

## Entregable 10. Implementación de operaciones de TDA V

```
#ifndef LISTACLIENTE_H_INCLUDED
#define LISTACLIENTE_H_INCLUDED

#include <string>

#include "nodocliente.h"
#include "cliente.h"
#include "listexception.h"

class ListaCliente{
private:
    NodoCliente* ultimo;
    NodoCliente* primerInsertado;
    NodoCliente* auxiliar;
    NodoCliente* auxiliar2;

    void intercambiar(NodoCliente*,NodoCliente*);
    void copiarTodo(const ListaCliente&);
public:
    ListaCliente();
    ListaCliente(const ListaCliente&);

    bool isEmpty();
    void insertar(const Cliente&);
    void eleminar(const Cliente&);
    NodoCliente* primerNodo();
    NodoCliente* ultimoNodo();
    NodoCliente* anterior(NodoCliente*);
    NodoCliente* siguiente(NodoCliente*);
    NodoCliente* localiza(const Cliente&);
    void ordena();
    void ordena(NodoCliente*,NodoCliente*);
    std::string recupera(const Cliente&);
    std::string toString();
    void eliminarTodo();

    ListaCliente& operator = (const ListaCliente&);
};

#endif // LISTACLIENTE_H_INCLUDED
```

```
#include "listacliente.h"
#include <iostream>
using namespace std;
```

```
void ListaCliente::intercambiar(NodoCliente*a, NodoCliente*b) {
    Cliente aux(a->getCliente());
    a->setCliente(b->getCliente());
    b->setCliente(aux);
}
```

```
void ListaCliente::copiarTodo(const ListaCliente&l) {
    NodoCliente* aux1=l.ultimo;
    NodoCliente* aux2=nullptr;

    while(aux1 != nullptr) {
        NodoCliente* nuevoNodo=new NodoCliente();
        if(aux2==nullptr) {
            aux2 = nuevoNodo;
            primerInsertado = aux2;
        }
        nuevoNodo->setCliente(aux1->getCliente());
        nuevoNodo->setSiguiente(ultimo);
        ultimo = nuevoNodo;
        aux1 = aux1->getSiguiente();
    }
}
```

```
ListaCliente::ListaCliente() : ultimo(nullptr), auxiliar(nullptr), auxiliar2(nullptr),
primerInsertado(nullptr) {
}
```

```
ListaCliente::ListaCliente(const ListaCliente&l) {
    if(isEmpty()==false) {
        eliminarTodo();
    }

    copiarTodo(l);
}
```

```
bool ListaCliente::isEmpty() {
    return ultimo == nullptr;
}
```

```
void ListaCliente::insertar(const Cliente&c) {
    NodoCliente* nuevo_nodo = new NodoCliente();
    if (primerInsertado == nullptr) {
        primerInsertado = nuevo_nodo;
    }
}
```

```

    }
    nuevo_nodo->setCliente(c);
    nuevo_nodo->setSiguiente(ultimo);
    ultimo = nuevo_nodo;
}

void ListaCliente::eliminar(const Cliente&c) {
    if(isEmpty()) {
        throw ListException("No hay datos,ListaCliente->eliminar");
    }
    auxiliar = ultimo;
    if(ultimo->getCliente()==c) {
        ultimo=ultimo->getSiguiente();
        delete auxiliar;
    } else {
        auxiliar = auxiliar->getSiguiente();
        while(auxiliar != nullptr) {
            if(auxiliar->getCliente() == c) {
                auxiliar2 = anterior(auxiliar);
                auxiliar2->setSiguiente(auxiliar->getSiguiente());
                delete auxiliar;
            }
            auxiliar=auxiliar->getSiguiente();
        }
    }
}

}

NodoCliente* ListaCliente::primerNodo() {
    if(isEmpty()) {
        throw ListException("No hay datos,ListaCliente->primerNodo");
    }
    return primerInsertado;
}

NodoCliente* ListaCliente::ultimoNodo() {
    if(isEmpty()) {
        throw ListException("No hay datos,ListaCliente->ultimoNodo");
    }
    return ultimo;
}

NodoCliente* ListaCliente::siguiente(NodoCliente*c) {
    if(isEmpty()) {
        throw ListException("No hay datos,ListaCliente->siguiente");
    }
}

