```
//COLA CIRCULAR

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

#define MAX 10

using namespace std;

/*
frente: referencia al primer elemento de la ColaCircular, aquél que será
devuelto. Esto es válido SOLO si la cola NO está VACÍA.
fondo: referencia a la próxima posición a ser ocupada en la próxima alta.
llena: true si la cola está llena, false en caso contrario.
vacia: true si la cola está 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) {}


//FIN COLA CIRCULAR
```

```
//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_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));
}

//FIN ESTRUCTURAS DE DATOS LINEALES
```

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

using namespace std;


// DEFINICI�N DE TIPOS.

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

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


// DECLARACI�N DE FUNCIONES.

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

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)
{
	listase_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;
		listase_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;
		listase_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 = listase_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;

	listase_eliminar_ocurrencias (listade, dato_eliminar);

	cout << endl;
	cout << endl;
}

//
void listase_mostrar (NListaDE* listade)
{
	cout << endl;
	cout << endl;
}

void listase_agregar_final (NListaDE* &listade, int ndato){}

void listase_agregar_ordenado (NListaDE* &listade, int ndato){}

bool listase_eliminar_ocurrencia (NListaDE* &listade, int datoe)
{   return false;   }

void listase_eliminar_ocurrencias (NListaDE* &listade, int datoe)
{
	while (listase_eliminar_ocurrencia (listade, datoe));
}
//FIN ESTRUCTURAS DE DATOS LINEALES NO TRADICIONALES
```

```
//ÁRBOLES 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 << "****** 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);
	}
}

//FIN ÁRBOLES BINARIOS
```

//BARRIDOS ITERATIVOS PSEUDOCÓDIGO

Notas:
* El identado delimita los bloques.
* Si la estructura de control "si" tiene solo una línea, no requiere el delimitador de bloque finSi.
* La variable "arbol" es recibida como parámetro, apunta a la raíz del árbol.
* La expresión pila <- arbol agrega arbol a la pila.
* La expresión aux <- pila saca un elemento de la pila y lo guarda en aux.
* La función procesar (aux) hace lo que sea que queramos hacer con cada nodo del árbol, para estos ejemplos asumamos que "muestra el valor del nodo por pantalla".
* La función derecho (aux) retorna el hijo derecho del nodo aux.
* La función izquierdo (aux) retorna el hijo izquierdo del nodo aux.

PREORDEN

```
si (arbol != NULL)
        pila <- arbol
mientras (!pila_vacia (pila))
        aux <- pila
        procesar (aux)
        si (derecho (aux) != NULL)
                pila <- derecho (aux)
        si (izquierdo (aux) != NULL)
                pila <- izquierdo (aux)
finMientras
```

INORDEN

```
si (arbol != NULL)
        pila <- (arbol, 1)
mientras (!pila_vacia (pila))
        (aux, bandera) <- pila
        si (bandera == 1)
                pila <- (aux, 2)
                si (izquierda (aux) != NULL)
                        pila <- (izquierda (aux), 1)
        sino
                procesar (aux)
                si (derecha (aux) != NULL)
                        pila <- (derecha (aux), 1)
        finSi
finMientras
```

POSTORDEN

```
si (arbol != NULL)
        pila <- (arbol, 1)
mientras (!pila_vacia (pila))
        (aux, bandera) <- pila_sacar (pila)
        si (bandera == 1)
                pila <- (aux, 2)
                si (izquierdo (aux) != NULL)
                        pila <- (izquierdo (aux), 1)
        finSi
        si (bandera == 2)
                pila <- (aux, 3)
```

```
                si (derecho (aux) != NULL)
                        pila <- (derecho (aux), 1)
        finSi
        si (bandera == 3)
                procesar (aux)
finMientras
```

Notas:
* La expresión cola <- aux agrega aux a la cola.
* La expresión aux <- cola saca un elemento de la cola y lo guarda en aux.

PORNIVELES

```
cola <- arbol
while (!cola_vacia (cola))
        aux <- cola
        procesar (aux)
        si (izquierdo (aux) != NULL)
                cola <- izquierdo (aux);
        si (derecho (aux) != NULL)
                cola <- derecho (aux);
finMientras
```

//FIN BARRIDOS ITERATIVOS EN PSEUDOCÓDIGO