Listas\_t.h  
#include <iostream>

#include <string.h>

#include <sstream>

using namespace std;

// Nodo class definition

template <class T>

class Nodo {

public:

    // O(1) - Constante

    Nodo(T \*info, Nodo<T> \*sig) : Info(info), Sig(sig) {}

    // O(1) - Constante

    Nodo(T \*info) : Sig(NULL), Info(info) {}

    // O(1) - Constante

    virtual void setSig(Nodo<T> \*sig) { Sig = sig; }

    // O(1) - Constante

    virtual void setInfo(T \*info) { Info = info; }

    // O(1) - Constante

    virtual Nodo<T> \*getSig() { return Sig; }

    // O(1) - Constante

    virtual T \*getInfo() { return Info; }

private:

    Nodo<T> \*Sig;

    T \*Info;

};

// List class definition

template <class T>

class List {

public:

    // O(1) - Constante

    List() : first(NULL) {}

    // O(n) - Lineal (en el peor caso)

    virtual void add(T \*value);

    // O(1) - Constante

    virtual Nodo<T> \*getFirst();

    // O(n) - Lineal (en el peor caso)

    virtual Nodo<T> \*getLast();

    // O(n) - Lineal (en el peor caso)

    virtual bool removeAt(int index);

    // O(1) - Constante

    virtual bool isEmpty();

    // O(n) - Lineal

    string toString() const;

private:

    Nodo<T> \*first;

};

template <class T>

void List<T>::add(T \*value) {

    Nodo<T> \*nodo = new Nodo<T>(value);

    if (first == NULL) {

        first = nodo;

    } else {

        Nodo<T> \*ultimateTeam = getLast();

        ultimateTeam->setSig(nodo);

    }

}

template <class T>

Nodo<T> \*List<T>::getFirst() {

    return first;

}

template <class T>

bool List<T>::isEmpty() {

    return first == NULL;

}

template <class T>

Nodo<T> \*List<T>::getLast() {

    if (isEmpty()) return NULL;

    Nodo<T> \*p = first;

    while (p->getSig() != NULL) {

        p = p->getSig();

    }

    return p;

}

template <class T>

bool List<T>::removeAt(int index) {

    Nodo<T> \*nodo = getFirst();

    Nodo<T> \*anterior = NULL;

    int contador = 0;

    while (nodo != NULL) {

        if (contador == index) {

            if (anterior == NULL) {

                first = nodo->getSig();

            } else {

                anterior->setSig(nodo->getSig());

            }

            delete nodo;

            return true;

        }

        contador++;

        anterior = nodo;

        nodo = nodo->getSig();

    }

    return false;

}

template <class T>

string List<T>::toString() const {

    Nodo<T> \*p = getFirst();

    string outputStr = "";

    while (p != NULL) {

        stringstream ss;

        ss << \*(p->getInfo());

        outputStr += ss.str() + " ";

        p = p->getSig();

    }

    return outputStr;

}

// O(n) - Lineal

template <class T>

ostream& operator<< (ostream& outs, const List<T>& obj) {

    return outs << obj.toString();

}

Clases

#include "Direccion.h"

#include <iostream>

using namespace std;

class Contacto {

    string nombre;

    int numero;

    Direccion direccion;

public:

    Contacto(int num, Direccion dir, string nom) : nombre(nom), numero(num), direccion(dir) {}

    int getNumero() const { return numero; }

    string getNombre() const { return nombre; }

    void setNombre(string nom) { nombre = nom; }

    void setNumero(int num) { numero = num; }

    void setDireccion(Direccion dir) { direccion = dir; }

    Direccion getDireccion() const { return direccion; }

    void print() {

        cout << "Nombre: " << nombre << endl;

        cout << "Numero: " << numero << endl;

        direccion.print();

    }

};

#ifndef DIRECCION\_H

#define DIRECCION\_H

class Direccion

{

    string calle, municipio, estado;

    int numero;

public:

    Direccion(int num,

              string muni,

              string est,

              string cal)

        : numero(num), municipio(muni), estado(est), calle(cal) {}

    ~Direccion() {}

    void setNumero(int num) { numero = num; }

    void setMunicipio(string muni) { municipio = muni; }

    void setEstado(string est) { estado = est; }

    void setCalle(string cal) { calle = cal; }

    int getNumero() { return numero; }

    string getMunicipio() { return municipio; }

    string getEstado() { return estado; }

    string getCalle() { return calle; }

    void print() {

        cout << "Calle: " << calle << endl;

        cout << "Numero: " << numero << endl;

        cout << "Municipio: " << municipio << endl;

        cout << "Estado: " << estado << endl;

    }

    ostream &operator<<(ostream &os)

    {

        os << "Calle: " << calle << endl;

        os << "Numero: " << numero << endl;

        os << "Municipio: " << municipio << endl;

        os << "Estado: " << estado << endl;

        return os;

    }

};

#endif // DIRECCION\_H

Main

#include <iostream>

#include "listas\_t.h"

#include "Contacto.h"

#include "Direccion.h"

using namespace std;

int main()

{

    List<Contacto> \*listita = new List<Contacto>();

    int opcion = 0;

    while (opcion != 4)

    {

        cout << "Seleccione que desea hacer: " << endl;

        cout << "1. Crear un contacto" << endl;

        cout << "2. Eliminar un contacto" << endl;

        cout << "3. Buscar un contacto" << endl;

        cout << "4. Salir" << endl;

        cin >> opcion;

        if (opcion == 1)

        {

            cout << "Ingrese el nombre del contacto: " << endl;

            string nombre;

            cin >> nombre;

            cout << "Ingrese el numero del contacto: " << endl;

            int numero;

            cin >> numero;

            cout << "Ingrese la direccion del contacto: " << endl;

            string direccion;

            cin >> direccion;

            cout << "Ingrese la ciudad del contacto: " << endl;

            string ciudad;

            cin >> ciudad;

            cout << "Ingrese el estado del contacto: " << endl;

            string estado;

            cin >> estado;

            Direccion \*dir = new Direccion(numero, ciudad, estado, direccion);

            Contacto \*contacto = new Contacto(numero, \*dir, nombre);

            listita->add(contacto);

        }

        if (opcion == 2)

        {

            cout << "Ingrese el numero de contacto que desea eliminar: " << endl;

            int numero;

            cin >> numero;

            listita->removeAt(numero);

        }

        if (opcion == 3)

        {

            cout << "Ingrese el numero de contacto que desea buscar: " << endl;

            int numero;

            cin >> numero;

            Nodo<Contacto> \*nodo = listita->getFirst();

            while (nodo != NULL)

            {

                if (nodo->getInfo()->getNumero() == numero)

                {

                    cout << "Nombre: " << nodo->getInfo()->getNombre() << endl;

                    cout << "Numero: " << nodo->getInfo()->getNumero() << endl;

                    cout << "Direccion: " << nodo->getInfo()->getDireccion().getCalle() << endl;

                    cout << "Ciudad: " << nodo->getInfo()->getDireccion().getMunicipio() << endl;

                    cout << "Estado: " << nodo->getInfo()->getDireccion().getEstado() << endl;

                }

                nodo = nodo->getSig();

            }

        }

    }

    return 0;

}