```
//ESTRUCTURAS DE DATOS LINEALES
// Version: 20200415

#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_cola 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 pila_agregar(NPila* &pila, int ndato)
{
  NPila* nuevo_nodo = new NPila;
  nuevo_nodo->dato = ndato;
  nuevo_nodo->link = pila;
  pila = nuevo_nodo;
}

int pila_obtener(NPila* &pila)
{
  if (pila == NULL)
  {
    cout << "ERROR: Pila vacía.\n";
    return -1; // O manejar de otro modo
  }

  NPila* aux = pila;
  int dato = aux->dato;
  pila = pila->link;
  delete aux;
  return dato;
}

bool pila_vacia(NPila* pila)
{
  return (pila == NULL);
}
void cola_agregar(NCola* &frente, NCola* &fondo, int ndato)
{
  NCola* nuevo_nodo = new NCola;
  nuevo_nodo->dato = ndato;
  nuevo_nodo->link = NULL;

  if (frente == NULL)
  {
    frente = fondo = nuevo_nodo;
  }
  else
  {
    fondo->link = nuevo_nodo;
    fondo = nuevo_nodo;
  }
}

int cola_obtener(NCola* &frente, NCola* &fondo)
{
  if (frente == NULL)
  {
    cout << "ERROR: Cola vacía.\n";
    return -1; // O manejar de otro modo
  }

  NCola* aux = frente;
  int dato = aux->dato;
  frente = frente->link;
  if (frente == NULL)
    fondo = NULL; // Se vació la cola

  delete aux;
  return dato;
}

bool cola_vacia(NCola* frente, NCola* fondo)
{
  return (frente == NULL && fondo == NULL);
}
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;
		delete aux;
		return true;
	} else if ((actual != NULL) and (anterior != NULL))
	{
		aux = actual;
		anterior->link = actual->link;
		delete aux;
		return true;
	}

	return false;
}

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

//FIN ESTRUCTURAS DE DATOS LINEALES
```

```
//COLA CIRCULAR
#include <iostream>
#include <fstream>
#include <stdlib.h>


#define MAX 10


using namespace std;


/*
frente: referencia al primer elemento de la ColaCircular, aqu?l que ser?
devuelto. Esto es v?lido SOLO si la cola NO est? VACIA.
fondo: referencia a la pr?xima posici?n a ser ocupada en la pr?xima alta.
llena: true si la cola est? llena, false en caso contrario.
vacia: true si la cola est? vacia, false en caso contrario.
*/
struct cola_circular
{
	int cola [MAX];
	int tamanio;
	int frente;
	int fondo;
	bool llena;
	bool vacia;
	int cantidad_elementos;
};
typedef struct cola_circular ColaCircular;

int alta (ColaCircular &cc, int dato);
int baja (ColaCircular &cc,int &dato);
void mostrar_cc (ColaCircular &cc);


int main (void)
{
	ColaCircular cc_ejemplo;
	cc_ejemplo.tamanio = MAX;
	cc_ejemplo.frente = 0;
	cc_ejemplo.fondo = 0;
	cc_ejemplo.vacia = true;
	cc_ejemplo.llena = false;
	cc_ejemplo.cantidad_elementos = 0;

	int contador = 0, retorno = 0;
	int dato;

	cout << "-------------------------" << endl;
	cout << "Estado de Cola al INICIO" << endl << endl;
	mostrar_cc (cc_ejemplo);
	cout << "-------------------------" << endl << endl;

	while (!cc_ejemplo.llena)
	{
		if ((retorno = alta (cc_ejemplo, contador++)) == 10)
		{
			cout << "Error " << retorno << ": Intento de alta en
ColaCircular llena.";
			break;
		}
		cout << "Elemento dado de alta: " << contador - 1 << endl <<
endl;
		mostrar_cc (cc_ejemplo);
	}

	cout << "----------------------------------------" << endl;
	cout << "Estado de Cola luego de las INSERCIONES" << endl << endl;
	mostrar_cc (cc_ejemplo);
	cout << "----------------------------------------" << endl << endl;
/*
	while (!cc_ejemplo.vacia)
	{
		if ((retorno = baja (cc_ejemplo, dato)) == 20)
		{
			cout << "Error " << retorno << ": Intento de baja en
ColaCircular vacia.";
			break;
		}
		cout << "Elemento obtenido: " << dato << endl << endl;
		mostrar_cc (cc_ejemplo);
	}
	cout << "------------------------------------------------" << endl;
	cout << "Estado de Cola luego de OBTENER todos los datos" << endl
<< endl;
	mostrar_cc (cc_ejemplo);
	cout << "------------------------------------------------" << endl << endl;
```

```
*/
	return retorno;
}

int alta (ColaCircular &cc, int dato)
{
	cc.vacia = false;

	if (cc.llena)
		return 10;

	cc.cola[cc.fondo] = dato;
	cc.cantidad_elementos++;

	if (cc.fondo == cc.tamanio -1)
		cc.fondo = 0;
	else
		cc.fondo ++;

	if (cc.fondo == cc.frente)
		cc.llena = true;

	return 0;
}
int baja (ColaCircular &cc, int &dato)
{
  if (cc.vacia)
    return 20;

  dato = cc.cola[cc.frente];
  cc.cantidad_elementos--;

  if (cc.frente == cc.tamanio - 1)
