

```

1  #include "SList.h"
2
3  /***** constructor/destructor definitions *****/
4
5  SList::SList()
6  : head( NULL ), size( 0 )
7  {
8      /* empty */
9  }
10
11 SList::~~SList()
12 {
13     this->clear();
14 }
15
16 /***** public function definitions *****/
17
18 void SList::insertHead( int headValue )
19 {
20     if( size > 0 )
21     {
22         SLNode *selectedNode    = this->head;
23
24         this->head = new SLNode( headValue );
25         size++;
26         head->setNextNode( selectedNode );
27     }
28     else
29     {
30         this->head = new SLNode( headValue );
31         size++;
32     }
33 }
34
35 void SList::insertTail( int tailValue )
36 {
37     if( size > 0 )
38     {
39         SLNode *selectedNode    = head;
40
41         while( selectedNode->getNextNode() != NULL )
42         {
43             selectedNode = selectedNode->getNextNode();
44         }
45
46         selectedNode->setNextNode( new SLNode( tailValue ) );
47     }
48     else
49     {
50         head = new SLNode( tailValue );

```

```

51     }
52
53     size++;
54 }
55
56 void SList::removeHead()
57 {
58     if( size > 0 )
59     {
60         SLNode *selectedNode    = (this->head)->getNextNode();
61
62         delete this->head;
63         this->head = selectedNode;
64         size--;
65     }
66 }
67
68 void SList::removeTail()
69 {
70     if( size > 1 )
71     {
72         SLNode *selectedNode    = head;
73
74         while( (selectedNode->getNextNode())->getNextNode() != NULL )
75         {
76             selectedNode = selectedNode->getNextNode();
77         }
78
79         delete selectedNode->getNextNode();
80         selectedNode->setNextNode( NULL );
81         size--;
82     }
83     else if( size == 1 )
84     {
85         delete this->head;
86         size--;
87     }
88 }
89
90 void SList::insert( int insertValue )
91 {
92     if( size > 0 )
93     {
94         SLNode* selectedNode    = head;
95         SLNode* insertionNode   = NULL;
96
97         while( selectedNode->getNextNode() != NULL )
98         {
99             if( ( ( selectedNode->getContents() < insertValue ) && ( selectedNode->getNextNode()->getContents() > insertValue ) )
100                || ( selectedNode->getContents() == insertValue ) )

```

```

101         {
102             insertionNode = new SLNode( insertValue );
103             insertionNode->setNextNode( selectedNode->getNextNode() );
104             selectedNode->setNextNode( insertionNode );
105             size++;
106
107             return;
108         }
109         else
110         {
111             selectedNode = selectedNode->getNextNode();
112         }
113     }
114
115     if( selectedNode->getContents() < insertValue )
116     {
117         insertTail( insertValue );
118         return;
119     }
120     else if( selectedNode->getContents() > insertValue )
121     {
122         insertHead( insertValue );
123         return;
124     }
125 }
126 else
127 {
128     head = new SLNode( insertValue );
129     size++;
130 }
131 }
132
133 bool SList::removeFirst( int removeValue )
134 {
135     if( size != 0 )
136     {
137         if( head->getContents() == removeValue )
138         {
139             removeHead();
140             return true;
141         }
142     }
143     else
144     {
145         return false;
146     }
147
148     if( size >= 2 )
149     {
150         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 {

```

```

251     if( size > 1 )
252     {
253         SLNode *selectedNode;
254
255         do {
256             selectedNode = (this->head)->getNextNode();
257             delete this->head;
258             this->head = selectedNode;
259
260         } while( (this->head)->getNextNode() != NULL );
261     }
262
263     if( size >= 1 )
264     {
265         delete this->head;
266         size = 0;
267     }
268 }
269
270 string SList::toString() const
271 {
272
273     if( size > 0 )
274     {
275         stringstream ss;
276
277         if( (this->head)->getNextNode() == NULL )
278         {
279             ss << (this->head)->getContents();
280             return ss.str();
281         }
282
283         SLNode *selectedNode    = head;
284
285         do {
286             ss << selectedNode->getContents() << ",";
287             selectedNode = selectedNode->getNextNode();
288
289         } while( selectedNode->getNextNode() != NULL );
290
291         ss << selectedNode->getContents();
292
293         return ss.str();
294     }
295     else
296     {
297         return "";
298     }
299 }
300

```

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