```
****************************************************************************
* PROGRAMMED BY : Nick Reardon
* CLASS         : CS1D
* SECTION       : MW - 2:30p
* Assignment #4 : Deque To Queue
****************************************************************************

                        Assignment #4 - Deque To Queue

    Implement a Queue interface with a class that is based on a
Deque using a wrapper. Do not use the STL. Highlight your Deque
and Queue classes.


           Queue  method         | Deque Implementation
          ----------------------|----------------------
           size()               | size()
          ----------------------|----------------------
           empty()              | empty()
          ----------------------|----------------------
           front()              | front()
          ----------------------|----------------------
           enqueue()            | insertBack()
          ----------------------|----------------------
           dequeue()            | eraseFront()
          ----------------------|----------------------

    Test all the member functions (size(), empty(), front(),
enqueue, and dequeue) of the queue class with the following
data.

        String string1 = "A man, a plan, a canal, Panamaö;
        String string2 = ôWas it a car or a cat I saw?";
        String string3 = ôSit on a potato pan, Otis";
        String string4 = ôNo lemon, no melon";


Due February 10th


****************************************************************************



 --- Using Deque implementation ---

deque currently empty - empty() method called in printAll() to show

Printing deque:
Container is empty

Reading into deque from file using insertBack() method

```

```
Printing deque:
A man, a plan, a canal, Panama
Was it a car or a cat I saw?
Sit on a potato pan, Otis
No lemon, no melon

deque - calling size method
Size is currently: 4

deque - Calling front() method
front value is currently: A man, a plan, a canal, Panama

deque - Calling eraseFront() method
front value is now currently: Was it a car or a cat I saw?

deque - Calling eraseFront() method

Printing deque:
Sit on a potato pan, Otis
No lemon, no melon


 --- Using Queue wrapper of deque implementation ---

Queue currently empty - empty() method called in printAll() to show

Printing Queue:
Container is empty

Reading into queue from file using enqueue() method

Printing Queue:
A man, a plan, a canal, Panama
Was it a car or a cat I saw?
Sit on a potato pan, Otis
No lemon, no melon

queue - calling size method
Size is currently: 4

queue - Calling front() method
front value is currently: A man, a plan, a canal, Panama

queue - Calling dequeue() method
front value is now currently: Was it a car or a cat I saw?

queue - Calling dequeue() method

Printing Queue:
Sit on a potato pan, Otis
No lemon, no melon

```

```
Press any key to continue . . .
```

```
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #4    : Deque To Queue
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 10 / 20
 *****************************************************************************/
#ifndef _MAIN_H_
#define _MAIN_H_

//Standard includes
#include <iostream>
#include <iomanip>
#include <string>
#include "PrintHeader.h"

//Program Specific
#include "LinkedDeque.h"



#endif // _HEADER_H_
```

```cpp
/**************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #4    : Deque To Queue
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 10 / 20
 **************************************************************************/
#include "main.h"

using std::cout; using std::endl;


int main()
{

    /*
     * HEADER OUTPUT
     */
    PrintHeader(cout, "Prompt.txt");

    /*********************************************************************/

    LinkedDeque<std::string> deque;
    LinkedQueue<std::string> queue;

    std::ifstream iFile;

    cout << endl << " --- Using Deque implementation ---" << endl << endl;


    cout << "deque currently empty - empty() method called in printAll() to show" 
       << endl;
    PrintWithLabel("Printing deque:", deque, cout);

    //---------------------------------------------------

    cout << "Reading into deque from file using insertBack() method" << endl;

