```
1  /*--------------------------------------------------------------------
2  Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
3  file: slist.hpp
4
5  --------------------------------------------------------------------*/
6
7  /*--------------------------------------------------------------------
8  This file has class definition
9  --------------------------------------------------------------------*/
10
11 /*--------------------------------------------------------------------
12 Definition of routines of slist class
13 --------------------------------------------------------------------*/
14
15 /*--------------------------------------------------------------------
16 slist Constructor
17 --------------------------------------------------------------------*/
18 template <typename T>
19 slist<T>::slist(void(*pv) (T& c), int(*cf) (const T& c1, const T& c2)) :
20 _first(nullptr), _last(nullptr), _pntr_to_compare_func(cf), _pntr_to_func_to_delete_data(pv),
21 _num_nodes_allocated(0), _num_nodes_freed(0)
22 {
23   if (display()) {
24     cout << "in slist constructor:" << endl;
25   }
26 }
27
28 /*--------------------------------------------------------------------
29 slist Destructor
30 --------------------------------------------------------------------*/
31 template <typename T>
32 slist<T>::~slist() {
33   if (display()) {
34     cout << "in slist destructor:" << endl;
35   }
36   node<T>* t = _first;
37   while (t) {
38     node<T>* ct = t;
39     t = t->_next;
40     _delete_a_node(ct);
41   }
42   if (_num_nodes_allocated != _num_nodes_freed) {
43     assert(0);
44   }
45   _first = nullptr;
46   _last = nullptr;
47 }
48
49 /*--------------------------------------------------------------------
50 change function
51 --------------------------------------------------------------------*/
52 template <typename T>
53 void slist<T>::change_functions(void(*pv) (T& c), int(*cf) (const T& c1, const T& c2)) {
54   _pntr_to_compare_func = cf;
55   _pntr_to_func_to_delete_data = pv;
56 }
57
58 /*--------------------------------------------------------------------
59 How many elements are in the list
60 --------------------------------------------------------------------*/
61 template <typename T>
62 int slist<T>::size() const {
63   int i = 0;
64   node<T> *f = _first;
65   while (f) {
66     i++;
```

```
67      f = f->_next;
68    }
69    return i;
70  }
71
72  /*-----------------------------------------------------------------
73  create a node in slist
74  -----------------------------------------------------------------*/
75  template <typename T>
76  node<T>* slist<T>::_create_a_node(const T& data){
77    _num_nodes_allocated++;
78    node<T>* x = new(node<T>)(data); /* if T is by value, copy const will be called for T */
79    return x;
80  }
81
82  /*-----------------------------------------------------------------
83  delete a node from slist
84  -----------------------------------------------------------------*/
85  template <typename T>
86  void slist<T>::_delete_a_node(node<T> *n){
87    _num_nodes_freed++;
88    if (_pntr_to_func_to_delete_data) {
89      _pntr_to_func_to_delete_data(n->_data); /* Distructor of T will be called*/
90    }
91    delete(n);
92  }
93
94  /*-----------------------------------------------------------------
95  append data to the end of the slist
96  -----------------------------------------------------------------*/
97  template <typename T>
98  void slist<T>::append(const T& data) {
99    node<T> *c = _create_a_node(data);
100   if (!_first) {
101     _first = c;
102   }
103   else {
104     _last->_next = c;
105   }
106   _last = c;
107 }
108
109 /*-----------------------------------------------------------------
110 append data after p.
