Práctica 2

Objetivos

- Familiarizar al estudiante con la implementación de una lista ligada simple usando clases.
- Que el estudiante sea capaz de implementar una lista doblemente ligada.

Problema

¿Recuerdas la agenda de contactos de la anterior practica? Bueno ¡Ahora quieres mejorarla! A ti te encanta tener tus contactos ordenados de acuerdo con quien te cae mejor. Entonces, quieres crear una agenda telefónica donde el orden lo formas tú. Por ejemplo, si tienes la lista

1- Joel
2- Juan
3- Claudia

Y se te ocurre añadir Santy después de Joel, tu lista queda como

1- Joel
2- Santy
3- Juan
4- Claudia

Pero de repente te pueden empezar a caer mal ciertos contactos. Entonces, quieres ser capaz de eliminarlos. Por ejemplo, si Juan te cae mal tu lista puede quedar como

1- Joel
2- Santy
3- Claudia

En esta práctica deberás implementar una lista doblemente ligada para ser capaz de llevar esta agenda de contactos y realizar las operaciones que se mencionaran a continuación.

Ejercicio

Para hacer más eficiente tu algoritmo siempre llevaras la posición del último contacto al que afectaste. Para evitar errores, tu lista de contactos siempre tiene un contacto 0 que es nulo. Las operaciones que debes ser capaz de realizar son:

- agregar_contacto(int n, Contacto* contacto): Deberás moverte n contactos hacia adelante y agregar un nuevo contacto inmediatamente después. El último contacto que afectaste queda como el nuevo que acabas de agregar.

- eliminar_contacto(int n): Deberás moverte n contactos hacia adelante y eliminar ese contacto. El último contacto que afectaste queda determinado como el anterior al que eliminaste. No se puede eliminar el contacto 0. Cuando se elimine un contacto deberá aparecer el mensaje:

```
Se ha eliminado el contacto:
    Nombre: nombre_del_contacto
    Numero: 1234556789
    Email: jose.vales@cimat.mx
```

- mostrar_lista(): Imprime el nombre de todos los contactos en el orden que están.

En caso de que n sea un numero negativo en lugar de moverse n lugares hacia adelante, se moverá n lugares hacia atrás. Si en algún momento se intenta acceder a una posición que no es válida se deberá imprimir el mensaje `[ERROR] Posicion no valida` y la instrucción no se ejecuta.

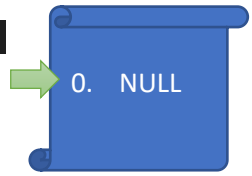
Deberán usar la siguiente estructura para las clases

```
class Contacto {
    public:
        string nombre;
        string numero;
        string email;

        Contacto* siguiente;
        Contacto* anterior;

        Contacto(string nombre, string numero, string email);
        ~Contacto();
};


class ListaContactos{
    public:
        Contacto* head; // Contacto 0
        Contacto* contacto_act; // Contacto en el que te encuentras

        ListaContactos();
        ~ListaContactos();

        void agregar_contacto(int n, Contacto* contacto);
        void eliminar_contacto(int n);
        void mostrar_lista();
};
```

Ejemplo

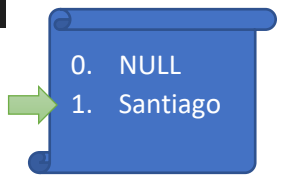
Se te adjunta un archivo main que te facilita ejecutar el ejemplo que se te mostrara a continuación.

```
ListaContactos lista;
```



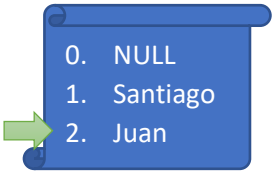
Primero se crea una lista vacía de contactos. El único “contacto” que hay es el nulo que corresponde a la posición 0. Y este es el contacto actual por el momento.

```
Contacto* c1 = new Contacto("Santiago", "1234567890", "jose.vales@cimat.mx");
lista.agregar_contacto(0, c1);
```



Se crear un nuevo contacto llamado Santiago y se añade a la lista. Nota que n es 0, por lo tanto, el contacto se añade justamente después del contacto actual. Y el contacto actual pasa a ser ese nuevo.

```
Contacto* c2 = new Contacto("Juan", "0987654321", "hola.mundo@cimat.mx");
lista.agregar_contacto(0, c2);
```

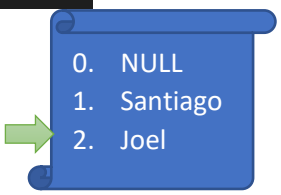


Se añade un nuevo contacto con el nombre Juan. Como n es 0 se añade justo después del contacto actual. En este caso nos queda después de Santiago. Y el contacto actual es el que añadimos.

```
Contacto* c3 = new Contacto("Joel", "1357924680", "adios.mundo@cimat.mx");
lista.agregar_contacto(-1, c3);
```

Luego, se crea un nuevo contacto con el nombre de Joel. Como n es -1, el contacto actual retrocede 1. Entonces el contacto actual pasa a ser ¨Santiago¨. Y el nuevo contacto se añade justo después de Santiago. Al final el contacto actual es el nuevo que añadimos

```
lista.eliminar_contacto(1);
```



Ahora se procede a eliminar un contacto. Como n es 1, el contacto actual avanza 1, es decir, el contacto actual se vuelve Juan y lo eliminamos. Al eliminar a Juan, el contacto actual se vuelve el anterior inmediato, en este caso Joel. Y se muestra en pantalla el mensaje

```
Se ha eliminado el contacto:
```

```
    Nombre: Juan
    Numero: 0987654321
    Email: hola.muundo@cimat.mx
```

```
lista.mostrar_lista();
```

Imprime en pantalla la lista actual del nombre de los contactos en orden.

```
#### Lista de contactos ####
1. Santiago
2. Joel
############################
```

Compilación

```
g++ --std=c++11 main.cpp ListaContactos.cpp -o ListaContactos
```