

```
1 #include <iostream>
2 #include <cstdlib>
3 #include "llcpInt.h"
4 using namespace std;
5
6 int FindListLength(Node* headPtr)
7 {
8     int length = 0;
9
10    while (headPtr != 0)
11    {
12        ++length;
13        headPtr = headPtr->link;
14    }
15
16    return length;
17 }
18
19 bool IsSortedUp(Node* headPtr)
20 {
21     if (headPtr == 0 || headPtr->link == 0) // empty or 1-node
22         return true;
23     while (headPtr->link != 0) // not at last node
24     {
25         if (headPtr->link->data < headPtr->data)
26             return false;
27         headPtr = headPtr->link;
28     }
29     return true;
30 }
31
32 void InsertAsHead(Node*& headPtr, int value)
33 {
34     Node *newNodePtr = new Node;
35     newNodePtr->data = value;
36     newNodePtr->link = headPtr;
37     headPtr = newNodePtr;
38 }
39
40 void InsertAsTail(Node*& headPtr, int value)
41 {
42     Node *newNodePtr = new Node;
43     newNodePtr->data = value;
```

```
44     newNodePtr->link = 0;
45     if (headPtr == 0)
46         headPtr = newNodePtr;
47     else
48     {
49         Node *cursor = headPtr;
50
51         while (cursor->link != 0) // not at last node
52             cursor = cursor->link;
53         cursor->link = newNodePtr;
54     }
55 }
56
57 void InsertSortedUp(Node*& headPtr, int value)
58 {
59     Node *precursor = 0,
60         *cursor = headPtr;
61
62     while (cursor != 0 && cursor->data < value)
63     {
64         precursor = cursor;
65         cursor = cursor->link;
66     }
67
68     Node *newNodePtr = new Node;
69     newNodePtr->data = value;
70     newNodePtr->link = cursor;
71     if (cursor == headPtr)
72         headPtr = newNodePtr;
73     else
74         precursor->link = newNodePtr;
75
76     ////////////////////////////////////
77     /* using-only-cursor (no precursor) version
78     Node *newNodePtr = new Node;
79     newNodePtr->data = value;
80     //newNodePtr->link = 0;
81     //if (headPtr == 0)
82     //    headPtr = newNodePtr;
83     //else if (headPtr->data >= value)
84     //{
85     //    newNodePtr->link = headPtr;
86     //    headPtr = newNodePtr;
```

```

87     //}
88     if (headPtr == 0 || headPtr->data >= value)
89     {
90         newNodePtr->link = headPtr;
91         headPtr = newNodePtr;
92     }
93     //else if (headPtr->link == 0)
94     //    head->link = newNodePtr;
95     else
96     {
97         Node *cursor = headPtr;
98         while (cursor->link != 0 && cursor->link->data < value)
99             cursor = cursor->link;
100        //if (cursor->link != 0)
101        //    newNodePtr->link = cursor->link;
102        newNodePtr->link = cursor->link;
103        cursor->link = newNodePtr;
104    }
105
106    ////////////////////////////////// commented lines removed //////////////////////////////////
107
108    Node *newNodePtr = new Node;
109    newNodePtr->data = value;
110    if (headPtr == 0 || headPtr->data >= value)
111    {
112        newNodePtr->link = headPtr;
113        headPtr = newNodePtr;
114    }
115    else
116    {
117        Node *cursor = headPtr;
118        while (cursor->link != 0 && cursor->link->data < value)
119            cursor = cursor->link;
120        newNodePtr->link = cursor->link;
121        cursor->link = newNodePtr;
122    }
123    */
124    //////////////////////////////////////////////////////////////////////
125 }
126
127 bool DelFirstTargetNode(Node*& headPtr, int target)
128 {
129     Node *precursor = 0,

```

```

130         *cursor = headPtr;
131
132     while (cursor != 0 && cursor->data != target)
133     {
134         precursor = cursor;
135         cursor = cursor->link;
136     }
137     if (cursor == 0)
138     {
139         cout << target << " not found." << endl;
140         return false;
141     }
142     if (cursor == headPtr) //OR precursor == 0
143         headPtr = headPtr->link;
144     else
145         precursor->link = cursor->link;
146     delete cursor;
147     return true;
148 }
149
150 bool DelNodeBefore1stMatch(Node*& headPtr, int target)
151 {
152     if (headPtr == 0 || headPtr->link == 0 || headPtr->data == target) r
153     Node *cur = headPtr->link, *pre = headPtr, *prepre = 0;
154     while (cur != 0 && cur->data != target)
155     {
156         prepre = pre;
157         pre = cur;
158         cur = cur->link;
159     }
160     if (cur == 0) return false;
161     if (cur == headPtr->link)
162     {
163         headPtr = cur;
164         delete pre;
165     }
166     else
167     {
168         prepre->link = cur;
169         delete pre;
170     }
171     return true;
172 }

