```cpp
/*
 * Class name:  DLList.h
 * Class description:  Serves as the apparatus for the node entries within a doubly linked list.  Provides variables
 *      indicating the head and tail nodes, along with the total size of the list.  The class is templated so that various
 *      data types can be used as doubly linked lists (such as int, double, float, char, etc.).
 *
 * Programmer:  Chad Philip Johnson
 * Date Created:  Thursday, March 28th, 2013
 * Last Date Modified:  Sunday, April 14th, 2013
 *
 * Sources Used:
 *      DLNode.h
 *          - to create instances of individual nodes containing values of the supplied type within a doubly linked list
 */

#include <iostream>

#include "DLNode.h"

using namespace std;

#ifndef DLLIST_H
#define DLLIST_H

template<typename T>
class DLList
{

    public:

    /***** constructor/destructor declarations *****/

        /**
         * Default constructor.  Sets the size of the doubly linked list to zero.
         */
        DLList()
        : uintCount( 0 ), pobjDLNodeHead( NULL ), pobjDLNodeTail( NULL )
        {
            /* empty */
        }

        /**
         * Default destructor.  Calls the clear() function to free memory associated with the current
         *      instance of the list.
         */
        ~DLList()
        {
            this->clear();
        }

```

```
    /***** overloaded operator definitions *****/

    /**
     * Friend overloaded insertion operator.  Displays all of the data members contained within each
     *      node of the doubly linked list.
     * @param objostreamOut The associated output stream.
     * @param objDLListSrc The doubly linked list to have its node data members reported.
     * @return The modified output stream containing reported values.
     */
    friend ostream& operator << ( ostream& objostreamOut, const DLList<T>& objDLListSrc )
    {
        if( objDLListSrc.uintCount > 0 )
        {
            DLNode<T>* pobjDLNodeSelectedNode   = objDLListSrc.pobjDLNodeHead;

            while( pobjDLNodeSelectedNode->getNext() != NULL )
            {
                objostreamOut << pobjDLNodeSelectedNode->getContents() << ",";
                pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();

            }
            objostreamOut << pobjDLNodeSelectedNode->getContents() << endl;
        }

        return objostreamOut;
    }

    /***** public function declarations *****/

        /**
         * Add a node to the front of the doubly linked list.
         * @param contents The value to be assigned to the added node.
         */
        void pushFront( T contents )
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeSelectedNode   = this->pobjDLNodeHead;

                this->pobjDLNodeHead = new DLNode<T>( contents );
                this->pobjDLNodeHead->setNext( pobjDLNodeSelectedNode );
                pobjDLNodeSelectedNode->setPrevious( this->pobjDLNodeHead );
            }
            else
            {
                this->pobjDLNodeHead = new DLNode<T>( contents );
                this->pobjDLNodeTail = this->pobjDLNodeHead;
            }

            uintCount++;
```

```
        }

        /**
         * Add a node to the end of the doubly linked list.
         * @param contents The value to be assigned to the added node.
         */
        void pushBack( T contents )
        {
            if( uintCount > 0 )
            {
                this->pobjDLNodeTail->setNext( new DLNode<T>( contents ) );
                this->pobjDLNodeTail->getNext()->setPrevious( this->pobjDLNodeTail );
                this->pobjDLNodeTail = this->pobjDLNodeTail->getNext();
            }
            else
            {
                this->pobjDLNodeTail = new DLNode<T>( contents );
                this->pobjDLNodeHead = this->pobjDLNodeTail;
            }

            uintCount++;
        }

        /**
         * Add a node into a sorted (ascending) order.
         * @param contents The value to be assigned to the added node.
         */
        void insert( T contents )
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeSelectedNode   = this->pobjDLNodeHead;
                DLNode<T>* pobjDLNodeInsertionNode  = NULL;

                while( pobjDLNodeSelectedNode->getNext() != NULL )
                {
                    if( ( ( pobjDLNodeSelectedNode->getContents() < contents ) && ( pobjDLNodeSelectedNode->getNext()->
                    getContents() > contents ) )
                            || ( pobjDLNodeSelectedNode->getContents() == contents ) )
                    {
                        pobjDLNodeInsertionNode = new DLNode<T>( contents );

                        pobjDLNodeInsertionNode->setNext( pobjDLNodeSelectedNode->getNext() );
                        pobjDLNodeSelectedNode->getNext()->setPrevious( pobjDLNodeInsertionNode );

                        pobjDLNodeInsertionNode->setPrevious( pobjDLNodeSelectedNode );
                        pobjDLNodeSelectedNode->setNext( pobjDLNodeInsertionNode );

                        uintCount++;
                        return;
```

```
                    }
                    else
                    {
                        pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();
                    }
                }

                if( pobjDLNodeSelectedNode->getContents() <= contents )
                {
                    pushBack( contents );
                    return;
                }
                else if( pobjDLNodeSelectedNode->getContents() > contents )
                {
                    pushFront( contents );
                    return;
                }
            }
            else
            {
                pushFront( contents );
                return;
            }
        }

        /**
         * Check to see if a value is contained within the doubly linked list.
