Instrucciones:

- ◇ Fecha de publicación: 21 de octubre de 2019 a las 9:00.
- ◇ Fecha de entrega: 29 de octubre de 2019 hasta las 23:55.
- ◇ Medio de entrega: `https://e-aulas.urosario.edu.co` (no se reciben entregas por correo electrónico u otros medios).
- ◇ La actividad **debe** realizarse **en grupos de tres** estudiantes.
- ◇ Formato de entrega: implementación e interfaz en C++14.
- ◇ Nombre archivos: definidos más abajo.
- ◇ Importante: no use acentos ni deje espacios en los nombres de los archivos que cree.
- ◇ Solamente un miembro del grupo debe realizar la entrega en formato `zip`. Liste los miembros del mismo como un comentario en el encabezado de la **implementación**.

Protocolo para la evaluación:

Los siguientes lineamientos serán seguidos de forma estricta y sin excepción.

1. Los grupos pueden consultar sus ideas con los profesores para recibir orientación; sin embargo, la solución y detalles del ejercicio deben realizarlos **los integrantes de cada grupo**. Cualquier tipo de fraude o plagio es causa de anulación directa de la evaluación y correspondiente proceso disciplinario.
2. El grupo de trabajo debe indicar en su entrega de la solución a la actividad cualquier asistencia que haya recibido.
3. El grupo no debe consultar ninguna solución a la actividad que no sea la suya.
4. El grupo no debe intentar ocultar ningún código que no sea propio en la solución a la actividad (a excepción del que se encuentra en las plantillas).
5. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
6. `e-aulas` se cerrará a la hora en punto acordada para el final de la evaluación. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de e-aulas será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0.

No habrán excepciones a estas reglas.

Enunciado:

Resuelva el siguiente ejercicio sobre listas enlazadas dobles. No olvide compilar usando el estándar C++14, usando `-std=c++14`, junto con los *flags* para detectar *warnings* y errores: `-Wall -Wextra -Werror`.

Escriba su código en archivos de implementación (`dlist.cpp`), interfaz (`dlist.hpp`) y un archivo principal (`main.cpp`). Solamente debe entregar los dos primeros archivos. Asegúrese de seguir cuidadosamente las indicaciones del ejercicio.

1. [*Doubly linked lists*.] Implemente una versión más simple del contenedor secuencial `std::list`, llamada `List`, usando el concepto de listas doblemente enlazadas. En una lista doblemente enlazada el orden y conexión entre elementos es mantenido asociando a cada elemento enlaces a los elementos anterior y siguiente.

   La implementación desarrollada debe tener las siguientes características:

   - Inserción y borrado de elementos al inicio y al final de la lista debe realizarse en tiempo constante $\Theta(1)$, manteniendo punteros al principio y final de la lista.
   - Una clase nodo/celda, `Node/Cell`, que contiene los datos del nodo y punteros a los nodos/celdas anterior y siguiente en la lista.
   - La clase `List` debe contener variables de instancia para los apuntadores a los nodos/celdas en los extremos de la lista y el tamaño de la lista, así como una colección básica de métodos.
   - Los métodos, con su descripción, que deben ser implementados se muestra a continuación. Estos tienen conexión con aquellos definidos para la ADT List.

```
template<typename dataType>
class List {
private:
    struct Cell {/* insert implementation */};

public:
    List();                     // create empty list
    List(const List & other); // new list from other list
    ~List();                    // destroy list & contents

    List & operator=(const List & other);

    int size() const;   // return number of elements
    bool empty() const; // check if container is empty
    void clear();       // clear contents of list

    dataType & back();  // access last element
    dataType & front(); // access first element

    void push_back(dataType & rhs);  // insert at end
    void push_front(dataType & rhs); // insert at beg
```

```
22      void pop_back();                    // remove last item
23      void pop_front();                   // remove first item
24
25  private:
26      int count; // number of elements
27      /* insert other fields for List */
28  };
```

Demuestre la correctitud de su implementación del contenedor `List` creando una función que cree un objeto de la clase `List` y utilice todos los métodos expuestos públicamente por esta clase.