    cc.frente = 0;
  else
    cc.frente++;

  cc.llena = false;

  if (cc.frente == cc.fondo)
    cc.vacia = true;

  return 0;
}
```

```
void mostrar_cc (ColaCircular &cc)
{
	cout << "OJO! - Esta operacion NO esta permitida para esta estructura
de datos.\nSe implementa solo a los fines de poder realizar un control del
comportamiento de la EDD.\n";
	for (int i=0; i<=cc.tamanio; i++)
	{
		if ((i == cc.frente) && (i != cc.fondo))
			cout << "Elemento [" << i << "]: " << cc.cola[i] << "<-
FRENTE" << endl;
		else if ((i == cc.fondo) && (i != cc.frente))
			cout << "Elemento [" << i << "]: " << cc.cola[i] << "<-
FONDO" << endl;
		else if ((i == cc.fondo) && (i == cc.frente))
			cout << "Elemento [" << i << "]: " << cc.cola[i] << "<-
FRENTE Y FONDO" << endl;
		else
			cout << "Elemento [" << i << "]: " << cc.cola[i] << endl;
	}
	cout << endl;
	cout << "TAMANIO: " << cc.tamanio << endl;
	cout << "FRENTE: " << cc.frente << endl;
	cout << "FONDO: " << cc.fondo << endl;
	cout << "LLENA: " << cc.llena << endl;
	cout << "VACIA: " << cc.vacia << endl;
	cout << "CANTIDAD DE ELEMENTOS: " << cc.cantidad_elementos <<
endl;

	cout << endl << endl;
}

// FIN COLA CIRCULAR
```

```
// ESTRUCTURAS DE DATOS LINEALES NO TRADICIONALES
#include <iostream>
#include <stdlib.h>

using namespace std;


// DEFINICI□N DE TIPOS.

struct nodo_listade
{
        int dato;
        struct nodo_listade* ant;
        struct nodo_listade* sig;
};
typedef struct nodo_listade NListaDE;

struct nodo_listac
{
        int dato;
        struct nodo_listac* link;
};
typedef struct nodo_listac NListaC;

struct nodo_sub {
   string nombre_elemento;
   nodo_sub* sig;
};
typedef nodo_sub NodoSub;

struct nodo_cab {
   string categoria;
   NodoSub* abajo;
   nodo_cab* sig;
};
typedef nodo_cab NodoCab;

// DECLARACI□N DE FUNCIONES.

void listade_mostrar (NListaDE* listade);
void listade_agregar_final (NListaDE* &listade, int ndato);
void listade_agregar_ordenado (NListaDE* &listade, int ndato);
bool listade_eliminar_ocurrencia (NListaDE* &listade, int datoe);
void listade_eliminar_ocurrencias (NListaDE* &listade, int datoe);
void listac_mostrar(NListaC* lista);
void listac_agregar_final(NListaC* &lista, int dato);
void multilista_agregar(NodoCab* &cabecera, string categoria, string
elemento);
void multilista_mostrar(NodoCab* cabecera);

void menu_opcion1 (NListaDE* listade);
void menu_opcion2 (NListaDE* &listade);
void menu_opcion3 (NListaDE* &listade);
void menu_opcion4 (NListaDE* &listade);
void menu_opcion5 (NListaDE* &listade);


// DEFINICI□N DE FUNCIONES.

int main (void)
{
//      NListaC* listac = NULL;
        NListaDE* listade = 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 (listade);
                        break;
```

```
                        case 2:
                                menu_opcion2 (listade);
                        break;
                        case 3:
                                menu_opcion3 (listade);
                        break;
                        case 4:
                                menu_opcion4 (listade);
                        break;
                        case 5:
                                menu_opcion5 (listade);
                        break;
                }
        } while ( opcion != 0);

        return 0;
}


void menu_opcion1 (NListaDE* listade)
{
        listade_mostrar (listade);
}

void menu_opcion2 (NListaDE* &listade)
{
        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;
                listade_agregar_final (listade, nuevo_dato);
                cout << nuevo_dato << "  ";
        }
        cout << endl;
        cout << endl;
        cout << endl;
}
```

```
void menu_opcion3 (NListaDE* &listade)
{
        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;
                listade_agregar_ordenado (listade, nuevo_dato);
                cout << nuevo_dato << "  ";
        }
        cout << endl;
        cout << endl;
        cout << endl;
}

void menu_opcion4 (NListaDE* &listade)
{
        int dato_eliminar;
        bool elimino;

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

        elimino = listade_eliminar_ocurrencia (listade, 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 (NListaDE* &listade)
{
        int dato_eliminar;

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

