```
 1: #ifndef CSCI180_SORTING_H
 2: #define CSCI180_SORTING_H
 3:
 4: #include <list>
 5: using std::list;
 6:
 7: namespace csci180 {
 8:
 9:   template <typename Object>
10:   void selectionSort(list<Object>& data) {
11:     typedef typename list<Object>::iterator iterator;  // for convenience
12:
13:     // Going from the end back to beginning.  Invariant is that
14:     // the k elements from the marker to the end are sorted and
15:     // are the largest k of the overall data set
16:     iterator marker = data.end();
17:
18:     while (marker != data.begin()) {
19:       // find largest item occurring strictly before the marker
20:       iterator step = marker;
21:       --step;                      // one before the marker
22:       iterator best = step;
23:       while (step != data.begin()) {
24:         --step;
25:         if (*step > *best)
26:           best = step;
27:       }
28:
29:       marker = data.insert(marker, *best); // insert in front of marker (and reset)
30:       data.erase(best);                    // remove original value
31:     }
32:   }
33:
34:   template <typename Object>
35:   void insertionSort(list<Object>& data) {
36:     typedef typename list<Object>::iterator iterator;  // for convenience
37:
38:     // Invariant is that all items from marker to end are relatively sorted
39:     iterator marker = data.end();
40:
41:     while (marker != data.begin()) {
42:       iterator temp = marker;        // record location of marker
43:       --marker;                      // step backward
44:
45:       // Goal:  determine where new marker belongs relative to later values
46:       while (temp != data.end() && *temp < *marker)
47:         ++temp;
48:
49:       // marked item belongs immediately before temp
50:       data.insert(temp, *marker);
51:       marker = data.erase(marker);
52:     }
53:
54:     --marker;
55:   }
```

```
56:   template <typename Object>
57:   void bubbleSort(list<Object>& data) {
58:     typedef typename list<Object>::iterator iterator;  // for convenience
59:
60:     iterator right = data.end();
61:     bool change = true;
62:     while (change) {
63:       change = false;
64:
65:       // single pass, sweep from left to right
66:       iterator sweep = data.begin();
67:       while (sweep != right) {
68:         iterator adv = sweep;
69:         ++adv;                 // look one step ahead
70:         if ((adv != right)  &&  ((*adv) < (*sweep)))   {
71:           change = true;
72:           data.insert(sweep, *adv);  // add copy of *adv before sweep;
73:           data.erase(adv);           // and remove original "adv" value
74:         } else {
75:           ++sweep;
76:         }
77:       }
78:      if (right != data.begin())
79:         --right;
80:     }
81:   }
82:
83: } // end of csci180 namespace
84:
85: #endif
```