```

```
173
174 void ShowAll(ostream& outs, Node* headPtr)
175 {
176     while (headPtr != 0)
177     {
178         outs << headPtr->data << " ";
179         headPtr = headPtr->link;
180     }
181     outs << endl;
182 }
183
184 void FindMinMax(Node* headPtr, int& minValue, int& maxValue)
185 {
186     if (headPtr == 0)
187     {
188         cerr << "FindMinMax() attempted on empty list" << endl;
189         cerr << "Minimum and maximum values not set" << endl;
190     }
191     else
192     {
193         minValue = maxValue = headPtr->data;
194         while (headPtr->link != 0)
195         {
196             headPtr = headPtr->link;
197             if (headPtr->data < minValue)
198                 minValue = headPtr->data;
199             else if (headPtr->data > maxValue)
200                 maxValue = headPtr->data;
201         }
202     }
203 }
204
205 double FindAverage(Node* headPtr)
206 {
207     if (headPtr == 0)
208     {
209         cerr << "FindAverage() attempted on empty list" << endl;
210         cerr << "An arbitrary zero value is returned" << endl;
211         return 0.0;
212     }
213     else
214     {
215         int sum = 0,
```

```

216         count = 0;
217
218         while (headPtr != 0)
219         {
220             ++count;
221             sum += headPtr->data;
222             headPtr = headPtr->link;
223         }
224
225         return double(sum) / count;
226     }
227 }
228
229 void ListClear(Node*& headPtr, int noMsg)
230 {
231     int count = 0;
232
233     Node *cursor = headPtr;
234     while (headPtr != 0)
235     {
236         headPtr = headPtr->link;
237         delete cursor;
238         cursor = headPtr;
239         ++count;
240     }
241     if (noMsg) return;
242     clog << "Dynamic memory for " << count << " nodes freed"
243         << endl;
244 }
245
246 void PromoteTarget(Node*& headPtr, int target) {
247     bool foundTarget = false;
248     if (headPtr == nullptr) {
249         Node *newNodePtr = new Node;
250         newNodePtr->data = target;
251         newNodePtr->link = nullptr;
252         headPtr = newNodePtr;
253     } else if (!(headPtr->link == nullptr && headPtr->data == target))
254         Node *cursor, *previous;
255         bool onlyTarget = false;
256         if (headPtr->data == target) {
257             foundTarget = true;
258             onlyTarget = true;

```

```
259         cursor = headPtr;
260         previous = headPtr;
261         while (cursor->data == target && cursor->link != nullptr){
262             previous = cursor;
263             cursor = cursor->link;
264             if (cursor->data != target)
265                 onlyTarget = false;
266         }
267     } else {
268         cursor = headPtr->link;
269         previous = headPtr;
270     }
271     while (!onlyTarget && cursor != nullptr) {
272         if (cursor->data == target) {
273             foundTarget = true;
274
275             if (cursor->link != nullptr)
276                 previous->link = cursor->link;
277             else
278                 previous->link = nullptr;
279
280             if (headPtr->data == target) {
281                 Node *savedCursor = cursor;
282                 cursor = headPtr;
283                 do {
284                     previous = cursor;
285                     cursor = cursor->link;
286                 }
287                 while (cursor->data == target);
288                 previous->link = savedCursor;
289                 savedCursor->link = cursor;
290             } else {
291                 Node *tempHeadPtr = headPtr;
292                 tempHeadPtr->link = headPtr->link;
293                 headPtr = cursor;
294                 cursor->link = tempHeadPtr;
295             }
296
297         }
298         previous = cursor;
299         cursor = cursor->link;
300     }
301
```

```
302         if (!foundTarget) {
303             Node *newNodePtr = new Node;
304             newNodePtr->data = target;
305             newNodePtr->link = nullptr;
306             previous->link = newNodePtr;
307         }
308     }
309 }
310
311
312
```