```

```

        return c->getSiguiente();
    }

    NodoCliente* ListaCliente::anterior(NodoCliente*c) {
        if(ultimo == c) {
            throw ListException("No hay anterior del ultimo insertado, ListaCliente->anterior");
        }
        auxiliar = ultimo;
        while(auxiliar != nullptr) {
            if(auxiliar->getSiguiente() == c) {
                return auxiliar;
            }
            auxiliar = auxiliar->getSiguiente();
        }
    }

    NodoCliente* ListaCliente::localiza(const Cliente&c) {
        if(isEmpty() == true) {
            throw ListException("No hay datos,ListaCliente->localiza");
        }
        auxiliar = ultimo;
        while(auxiliar != nullptr) {
            if(auxiliar->getCliente() == c) {
                return auxiliar;
            }
            auxiliar = auxiliar->getSiguiente();
        }
        if(auxiliar == nullptr) {
            throw ListException("No encontrado,ListaCliente->localiza");
        }
    }

    void ListaCliente::ordena()
    {
        ordena(ultimo, primerInsertado);
    }

    void ListaCliente::ordena(NodoCliente*leftedge, NodoCliente*rightedge) {
        if(leftedge == rightedge) {
            return;
        }

        if(leftedge->getSiguiente() == rightedge) {

```

```

    if(leftedge->getCliente() > rightedge->getCliente()) {

        intercambiar(leftedge, rightedge);

    }
    return;
}

NodoCliente* i(leftedge);
NodoCliente* j(rightedge);

while(i != j) {
    while(i != j and i->getCliente() <= rightedge->getCliente()) {
        i = i->getSiguiente();
    }

    while(i != j and j->getCliente() >= rightedge->getCliente()) {
        j = anterior(j);
    }

    intercambiar(i, j);
}
intercambiar(i, rightedge);

if( i!= leftedge) {
    ordena(leftedge, anterior(i));
}

if(i != leftedge) {
    ordena(i->getSiguiente(), rightedge);
}
}

string ListaCliente::recupera(const Cliente&c) {
    auxiliar = ultimo;
    while(auxiliar != nullptr) {
        if(auxiliar->getCliente() == c) {
            Cliente aux(auxiliar->getCliente());
            return aux.toString();
        }
        auxiliar = auxiliar->getSiguiente();
    }
    throw ListException("No encontrado,ListaCliente->recupera");
}

string ListaCliente::toString() {

```

```

    string resultado;
    if(isEmpty()) {
        return resultado;
    }

    auxiliar = ultimo;
    while(auxiliar != nullptr) {
        resultado += auxiliar->toString();
        resultado += "\n";
        auxiliar = auxiliar->getSiguiente();
    }
    return resultado;
}

void ListaCliente::eliminarTodo() {
    if(isEmpty()) {
        throw ListException("No hay datos,ListaCliente->eliminarTodo");
    }
    auxiliar = ultimo;
    while(auxiliar != nullptr) {
        auxiliar2 = auxiliar;
        auxiliar = auxiliar->getSiguiente();
        delete auxiliar2;
    }
    ultimo = nullptr;
    primerInsertado = nullptr;
}

ListaCliente& ListaCliente::operator=(const ListaCliente&l) {
    if(isEmpty()==false) {
        eliminarTodo();
    }
    copiarTodo(l);
    return *this;
}

```

```

#ifdef LISTAAGENTES_H_INCLUDED
#define LISTAAGENTES_H_INCLUDED

```

```

#include <string>

```

```

#include "nodoagente.h"
#include "agente.h"
#include "listexception.h"
#include "menucliente.h"

```

```

class ListaAgentes{

```

**private:**

```
NodoAgente *ultimoInsertado;  
NodoAgente *primerInsertado;  
NodoAgente *auxiliar1;  
NodoAgente *auxiliar2;
```

```
void intercambiar(NodoAgente*,NodoAgente*);  
void copiarTodo(const ListaAgentes&);
```

