Listas doblemente enlazadas

DoubleLinkedList.h

```
#ifndef LISTASMODIFICADAS_DOUBLELINKEDLIST_H
#define LISTASMODIFICADAS_DOUBLELINKEDLIST_H

#include <iostream>
#include <cassert>

using namespace std;

template<typename T>
struct Node{
   T data;
   Node<T>* prev;
   Node<T>* next;
};

template<typename T>
class DoubleLinkedList {
private:
   Node<T>* begin;
   int count;
   Node<T>* makeNode(const T& value);
public:
   DoubleLinkedList();
   ~DoubleLinkedList();
   void insert(int pos, const T& value);
   void erase(int pos);
   T& get(int pos) const;
   void print() const;
   void printReverse() const;
   int size() const;
};

#endif //LISTASMODIFICADAS_DOUBLELINKEDLIST_H
```

DoubleLinkedList.cpp

```
#ifndef LISTASMODIFICADAS_DOUBLELINKEDLIST_CPP
#define LISTASMODIFICADAS_DOUBLELINKEDLIST_CPP

#include "DoubleLinkedList.h"

template<typename T>
DoubleLinkedList<T>::DoubleLinkedList(): begin(0), count(0){}

template<typename T>
DoubleLinkedList<T>::~DoubleLinkedList(){
   Node<T>* del = begin;
   while (begin){
       begin = begin->next;
       delete del;
       del = begin;
   }
```

```
}

template<typename T>
void DoubleLinkedList<T>::insert(int pos, const T& value){
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    Node<T>* add = makeNode(value);
    if(pos == 0){
        if(count > 0)
            begin->prev = add;
        add->next = begin;
        begin = add;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        add->next = cur->next;
        cur->next = add;
        if(pos < count)
            add->next->prev = add;
        add->prev = cur;
    }
    count++;
}

template<typename T>
void DoubleLinkedList<T>::erase(int pos){
    if(pos < 0 || pos>=count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    if(pos == 0){
        Node<T>* del = begin;
        begin = begin->next;
        begin->prev = 0;
        delete del;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        Node<T>* del = cur->next;
        cur->next = del->next;
        if(pos < count - 1)
            cur->next->prev = cur;
        delete del;
    }
    count--;
}

template<typename T>
T& DoubleLinkedList<T>::get(int pos) const{
    if(pos < 0 || pos>count-1){
        cout << "Error! The position is out of range." << endl;
        assert(false);
    }
    if(pos == 0){
```

```
        return begin->data;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos; i++){
            cur = cur->next;
        }
        return cur->data;
    }
}

template<typename T>
void DoubleLinkedList<T>::print() const{
    if(count == 0){
        cout << "List is empty." << endl;
        return;
    }
    Node<T>* cur = begin;
    while(cur){
        cout << cur->data << " ";
        cur = cur->next;
    }
}

template<typename T>
void DoubleLinkedList<T>::printReverse() const{
    if(count == 0){
        cout << "List is empty." << endl;
        return;
    }
    Node<T>* cur = begin;
    while(cur->next){
        cur = cur->next;
    }
    while(cur){
        cout << cur->data << " ";
        cur = cur->prev;
    }
}

template<typename T>
int DoubleLinkedList<T>::size() const{
    return count;
}

template<typename T>
Node<T>* DoubleLinkedList<T>::makeNode(const T& value){
    Node<T>* temp = new Node<T>;
    temp->data = value;
    temp->next = 0;
    temp->prev = 0;
    return temp;
}

#endif
```

main.cpp

```cpp
#include <iostream>
#include "DoubleLinkedList.h"

using namespace std;
int main() {
    DoubleLinkedList<string> nombres;
    nombres.insert(0, "nestor");
    nombres.insert(1, "eduardo");
    nombres.insert(2, "suat");
    nombres.insert(3, "rojas");

    nombres.erase(3);
    cout << nombres.get(1) << endl;
    cout << nombres.size() << endl;
    nombres.print();
    cout << endl;
    nombres.printReverse();
    cout << endl;
    return 0;
}
```

Listas circulares

CircularLinkedList.h

```cpp
#ifndef LISTASMODIFICADAS_CIRCULARLINKEDLIST_H
#define LISTASMODIFICADAS_CIRCULARLINKEDLIST_H

#include <iostream>
#include <cassert>

using namespace std;

template<typename T>
struct Node{
    T data;
    Node<T>* next;
};

template<typename T>
class CircularLinkedList {
public:
    Node<T>* begin;
    Node<T>* end;
    int count;
    Node<T>* makeNode(const T& value);
public:
    CircularLinkedList();
    ~CircularLinkedList();
    void insert(int pos, const T& value);
    void erase(int pos);
    T& get(int pos) const;
    void print() const;
    int size() const;
};
```

```
#endif //LISTASMODIFICADAS_CIRCULARLINKEDLIST_H
```

