DoublyLinkedNodeIteratorPS5.h

```
#pragma once

#include "DoublyLinkedNode.h"

template< class DataType >
class DoublyLinkedNodeIterator {
private:
    enum IteratorStates {
        BEFORE, DATA, AFTER
    };

    IteratorStates fState;

    typedef DoublyLinkedNode< DataType > Node;

    const Node *fLeftmost;
    const Node *fRightmost;
    const Node *fCurrent;

public:
    typedef DoublyLinkedNodeIterator< DataType > Iterator;

    DoublyLinkedNodeIterator(const Node &aList)
    {
        fLeftmost = &aList;
        while (&fLeftmost->getPrevious() != &Node::NIL)
            fLeftmost = &fLeftmost->getPrevious();
        fCurrent = fLeftmost;
        fRightmost = &aList;
        while (&fRightmost->getNext() != &Node::NIL)
            fRightmost = &fRightmost->getNext();

        fState = (fCurrent != &Node::NIL) ? DATA : AFTER;

    }

    const DataType &operator*() const    // dereference
    {
```

```
    return fCurrent->getValue();
}

Iterator &operator++()              // prefix increment
{
    if (fState == BEFORE)
        fCurrent = fLeftmost;
    else if (fState == DATA)
        fCurrent = &fCurrent->getNext();
    if (fCurrent == &Node::NIL)
        fState = AFTER;
    else
        fState = DATA;
    return *this;
}

Iterator operator++(int)            // postfix increment
{
    Iterator ITemp = *this;
    ++*this;
    return ITemp;
}

Iterator &operator--()              // prefix decrement
{
    if (fState == AFTER)
        fCurrent = fRightmost;
    else if (fState == DATA)
        fCurrent = &fCurrent->getPrevious();
    if (fCurrent == &Node::NIL)
        fState = BEFORE;
    else
        fState = DATA;
    return *this;
}

Iterator operator--(int)            // postfix decrement
{
    Iterator ITemp = *this;
    --*this;
```

```
    return ITemp;
}


bool operator==(const Iterator &aOtherIter) const
{
    return (fCurrent == aOtherIter.fCurrent) &&
        (fLeftmost == aOtherIter.fLeftmost) &&
        (fRightmost == aOtherIter.fRightmost) &&
        (fState == aOtherIter.fState);
}

bool operator!=(const Iterator &aOtherIter) const
{
    return !(*this == aOtherIter);
}

Iterator begin() const
{
    Iterator IBegin = rend();
    ++IBegin;
    return IBegin;
}

Iterator end() const
{
    Iterator IEnd = *this;
    IEnd.fCurrent = &Node::NIL;
    IEnd.fState = AFTER;
    return IEnd;
}

Iterator rbegin() const
{
    Iterator IRBegin = end();
    IRBegin--;
    return IRBegin;
}

Iterator rend() const
```

```
    {
        Iterator IREnd = *this;
        IREnd.fState = BEFORE;
        IREnd.fCurrent = &Node::NIL;
        return IREnd;
    }
};
```

```
Forward iteration I:
One
Two
Three
Four
Five
Six
Backward iteration I:
Six
Five
Four
Three
Two
One
Forward iteration II:
One
Two
Three
Four
Five
Six
Backward iteration II:
Six
Five
Four
Three
Two
One
Yes
Yes
Yes
Program ended with exit code: 0
```