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

using namespace std;

// Version: 20170419
// DEFINICI□N DE TIPOS.

struct nodo_pila
{
    int dato;
    struct nodo_pila* link;
};
typedef struct nodo_pila NPila;//Npila sera de aqui en mas el nombre de mi nodo
para pila

struct nodo_cola
{
    int dato;
    struct nodo_cola* link;
};
typedef struct nodo_cola NCola;//Ncola sera de aqui en mas el nombre de mi nodo
para estructuras de cola.

struct nodo_listase
{
    int dato;
    struct nodo_listase* link;
};
typedef struct nodo_listase NListaSE;//NListaSE sera de aqui en mas el nombre de
mi nodo para estructuras de lista simplemente enlazadas

// DECLARACI□N DE FUNCIONES.
//Pila
void pila_agregar (NPila* &pila, int ndato);
int pila_obtener (NPila* &pila, int dato_ob);
bool pila_vacia (NPila* pila);
//Cola
void cola_agregar (NCola* &frente, NCola* &fondo, int ndato);//encolar
int cola_obtener (NCola* &frente, NCola* &fondo, int dato_ob);//desencolar
bool cola_vacia (NCola* frente);
//Lista
void listase_mostrar (NListaSE* listase);//mostrar los elementos de una lista
void listase_buscar(NListaSE*&listase,int ndato);//buscar un elemento cualquiera
a la lista
void listase_agregar_final (NListaSE* &listase, int ndato);//agregar datos a una
lista de forma desordenada , siempre al final
void listase_agregar_ordenado (NListaSE* &listase, int ndato);//agregar datos a
una lista , de forma ordenada
void listase_eliminar_ocurrencia (NListaSE* &listase, int datoe);//eliminar un
dato cualquiera dentro de una lista
void listase_eliminar_ocurrencias (NListaSE* &listase);//limpiar la lista ,
borrar todo sus elementos
void listase_Calc_MAY_men(NListaSE*listase);//calcular mayor y menor en una
lista desordenada
void listase_cargarVec(NListaSE*& listase, int *vec, int size);//Dado un vector
cargar una lista
```

```
void listase_enc_Antcesor(NListaSE* lista,int dato_e);//encontrar antecesor,
muestra ese nodo
bool listase_verif_ordAscente(NListaSE* lista);//Verificar si la lista esta
ordenada ascendentemente
void listase_invert(NListaSE* lista);//invertir lista
int main()
{

    return 0;
}
//Algoritmo para agregar un elemento en pila.
void pila_agregar(NPila *&pila, int ndato){
    NPila* nuevo_nodo =new NPila();
    nuevo_nodo->dato=ndato;
    nuevo_nodo->link=pila;
    pila=nuevo_nodo;
}
//Algoritmo para quitar/eliminar elementor de una pila, devuelve el valor que
eliminamos.
int pila_obtener(NPila *&pila, int dato_ob){
    NPila* aux=pila;
    dato_ob=aux->dato;
    pila=aux->link;
    delete aux;
    return dato_ob;
}
//Algoritmo para corroborar si la pila esta vacia o no.
bool pila_vacia(NPila *pila){
    bool band=false;
    if(pila==NULL) band=true;
    return band;
}
//-------------------------------------------------------------------------
----------
//Algoritmo para corroborar si una cola esta vacia.
bool cola_vacia(NCola *frente){
    bool band=false;
    if(frente==NULL)band=true;
    return band;
}

//Algoritmo dar de alta un elemento en cola.
void cola_agregar(NCola *&frente, NCola *&fondo, int ndato){ //chequear
igualacion !!
    NCola* nuevo_nodo=new NCola();
    nuevo_nodo->dato=ndato;
    nuevo_nodo->link=NULL;
    if(cola_vacia(frente))frente=nuevo_nodo;
    else fondo->link=nuevo_nodo;
    fondo=nuevo_nodo;
}

//Algoritmo para eliminar un elemento de la pila, devolviendo su valor para ser
mostrado.
int cola_obtener(NCola *&frente, NCola *&fondo,int dato_ob){
    dato_ob=frente->dato;
    NCola* aux=frente;
    if(frente==fondo){//cola vacia o con un solo elemento
        frente=NULL;
        fondo=NULL;
```

