```
 1: #include "List.h"
 2: #include "ListNode.h"
 3: #include <stdio.h>
 4:
 5: List::List()
 6: {
 7:     m_first = NULL;
 8:     m_last = NULL;
 9:     m_numElems = 0;
10: }
11:
12: List::~List()
13: {
14:     // DOES NOTHING
15: }
16:
17: ListNode* List::insertAtEnd(int content)
18: {
19:     ListNode * node = new ListNode(content);
20:     ListNode * previous = NULL;
21:
22:     /* Primeiro elemento */
23:     if ((m_first == NULL) && (m_last == NULL)) {
24:         m_first = node;
25:         m_last = node;
26:     } else {
27:         node->setPrevious( m_last );
28:         previous = m_last;
29:         m_last->setNext(node);
30:         m_last = node;
31:     }
32:
33:     m_numElems++;
34:
35:     return node;
36: }
37:
38: int List::insertAtFront(int content)
39: {
40:     ListNode * node = new ListNode(content);
41:
42:     /* Primeiro elemento */
43:     if ((m_first == NULL) && (m_last == NULL)) {
44:         m_first = node;
45:         m_last = node;
46:     } else {
47:         node->setNext(m_first);
48:         m_first = node;
49:     }
50:
51:     m_numElems++;
52:
53:     return 0;
54: }
55:
56: int List::removeFirst()
57: {
58:     if (m_numElems == 0) {
59:         return -1;
60:     }
61:
62:     ListNode * node = m_first;
63:
64:     int content = node->getVertex();
65:
66:     m_first = m_first->next();
67:     m_first->setPrevious( NULL );
68:     delete node;
69:
70:     m_numElems--;
71:
72:     return content;
73: }
74:
75: int List::size()
76: {
77:     return m_numElems;
78: }
79:
80: void List::erase(int content)
81: {
82:     ListNode * previous = NULL;
83:
84:     for(ListNode * node = m_first; node != NULL; node = node->next()) {
85:
86:         if(content == node->getVertex()) {
87:
88:             // Primeiro da lista
89:             if (node == m_first) {
90:                 if (m_numElems == 1) {
91:                     m_first = NULL;
92:                     m_last = NULL;
93:                 } else {
94:                     m_first = node->next();
```

```
 95:                     m_first->setPrevious( NULL );
 96:                 }
 97:             } else {
 98:                 previous->setNext(node->next());
 99:
100:                 if (node == m_last) {
101:                     m_last = previous;
102:                 }
103:                 else
104:                 {
105:                     node->next()->setPrevious( previous );
106:                 }
107:             }
108:
109:
110:                 m_numElems--;
111:             delete node;
112:             return;
113:         }
114:         previous = node;
115:     }
116: }
117:
118:
119: void List::remove( ListNode* node )
120: {
121:     if (node == NULL)
122:         return;
123:
124:     ListNode* previous = node->previous();
125:
126:     // primeiro da lista
127:     if (node == m_first)
128:     {
129:         if (m_numElems == 1)
130:         {
131:             m_first = NULL;
132:             m_last = NULL;
133:         }
134:         else
135:         {
136:             m_first = node->next();
137:             m_first->setPrevious( NULL );
138:         }
139:     }
140:     else
141:     {
142:         previous->setNext(node->next());
143:
144:         if (node == m_last)
145:         {
146:             m_last = previous;
147:         }
148:         else
149:         {
150:             node->next()->setPrevious( previous );
151:         }
152:     }
153:
154:     m_numElems--;
155:
156:     delete node;
157: }
```