```
/*-----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: slist.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has slist class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef slist_H
#define slist_H

#include "../util/util.h"
#include "../darray/darray.h"

/*---------------------------------------------------------
static definition - only once at the start
Change to false, if you don't need verbose
---------------------------------------------------------*/
template <typename T>
bool darray<T>::_display = true;

/*-----------------------------------------------------------------
All forward declaration
-----------------------------------------------------------------*/
template <typename T>
class node;

template <typename T>
class slist;

template <typename T>
class slist_iterator;


/*---------------------------------------------------------
typename node
---------------------------------------------------------*/
template <typename T>
class node {
public:
  node(const T& data) :_data(data), _next(nullptr){}
  ~node() {
    _next = nullptr;
  }
  T& get_data() { return _data; }
  friend class slist<T>; //slist can access nodes private part
  friend class slist_iterator<T>; //slist_iterator can access slist private part

  /* no body can copy node or equal node */
  node(const node<T>& x) = delete;
  node& operator=(const node<T>& x) = delete;

private:
  T  _data;
  node<T>* _next;
};

/*---------------------------------------------------------
typename slist iterator
---------------------------------------------------------*/
template <typename T>
class slist_iterator {
```

```cpp
public:
  slist_iterator(node<T>* x = nullptr) :_current(x) {}
  ~slist_iterator() {}
  slist_iterator(const slist_iterator<T>& x) { _current = x._current; }
  slist_iterator& operator=(const slist_iterator<T>& x) { _current = x._current; return *this; }
  // *(itt)
  T& operator*() const{
    return (_current->_data);
  }

  //++itt
  slist_iterator<T>& operator++() {
    _current = _current->_next;
    return *this;
  }

  //if (itt != x.end()
  bool operator!=(const slist_iterator<T>& rhs) const {
    return (_current != rhs._current);
  }
  //No body can call itt++
  slist_iterator<T> operator++(int i) = delete;
  //No body can call --itt or itt--
  slist_iterator<T>& operator--() = delete;
  slist_iterator<T> operator--(int i) = delete;

private:
  node<T>* _current;
};

/*---------------------------------------------------------
typename slist
---------------------------------------------------------*/
template <typename T>
class slist {
public:
  typedef slist_iterator<T> iterator;
  friend class slist_iterator<T>; //slist_iterator can access slist private part
  slist(void(*pv) (T& c) = nullptr, int(*cf) (const T& c1, const T& c2) = NULnullptr);
  ~slist();

  void change_functions(void(*pv) (T& c), int(*cf) (const T& c1, const T& c2));
  int size() const;
  void append(const T& data);
  void append_after(const T& p, const T& data);
  bool find(const T& data);
  bool unlink_data(const T& data);
  void reverse();
  void reverse_recur();
  void print_in_reverse_order_without_recursion() const;
  void print_in_reverse_order_with_recursion() const;

  void create_a_loop(int from, int to);
  bool detect_loop() const;
  bool display()const { return _display; }
  static void set_display(bool x) {
    darray<T>::set_display(x);
    _display = x;
  }

  /* for iterator */
  iterator begin() { return iterator(_first); }
  iterator end()  { return iterator(); }

  /* no body can copy slist or equal slist */
  slist(const slist<T>& x) = delete;
  slist& operator=(const slist<T>& x) = delete;

private:
  node<T>* _first;
  node<T>* _last;
  void(*_pntr_to_func_to_delete_data) (T& c);
  int(*_pntr_to_compare_func) (const T& c1, const T& c2);
  int _num_nodes_allocated;
  int _num_nodes_freed;
  static bool _display;

  node<T>* _create_a_node(const T& data);
  void _delete_a_node(node<T> *n);
  node<T>* _find(const T& data);
  bool _unlink_data(const node<T>* p);
  void _reverse_recur(node<T>* f);
  void _get_last_and_last_but_one(node<T>*& last, node<T>*& last_but_one) const;
  void _print_in_reverse_order_with_recursion_r(node<T>* c) const;
};


#include "slist.hpp"

#endif
//EOF


```