```
#ifndef LIST_H
#define LIST_H

// forward declcarations
template <typename T> class List;
template <typename T> std::ostream& operator<<(std::ostream&, const List<T>&);

template <typename T>
class List {
        private:
        struct Node     // an all public class with data only, no member ftns
        {
                T       data;
                Node    *prev;
                Node    *next;

                Node(const T& d = T(), Node *p = NULL, Node *n = NULL) : \
                        data(d), prev(p), next(n) \
                                                { };

        };

        public:
        class const_iterator    // returns const reference when *'ed
        {
                public:
                const_iterator() : curp(NULL)           { };

                const T& operator*() const
                {
                  return retrieve();
                }

                const_iterator&  operator++()           // prefix ++itr
                {
                  curp = curp->next;
                  return *this;
                }

                const_iterator operator++(int)          // postfix itr++
                {
                        const_iterator old = *this;

                  ++(*this);
                  return old;
                }

                bool operator==(const const_iterator& rhs) const
                {
                  return curp == rhs.curp;
                }

                bool operator!=(const const_iterator& rhs) const
                {
                  return !(curp == rhs.curp);
                }

                protected:
                Node    *curp;

                T& retrieve() const
                {
                  return curp->data;
                }

                const_iterator(Node *p) : curp(p)       { }

                friend class List<T>;
        };

        class iterator : public const_iterator  // returns reference *'ed
        {
                public:
                iterator()                              { }

                T& operator*()
                {
                  return retrieve();
                }

                const T& operator*() const
                {
                  return const_iterator::operator*();
                }

                iterator& operator++()                  // prefix ++itr
                {
                  curp = curp->next;
                  return *this;
                }

                iterator operator++(int)                // postfix itr++
                {
                        iterator old = *this;
                  ++(*this);
                  return old;
                }

                protected:
                iterator(Node *p) : const_iterator(p)   { };

                friend class List<T>;
        };

        public:
        // constructors (overloaded)
        List();

        // copy constructor
        List(const List& rhs);

        // destructors
        ~List()                                 {
                                                  clear();
                                                  delete head;
                                                  delete tail;
                                                }

        // friends -- note the extra <> telling the compiler to instantiate
```

```
        // a templated version of the operator<< -- <T> is also legal, i.e.,
        // friend std::ostream& operator<< <T>(std::ostream& s, const List&);
friend  std::ostream& operator<< <>(std::ostream& s, const List& rhs);
friend  std::ostream& operator<<(std::ostream& s, List *rhs)
                                        { return(s << (*rhs)); }

        // assignment operator
        const List&     operator=(const List&);

        // operators

        // iterator functions
        iterator        begin()                 { return iterator(head->next); }
        const_iterator  begin() const           { return const_iterator(head->next); }
        iterator        end()                   { return iterator(tail); }
        const_iterator  end() const             { return const_iterator(tail); }

        iterator        insert(iterator, const T&);
        iterator        erase(iterator);
        iterator        erase(iterator, iterator);

        // members
        int             size() const            { return sz; }
        bool            empty() const           { return size() == 0; }
        void            clear()                 { while(!empty()) pop_front(); }

        T&              front()                 { return *begin(); }
        const T&        front() const           { return *begin(); }
        T&              back()                  { return *--end(); }
        const T&        back() const            { return *--end(); }
        iterator        push_front(const T& o)  { return insert(begin(), o); }
        iterator        push_back(const T& o)   { return insert(end(), o); }
        iterator        pop_front()             { return erase(begin()); }
        iterator        pop_back()              { return erase(--end()); }

        // private: only available to this class
        private:
        int     sz;
        Node    *head;
        Node    *tail;
};

#endif
```