**Estructuras Lineales: AED 2018**

#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;

}