```
        listade_eliminar_ocurrencias (listade, dato_eliminar);

        cout << endl;
        cout << endl;
}

void listade_mostrar(NListaDE* listade)
{
   cout << "Lista Doblemente Enlazada:\n\n";
   while (listade != NULL)
   {
      cout << listade->dato << " <-> ";
      listade = listade->sig;
   }
   cout << "NULL\n\n\n";
}
void listade_agregar_final(NListaDE* &listade, int ndato)
{
   NListaDE* nuevo_nodo = new NListaDE;
   nuevo_nodo->dato = ndato;
   nuevo_nodo->sig = NULL;
   nuevo_nodo->ant = NULL;

   if (listade == NULL)
   {
      listade = nuevo_nodo;
   }
   else
   {
      NListaDE* aux = listade;
      while (aux->sig != NULL)
         aux = aux->sig;

      aux->sig = nuevo_nodo;
      nuevo_nodo->ant = aux;
   }
}
void listade_agregar_ordenado(NListaDE* &listade, int ndato)
{
   NListaDE* nuevo_nodo = new NListaDE;
   nuevo_nodo->dato = ndato;
   nuevo_nodo->sig = NULL;
   nuevo_nodo->ant = NULL;

   NListaDE* actual = listade;

   while (actual != NULL && actual->dato < ndato)
      actual = actual->sig;

   if (actual == listade)
   {
      // Insertar al inicio
      nuevo_nodo->sig = listade;
      if (listade != NULL)
         listade->ant = nuevo_nodo;
      listade = nuevo_nodo;
   }
   else if (actual == NULL)
   {
      listade_agregar_final(listade, ndato);
      delete nuevo_nodo;
   }
   else
   {
      nuevo_nodo->sig = actual;
      nuevo_nodo->ant = actual->ant;
      actual->ant->sig = nuevo_nodo;
      actual->ant = nuevo_nodo;
   }
}
bool listade_eliminar_ocurrencia(NListaDE* &listade, int datoe)
{
   NListaDE* actual = listade;

   while (actual != NULL && actual->dato != datoe)
      actual = actual->sig;

   if (actual == NULL)
      return false;  // no encontrado

   if (actual == listade)
   {
      listade = actual->sig;
      if (listade != NULL)
         listade->ant = NULL;
   }
   else
```

```
  {
    if (actual->ant != NULL)
      actual->ant->sig = actual->sig;
    if (actual->sig != NULL)
      actual->sig->ant = actual->ant;
  }

  delete actual;
  return true;
}
void listade_eliminar_ocurrencias (NListaDE* &listade, int datoe)
{
        while (listade_eliminar_ocurrencia (listade, datoe));
}

void listac_mostrar(NListaC* lista) {
  if (lista == NULL) {
    cout << "Lista circular vacía.\n";
    return;
  }

  NListaC* aux = lista;
  do {
    cout << aux->dato << " -> ";
    aux = aux->link;
  } while (aux != lista);
  cout << "(vuelve al inicio)\n\n";
}
void listac_agregar_final(NListaC* &lista, int dato) {
  NListaC* nuevo = new NListaC;
  nuevo->dato = dato;

  if (lista == NULL) {
    lista = nuevo;
    lista->link = lista;
  } else {
    NListaC* aux = lista;
    while (aux->link != lista)
      aux = aux->link;

    aux->link = nuevo;
    nuevo->link = lista;
  }
}
```

```
void multilista_agregar(NodoCab* &cabecera, string categoria, string
elemento) {
  NodoCab* cat = cabecera;
  while (cat != NULL && cat->categoria != categoria)
    cat = cat->sig;

  if (cat == NULL) {
    cat = new NodoCab;
    cat->categoria = categoria;
    cat->abajo = NULL;
    cat->sig = cabecera;
    cabecera = cat;
  }

  NodoSub* nuevo = new NodoSub;
  nuevo->nombre_elemento = elemento;
  nuevo->sig = cat->abajo;
  cat->abajo = nuevo;
}
void multilista_mostrar(NodoCab* cabecera) {
  NodoCab* cat = cabecera;
  while (cat != NULL) {
    cout << "Categoria: " << cat->categoria << endl;
    NodoSub* sub = cat->abajo;
    while (sub != NULL) {
      cout << "   - " << sub->nombre_elemento << endl;
      sub = sub->sig;
    }
    cat = cat->sig;
  }
}
// FIN ESRTUCTURAS DE DATOS LINEALES NO TRADICIONALES
```