CircularLinkedList.cpp

```
#ifndef LISTASMODIFICADAS_CIRCULARLINKEDLIST_CPP
#define LISTASMODIFICADAS_CIRCULARLINKEDLIST_CPP

#include "CircularLinkedList.h"

template<typename T>
CircularLinkedList<T>:: CircularLinkedList(): begin(0), end(0), count(0){

}
template<typename T>
CircularLinkedList<T>:: ~CircularLinkedList(){
    Node<T>* del = begin;
    end->next = 0;
    while (begin){
        begin = begin->next;
        delete del;
        del = begin;
    }
}
template<typename T>
Node<T>* CircularLinkedList<T>::makeNode(const T &value) {
    Node<T>* temp = new Node<T>;
    temp->data = value;
    temp->next = 0;
    return temp;
}
template<typename T>
void CircularLinkedList<T>::insert(int pos, const T &value) {
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    Node<T>* add = makeNode(value);
    if(pos == 0){
        if(count == 0){
            begin = add;
            end = add;
            add->next = begin;
        }else{
            add->next = begin;
            begin = add;
            end->next = add;
        }

    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        add->next = cur->next;
        cur->next = add;
```

```
        // solo cuando se agrega al final
        if(pos == count ){
            end = add;
        }
    }
    count++;
}

template<typename T>
void CircularLinkedList<T>::erase(int pos) {
    if(pos < 0 || pos>=count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    if(pos == 0){
        Node<T>* del = begin;
        if(count <= 1){
            begin = 0;
            end = 0;
        }else{
            begin = begin->next;
            end->next = begin;
        }
        delete del;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        Node<T>* del = cur->next;
        cur->next = del->next;
        // solo cuando se borra el último elemento
        if(pos == count - 1)
            end = cur;
        delete del;
    }
    count--;
}
template<typename T>
T& CircularLinkedList<T>::get(int pos) const{
    if(pos < 0 || pos>count-1){
        cout << "Error! The position is out of range." << endl;
        assert(false);
    }
    if(pos == 0){
        return begin->data;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos; i++){
            cur = cur->next;
        }
        return cur->data;
    }
}
template<typename T>
void CircularLinkedList<T>::print() const{
    if(count == 0){
        cout << "List is empty." << endl;
        return;
    }
```

```
    Node<T>* cur = begin;
    while(cur != end){
        cout << cur->data << " ";
        cur = cur->next;
    }
    cout << cur->data << " ";
}
template<typename T>
int CircularLinkedList<T>::size() const {
    return count;
}
#endif
```

main.cpp

```
#include <iostream>
#include "CircularLinkedList.h"

using namespace std;
int main() {
    CircularLinkedList<string> nombres;
    nombres.insert(0, "nestor");
    nombres.insert(1, "eduardo");
    nombres.insert(2, "suat");
    nombres.insert(3, "rojas");

    nombres.erase(0);
    cout << "Begin: " << nombres.begin->data << endl;
    cout << "End: " << nombres.end->data << endl;
    cout << "Get: " << nombres.get(1) << endl;
    cout << "Size: " << nombres.size() << endl;
    nombres.print();
    cout << endl;
    return 0;
}
```

Colas Dobles

DoubleQueue.h

```
#ifndef LISTASMODIFICADAS_DOUBLEQUEUE_H
#define LISTASMODIFICADAS_DOUBLEQUEUE_H

#include "List.h"

template<typename T>
class DoubleQueue {
private:
    List<T> list;
public:
    void push_front(const T& value);
    void push_back(const T& value);
    void pop_front();
    void pop_back();
    T& front() const;
```

```
    T& back() const;
    int size() const;
    void print() const;
    bool isEmpty() const;
};


#endif //LISTASMODIFICADAS_DOUBLEQUEUE_H
```

DoubleQueue.cpp

```
#ifndef LISTASMODIFICADAS_DOUBLEQUEUE_CPP
#define LISTASMODIFICADAS_DOUBLEQUEUE_CPP
#include <iostream>
#include "DoubleQueue.h"

template<typename T>
void DoubleQueue<T>::push_front(const T &value) {
    list.insert(0, value);
}

template<typename T>
void DoubleQueue<T>::push_back(const T &value) {
    list.insert(list.size(), value);
}

template<typename T>
void DoubleQueue<T>::pop_front(){
    list.erase(0);
}

template<typename T>
void DoubleQueue<T>::pop_back(){
    list.erase(list.size()-1);
}

template<typename T>
T& DoubleQueue<T>::front() const {
    return list.get(0);
}

template<typename T>
T& DoubleQueue<T>::back() const {
    return list.get(list.size()-1);
}

template<typename T>
int DoubleQueue<T>::size() const {
    return list.size();
}

template<typename T>
void DoubleQueue<T>::print() const{
    list.print();
}

template<typename T>
```

```cpp
bool DoubleQueue<T>::isEmpty() const {
   return size() == 0 ;
}

#endif
```

main.cpp

```cpp
#include <iostream>
#include "List.cpp"
#include "DoubleQueue.cpp"

using namespace std;
int main() {
   DoubleQueue<string> queue;
   cout << boolalpha << queue.isEmpty() << endl;

   queue.push_back("Maria");
   queue.push_back("Juan");
   queue.push_front("Victor");
   queue.push_front("Mariana");

   cout << queue.size() << endl;
   cout << boolalpha << queue.isEmpty() << endl;
   queue.print();
   cout << endl;
   queue.push_back("Luis");
   queue.print();
   cout << endl;

   queue.pop_back();
   queue.pop_front();
   queue.print();
   cout << endl;

   cout << "Front: " << queue.front() << endl;
   cout << "Back: " << queue.back() << endl;
   return 0;
}
```

```
def verificarPalindromo(cadena):

    colaDobleCaracteres = ColaDoble()
    for  caracter in cadena:
        colaDobleCaracteres.agregarFinal(caracter)

    aunIguales = True

    while colaDobleCaracteres.tamano() > 1 and aunIguales:
        primero = colaDobleCaracteres.removerFrente()
        ultimo = colaDobleCaracteres.removerFinal()
        if primero != ultimo:
            aunIguales = False

    return aunIguales

print(verificarPalindromo("lsdkjfskf"))
print(verificarPalindromo("radar"))

```