```
#include "SList.h"

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

SList::SList()
: head( NULL ), size( 0 )
{
    /* empty */
}

SList::~SList()
{
    this->clear();
}

/***** public function definitions *****/

void SList::insertHead( int headValue )
{
    if( size > 0 )
    {
        SLNode *tempValue   = this->head;

        this->head = new SLNode( headValue );
        size++;
        head->setNextNode( tempValue );
    }
    else
    {
        this->head = new SLNode( headValue );
        size++;
    }
}

void SList::insertTail( int tailValue )
{
    if( size > 0 )
    {
        SLNode *tempValue   = head;

        while( tempValue->getNextNode() != NULL )
        {
            tempValue = tempValue->getNextNode();
        }

        tempValue->setNextNode( new SLNode( tailValue ) );
    }
    else
    {
        head = new SLNode( tailValue );
```

```
    }

    size++;
}

void SList::removeHead()
{
    if( size > 0 )
    {
        SLNode *tempValue   = (this->head)->getNextNode();

        delete this->head;
        this->head = tempValue;
        size--;
    }
}

void SList::removeTail()
{
    if( size > 1 )
    {
        SLNode *tempValue   = head;

        while( (tempValue->getNextNode())->getNextNode() != NULL )
        {
            tempValue = tempValue->getNextNode();
        }

        delete tempValue->getNextNode();
        tempValue->setNextNode( NULL );
        size--;
    }
    else if( size == 1 )
    {
        delete this->head;
        size--;
    }
}

void SList::insert( int insertValue )
{
    if( size > 0 )
    {
        SLNode* tempValue       = head;
        SLNode* insertionNode   = NULL;

        while( tempValue->getNextNode() != NULL )
        {
            if( ( ( tempValue->getContents() < insertValue ) && ( tempValue->getNextNode()->getContents() > insertValue ) )
                    || ( tempValue->getContents() == insertValue ) )
```

```
            {
                insertionNode   = new SLNode( insertValue );
                insertionNode->setNextNode( tempValue->getNextNode() );
                tempValue->setNextNode( insertionNode );
                size++;

                return;
            }
            else
            {
                tempValue = tempValue->getNextNode();
            }
        }

        if( tempValue->getContents() < insertValue )
        {
            insertTail( insertValue );
            return;
        }
        else if( tempValue->getContents() > insertValue )
        {
            insertHead( insertValue );
            return;
        }
    }
    else
    {
        head = new SLNode( insertValue );
        size++;
    }
}

bool SList::removeFirst( int removeValue )
{
    if( size != 0 )
    {
        if( head->getContents() == removeValue )
        {
            removeHead();
            return true;
        }
    }
    else
    {
        return false;
    }

    if( size >= 2 )
    {
        SLNode* tempValue   = head;
```

```
        SLNode* removeNode  = NULL;

        if( !( tempValue->getNextNode()->getContents() == removeValue ) )
        {
            while( tempValue->getNextNode()->getNextNode() != NULL )
            {
                if( tempValue->getNextNode()->getContents() == removeValue )
                {
                    break;
                }

                tempValue = tempValue->getNextNode();
            }
        }

        if( tempValue->getNextNode()->getContents() == removeValue )
        {
            if( tempValue->getNextNode()->getNextNode() == NULL )
            {
                removeTail();
                return true;
            }
            else
            {
                removeNode = tempValue->getNextNode();
                tempValue->setNextNode( removeNode->getNextNode() );

                delete removeNode;
                removeNode = NULL;

                size--;
                return true;
            }
        }
    }
}

void SList::clear()
{
    if( size > 1 )
    {
        SLNode *tempValue;

        do {
            tempValue = (this->head)->getNextNode();
            delete this->head;
            this->head = tempValue;

        } while( (this->head)->getNextNode() != NULL );
    }
```

```
    if( size >= 1 )
    {
        delete this->head;
        size = 0;
    }

}

string SList::toString() const
{

    if( size > 0 )
    {
        stringstream ss;

        if( (this->head)->getNextNode() == NULL )
        {
            ss << (this->head)->getContents();
            return ss.str();
        }

        SLNode *tempValue   = head;

        do {
            ss << tempValue->getContents() << ",";
            tempValue = tempValue->getNextNode();

        } while( tempValue->getNextNode() != NULL );

        ss << tempValue->getContents();

        return ss.str();
    }
    else
    {
        return "";
    }
}

/***** accessor/mutator function definitions *****/
unsigned int SList::getSize() const
{
    return this->size;
}

```