         * @param target The value to be checked.
         * @return bool True if value exists in the list; false if not.
         */
        bool get( T target )
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeSelectedNode   = pobjDLNodeHead;

                while( pobjDLNodeSelectedNode->getNext() != NULL )
                {
                    if( pobjDLNodeSelectedNode->getContents() == target )
                    {
                        return true;
                    }
                    else
                    {
                        pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();
                    }
                }

                if( pobjDLNodeSelectedNode->getContents() == target )
                {
```

```
                    return true;
                }
            }

            return false;
        }

        /**
         * Remove the node at the front of the list.
         */
        void popFront()
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeDeleteNode = this->pobjDLNodeHead;
                if( pobjDLNodeDeleteNode->getNext() != NULL )
                {
                    this->pobjDLNodeHead    = pobjDLNodeDeleteNode->getNext();
                    pobjDLNodeHead->setPrevious( NULL );
                }

                delete pobjDLNodeDeleteNode;
                uintCount--;
            }
            else
            {
                return;
            }
        }

        /**
         * Remove the node at the end of the list.
         */
        void popBack()
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeDeleteNode = this->pobjDLNodeTail;
                if( pobjDLNodeDeleteNode->getPrevious() != NULL )
                {
                    this->pobjDLNodeTail    = pobjDLNodeDeleteNode->getPrevious();
                    pobjDLNodeTail->setNext( NULL );
                }

                delete pobjDLNodeDeleteNode;
                uintCount--;
            }
            else
            {
                return;
```

```
            }
        }

        /**
         * Remove the first appearance of a value within the list, if it exists.
         * @param target The value to be removed.
         * @return bool True if successful; false if not.
         */
        bool removeFirst( T target )
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeSelectedNode   = this->pobjDLNodeHead;
                DLNode<T>* pobjDLNodeDeleteNode     = NULL;

                if( pobjDLNodeSelectedNode->getContents() == target )
                {
                    this->pobjDLNodeHead = pobjDLNodeSelectedNode->getNext();

                    if( pobjDLNodeHead != NULL )
                    {
                        this->pobjDLNodeHead->setPrevious( NULL );
                    }

                    pobjDLNodeDeleteNode = pobjDLNodeSelectedNode;
                }
                else
                {
                    while( pobjDLNodeSelectedNode->getNext() != NULL )
                    {
                        if( pobjDLNodeSelectedNode->getNext()->getContents() == target )
                        {
                            pobjDLNodeDeleteNode    = pobjDLNodeSelectedNode->getNext();
                            pobjDLNodeSelectedNode->setNext( pobjDLNodeDeleteNode->getNext() );

                            if( pobjDLNodeDeleteNode->getNext() != NULL )
                            {
                                pobjDLNodeDeleteNode->getNext()->setPrevious( pobjDLNodeSelectedNode );
                            }

                            break;
                        }
                        else
                        {
                            pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();
                        }
                    }

                    if( pobjDLNodeDeleteNode == this->pobjDLNodeTail )
                    {
```

```
                        this->pobjDLNodeTail = pobjDLNodeDeleteNode->getPrevious();
                        this->pobjDLNodeTail->setNext( NULL );
                    }
                }

                if( pobjDLNodeDeleteNode != NULL )
                {
                    delete pobjDLNodeDeleteNode;
                    uintCount--;
                    return true;
                }
            }

            return false;
        }

        /**
         * Remove all occurrences of a value within a list, provided at least one exists.
         * @param target The value to be removed.
         * @return bool True if value was removed; false if not.