**public:**

```
ListaAgentes();  
ListaAgentes(const ListaAgentes&);
```

```
bool isEmpty();  
void insertar(const Agente&);  
void eliminar(const Agente&);  
NodoAgente* primerNodo();  
NodoAgente* ultimoNodo();  
NodoAgente* anterior(NodoAgente*);  
NodoAgente* siguiente(NodoAgente*);  
NodoAgente* localiza(const Agente&);  
void ordenaPorNombre();  
void ordenaPorNombre(NodoAgente*,NodoAgente*);  
void ordenaPorEspecialidad();  
void ordenaPorEspecialidad(NodoAgente*,NodoAgente*);  
std::string recupera(const Agente&);  
std::string toString();  
void eliminarTodo();
```

```
ListaAgentes& operator = (const ListaAgentes&);  
};
```

```
#endif // LISTAAGENTES_H_INCLUDED
```

```
#include "listaagentes.h"
```

```
using namespace std;
```

```
void ListaAgentes::intercambiar(NodoAgente*a, NodoAgente*b) {  
    NodoAgente aux;  
    aux = *a;  
    *a = *b;  
    *b = aux;  
}
```

```
void ListaAgentes::copiarTodo(const ListaAgentes&l) {  
    NodoAgente* aux1=l.ultimoInsertado;
```

```

    NodoAgente* aux2=nullptr;

    while(aux1 != nullptr) {
        NodoAgente* nuevoNodo=new NodoAgente();
        if(aux2==nullptr) {
            aux2 = nuevoNodo;
            primerInsertado = aux2;
        }
        nuevoNodo->setAgente(aux1->getAgente());
        nuevoNodo->setSiguiente(ultimoInsertado);
        nuevoNodo->setListaDeClientes(*aux1->getListaDeClientes());
        ultimoInsertado->setAnterior(nuevoNodo);
        ultimoInsertado = nuevoNodo;
        aux1 = aux1->getSiguiente();
    }
}

ListaAgentes::ListaAgentes() : ultimoInsertado(nullptr), primerInsertado(nullptr),
auxiliar1(nullptr), auxiliar2(nullptr) {

}

ListaAgentes::ListaAgentes(const ListaAgentes&l) {
    copiarTodo(l);
}

bool ListaAgentes::isEmpty() {
    return ultimoInsertado == nullptr;
}

void ListaAgentes::insertar(const Agente&a) {
    NodoAgente* nuevo_nodo = new NodoAgente();
    if (primerInsertado == nullptr) {
        primerInsertado = nuevo_nodo;
        nuevo_nodo->setAgente(a);
        nuevo_nodo->setSiguiente(ultimoInsertado);
        ultimoInsertado = nuevo_nodo;
    } else {
        nuevo_nodo->setAgente(a);
        nuevo_nodo->setSiguiente(ultimoInsertado);
        ultimoInsertado->setAnterior(nuevo_nodo);
        ultimoInsertado = nuevo_nodo;
    }
}
}

```



```

void ListaAgentes::eliminar(const Agente&a) {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    auxiliar1 = ultimoInsertado;
    if(ultimoInsertado->getAgente()==a) {
        ultimoInsertado=ultimoInsertado->getSiguiente();
        delete auxiliar1;
    } else {
        while(auxiliar1 != nullptr) {
            if(auxiliar1->getAgente() == a) {
                auxiliar2 = auxiliar1->getAnterior();
                auxiliar2->setSiguiente(auxiliar1->getSiguiente());
                delete auxiliar1;
            }
            auxiliar1=auxiliar1->getSiguiente();
        }
    }
}

```

```

NodoAgente* ListaAgentes::primerNodo() {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    return primerInsertado;
}

```

```

NodoAgente* ListaAgentes::ultimoNodo() {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    return ultimoInsertado;
}

```

```

NodoAgente* ListaAgentes::anterior(NodoAgente*a) {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    return a->getAnterior();
}

```

```

NodoAgente* ListaAgentes::siguiente(NodoAgente*a) {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
}

```

```

    }
    return a->getSiguiente();
}

```

```

NodoAgente* ListaAgentes::localiza(const Agente&a) {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    auxiliar1 = ultimoInsertado;
    while(auxiliar1 != nullptr) {
        if(auxiliar1->getAgente() == a) {
            return auxiliar1;
        }
        auxiliar1 = auxiliar1->getSiguiente();
    }
    throw ListException("No encontrado");
}

