

```

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

```

```

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

```

```

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

```

```

150     {
151         if( head->getContents() == removeValue )
152         {
153             removeHead();
154             return true;
155         }
156     }
157     else
158     {
159         return false;
160     }
161
162     if( size >= 2 )
163     {
164         SLNode<T>* selectedNode      = head;
165         SLNode<T>* removeNode       = NULL;
166
167         if( !( selectedNode->getNextNode()->getContents() == removeValue ) )
168         {
169             while( selectedNode->getNextNode()->getNextNode() != NULL )
170             {
171                 if( selectedNode->getNextNode()->getContents() == removeValue )
172                 {
173                     break;
174                 }
175
176                 selectedNode = selectedNode->getNextNode();
177             }
178         }
179
180         if( selectedNode->getNextNode()->getContents() == removeValue )
181         {
182             if( selectedNode->getNextNode()->getNextNode() == NULL )
183             {
184                 removeTail();
185                 return true;
186             }
187             else
188             {
189                 removeNode = selectedNode->getNextNode();
190                 selectedNode->setNextNode( removeNode->getNextNode() );
191
192                 delete removeNode;
193                 removeNode = NULL;
194
195                 size--;
196                 return true;
197             }
198         }
199     }

```

```
200
201     return false;
202 }
203
204 bool removeAll( T removeValue )
205 {
206     if( size > 0 )
207     {
208
209         SLNode<T>* selectedNode    = this->head;
210         SLNode<T>* removeNode     = NULL;
211
212         while( selectedNode->getContents() == removeValue )
213         {
214             removeNode      = selectedNode;
215             this->head       = removeNode->getNextNode();
216
217             delete removeNode;
218             size--;
219
220             selectedNode = this->head;
221             if( selectedNode == NULL )
222             {
223                 return true;
224             }
225         }
226
227         SLNode<T>* followerNode = NULL;
228         if( selectedNode != NULL )
229         {
230             while( selectedNode->getNextNode() != NULL )
231             {
232                 followerNode    = selectedNode;
233                 if( selectedNode->getNextNode()->getContents() == removeValue )
234                 {
235                     removeNode = selectedNode->getNextNode();
236
237                     selectedNode->setNextNode( removeNode->getNextNode() );
238
239                     delete removeNode;
240                     size--;
241
242                     continue;
243                 }
244
245                 selectedNode = selectedNode->getNextNode();
246             }
247
248             if( selectedNode->getContents() == removeValue )
249             {
```

```

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

```

```

300         }
301
302         SLNode<T>* selectedNode = head;
303
304         do {
305             ss << selectedNode->getContents() << ",";
306             selectedNode = selectedNode->getNextNode();
307
308         } while( selectedNode->getNextNode() != NULL );
309
310         ss << selectedNode->getContents();
311
312         return ss.str();
313     }
314     else
315     {
316         return "";
317     }
318 }
319
320 /***** accessor/mutator function definitions *****/
321 unsigned int getSize() const
322 {
323     return this->size;
324 }
325
326 private:
327
328 /***** private variable declarations *****/
329
330     SLNode<T>* head;
331     unsigned int size;
332
333 };
334
335 #endif
336

```