Laboratorio 10

Punteros, POO y Listas

Nombre: Gabriel Fernando Rodriguez Cutimbo

CUI: 20212157

Grupo: B

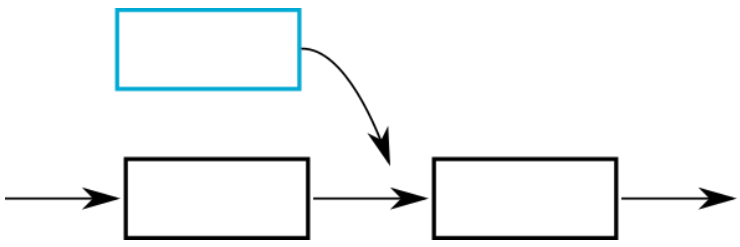
Repositorio GitHub:

https://github.com/gaco123/EPCC_CCII.git

1. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Defina una lista enlazada que permita insertar elementos al final de todos los elementos que ya se hayan ingresado. Por el momento no es necesario preservar un orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
2. Con la implementación de la lista enlazada anterior, desarrolle una función que permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que modificar el comportamiento del puntero que tiene referencia al primer elemento para que sea redireccionado al nuevo elemento por ingresar.
3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar en esa posición.



4. Elabore una función que permita eliminar el último elemento de una lista enlazada. (Evite copiar los elementos en una nueva lista para completar la eliminación del elemento)
5. Desarrolle una función que permita eliminar el primer elemento de una lista sin perder referencia de los demás elementos que ya se encuentran almacenados en la estructura .(Evite copiar los elementos en una nueva lista para completar la eliminación de los elementos)
6. Dado una posición válida dentro de la lista, permita al usuario eliminar un elemento de cualquier posición sin perder referencia de los demás elementos.
7. Desarrolle un algoritmo de ordenamiento que permita ordenar los elementos de forma ascendente y descendente de acuerdo a la elección del usuario. Se debe poder simular el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor tiempo posible (< 2 seg).

Este es el código de todos los ejercicios:

Código "Nodo.h":

```
#ifndef NODO_H
#define NODO_H
#include <cstddef>

class Nodo{
public:
	Nodo();
	~Nodo();
	Nodo* sig;
	int val;
};

#endif
```

Código "Nodo.cpp":

```
#include "Nodo.h"

Nodo::Nodo(){
	sig=NULL;
	val={};
}
Nodo::~Nodo(){
	delete sig;
}
```