```
//ARBOLES BINARIOS
# 20200610

#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 << "****** ARBOL BINARIO DE BUSQUEDA ******\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)
{
        return (pila.tope == 0);
}
void pila_agregar (PilaE &pila, NABinario* nodo)
{
        if (pila.tope < pila.tamanio)
                pila.dato[pila.tope++] = nodo;
        else
                cout << "Pila llena.\n";
}
NABinario* pila_sacar (PilaE &pila)
{
        if (pila.tope > 0)
                return pila.dato[--pila.tope];
        else
                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);
        }
}
```

```
//BARRIDOS ARBOL BINARIO ITERATIVOS
struct NodoPila {
  NABinario* nodo;
  int bandera;
};

struct NodoCola {
  NABinario* nodo;
  NodoCola* sig;
};

//DECLARACION DE FUNCIONES
void pila_push(NodoPilaBandera* &pila, NABinario* nodo, int bandera);
bool pila_vacia(NodoPilaBandera* pila);
void pila_pop(NodoPilaBandera* &pila, NABinario* &nodo, int &bandera);

void recorrido_preorden_iterativo(NABinario* arbol)
void recorrido_inorden_iterativo(NABinario* arbol)
void recorrido_postorden_iterativo(NABinario* arbol);
void recorrido_por_niveles(NABinario* arbol);

//DEFINICION DE FUNCIONES
void pila_bandera_push(NodoPilaBandera* &pila, NABinario* nodo, int
bandera) {
  NodoPilaBandera* nuevo = new NodoPilaBandera;
  nuevo->nodo = nodo;
  nuevo->bandera = bandera;
  nuevo->sig = pila;
  pila = nuevo;
}

bool pila_bandera_vacia(NodoPilaBandera* pila) {
  return pila == NULL;
}

void pila_bandera_pop(NodoPilaBandera* &pila, NABinario* &nodo, int
&bandera) {
  if (pila != NULL) {
    NodoPilaBandera* aux = pila;
    nodo = aux->nodo;
    bandera = aux->bandera;
    pila = aux->sig;
    delete aux;
  }
}

void recorrido_preorden_iterativo(NABinario* arbol) {
  if (arbol == NULL) return;

  NPila* pila = NULL;
  pila_agregar(pila, arbol);

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

    if (aux->de != NULL)
      pila_agregar(pila, aux->de);
    if (aux->iz != NULL)
      pila_agregar(pila, aux->iz);
  }
}
void recorrido_inorden_iterativo(NABinario* arbol) {
  if (arbol == NULL) return;

  NodoPilaBandera* pila = NULL;
  pila_bandera_push(pila, arbol, 1);

  NABinario* aux;
  int bandera;

  while (!pila_bandera_vacia(pila)) {
    pila_bandera_pop(pila, aux, bandera);

    if (bandera == 1) {
      pila_bandera_push(pila, aux, 2);
      if (aux->iz != NULL)
        pila_bandera_push(pila, aux->iz, 1);
    } else {
      cout << aux->dato << " ";
      if (aux->de != NULL)
        pila_bandera_push(pila, aux->de, 1);
    }
  }
}
void recorrido_postorden_iterativo(NABinario* arbol) {
  if (arbol == NULL) return;
```

```
    NodoPilaBandera* pila = NULL;
    pila_bandera_push(pila, arbol, 1);

    NABinario* aux;
    int bandera;

    while (!pila_bandera_vacia(pila)) {
        pila_bandera_pop(pila, aux, bandera);

        if (bandera == 1) {
            pila_bandera_push(pila, aux, 2);
            if (aux->iz != NULL)
                pila_bandera_push(pila, aux->iz, 1);
        } else if (bandera == 2) {
            pila_bandera_push(pila, aux, 3);
            if (aux->de != NULL)
                pila_bandera_push(pila, aux->de, 1);
        } else {
            cout << aux->dato << " ";
        }
    }
}
void recorrido_por_niveles(NABinario* arbol) {
    if (arbol == NULL) return;

    NodoColaArbol* frente = NULL;
    NodoColaArbol* fondo = NULL;

    cola_arbol_agregar(frente, fondo, arbol);

    while (!cola_arbol_vacia(frente)) {
        NABinario* aux = cola_arbol_obtener(frente, fondo);
        cout << aux->dato << " ";

        if (aux->iz != NULL)
            cola_arbol_agregar(frente, fondo, aux->iz);
        if (aux->de != NULL)
            cola_arbol_agregar(frente, fondo, aux->de);
    }
}
//FIN BARRIDOS ITERATIVOS
```