```
#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 *selectedNode    = this->head;

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

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

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

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

```
    }

    size++;
}

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

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

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

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

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

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

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

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

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

        if( selectedNode->getContents() < insertValue )
        {
            insertTail( insertValue );
            return;
        }
        else if( selectedNode->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* selectedNode    = head;
```

```
151         SLNode* removeNode      = NULL;
152
153         if( !( selectedNode->getNextNode()->getContents() == removeValue ) )
154         {
155             while( selectedNode->getNextNode()->getNextNode() != NULL )
156             {
157                 if( selectedNode->getNextNode()->getContents() == removeValue )
158                 {
159                     break;
160                 }
161
162                 selectedNode = selectedNode->getNextNode();
163             }
164         }
165
166         if( selectedNode->getNextNode()->getContents() == removeValue )
167         {
168             if( selectedNode->getNextNode()->getNextNode() == NULL )
169             {
170                 removeTail();
171                 return true;
172             }
173             else
174             {
175                 removeNode = selectedNode->getNextNode();
176                 selectedNode->setNextNode( removeNode->getNextNode() );
177
178                 delete removeNode;
179                 removeNode = NULL;
180
181                 size--;
182                 return true;
183             }
184         }
185     }
186 }
187
188 bool SList::removeAll( int removeValue )
189 {
190     if( uintCount > 0 )
191     {
192
193         SLNode* selectedNode    = this->head;
194         SLNode* removeNode      = NULL;
195
196         while( selectedNode->getContents() == removeValue )
197         {
198             removeNode      = selectedNode;
199             this->head      = removeNode->getNextNode();
200
```

```
201             delete removeNode;
202             size--;
203
204             selectedNode = this->head;
205             if( selectedNode == NULL )
206             {
207                 return true;
208             }
209         }
210
211         SLNode* followerNode    = NULL;
212         if( selectedNode != NULL )
213         {
214             while( selectedNode->getNext() != NULL )
215             {
216                 followerNode    = selectedNode;
217                 if( selectedNode->getNext()->getContents() == removeValue )
218                 {
219                     removeNode  = selectedNode->getNextNode();
220
221                     selectedNode->setNextNode( removeNode->getNextNode() );
222
223                     delete removeNode;
224                     uintCount--;
225                 }
226
227                 selectedNode = selectedNode->getNextNode();
228             }
229
230             if( selectedNode->getContents() == removeValue )
231             {
232                 removeNode = selectedNode;
233                 followerNode->setNext( NULL );
234
235                 delete removeNode;
236                 uintCount--;
237             }
238         }
239
240         if( removeNode != NULL )
241         {
242             return true;
243         }
244     }
245
246     return false;
247 }
248
249 void SList::clear()
250 {
```

```
    if( size > 1 )
    {
        SLNode *selectedNode;

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

        } 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 *selectedNode    = head;

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

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

        ss << selectedNode->getContents();

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

```
301
302 /***** accessor/mutator function definitions *****/
303 unsigned int SList::getSize() const
304 {
305     return this->size;
306 }
307
```