    std::string temp;
    iFile.open("Input.txt");

    while ( getline(iFile, temp) )
    {
        deque.insertBack(temp);
    }
    iFile.close();

    PrintWithLabel("Printing deque:", deque, cout);

    //---------------------------------------------------

    cout << "deque - calling size method" << endl;
    cout << "Size is currently: " << deque.size() << endl << endl;

    //-------------------------------------------------

    cout << "deque - Calling front() method" << endl;
    cout << "front value is currently: " << deque.front() << endl << endl;

    //-------------------------------------------------

    cout << "deque - Calling eraseFront() method" << endl;
    deque.eraseFront();
    cout << "front value is now currently: " << deque.front() << endl << endl;

    cout << "deque - Calling eraseFront() method" << endl;
    deque.eraseFront();
    PrintWithLabel("Printing deque:", deque, cout);

    //-------------------------------------------------
    //-------------------------------------------------

    cout << endl << " --- Using Queue wrapper of deque implementation ---" <<
      endl << endl;


    cout << "Queue currently empty - empty() method called in printAll() to show"
       << endl;
    PrintWithLabel("Printing Queue:", queue, cout);

    //-------------------------------------------------

    cout << "Reading into queue from file using enqueue() method" << endl;

    temp.clear();
    iFile.open("Input.txt");

    while (getline(iFile, temp))
    {
        queue.enqueue(temp);
    }
    iFile.close();

    PrintWithLabel("Printing Queue:", queue, cout);

    //-------------------------------------------------

    cout << "queue - calling size method" << endl;
    cout << "Size is currently: " << queue.size() << endl << endl;

    //-------------------------------------------------

    cout << "queue - Calling front() method" << endl;
    cout << "front value is currently: " << queue.front() << endl << endl;
```

```cpp

    //------------------------------------------------

    cout << "queue - Calling dequeue() method" << endl;
    queue.dequeue();
    cout << "front value is now currently: " << queue.front() << endl << endl;

    cout << "queue - Calling dequeue() method" << endl;
    queue.dequeue();
    PrintWithLabel("Printing Queue:", queue, cout);


    system("pause");
    return 0;
}


```

```
/***************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #4    : Deque To Queue
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 10 / 20
 ***************************************************************************/
#ifndef _LINKEDDEQUE_H_
#define _LINKEDDEQUE_H_
#include <exception>
#include <sstream>



class Except : virtual public std::runtime_error {

protected:

    int error_number;               ///< Error number
    int error_offset;               ///< Error offset

public:

    /** Constructor (C++ STL string, int, int).
     *  @param  msg      The error message
     *  @param  err_num  Error number
     *  @param  err_off  Error offset
     */
    explicit
        Except(const std::string& msg, int err_num, int err_off) :
        std::runtime_error(msg)
    {
        error_number = err_num;
        error_offset = err_off;

    }

    /** Destructor.
     *  Virtual to allow for subclassing.
     */
    virtual ~Except() throw () {}

    /** Returns error number.
     *  @return #error_number
     */
    virtual int getErrorNumber() const throw() {
        return error_number;
    }

    /**Returns error offset.
     * @return #error_offset
     */
    virtual int getErrorOffset() const throw() {
        return error_offset;
    }

};

enum ERROR_TYPE
{
    DEFUALT,
    FULL,
    EMPTY,
    OUT_OF_RANGE
};

template <class Type>
struct Node
{
    Type value;

    Node<Type>* prev;
    Node<Type>* next;

    Node<Type>(const Type& newValue, Node<Type>* prevNode, Node<Type>* nextNode)
    {
        value = newValue;

        prev = prevNode;
        next = nextNode;
    }

};

template <class Type>
class LinkedDeque
{
private:
    Node<Type>* head;
    Node<Type>* tail;

    int capacity;
    int currentSize;


protected:

public:

    LinkedDeque<Type>(const int newCapacity = 32);

    LinkedDeque<Type>(const LinkedDeque<Type>& otherDeque);