```
    }else{
        frente=frente->link;
    }
    delete aux;
    return dato_ob;
}
//-------------------------------------------------------------------------------
----------------------
//Algoritmo insertar en lista de forma ordenada
void listase_agregar_ordenado(NListaSE *&listase, int ndato){
 NListaSE* nuevo_nodo=new NListaSE();
 nuevo_nodo->dato=ndato;
 NListaSE*aux_1=listase;
 NListaSE*aux_2=NULL;
 while ((aux_1!=NULL)&&(aux_1->dato<ndato))
 {  aux_2=aux_1;
    aux_1=aux_1->link;
 }
 if(listase==aux_1)
 {
     listase=nuevo_nodo;
 }else{
     nuevo_nodo->link=nuevo_nodo;
 }
     nuevo_nodo->link=aux_1;
}
//Mostrar los elementos de una lista.
void listase_mostrar(NListaSE* listase){
    NListaSE*actual=new NListaSE();
    actual=listase;
    while(actual!=NULL)
    { cout<<actual->dato<<"->"<<endl;
        actual=actual->link;
    }
}
//Buscar elementos en una lista
void listase_buscar(NListaSE*&listase,int ndato){
    NListaSE* actual=new NListaSE();
    bool band=false;
    actual=listase;
    while ((actual!=NULL)&&(actual->dato<=ndato))//condicion para buscar en
lista ordenada , sino sacar el menos y que sea solo hasta que se el numero
    {   if(actual->dato==ndato)band=true;
        actual=actual->link;
    }
    if(band==true) cout<<"Elemento "<<ndato<<" encontrado en la lista"<<endl;
    else cout<<"Elemento "<<ndato<<" no encontrado en la lista"<<endl;
}
//Eliminar ocurrencia en una lista
void listase_eliminar_ocurrencia(NListaSE *&listase, int datoe){
    if(listase!=NULL)//Pregunta si la lista esta vacia
    {   NListaSE* aux_borrar;
        NListaSE* anterior=NULL;
        aux_borrar=listase;
        while((aux_borrar!=NULL)&&(aux_borrar->dato=datoe))//Recorre la lista
        { anterior=aux_borrar;
            aux_borrar=aux_borrar->link;
        }

         if (aux_borrar==NULL){// si el elemento no se encuentra en la lista
             cout<<"Elemento no encontrado dentro de la lista"<<endl;
```

```
        }
        else if (anterior==NULL){//el primer elemento es el que vamos a
eliminar
               listase=listase->link;
               delete aux_borrar;
        }else{//El elemento esta en la lista , pero no es el 1ero
            anterior->link=aux_borrar->link;
            delete aux_borrar;
        }
    }
}
void listase_eliminar_ocurrencias (NListaSE* &listase){
    //Dejar la lista vacia , mostrando todo su contenido, en este se puede
evitar el mostrado pero seria lo mismo.
    while(listase!=NULL)
    {   NListaSE*aux=listase;
        int dato=aux->dato;
        listase=aux->link;
        delete aux;
        cout<<dato<<"->";
    }
}
void listase_agregar_final (NListaSE* &listase, int ndato){//insertar elementos
de forma desordenada
    NListaSE*nuevo_nodo=new NListaSE();
    NListaSE*aux;
    nuevo_nodo->dato=ndato;
    nuevo_nodo->link=NULL;
    if(listase==NULL)listase=nuevo_nodo;//si la lista esta vacia
    else aux=listase;//aux apunta al inicio de la lista
    while(aux->link!=NULL)
    {   aux=aux->link;//avanzamos posiciones
    }
    aux->link=nuevo_nodo;//agregar el nodo a la lista
}
void listase_Calc_MAY_men(NListaSE*listase){
 int MAY=0,men=999999;
 while (listase!=NULL) {
    if(listase->dato>MAY)MAY=listase->dato;
    if(listase->dato<men)men=listase->dato;
    listase=listase->link;
 }
 cout<<"El elemento mayor es  :"<<MAY<<endl;
 cout<<"El elemento menor es  :"<<men<<endl;
}

//Algoritmo cargar una lista con un vector
void listase_cargarVec(NListaSE*& listase,int* vec,int size){
    NListaSE* aux=listase;
    NListaSE* nodo_nuevo=new NListaSE();
    for(int i=0;i<=size;i++){
        nodo_nuevo->dato=vec[i];
        aux=aux->link;
    }
    listase=nodo_nuevo;
}
//encontrar antecesor, devuelve puntero a ese nodo
void listase_enc_Antcesor(NListaSE* lista,int dato_e){
    NListaSE* ant=NULL;
    NListaSE* act=lista;
    while((act!=NULL)&&(act->dato!=dato_e))
```

```
    {
        ant=act;
        act=act->link;
    }
    if(act->dato==dato_e)
    {
        cout<<"Antecesor encontrado...."<<endl;
        cout<<"El nodo antecesor es :"<<act->dato<<endl;
    }
}
//Verificar si la lista esta ordenada ascendentemente
bool listase_verif_ordAscente(NListaSE* lista){
    NListaSE* ant=NULL;
    NListaSE* act=lista;
    bool band=true;
    while(act!=NULL)
    {   if(ant->dato<act->dato){
            ant=act;
            act=act->link;
        }else band=false;
    }
    return band;
}
void listase_invert(NListaSE* lista){
    int cant=0;
    NListaSE*aux=lista;
    NListaSE*ant,reinicio;
    while(aux->link!=NULL)
    {
        cant ++;
        ant=aux;
        aux=aux->link;
    }
    reinicio=aux;
    for(int i=0;i<=cant;i++)
    {
        while(aux->link!=NULL){
            ant=aux;
            aux=aux->link;
        }
        NListaSE*cambio=aux->link;
        aux->link=ant->link;
        ant->link=cambio;

    }
    lista=reinicio;
}
```