         */
        bool removeAll( T target )
        {
            if( uintCount > 0 )
            {
                DLNode<T>* pobjDLNodeSelectedNode   = this->pobjDLNodeHead;
                DLNode<T>* pobjDLNodeDeleteNode     = NULL;

                while( pobjDLNodeSelectedNode->getContents() == target )
                {
                    pobjDLNodeDeleteNode    = pobjDLNodeSelectedNode;
                    this->pobjDLNodeHead    = pobjDLNodeDeleteNode->getNext();

                    delete pobjDLNodeDeleteNode;
                    uintCount--;

                    pobjDLNodeSelectedNode = this->pobjDLNodeHead;
                    if( pobjDLNodeSelectedNode == NULL )
                    {
                        this->pobjDLNodeTail    = NULL;
                        return true;
                    }
                }

                if( pobjDLNodeSelectedNode->getPrevious() != NULL )
                {
                    this->pobjDLNodeHead->setPrevious( NULL );
                }

                pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();
```

```
350
351                 if( pobjDLNodeSelectedNode != NULL )
352                 {
353                     while( pobjDLNodeSelectedNode->getNext() != NULL )
354                     {
355                         if( pobjDLNodeSelectedNode->getContents() == target )
356                         {
357                             pobjDLNodeDeleteNode    = pobjDLNodeSelectedNode;
358
359                             pobjDLNodeDeleteNode->getPrevious()->setNext( pobjDLNodeDeleteNode->getNext() );
360                             pobjDLNodeDeleteNode->getNext()->setPrevious( pobjDLNodeDeleteNode->getPrevious() );
361
362                             pobjDLNodeSelectedNode  = pobjDLNodeDeleteNode->getPrevious();
363
364                             delete pobjDLNodeDeleteNode;
365                             uintCount--;
366                         }
367
368                         pobjDLNodeSelectedNode = pobjDLNodeSelectedNode->getNext();
369                     }
370
371                     if( pobjDLNodeSelectedNode->getContents() == target )
372                     {
373                         pobjDLNodeDeleteNode = pobjDLNodeSelectedNode;
374                         pobjDLNodeDeleteNode->getPrevious()->setNext( NULL );
375                         this->pobjDLNodeTail = pobjDLNodeDeleteNode->getPrevious();
376
377                         delete pobjDLNodeDeleteNode;
378                         uintCount--;
379                     }
380                 }
381
382                 if( pobjDLNodeDeleteNode != NULL )
383                 {
384                     return true;
385                 }
386             }
387
388             return false;
389         }
390
391         /**
392          * Free all memory associated with this doubly linked list and reset the list to its default (empty) configuration.
393          */
394         void clear()
395         {
396             if( uintCount > 0 )
397             {
398                 DLNode<T>* pobjDLNodeDeleteNode     = this->pobjDLNodeHead;
399                 DLNode<T>* pobjDLNodeSelectedNode       = this->pobjDLNodeHead->getNext();
```

```
                do
                {
                    delete pobjDLNodeDeleteNode;

                    pobjDLNodeDeleteNode    = pobjDLNodeSelectedNode;

                    if( pobjDLNodeDeleteNode != NULL )
                    {
                        pobjDLNodeSelectedNode  = pobjDLNodeSelectedNode->getNext();
                    }
                }
                while( pobjDLNodeDeleteNode != NULL );

                this->pobjDLNodeHead    = NULL;
                this->pobjDLNodeTail    = NULL;
                uintCount = 0;
            }
        }

    /***** accessor/mutator function declarations *****/

        /**
         * Report the current size of the doubly linked list.
         * @return unsigned int The current size of the list.
         */
        unsigned int getSize() const
        {
            return this->uintCount;
        }

        /**
         * Report the data stored within the node at the front of the list.
         * @return T The value stored within the node at the front of the list.
         */
        T getFront() const
        {
            if( uintCount == 0 )
            {
                throw "LIST EMPTY";
            }
            else
            {
                return this->pobjDLNodeHead->getContents();
            }
        }

        /**
         * Report the data stored within the node at the end of the list.
         * @return T The value stored within the node at the end of the list.
```

```
450          */
451         T getBack() const
452         {
453             if( uintCount == 0 )
454             {
455                 throw "LIST EMPTY";
456             }
457             else
458             {
459                 return this->pobjDLNodeTail->getContents();
460             }
461         }
462
463     private:
464
465     /***** private variable declarations *****/
466
467         unsigned int uintCount;
468         DLNode<T>* pobjDLNodeHead;
469         DLNode<T>* pobjDLNodeTail;
470
471 };
472
473 #endif
474
```