    ~LinkedDeque();
```

```cpp

    void destroy();

    bool empty() const;
    bool full() const;
    int size() const;

    void insertBefore(const Type& newItem, const int index);
    void insertAfter(const Type& newItem, const int index);

    void insertFront(const Type& newItem);
    void insertBack(const Type& newItem);

    void eraseFront();
    void eraseBack();


    Type front() const;
    Type back() const;

    void printAll(std::ostream& output) const;

};


template<class Type>
inline LinkedDeque<Type>::LinkedDeque(const int newCapacity)
{
    head = nullptr;
    tail = nullptr;

    capacity = newCapacity;
    currentSize = 0;
}

template<class Type>
inline LinkedDeque<Type>::LinkedDeque(const LinkedDeque<Type>& otherDeque)
{
    capacity = otherDeque.capacity;
    currentSize = 0;

    head = nullptr;
    tail = nullptr;

    for (Node<Type>* temp = otherDeque.head; temp != nullptr; temp = temp->next)
    {
        insertBack(temp->value);
    }
}


template<class Type>
```

```cpp
inline LinkedDeque<Type>::~LinkedDeque()
{
    destroy();
}

template<class Type>
inline bool LinkedDeque<Type>::empty() const
{
    bool value = (currentSize == 0 && head == nullptr && tail == nullptr);

    return value;
}

template<class Type>
inline bool LinkedDeque<Type>::full() const
{
    bool value = (currentSize == capacity);

    return value;
}

template<class Type>
inline void LinkedDeque<Type>::destroy()
{
    for (Node<Type>* temp = head; temp != nullptr; )
    {
        Node<Type>* hold = temp;
        temp = temp->next;
        delete hold;
    }

    head = nullptr;
    tail = nullptr;

}

template<class Type>
inline void LinkedDeque<Type>::insertBefore(const Type& newItem, const int index)
{
    if (empty())
    {
        throw Except("container is empty", EMPTY, 5);
    }
    else if (full())
    {
        throw Except("container is full", FULL, 5);
    }
    else
    {
        if (index <= (size() - 1 / 2))
        {
            Node<Type>* ptr = head;
            for (int i = 0; i < index; i++)
            {
                ptr = ptr->next;
            }
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            temp->prev = ptr->prev;
            temp->next = ptr;

            ptr->prev = temp;
            temp->prev->next = temp;

            temp = nullptr;
            ptr = nullptr;

            currentSize++;
        }
        else
        {
            Node<Type>* ptr = head;
            for (int i = size() - 1; i >= size() - 1 - index; i--)
            {
                ptr = ptr->next;
            }
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            temp->prev = ptr->prev;
            temp->next = ptr;

            ptr->prev = temp;
            temp->prev->next = temp;

            temp = nullptr;
            ptr = nullptr;

            currentSize++;
        }
    }

}

template<class Type>
inline void LinkedDeque<Type>::insertAfter(const Type& newItem, const int index)
{
    if (empty())
    {
        throw Except("container is empty", EMPTY, 5);
    }
    else if (full())
    {
        throw Except("container is full", FULL, 5);
    }
    else
    {
        if (index <= (size() - 1 / 2))
        {
            Node<Type>* ptr = head;
            for (int i = 1; i <= index; i++)
            {
                ptr = ptr->next;
            }
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            temp->next = ptr->next;
            ptr->next = temp;

            temp->prev = ptr;
            temp->next->prev = temp;

            temp = nullptr;
            ptr = nullptr;

            currentSize++;
        }
        else
        {
            Node<Type>* ptr = head;
            for (int i = size() - 1; i >= size() - 1 - index; i--)
            {
                ptr = ptr->next;
            }
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            temp->next = ptr->next;
            ptr->next = temp;

            temp->prev = ptr;
            temp->next->prev = temp;

            temp = nullptr;
            ptr = nullptr;

            currentSize++;
        }
    }

}

template<class Type>
inline void LinkedDeque<Type>::insertFront(const Type& newItem)
{

    if (!full())
    {
        if (head == 0)