Código "Lab_10.cpp":

```
#include <iostream>
#include <clocale>//Cáracteres Español
#include "Nodo.h"
using namespace std;

/*
1. Defina una lista enlazada que permita insertar elementos al final de todos los
elementos que ya se hayan ingresado. Por el momento no es necesario preservar un
orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
*/
void adirUltNodo(Nodo* a, int val){
	//Búsqueda del último elemento
	Nodo* b = a;
	while(b->sig!=NULL){
		b=b->sig;
	}
	//Creación de un nuevo nodo
	Nodo* c = new Nodo;
	c->val=val;
	//Asignación del nuevo nodo al ultimo elemento de la lista
	b->sig=c;
}
/*
2. Con la implementación de la lista enlazada anterior, desarrolle una función que
permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que
modificar el comportamiento del puntero que tiene referencia al primer elemento para
que sea redireccionado al nuevo elemento por ingresar.
*/
Nodo* adirPrimNodo(Nodo* a, int val){
	//Asignar el sig del nuevo nodo al nodo a
	Nodo* c = new Nodo;
	c->val=val;
	c->sig=a;
	//Poner al nuevo nodo como la cabeza de la lista
	return c;
}
/*
3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos
de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese
una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar
en esa posición.
*/
Nodo* adirPosNodo(Nodo* a, int val, int pos){
	//Tamaño valido del vector
	Nodo* b = a;
	int n=1;
	while(b->sig!=NULL){
		b=b->sig;
```

```
            n++;
        }
        if(pos<n&&pos>=0){
            //Búsqueda del elemento en la posición indicada
            b=a;//Reinicio de la posicion del puntero b
            if(pos==0){
                    Nodo* c = new Nodo;
                    c->val=val;
                    c->sig=a;
                    cout<<"Se agrego un nodo de la posición "<<pos<<"\n\n";
                    return c;
            }
            for(int i=0; i<pos-1; i++){
                    b=b->sig;
            }
            Nodo* c = new Nodo;
            c->val=val;
            c->sig=b->sig;
            b->sig=c;
            cout<<"Se agrego un nodo de la posición "<<pos<<"\n\n";
            return a;
        }
        else{
            cout<<"Error al insertar el nodo en la posición indicada...\n\n";
            return a;
        }
}
/*
4. Elabore una función que permita eliminar el último elemento de una lista enlazada.
(Evite copiar los elementos en una nueva lista para completar la eliminación del
elemento)
*/
void borrarUltNodo(Nodo* a){
        Nodo* b = a;
        //Bucle para apuntar al penúltimo elemento
        while(b->sig->sig!=NULL){
            b=b->sig;
        }
        //Desde el penúltimo elemento se borra el último elemento y se hace que penúltimo elemento apunte a NULL
        delete b->sig;
        b->sig=NULL;
}
/*
5. Desarrolle una función que permita eliminar el primer elemento de una lista sin perder
referencia de los demás elementos que ya se encuentran almacenados en la estructura
.(Evite copiar los elementos en una nueva lista para completar la eliminación de los
elementos)
*/
Nodo* borrarPrimNodo(Nodo* a){
        Nodo* b = a;
        //Asigna al apuntador de la cabeza que seleccione el siguiente nodo
```

```
        a=a->sig;
        //Elimina la referencia del nodo en la cabeza al siguiente nodo
        b->sig=NULL;
        //Se elimina el primer nodo
        delete b;
        return a;
}
/*
6. Dado una posición válida dentro de la lista, permita al usuario eliminar un elemento
de cualquier posición sin perder referencia de los demás elementos.
*/
Nodo* borrarPosNodo(Nodo* a, int pos){
        //Tamaño valido del vector
        Nodo* b = a;
        int n=1;
        while(b->sig!=NULL){
                b=b->sig;
                n++;
        }
        if(pos<n&&pos>=0){
                //Búsqueda del elemento en la posición indicada
                b=a;//Reinicio de la posicion del puntero b
                if(pos==0){
                        Nodo* c = a;
                        a=a->sig;
                        c->sig=NULL;
                        delete c;
                        cout<<"Se borro el nodo de la posición "<<pos<<"\n\n";
                        return a;
                }
                for(int i=0; i<pos-1; i++){
                        b=b->sig;
                }
                Nodo* c = b;
                b=b->sig;
                c->sig=b->sig;
                b->sig=NULL;
                delete b;
                cout<<"Se borro el nodo de la posición "<<pos<<"\n\n";
                return a;
        }
        else{
                cout<<"Error al borrar el nodo en la posición indicada...\n\n";
                return a;
        }
}
//Imprime la lista enlazada
void impLista(Nodo* a){
        Nodo* b = a;
        cout<<"[";
        //Recorre toda la lista enlazada hasta que el "nodo* sig" sea null
```

```
        while(b->sig!=NULL){
                cout<<b->val<< ", ";
                b=b->sig;
        }
        //Imprime el ultimo elemento
        cout<<b->val<< "]\n\n";
}
/*
7. Desarrolle un algoritmo de ordenamiento que permita ordenar los elementos de forma
ascendente y descendente de acuerdo a la elección del usuario. Se debe poder simular
el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor tiempo
posible ( < 2 seg).
*/
void descendNodo(Nodo* a){
        //Buscar tamaño del nodo
        Nodo* b = a;
        int n=1;
        while(b->sig!=NULL){
                b=b->sig;
                n++;
        }

        //Ordenamiento Descendente
        int it = 0;
        Nodo* temp = a;
        int tempval = temp->val;
        int temppos = 0;
        bool comp=false;
        for(int i=it; i<n; i++){
                //Evita que el código se ejecute cuando llegue al final
                if(it!=n-1){
                        //Hallar el valor b en cada posición indicada
                        b = a;
                        for(int j=0; j<i; j++){
                                b=b->sig;
                        }
                        //Comparar el valor de cada posición del vector con la posicion con mayor valor actual
                        if(tempval<b->val){
                                temppos=i;
                                tempval=b->val;
                                comp=true;
                        }
                        if(i==n-1&&it<n){
                                //Hallar la posicion del mayor valor
                                b = a;
                                for(int e=0; e<temppos; e++){
                                        b=b->sig;
                                }
                                //Cuando se ha encontrado un numero mayor que el numero seleccionado por temp
                                if(comp==true){
                                        //Intercambiar ambos valores
```

```
                                        swap(temp->val,b->val);
                                        //Ajustar el temporal a la nueva posición
                                        it++;
                                        i=it-1;
                                        temp = a;
                                        for(int d=0; d<it; d++){
                                                temp=temp->sig;
