__const_iterator.hpp

```cpp
#ifndef __CONST_ITERATOR_HPP__
#define __CONST_ITERATOR_HPP__

#include "franklist.h"

using namespace vhuk;

template <typename T>
FrankList<T>::const_iterator::const_iterator(Node* ptr)

    : base_iterator(ptr){}


template <typename T>
FrankList<T>::const_iterator::const_iterator(const base_iterator& rhv)

    : base_iterator(rhv.ptr){}

template <typename T>
FrankList<T>::const_iterator::const_iterator(base_iterator&& rhv)

    : base_iterator(rhv.ptr){}


template <typename T>
const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator=(const base_iterator& rhv)
{
    return (*this = std::move(rhv));
}

template <typename T>
const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator=(base_iterator&& rhv)
{
    if (this != &rhv)
        this->ptr = rhv.ptr;
    return (*this);
}

template <typename T>
typename FrankList<T>::const_reference FrankList<T>::const_iterator::operator*() const
{
    return (this->ptr->val);
}

template <typename T>
typename FrankList<T>::const_pointer FrankList<T>::const_iterator::operator->() const
{
    return &(this->ptr->val);
}

template <typename T>
const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator++()
{
    this->ptr = this->ptr->next;
    return (*this);
}

template <typename T>
const typename FrankList<T>::const_iterator FrankList<T>::const_iterator::operator++(value_type)
{
    FrankList::Node* tmp = this->ptr;

    ++(*this);

    return (base_iterator(tmp));
}

template <typename T>
const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator--()
{
    this->ptr = this->ptr->prev;
    return (*this);
}

template <typename T>
const typename FrankList<T>::const_iterator FrankList<T>::const_iterator::operator--(value_type)
{
    FrankList::Node* tmp = this->ptr;

    --(*this);

    return (base_iterator(tmp));
}


#endif // __CONST_ITERATOR_HPP__
```