemente Enlazada ******\n";
                cout << endl;
                cout << "1- Mostrar.\n";
                cout << "2- Insertar N elementos al final.\n";
                cout << "3- Insertar N elementos ordenados.\n";
                cout << "4- Elminar primer ocurrencia de N.\n";
                cout << "5- Elminar todas las ocurrencias de N.\n";
                cout << endl;
                cout << "    0- Salir\n";
                cout << endl;
                cout << "                          Ingrese opcion: ";
                cin >> opcion;
                cout << endl;
                cout << endl;

                switch(opcion)
                {
                        case 1:
                                menu_opcion1 (listase);
                        break;
                        case 2:
                                menu_opcion2 (listase);
                        break;
                        case 3:
                                menu_opcion3 (listase);
                        break;
                        case 4:
                                menu_opcion4 (listase);
                        break;
                        case 5:
                                menu_opcion5 (listase);
                        break;
                }
        } while ( opcion != 0);
```

```
        return 0;
}


void menu_opcion1 (NListaSE* listase)
{
        listase_mostrar (listase);
}

void menu_opcion2 (NListaSE* &listase)
{
        int nuevo_dato, cantidad;

        cout << "Cuantos datos aleatorios desea cargar?: ";
        cin >> cantidad;
        cout << endl;
        cout << "Lista de datos cargados:\n";
        cout << endl;
        for (int i=0; i<cantidad; i++)
        {
                nuevo_dato = (rand () % 100) + 1;
                listase_agregar_final (listase, nuevo_dato);
                cout << nuevo_dato << "  ";
        }
        cout << endl;
        cout << endl;
        cout << endl;
}

void menu_opcion3 (NListaSE* &listase)
{
        int nuevo_dato, cantidad;

        cout << "Cuantos datos aleatorios desea cargar?: ";
        cin >> cantidad;
        cout << endl;
        cout << "Lista de datos cargados:\n";
        cout << endl;
        for (int i=0; i<cantidad; i++)
        {
                nuevo_dato = (rand () % 100) + 1;
                listase_agregar_ordenado (listase, nuevo_dato);
                cout << nuevo_dato << "  ";
        }
        cout << endl;
        cout << endl;
        cout << endl;
}

void menu_opcion4 (NListaSE* &listase)
{
        int dato_eliminar;
        bool elimino;
```

```
        cout << "Que dato desea eliminar?: ";
        cin >> dato_eliminar;
        cout << endl;

        elimino = listase_eliminar_ocurrencia (listase, dato_eliminar);

        if (elimino)
                cout << "Fue encontrado y eliminado un dato.\n\n";
        else
                cout << "No fue encontrado el dato.\n\n";
        cout << endl;
}

void menu_opcion5 (NListaSE* &listase)
{
        int dato_eliminar;

        cout << "Que dato desea eliminar?: ";
        cin >> dato_eliminar;

        listase_eliminar_ocurrencias (listase, dato_eliminar);

        cout << endl;
        cout << endl;
}


void listase_mostrar (NListaSE* listase)
{
        cout << "Lista Simplemente Enlazada:\n\n";
        while (listase != NULL)
        {
                cout << listase->dato << " -> ";
                listase = listase->link;
        }
        cout << "NULL\n";
        cout << endl;
        cout << endl;
}

void listase_agregar_final (NListaSE* &listase, int ndato)
{
        NListaSE* aux_lse = listase;
        NListaSE* nuevo_nodo = new (NListaSE);
        nuevo_nodo->dato = ndato;
        nuevo_nodo->link = NULL;

        if (aux_lse == NULL)
                listase = nuevo_nodo;
        else
        {
                while (aux_lse->link != NULL)
```

```
                        aux_lse = aux_lse->link;
                aux_lse->link = nuevo_nodo;
        }
}

void listase_agregar_ordenado (NListaSE* &listase, int ndato)
{
        NListaSE* actual = listase;
        NListaSE* anterior = NULL;
        NListaSE* nuevo_nodo = new (NListaSE);
        nuevo_nodo->dato = ndato;

        while (actual != NULL && actual->dato < ndato)
        {
                anterior = actual;
                actual = actual->link;
        }

        if (anterior == NULL)
        {
                nuevo_nodo->link = listase;
                listase = nuevo_nodo;
        } else
        {
                nuevo_nodo->link = anterior->link;
                anterior->link = nuevo_nodo;
        }
}

bool listase_eliminar_ocurrencia (NListaSE* &listase, int datoe)
{
        NListaSE* actual = listase;
        NListaSE* anterior = NULL;
        NListaSE* aux = NULL;

        while ((actual != NULL) and (actual->dato != datoe))
        {
                anterior = actual;
                actual = actual->link;
        }

        if ((actual != NULL) and (anterior == NULL))
        {
                aux = actual;
                listase = listase->link;
                free (aux);
                return true;
        } else if ((actual != NULL) and (anterior != NULL))
        {
                aux = actual;
                anterior->link = actual->link;
                free (aux);
                return true;
```

```
        }

        return false;
}

void listase_eliminar_ocurrencias (NListaSE* &listase, int datoe)
{
        while (listase_eliminar_ocurrencia (listase, datoe));
}
```

LISTA DOBLEMENTE ENLAZADA

```
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;
}
```