ReadMe Listas

Aquí se vera la documentación de una lista creada con témplate en c++

```
#ifndef SLLIST_H
#define SLLIST_H

#include <iostream>
#include <utility>
// declaras que es una template
template <typename Object>
// de la clase lista
class SLList {
    // atributos privados
private:
    //Cada Cuadrito
    struct Node  {
        Object data;
        Node *next;// anya que apunta meme.jpg
        // Constructores        // por copia        Node(const Object &d =
Object{}, Node *n = nullptr)
            : data{d}, next{n} {}
            // por referencia
        Node(Object &&d, Node *n = nullptr)
            : data{std::move(d)}, next{n} {}
    };
// atributos publicos
public:
    // iterador
    class iterator {
        // atributos publicos del iterador
    public:

        // constructor default
        iterator() : current{nullptr} {}
        // declaracion de puntero
        Object &operator*() {
            if(current == nullptr)
                throw std::logic_error("Trying to dereference a null pointer.");
            return current->data;
        }
        // Declaracion de adicion para el proximo puntero
        iterator &operator++() {
```

```
            if(current)
                current = current->next;
            else                throw std::logic_error("Trying to increment past
the end.");
            return *this;
        }
        // usas el ++ para determinar la posicion
        iterator operator++(int) {
            iterator old = *this;
            ++(*this);
            return old;
        }
        // Declaras como usar el ==
        bool operator==(const iterator &rhs) const {
            return current == rhs.current;
        }
        // Declaras como usar el !=
        bool operator!=(const iterator &rhs) const {
            return !(*this == rhs);
        }
    // atributos privados del iterador
    private:
        //anya que apunta
        Node *current;
        // constructor
        iterator(Node *p) : current{p} {
        friend class SLList<Object>;
    };

public:
        // constructor default de las listas
    SLList() : head(new Node()), tail(new Node()), theSize(0) {
        head->next = tail;
    }
    // destructor
    ~SLList() {
        clear();
        delete head;
        delete tail;
    }
    // optienes el inicio de las listas
    iterator begin() { return {head->next}; }
    // optienes el final de las listas
    iterator end() { return {tail}; }
    // optienes el tamaño de las listas
    int size() const { return theSize; }
```

```
    // revisas si la lista esta vacia si es asi devuelves verdadero de otra forma
es falso
    bool empty() const { return size() == 0; }
    // si la lista no esta vacia la limpia
    void clear() { while (!empty()) pop_front(); }
    // inicio de la lista
    Object &front() {
            // checa si el primer elemeto esta vacio si no da el valor
        }
        if(empty())
            throw std::logic_error("List is empty.");
        return *begin();
    }
    // añade nuevos elementos a la lista
    void push_front(const Object &x) { insert(begin(), x); }
    void push_front(Object &&x) { insert(begin(), std::move(x)); }
    // elimina el primer elemento de la lista
    void pop_front() {
        if(empty())
            throw std::logic_error("List is empty.");
        erase(begin());
    }
    // inserta el elemento hacia donde está el puntero.
    // por copia    iterator insert(iterator itr, const Object &x) {
        Node *p = itr.current;
        head->next = new Node{x, head->next};
        theSize++;
        return iterator(head->next);
    }
    // por referencia
    iterator insert(iterator itr, Object &&x) {
        Node *p = itr.current;
        head->next = new Node{std::move(x), head->next};
        theSize++;
        return iterator(head->next);
    }

    // elimina el elemento donde está el puntero
    iterator erase(iterator itr) {
        if (itr == end())
            throw std::logic_error("Cannot erase at end iterator");
        Node *p = head;
        while (p->next != itr.current) p = p->next;
        p->next = itr.current->next;
        delete itr.current;
        theSize--;
```

```
        return iterator(p->next);
    }
// imprime la lista
    void printList() {
        iterator itr = begin();
        while (itr != end()) {
            std::cout << *itr << " ";
            ++itr;
        }
        std::cout << std::endl;
    }

private:
    // inicio de la lista
    Node *head;
    // final de la lista
    Node *tail;
    // valor
    int theSize;
    // funcion para inicializar variables
    void init() {
        theSize = 0;
        head->next = tail;
    }
};
#endif
```