111 if p is not there, add data at the end
112 -----------------------------------------------------------------*/
113 template <typename T>
114 void append_after(const T& p, const T& data) {
115   //WRITE YOUR CODE HERE
116 }
117
118 /*-----------------------------------------------------------------
119 find data in the slist
120 -----------------------------------------------------------------*/
121 template <typename T>
122 node<T>* slist<T>::_find(const T& data){
123   if (!(_pntr_to_compare_func)) {
124     return NULL;
125   }
126   node<T> *f = _first;
127   while (f) {
128     if (_pntr_to_compare_func(data, f->_data) == 0) {
129       return f;
130     }
131     f = f->_next;
132   }
```

```
133   return NULL;
134 }
135
136 /*-------------------------------------------------------------------
137 find data in the slist
138 -------------------------------------------------------------------*/
139 template <typename T>
140 bool slist<T>::find(const T& data){
141   node<T>* f = _find(data);
142   return (f) ? true : false;
143 }
144
145 /*-------------------------------------------------------------------
146 Unlink node p from slist
147 -------------------------------------------------------------------*/
148 template <typename T>
149 bool slist<T>::_unlink_data(const node<T>* f) {
150   bool unlinked = false;
151   //WRITE YOUR CODE HERE
152
153   return unlinked;
154 }
155
156 /*-------------------------------------------------------------------
157 Unlink node p from slist
158 -------------------------------------------------------------------*/
159 template <typename T>
160 bool slist<T>::unlink_data(const T& data) {
161   node<T>* f = _find(data);
162   if (f) {
163     return _unlink_data(f);
164   }
165   return false;
166 }
167
168 /*-------------------------------------------------------------------
169 1->4->8->7
170
171 7->8->4->1
172 -------------------------------------------------------------------*/
173 template <typename T>
174 void slist<T>::reverse() {
175   node<T>* c = _first;
176   if (c) {
177     _last = _first;
178     node<T>* n = c->_next;
179     while (n) {
180       node<T>* t = n->_next;
181       n->_next = c;
182       c = n;
183       n = t;
184     }
185     _first = c;
186     _last->_next = NULL;
187   }
188 }
189
190 /*-------------------------------------------------------------------
191 DO NOT CHANGE ANYTHING BELOW
192 -------------------------------------------------------------------*/
193 template <typename T>
194 void slist<T>::_get_last_and_last_but_one(node<T>*& last, node<T>*& last_but_one) const{
195   last = 0;
196   last_but_one = 0;
197   node<T>* c = _first;
198   while (c) {
```

```
199     last_but_one = last;
200     last = c;
201     c = c->_next;
202   }
203 }
204
205
206 /*-------------------------------------------------------------
207 1->4->8->7
208
209 7->8->4->1
210 -------------------------------------------------------------*/
211 template <typename T>
212 void slist<T>::reverse_recur() {
213   //WRITE YOUR CODE HERE
214 }
215
216
217 /*-------------------------------------------------------------
218 create a loop
219 DO NOT CHANGE ANYTHING BELOW
220 -------------------------------------------------------------*/
221 template <typename T>
222 void slist<T>::create_a_loop(int from, int to) {
223   int i = 0;
224   node<T> *fp = NULL;
225   node<T> *tp = NULL;
226   node<T> *f = _first;
227   while (f) {
228     i++;
229     if (i == from) {
230       fp = f;
231     }
232     if (i == to) {
233       tp = f;
234     }
235     f = f->_next;
236   }
237   if (fp && tp) {
238     fp->_next = tp;
239   }
240 }
241
242 /*-------------------------------------------------------------
243 detect a loop
244 -------------------------------------------------------------*/
245 template <typename T>
246 bool slist<T>::detect_loop() const {
247   bool loopfound = false;
248   //WRITE CODE BELOW
249   return loopfound;
250 }
251
252 /*-------------------------------------------------------------
253 Helper routine
254 -------------------------------------------------------------*/
255 template <typename T>
256 void slist<T>::_print_in_reverse_order_with_recursion_r(node<T>* c) const{
257   //WRITE YOUR CODE HERE
258 }
259
260 /*-------------------------------------------------------------
261 Print linked list in reverse order using recursion
262 -------------------------------------------------------------*/
263 template <typename T>
264 void slist<T>::print_in_reverse_order_with_recursion() const{
```

```
265   _print_in_reverse_order_with_recursion_r(_first);
266   cout << endl;
267 }
268
269 /*-------------------------------------------------------------
270 Print linked list in reverse order without using recursion
271 -------------------------------------------------------------*/
272 template <typename T>
273 void slist<T>::print_in_reverse_order_without_recursion() const{
274   //WRITE YOUR CODE HERE
275 }
276
277
278 //EOF
279
280
```