```

```

void ListaAgentes::ordenaPorNombre() {
    ordenaPorNombre(ultimoInsertado, primerInsertado);
}

```

```

void ListaAgentes::ordenaPorNombre(NodoAgente* leftedge,
NodoAgente* rightedge) {
    if(leftedge == rightedge) {
        return;
    }

```

```

    if(leftedge->getSiguiente() == rightedge) {

        if(leftedge->getAgente() > rightedge->getAgente()) {

            intercambiar(leftedge, rightedge);

        }
        return;
    }

```

```

    NodoAgente* i(leftedge);
    NodoAgente* j(rightedge);

```

```

    while(i != j) {
        while(i != j and i->getAgente() <= rightedge->getAgente()) {
            i = i->getSiguiente();
        }

```

```

        while(i != j and j->getAgente() >= rightedge->getAgente()) {

```

```

        j = j->getAnterior();
    }

    intercambiar(i, j);
}
intercambiar(i, rightedge);

if( i!= leftedge) {
    ordenaPorNombre(leftedge, i->getAnterior());
}

if(i != leftedge) {
    ordenaPorNombre(i->getSiguiente(), rightedge);
}

}

void ListaAgentes::ordenaPorEspecialidad() {
    ordenaPorEspecialidad(ultimoInsertado, primerInsertado);
}

void ListaAgentes::ordenaPorEspecialidad(NodoAgente* leftedge,
NodoAgente* rightedge) {
    if(leftedge == rightedge) {
        return;
    }

    if(leftedge->getSiguiente() == rightedge) {

        if(leftedge->getAgente().getEspecialidad() > rightedge-
>getAgente().getEspecialidad()) {

            intercambiar(leftedge, rightedge);

        }
        return;
    }

    NodoAgente* i(leftedge);
    NodoAgente* j(rightedge);

    while(i != j) {
        while(i != j and i->getAgente().getEspecialidad() <= rightedge-
>getAgente().getEspecialidad()) {
            i = i->getSiguiente();
        }
    }

```

```

        while(i != j and j->getAgente().getEspecialidad() >= rightedge-
>getAgente().getEspecialidad()) {
            j = j->getAnterior();
        }

        intercambiar(i, j);
    }
    intercambiar(i, rightedge);

    if( i!= leftedge) {
        ordenaPorNombre(leftedge, i->getAnterior());
    }

    if(i != leftedge) {
        ordenaPorNombre(i->getSiguiente(), rightedge);
    }
}

```

```

string ListaAgentes::recupera(const Agente&a) {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    auxiliar1 = ultimoInsertado;
    while(auxiliar1->getSiguiente() != nullptr) {
        if(auxiliar1->getAgente() == a) {
            Agente aux(auxiliar1->getAgente());
            return aux.toString();
        }
        auxiliar1 = auxiliar1->getSiguiente();
    }
    throw ListException("No encontrado");
}

```

```

string ListaAgentes::toString() {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    string resultado;
    auxiliar1 = ultimoInsertado;
    while(auxiliar1 != nullptr) {
        resultado += auxiliar1->toString();
        resultado += "\n";
        auxiliar1 = auxiliar1->getSiguiente();
    }
    return resultado;
}

```

```

void ListaAgentes::eliminarTodo() {
    if(isEmpty()) {
        throw ListException("No hay datos");
    }
    auxiliar1 = ultimoInsertado;
    while(auxiliar1 != nullptr) {
        auxiliar2 = auxiliar1;
        auxiliar1 = auxiliar1->getSiguiente();
        delete auxiliar2;
    }
    ultimoInsertado = nullptr;
    primerInsertado = nullptr;
}

ListaAgentes& ListaAgentes::operator=(const ListaAgentes&l) {
    if(isEmpty()==false) {
        eliminarTodo();
    }
    copiarTodo(l);

    return *this;
}