                                        }
                                        tempval=temp->val;
                                        comp=false;
                                }
                                //Solo si el número seleccionado por temp ya es mayor que todos los números
                                else{
                                        it++;
                                        i=it-1;
                                        if(b->sig!=NULL){
                                                temp = a;
                                                for(int d=0; d<it; d++){
                                                        temp=temp->sig;
                                                }
                                                tempval=temp->val;
                                        }
                                }
                        }
                }
        }
}
/*
7. Desarrolle un algoritmo de ordenamiento que permita ordenar los elementos de forma
ascendente y descendente de acuerdo a la elección del usuario. Se debe poder simular
el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor tiempo
posible ( < 2 seg).
*/
void ascendNodo(Nodo* a){
        //Buscar tamaño del nodo
        Nodo* b = a;
        int n=1;
        while(b->sig!=NULL){
                b=b->sig;
                n++;
        }

        //Ordenamiento Ascendente
        int it = 0;
        Nodo* temp = a;
        int tempval = temp->val;
        int temppos = 0;
        bool comp=false;
        for(int i=it; i<n; i++){
                //Evita que el código se ejecute cuando llegue al final
                if(it!=n-1){
```

```
                    //Hallar el valor b en cada posición indicada
                    b = a;
                    for(int j=0; j<i; j++){
                            b=b->sig;
                    }
                    //Comparar el valor de cada posición del vector con la posicion con menor valor actual
                    if(tempval>b->val){
                            temppos=i;
                            tempval=b->val;
                            comp=true;
                    }
                    if(i==n-1&&it<n){
                            //Hallar la posicion del mayor valor
                            b = a;
                            for(int e=0; e<temppos; e++){
                                    b=b->sig;
                            }
                            //Cuando se ha encontrado un numero menor que el numero seleccionado por temp
                            if(comp==true){
                                    //Intercambiar ambos valores
                                    swap(temp->val,b->val);
                                    //Ajustar el temporal a la nueva posición
                                    it++;
                                    i=it-1;
                                    temp = a;
                                    for(int d=0; d<it; d++){
                                            temp=temp->sig;
                                    }
                                    tempval=temp->val;
                                    comp=false;
                            }
                            //Solo si el número seleccionado por temp ya es menor que todos los números
                            else{
                                    it++;
                                    i=it-1;
                                    if(b->sig!=NULL){
                                            temp = a;
                                            for(int d=0; d<it; d++){
                                                    temp=temp->sig;
                                            }
                                            tempval=temp->val;
                                    }
                            }
                    }
            }
        }
}

int main(int argc, char* argv[]){
      setlocale(LC_CTYPE, "Spanish");//Cáracteres Español
```

```
bool comp=true;
bool comporg=false;
int op;

Nodo* a;
while(comp==true){
	cout<<"MENÚ PRINCIPAL DE LA LISTA ENLAZADA\n";
	cout<<"1. Crear el Primer Nodo\n";
	cout<<"2. Añadir Nodo al Final\n";
	cout<<"3. Añadir Nodo al Principio\n";
	cout<<"4. Añadir Nodo en la Posición Seleccionada\n";
	cout<<"5. Eliminar Nodo al Final\n";
	cout<<"6. Eliminar Nodo al Principio\n";
	cout<<"7. Eliminar Nodo en la Posición Seleccionada\n";
	cout<<"8. Ordenar Nodos de Manera Descendente\n";
	cout<<"9. Ordenar Nodos de Manera Ascendente\n";
	cout<<"10. Imprimir Nodos\n";
	cout<<"11. Salir del Programa\n\n";
	cout<<"Ingrese una opción: ";
	cin>>op;
	if(op==11){
		return 0;
	}
	if(op==1&&comporg==false){
		a=new Nodo;
		cout<<"Ingrese un valor para el nodo: ";
		cin>>a->val;
		cout<<endl;
		comporg=true;
	}
	else if(op==1&&comporg==true){
		cout<<"Error, ya se creó el Primer Nodo...\n\n";
	}
	if(comporg==true){
		if(op==2){
			int temp;
			cout<<"Ingrese un valor para el nuevo nodo: ";
			cin>>temp;
			cout<<"Se agrego un ultimo nodo a la lista\n\n";
			adirUltNodo(a,temp);
		}
		else if(op==3){
			int temp;
			cout<<"Ingrese un valor para el nuevo nodo: ";
			cin>>temp;
			cout<<"Se agrego un primer nodo a la lista\n\n";
			a=adirPrimNodo(a,temp);
		}
		else if(op==4){
			int temp1;
			cout<<"Ingrese un valor para el nuevo nodo: ";
```

```
                        cin>>temp1;
                        int temp2;
                        cout<<"Ingrese la posición donde se agregara al nuevo nodo: ";
                        cin>>temp2;
                        a=adirPosNodo(a,temp1,temp2);
                }
                else if(op==5){
                        borrarUltNodo(a);
                        cout<<"Se borro el ultimo nodo de la lista\n\n";
                }
                else if(op==6){
                        a=borrarPrimNodo(a);
                        cout<<"Se borro el primer nodo de la lista\n\n";
                }
                else if(op==7){
                        int temp;
                        cout<<"Ingrese la posición donde se borrara un nodo: ";
                        cin>>temp;
                        a=borrarPosNodo(a,temp);
                }
                else if(op==8){
                        descendNodo(a);
                        cout<<"Se ordeno de manera descendente la lista\n\n";
                }
                else if(op==9){
                        ascendNodo(a);
                        cout<<"Se ordeno de manera ascendente la lista\n\n";
                }
                else if(op==10){
                        cout<<"\nLISTA ENLAZADA\n";
                        impLista(a);
                }
                else if (op<1||op>10){
                        cout<<"Error, escriba un número de opción valida...\n\n";
                }
        }
        else{
                cout<<"Error, necesita crear el Primer Nodo...\n\n";
        }
    }
}
```

5. Entregables

Al final el estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB10_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB10_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB10_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB10_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.