Handout #16 **CS 400**

J. Dichter

```
#ifndef TmpLIST__SEEN
#define TmpLIST__SEEN
///////    File: List.h    ///////
#include <iostream.h>
// linked list template

// forward teamplate class declaration
template <class T> class Cell;
template <class T> class ListIterator;

// type T must define operator== and overload <<

template <class T>
class List
{   friend class ListIterator<T>;
 public:
   List() : head(NULL) { }              // empty list constructor
   List(T& c)                           // list with first cell
   //: head (new Cell<T>(c,NULL)) { }
   : head (new Cell<T>(c)) { }
   Cell<T>* first() { return head; }   // first cell
   static Cell<T>* next(Cell<T>* p)     // next cell
     { return (p ? p->next : p); }
   static int is_end(Cell<T>* p)        // end test
     { return(p == NULL); }
   Cell<T>* last();                     // last cell
   Cell<T>* find(T& c);                // first value equals c
   int substitute(T& r, T& s);          // r for first s on list
   int remove(T& c);                    // c from entire list
   void remove(Cell<T>* cell);          // remove given cell
   int shorten(int n);                  // remove first n cells
   static T& content(Cell<T>* p)
     {  return p->value();  }
   int put_on(T& c);                    // insert in front
   int insert(T& c, Cell<T>* cell);     // insert after cell
   int append(T& c)                     // insert at end
     { return insert(c, last()); }
   int is_empty()
     { return(is_end(head)); }
   void display(ostream& out,           // display from p to end
                Cell<T>* p) const;
   void display(ostream& out) const
   { display(out, head); }              // display whole list
   ~List();                             // destructor
 private:
   int equal(T&, T&);                   // equality test
   Cell<T>* head;                       // first cell of list
   void free();                         // free all cells
};
```

```
template <class T>
class ListIterator
{  public:
      ListIterator(List<T>& x) : l(x)
      { cur = pre = x.first(); }
      int operator()(T& n); // returns next entry in n
      void del();           // erases last retrieved entry from list
  private:
      List<T>& l;
      Cell<T>* cur;
      Cell<T>* pre;
};

template <class T>
class Cell
{ friend class List<T>;
 friend class ListIterator<T>;
 private:
   T& value() { return(*val); }
   void value(T& v) { val = &v; } // set value
   T* val;                        // store pointer to type T
   Cell<T>* next;
   Cell(T& c, Cell<T>* ptr = NULL)
   :  val(&c), next(ptr) { }   // constructor
};
#endif
```

```
///////    File: List.C    ///////
#include "List.h"

template <class T>
inline int List<T>::equal(T& r, T& s)
{  return( r == s );  }

template <class T>
List<T>::~List() {   free();  }

template <class T>
void List<T>::free() // private method
{        Cell<T>* n, *p = head;
        while (p)
        {   n = p->next;
            delete p;
            p=n; } }

template <class T>
int List<T>::shorten(int n)
{      while (n-- && head)
     {   Cell<T>* tmp=head;
         head = head->next;
         delete(tmp);
     }
     return(-++n); }

template <class T>
int List<T>::remove(T& c)
{  Cell<T> *tmp, *p = head;
  int count = 0;
     while (p->next)       // treat all but head cell
     {  if ( equal((p->next)->value(), c) )
        {   count++;
            tmp = p->next;
            p->next = tmp->next;
            delete(tmp); // free up store
        }
        else
               p = p->next;
     }
     if ( equal(head->value(), c) ) // treat head cell
     {  tmp = head;
        head = head->next;
        delete(tmp);
        count++;
     }
     return(count);         // number of items removed
}
```