```
/*
 * Class name:  DLNode.h
 * Class description:  Individual node entry for a doubly linked list.  Provides variables representing the previous
 *      and next nodes within the list, along with a variable for the data being held by the node.  The class is
 *      templated so that it may be used with doubly linked lists of various data types (such as int, char, double,
 *      float, etc.).
 *
 * Programmer:  Chad Philip Johnson
 * Date Created:  Thursday, March 28th, 2013
 * Last Date Modified:  Sunday, April 14th, 2013
 *
 */

#include <cstdlib>

using namespace std;

#ifndef DLNODE_H
#define DLNODE_H

template<typename T>
class DLNode
{

    public:

    /***** constructor/destructor definitions *****/

        /**
         * Default constructor.  Sets the value of contents to zero and the previous and next node pointer values to NULL.
         */
        DLNode()
        : contents( 0 ), pobjDLNodePrevious( NULL ), pobjDLNodeNext( NULL )
        {
            /* empty */
        }

        /**
         * Overloaded constructor.  Assigns a given value to the node and sets the previous and next node pointer values to NULL.
         */
        DLNode( T contents )
        : contents( contents ), pobjDLNodePrevious( NULL ), pobjDLNodeNext( NULL )
        {
            /* empty */
        }

        /**
         * Default destructor (does nothing--no memory within the node to delete).
         */
        virtual ~DLNode()
```

```
        {
            /* empty */
        }

    /***** accessor/mutator function definitions *****/

        /**
         * Set the contents of the node.
         * @param contents The new value to be assigned to the node.
         */
        void setContents( T contents )
        {
            this->contents = contents;
        }

        /**
         * Report the current value of the node.
         * @return The current value of the node.
         */
        T getContents() const
        {
            return this->contents;
        }

        /**
         * Associate the current node with the next node in the doubly linked list.
         * @param pobjDLNodeNext The memory address of the next node in the doubly linked list.
         */
        void setNext( DLNode<T>* pobjDLNodeNext )
        {
            this->pobjDLNodeNext = pobjDLNodeNext;
        }

        /**
         * Report the memory address of the next node in the list.
         * @return The memory address of the next node in the list.
         */
        DLNode<T>* getNext() const
        {
            return this->pobjDLNodeNext;
        }

        /**
         * Associate the current node with the previous node in the doubly linked list.
         * @param pobjDLNodePrevious The memory address of the previous node in the doubly linked list.
         */
        void setPrevious( DLNode<T>* pobjDLNodePrevious )
        {
            this->pobjDLNodePrevious = pobjDLNodePrevious;
        }
```

```
        /**
         * Friend overloaded insertion operator.  Displays all of the data members contained within each
         *      node of the doubly linked list.
         * @return The memory address of the previous node in the list.
         */
        DLNode<T>* getPrevious() const
        {
            return this->pobjDLNodePrevious;
        }

    private:

    /***** private variable definitions *****/

        T contents;
        DLNode<T>* pobjDLNodePrevious;
        DLNode<T>* pobjDLNodeNext;

};

#endif
```