        {
            head = new Node<Type>(newItem, nullptr, nullptr);
            tail = head;
        }
        else
        {
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            head->prev = temp;

            temp->next = head;
            temp->prev = nullptr;

            head = temp;

            temp = nullptr;
        }
        currentSize++;

    }
    else
    {
        throw Except("container is full", FULL, 5);
    }
}

template<class Type>
inline void LinkedDeque<Type>::insertBack(const Type& newItem)
{

    if (!full())
    {
        if (head == 0)
        {
            head = new Node<Type>(newItem, nullptr, nullptr);
            tail = head;
        }
        else
        {
            Node<Type>* temp = new Node<Type>(newItem, nullptr, nullptr);

            tail->next = temp;

            temp->prev = tail;
            temp->next = nullptr;

            tail = temp;

            temp = nullptr;
        }
        currentSize++;

    }
    else
    {
        throw Except("container is full", FULL, 5);
    }

}

template<class Type>
inline void LinkedDeque<Type>::eraseFront()
{

    if (!empty())
    {
        if (head == tail)
        {
            delete head;
            head = nullptr;
            tail = nullptr;
        }
        else
        {
            Node<Type>* temp = head;
            head = head->next;
            head->prev = nullptr;
            delete temp;
        }
        currentSize--;

    }
    else
    {
        throw Except("container is empty", EMPTY, 5);
    }

}

template<class Type>
inline void LinkedDeque<Type>::eraseBack()
{
    if (!empty())
    {
        if (head == tail)
        {
            delete head;
            head = nullptr;
            tail = nullptr;
        }
        else
        {
            Node<Type>* temp = tail;
            tail = tail->prev;
            tail->next = nullptr;
            delete temp;
        }
        currentSize--;

    }
    else
    {
        throw Except("container is empty", EMPTY, 5);
    }

}

template<class Type>
inline int LinkedDeque<Type>::size() const
{
    return currentSize;
}


template<class Type>
inline Type LinkedDeque<Type>::front() const
{
    return head->value;
}

template<class Type>
inline Type LinkedDeque<Type>::back() const
{
    return tail->value;
}

template<class Type>
inline void LinkedDeque<Type>::printAll(std::ostream& output) const
{
    for (Node<Type>* temp = head; temp != nullptr; temp = temp->next)
    {
        output << temp->value << '\n';
    }
}



template<class Type>
class LinkedQueue : protected LinkedDeque<Type>
{
private:
    LinkedDeque<Type> deque;

protected:

```

```
public:

    LinkedQueue<Type>(const int newCapacity = 32);

    LinkedQueue<Type>(LinkedQueue<Type>& otherQueue);
    LinkedQueue<Type>(LinkedDeque<Type>& otherDeque);

    ~LinkedQueue();

    void destroy();

    bool empty() const;
    bool full() const;
    int size() const;

    void enqueue(const Type& newItem);
    void dequeue();

    Type front() const;
    Type back() const;

    void printAll(std::ostream& output) const;

};



#endif // !_LINKEDDEQUE_H_

template<class Type>
inline LinkedQueue<Type>::LinkedQueue(const int newCapacity) :
    LinkedDeque<Type>(newCapacity)
{

}

template<class Type>
inline LinkedQueue<Type>::LinkedQueue(LinkedQueue<Type>& otherQueue) :
    LinkedDeque<Type>(otherQueue)
{

}

template<class Type>
inline LinkedQueue<Type>::LinkedQueue(LinkedDeque<Type>& otherDeque) :
    LinkedDeque<Type>(otherDeque)
{

}
template<class Type>
inline void LinkedQueue<Type>::destroy()
{
    deque.destroy();
}

template<class Type>
inline LinkedQueue<Type>::~LinkedQueue()
{
    deque.destroy();
}

template<class Type>
inline bool LinkedQueue<Type>::empty() const
{
    return deque.empty();
}

template<class Type>
inline bool LinkedQueue<Type>::full() const
{
    return deque.full();
}

template<class Type>
inline int LinkedQueue<Type>::size() const
{
    return deque.size();
}

template<class Type>
inline void LinkedQueue<Type>::enqueue(const Type& newItem)
{
    deque.insertBack(newItem);
}

template<class Type>
inline void LinkedQueue<Type>::dequeue()
{
    deque.eraseFront();
}

template<class Type>
inline Type LinkedQueue<Type>::front() const
{
