ReadMe DLList

```
template <typename Object>
class DLList{
private:
    struct Node{
        Node *prev;
        Object data;
        Node *next;

        // constructor por copia
        Node(const Object &d = Object{}, Node *n = nullptr)
                : data{d}, next{n}, prev{n} {}
                // constructor por referencia
        Node(Object &&d, Node *n = nullptr)
            :data{std::move(d)}, next{n},prev{n}{}
    }; // final struct node
    public:
    class iterator{
    public:
        // constructor implicito
        iterator() : current{nullptr} {}
        // define * como puntero
        Object &operator *(){
            if (current == nullptr)
                throw std::logic_error("");
            return current -> data;
        }
        //enseña como moverse por la lista
        iterator &operator++() {
            if (current)
                current = current->next;
            else
                throw std::logic_error("Trying to increment past the end.");
            return *this;
        }
        iterator operator++(int) {
            iterator old = *this;
            ++(*this);
            return old;
        }
        iterator &operator--() {
```

```
            if(current)
                current = current->prev;
            else
                throw std::logic_error("Trying to decrease past the beginning.");
            return *this;
        }
        iterator operator--(int) {
            iterator old = *this;
            --(*this);
            return old;
        }

        bool operator==(const iterator &rhs) const {
            return current == rhs.current;
        }

        bool operator!=(const iterator &rhs) const {
            return !(*this == rhs);
        }
        // fin atributos publicos
    private:
        //apunta al nodo al que estoy trabajando en ese momento
        Node *current;
        iterator(Node *p) : current{p} {}
        //da accesos a los tributos privados
        friend class DLList<Object>;
        // fin atributos privados
    }; // fin iterator
public: // valores publicos de la DLList
    // como quisiera construir nuestro amor,
    // pero solo puedo construir esta lista "doblemente enlazada"
    DLList() : head(new Node()), tail(new Node()), theSize(0) {
        head->next = tail;
    }
    // destructor
    ~DLList() {
        clear();
        delete head;
        delete tail;
    }

    // mete el iterador ya sea al principio o al final.
    iterator begin() { return {head->next}; }
    iterator end() { return {tail}; }
    // dice el tamaño de la lista para que tu iterador sepa
    // que tanto recorer
```

```
    int size() const { return theSize; }
    bool empty() const { return size() == 0; }

    // Si no está vacia borra el objeto
    void clear() { while (!empty()) pop_front(); }
     // te dice si la lista esta vaciá si es asi da error si no es asi
     // te lleva al inicio de la lista
     Object &front(){
        if(empty())
            throw std::logic_error("la lista esta vacia");
        return *begin();
    }
     //funciones de push
     void push_front(const Object &x) { insert(begin(), x); }
     void push_front(Object &&x) { insert(begin(), std::move(x)); }
     void push_back(const Object &x) { insert(end(), x); }
     void push_back(Object &&x) { insert(end(), std::move(x)); }

     // elimina el valor de enfrente
     void pop_front() {
        if(empty())
        throw std::logic_error("List is empty.");
            erase(begin());
     }

     //  Inserta el valor en la posicion que le demos
     iterator insert(iterator itr, const Object &x) {
        Node *p = itr.current;
        head->next = new Node{x, head->next};
        theSize++;
```