```

```
C:\Users\Alber\OneDrive\Documents\Administracion call-center Alberto García Ramírez\bin\Debug\Ad...
1)Ordenar por nombre de los agentes
2)Ordenar por especialidad
3)Salir
2
Nombre del agente : Zidal Flores Jimena
Horario de atencion : 14:00
Numero de extension : +52
Horas extras trabajadas : 0
Especialidad : Portatiles

Lista de clientes :
Hora de llamada: 12:00 | Duracion: 00:01:30 | Nombre: Garcia Ramirez Alberto
Hora de llamada: 10:00 | Duracion: 00:01:30 | Nombre: Garcia Gomez Eric
Hora de llamada: 20:00 | Duracion: 00:02:59 | Nombre: Enciso Borrayo Jonatan
Hora de llamada: 04:15 | Duracion: 00:07:50 | Nombre: Sanz Diez Nicolás
Hora de llamada: 02:20 | Duracion: 00:20:48 | Nombre: Lozano Jimenez Marc
Hora de llamada: 08:48 | Duracion: 14:14:24 | Nombre: Mendez Herrero César
Hora de llamada: 12:15 | Duracion: 00:04:50 | Nombre: Aguilar Fuentes Omar

Nombre del agente : Soler Garrido Zrika
Horario de atencion : 14:00
Numero de extension : +52
Horas extras trabajadas : 0
Especialidad : Servidores

Lista de clientes :
Hora de llamada: 12:15 | Duracion: 00:04:50 | Nombre: Aguilar Fuentes Omar
Hora de llamada: 08:48 | Duracion: 14:14:24 | Nombre: Mendez Herrero César
Hora de llamada: 02:20 | Duracion: 00:20:48 | Nombre: Lozano Jimenez Marc
Hora de llamada: 04:15 | Duracion: 00:07:50 | Nombre: Sanz Diez Nicolás
Hora de llamada: 20:00 | Duracion: 00:02:59 | Nombre: Enciso Borrayo Jonatan
Hora de llamada: 10:00 | Duracion: 00:01:30 | Nombre: Garcia Gomez Eric
Hora de llamada: 12:00 | Duracion: 00:01:30 | Nombre: Garcia Ramirez Alberto

Nombre del agente : Marti Gonzalez Elsa
Horario de atencion : 07:00
Numero de extension : +52
Horas extras trabajadas : 0
Especialidad : Servidores

Lista de clientes :
Hora de llamada: 12:00 | Duracion: 00:01:30 | Nombre: Garcia Ramirez Alberto
Hora de llamada: 10:00 | Duracion: 00:01:30 | Nombre: Garcia Gomez Eric
Hora de llamada: 20:00 | Duracion: 00:02:59 | Nombre: Enciso Borrayo Jonatan
Hora de llamada: 04:15 | Duracion: 00:07:50 | Nombre: Sanz Diez Nicolás
Hora de llamada: 02:20 | Duracion: 00:20:48 | Nombre: Lozano Jimenez Marc
Hora de llamada: 08:48 | Duracion: 14:14:24 | Nombre: Mendez Herrero César
Hora de llamada: 12:15 | Duracion: 00:04:50 | Nombre: Aguilar Fuentes Omar

Ordenar la Lista:
1)Ordenar por nombre de los agentes
2)Ordenar por especialidad
3)Salir
1
Nombre del agente : Marti Gonzalez Elsa
Horario de atencion : 07:00
Numero de extension : +52
Horas extras trabajadas : 0
Especialidad : Servidores

Lista de clientes :
Hora de llamada: 12:15 | Duracion: 00:04:50 | Nombre: Aguilar Fuentes Omar
Hora de llamada: 08:48 | Duracion: 14:14:24 | Nombre: Mendez Herrero César
Hora de llamada: 02:20 | Duracion: 00:20:48 | Nombre: Lozano Jimenez Marc
Hora de llamada: 04:15 | Duracion: 00:07:50 | Nombre: Sanz Diez Nicolás
Hora de llamada: 20:00 | Duracion: 00:02:59 | Nombre: Enciso Borrayo Jonatan
Hora de llamada: 10:00 | Duracion: 00:01:30 | Nombre: Garcia Gomez Eric
Hora de llamada: 12:00 | Duracion: 00:01:30 | Nombre: Garcia Ramirez Alberto

Nombre del agente : Soler Garrido Zrika
Horario de atencion : 14:00
Numero de extension : +52
```

Primero seleccione el ordenamiento por especialidad y luego por nombre, tengo un problema porque no funcionan la segunda opción si no uso el ordenamiento por nombre, y no encuentro el problema.