

```

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 *tempValue  = this->head;
23
24         this->head = new SLNode( headValue );
25         size++;
26         head->setNextNode( tempValue );
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 *tempValue  = head;
40
41         while( tempValue->getNextNode() != NULL )
42         {
43             tempValue = tempValue->getNextNode();
44         }
45
46         tempValue->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 *tempValue   = (this->head)->getNextNode();
61
62         delete this->head;
63         this->head = tempValue;
64         size--;
65     }
66 }
67
68 void SList::removeTail()
69 {
70     if( size > 1 )
71     {
72         SLNode *tempValue   = head;
73
74         while( (tempValue->getNextNode())->getNextNode() != NULL )
75         {
76             tempValue = tempValue->getNextNode();
77         }
78
79         delete tempValue->getNextNode();
80         tempValue->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* tempValue      = head;
95         SLNode* insertionNode  = NULL;
96
97         while( tempValue->getNextNode() != NULL )
98         {
99             if( ( tempValue->getContents() < insertValue ) && ( tempValue->getNextNode()->getContents() > insertValue ) )
100                 || ( tempValue->getContents() == insertValue ) )

```

```

101         {
102             insertionNode = new SLNode( insertValue );
103             insertionNode->setNextNode( tempValue->getNextNode() );
104             tempValue->setNextNode( insertionNode );
105             size++;
106
107             return;
108         }
109         else
110         {
111             tempValue = tempValue->getNextNode();
112         }
113     }
114
115     if( tempValue->getContents() < insertValue )
116     {
117         insertTail( insertValue );
118         return;
119     }
120     else if( tempValue->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* tempValue = head;

```

```

151     SLNode* removeNode = NULL;
152
153     if( !( tempValue->getNextNode()->getContents() == removeValue ) )
154     {
155         while( tempValue->getNextNode()->getNextNode() != NULL )
156         {
157             if( tempValue->getNextNode()->getContents() == removeValue )
158             {
159                 break;
160             }
161
162             tempValue = tempValue->getNextNode();
163         }
164     }
165
166     if( tempValue->getNextNode()->getContents() == removeValue )
167     {
168         if( tempValue->getNextNode()->getNextNode() == NULL )
169         {
170             removeTail();
171             return true;
172         }
173         else
174         {
175             removeNode = tempValue->getNextNode();
176             tempValue->setNextNode( removeNode->getNextNode() );
177
178             delete removeNode;
179             removeNode = NULL;
180
181             size--;
182             return true;
183         }
184     }
185 }
186
187 void SList::clear()
188 {
189     if( size > 1 )
190     {
191         SLNode *tempValue;
192
193         do {
194             tempValue = (this->head)->getNextNode();
195             delete this->head;
196             this->head = tempValue;
197         } while( (this->head)->getNextNode() != NULL );
198     }
199 }
200

```

```

201
202     if( size >= 1 )
203     {
204         delete this->head;
205         size = 0;
206     }
207
208 }
209
210 string SList::toString() const
211 {
212
213     if( size > 0 )
214     {
215         stringstream ss;
216
217         if( (this->head)->getNextNode() == NULL )
218         {
219             ss << (this->head)->getContents();
220             return ss.str();
221         }
222
223         SLNode *tempValue  = head;
224
225         do {
226             ss << tempValue->getContents() << ",";
227             tempValue = tempValue->getNextNode();
228
229         } while( tempValue->getNextNode() != NULL );
230
231         ss << tempValue->getContents();
232
233         return ss.str();
234     }
235     else
236     {
237         return "";
238     }
239 }
240
241 /***** accessor/mutator function definitions *****/
242 unsigned int SList::getSize() const
243 {
244     return this->size;
245 }
246

```