    return deque.front();
}

template<class Type>
inline Type LinkedQueue<Type>::back() const
{
    return deque.back();
}

template<class Type>
```

```
inline void LinkedQueue<Type>::printAll(std::ostream& output) const
{
    deque.printAll(output);
}



template<class Type>
void PrintWithLabel(const std::string& label, const LinkedDeque<Type> &container, 
   std::ostream& output)
{
    output << '\n' << label << '\n';

    if (container.empty())
    {
        output << "Container is empty \n\n";

    }
    else
    {
        container.printAll(output);
        output << '\n';
    }
}

template<class Type>
void PrintWithLabel(const std::string& label, const LinkedQueue<Type>& container, 
   std::ostream& output)
{
    output << '\n' << label << '\n';

    if (container.empty())
    {
        output << "Container is empty \n\n";

    }
    else
    {
        container.printAll(output);
        output << '\n';
    }
}
```

```
/****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #4    : Deque To Queue
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 10 / 20
 ****************************************************************************/
#ifndef _PRINTHEADER_H_
#define _PRINTHEADER_H_

#include <iostream>
#include <iomanip>
#include <ostream>
#include <string>
#include <fstream>

/****************************************************************************
* PrintHeader
* ---------------------------------------------------------------------------
* This function will output a class header through the use of ostream.
* It also will output the program description
* ----------------------------------------------------------------------------
*  Call
* ------
* The function call requires 1 parameters. The following example uses an
* output file in the ostream parameter. Ex:
*
*       PrintHeader (oFile);
*
* ---------------------------------------------------------------------------
*  Output
* --------
*   The function will output as follows. Ex:
*
*       *********************************************************
*       * PROGRAMMED BY : Parsa Khazravi and Nick Reardon
*       * CLASS         : CS1B
*       * SECTION       : MW: 7:30pm
*       * Lab #3        : Functions - GCD
*       ********************************************************
*
* ---------------------------------------------------------------------------
* CONSTANTS
* ---------------------------------------------------------------------------
* OUTPUT - USED FOR CLASS HEADING
* ---------------------------------------------------------------------------
* PROGRAMMER        : Name(s) of programmer(s)  - Nick Reardon
* SECTION           : Class times               - MW - 7:30p
* CLASS             : Class label               - CS1B
* PROGRAM_NUM       : # of the program
* PROGRAM_NAME      : Title of the program
* PROGRAM_TYPE      : Type of program - Lab, Assignment, etc.
```

```cpp
*
* ---------------------------------------------------------------------------
* MAX_OUTPUT        : Max movies to be output at once
*****************************************************************************/
const std::string PROGRAMMER = "Nick Reardon";
const std::string SECTION = "MW - 2:30p";
const std::string CLASS = "CS1D";
const int PROGRAM_NUM = 4;
const std::string PROGRAM_NAME = "Deque To Queue";
const std::string PROGRAM_TYPE = "Assignment";


void PrintHeader(std::ostream &output, std::string inputText)
{
    std::string typeNum = PROGRAM_TYPE + " #" + std::to_string(PROGRAM_NUM);

    output << std::left
        << std::string(76, '*')
        << std::endl
        << "* PROGRAMMED BY : " << PROGRAMMER << std::endl
        << "* " << std::setw(14) << "CLASS" << ": " << CLASS << std::endl
        << "* " << std::setw(14) << "SECTION" << ": " << SECTION << std::endl
        << "* " << std::setw(14) << typeNum << ": " << PROGRAM_NAME << std::endl
        << std::string(76, '*')
        << std::endl << std::endl
        << std::string(((76 - typeNum.length() - PROGRAM_NAME.length() ) / 2), ' ')
        << typeNum + " - " + PROGRAM_NAME
        << std::endl << std::endl
        << std::ifstream(inputText).rdbuf()
        << std::endl
        << std::string(76, '*')
        << std::endl << std::endl;

}

#endif //_PRINTHEADER_H_
```