```
#include <iostream>

#include "list.h"

template <typename T>
std::ostream& operator<<(std::ostream& s, const List<T>& rhs)
{
  return s;
}

template <typename T>
List<T>::List()
{
  // this constructor creates an emply List object
  sz = 0;
  head = new Node;
  tail = new Node;
  head->next = tail;    // head->prev = NULL done in Node constructor
  tail->prev = head;    // tail->next = NULL done in Node constructor
}

template <typename T>
List<T>::List(const List<T>& rhs)
{
  // this constructor creates the List object given a constant
  // reference to another List object (result is a copy of
  // the other object)
  sz = 0;
  head = new Node;
  tail = new Node;
  head->next = tail;    // head->prev = NULL done in Node constructor
  tail->prev = head;    // tail->next = NULL done in Node constructor
  *this = rhs;
}

template <typename T>
const List<T>& List<T>::operator=(const List<T>& rhs)
{
  if(this == &rhs)      // standard alias test
    return *this;

  clear();

  for(const_iterator itr = rhs.begin(); itr != rhs.end(); ++itr)
    push_back(*itr);

  return *this;
}

template <typename T>
List<T>::iterator List<T>::insert(List<T>::iterator itr, const T& rhs)
{
        Node *p = itr.current;

  // insert rhs before itr
  sz++;
  return iterator(p->prev = p->prev->next = new Node(rhs,p->prev,p));
}

template <typename T>
List<T>::iterator List<T>::erase(List<T>::iterator)
{
        Node                    *p = itr.current;
        List<T>::iterator       ret(p->next);

  // erase item at itr
  p->prev->next = p->next;
  p->next->prev = p->prev;
  delete p;
  sz--;

  return ret;
}

template <typename T>
List<T>::iterator List<T>::erase(List<T>::iterator start, List<T>::iterator end)
{
  for(List<T>::iterator itr = start; itr != end; ) itr = erase(itr);
}

////////////////////////// specializations ///////////////////////////////////
template class List<int>;
template std::ostream& operator<<(std::ostream&, const List<int>&);

template class List<float>;
template std::ostream& operator<<(std::ostream&, const List<float>&);

template class List<double>;
template std::ostream& operator<<(std::ostream&, const List<double>&);
```

```
#include <iostream>
#include <cstdlib>
#include <sys/time.h>

#include "array.h"

int main()
{
	int		num,seeknum;
	List<int>	list;

	struct timeval	tp;
	unsigned int	seed;

  // get time of day
  gettimeofday(&tp,NULL);

  // use microseconds as the seed
  seed = (unsigned int)tp.tv_usec;
  srand(seed);
  std::cerr << "seed = " << seed << std::endl;

  for(int i=0;i<10;i++) {
    num = static_cast<int>((float)rand()/(float)RAND_MAX*100.0);
    list.push_back(num);
  }

  std::cout << "List:\n" << list << std::endl;
  for(List<int>::iterator itr = list.begin(); itr != list.end(); itr++) {
    std::cout << *itr;
  }
}
```