

```
1 /*-----
2 Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
3 file: slist.h
4 -----*/
5
6 /*-----
7 This file has slist class declaration
8 -----*/
9
10 /*-----
11 All includes here
12 -----*/
13 #ifndef slist_H
14 #define slist_H
15
16 #include "../util/util.h"
17 #include "../darray/darray.h"
18
19 /*-----
20 static definition - only once at the start
21 Change to false, if you don't need verbose
22 -----*/
23 template <typename T>
24 bool darray<T>::_display = true;
25
26 /*-----
27 All forward declaration
28 -----*/
29 template <typename T>
30 class node;
31
32 template <typename T>
33 class slist;
34
35 template <typename T>
36 class slist_iterator;
37
38
39 /*-----
40 typename node
41 -----*/
42 template <typename T>
43 class node {
44 public:
45     node(const T& data) :_data(data), _next(nullptr){}
46     ~node() {
47         _next = nullptr;
48     }
49     T& get_data() { return _data; }
50     friend class slist<T>; //slist can access nodes private part
51     friend class slist_iterator<T>; //slist_iterator can access slist private part
52
53     /* no body can copy node or equal node */
54     node(const node<T>& x) = delete;
55     node& operator=(const node<T>& x) = delete;
56
57 private:
58     T _data;
59     node<T>* _next;
60 };
61
62 /*-----
63 typename slist iterator
64 -----*/
65 template <typename T>
66 class slist_iterator {
```

```

67 public:
68     slist_iterator(node<T>* x = nullptr) :_current(x) {}
69     ~slist_iterator() {}
70     slist_iterator(const slist_iterator<T>& x) { _current = x._current; }
71     slist_iterator& operator=(const slist_iterator<T>& x) { _current = x._current; return *this; }
72     // *(itt)
73     T& operator*() const{
74         return (_current->_data);
75     }
76
77     //++itt
78     slist_iterator<T>& operator++() {
79         _current = _current->_next;
80         return *this;
81     }
82
83     //if (itt != x.end())
84     bool operator!=(const slist_iterator<T>& rhs) const {
85         return (_current != rhs._current);
86     }
87     //No body can call itt++
88     slist_iterator<T> operator++(int i) = delete;
89     //No body can call --itt or itt--
90     slist_iterator<T>& operator--() = delete;
91     slist_iterator<T> operator--(int i) = delete;
92
93 private:
94     node<T>* _current;
95 };
96
97 /*-----
98 typename slist
99 -----*/
100 template <typename T>
101 class slist {
102 public:
103     typedef slist_iterator<T> iterator;
104     friend class slist_iterator<T>; //slist_iterator can access slist private part
105     slist(void(*pv) (T& c) = nullptr, int(*cf) (const T& c1, const T& c2) = NULnullptr);
106     ~slist();
107
108     void change_functions(void(*pv) (T& c), int(*cf) (const T& c1, const T& c2));
109     int size() const;
110     void append(const T& data);
111     void append_after(const T& p, const T& data);
112     bool find(const T& data);
113     bool unlink_data(const T& data);
114     void reverse();
115     void reverse_recur();
116     void print_in_reverse_order_without_recursion() const;
117     void print_in_reverse_order_with_recursion() const;
118
119     void create_a_loop(int from, int to);
120     bool detect_loop() const;
121     bool display()const { return _display; }
122     static void set_display(bool x) {
123         darray<T>::set_display(x);
124         _display = x;
125     }
126
127     /* for iterator */
128     iterator begin() { return iterator(_first); }
129     iterator end() { return iterator(); }
130
131     /* no body can copy slist or equal slist */
132     slist(const slist<T>& x) = delete;

```

```
133  slist& operator=(const slist<T>& x) = delete;
134
135 private:
136     node<T>* _first;
137     node<T>* _last;
138     void(*_pntr_to_func_to_delete_data) (T& c);
139     int(*_pntr_to_compare_func) (const T& c1, const T& c2);
140     int _num_nodes_allocated;
141     int _num_nodes_freed;
142     static bool _display;
143
144     node<T>* _create_a_node(const T& data);
145     void _delete_a_node(node<T> *n);
146     node<T>* _find(const T& data);
147     bool _unlink_data(const node<T>* p);
148     void _reverse_recur(node<T>* f);
149     void _get_last_and_last_but_one(node<T>*& last, node<T>*& last_but_one) const;
150     void _print_in_reverse_order_with_recursion_r(node<T>* c) const;
151 };
152
153
154 #include "slist.hpp"
155
156 #endif
157 //EOF
158
159
```