

```
#include <iostream>

#include "list.h"

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

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

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

template <typename T>
const list_t<T>& list_t<T>::operator=(const list_t<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>
typename
list_t<T>::iterator list_t<T>::insert(list_t<T>::iterator itr, const T& rhs)
{
    node_t *p = itr.curp;

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

template <typename T>
typename
```

```
list_t<T>::iterator list_t<T>::erase(list_t<T>::iterator itr)
{
    typename list_t<T>::node_t      *p = itr.curp;
    typename list_t<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>
typename
list_t<T>::iterator list_t<T>::erase(list_t<T>::iterator start, list_t<T>::iterator end
)
{
    for(typename list_t<T>::iterator itr = start; itr != end; ) itr = erase(itr);
}

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

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

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

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

#include "list.h"

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

    struct timeval tp;
    unsigned int seed=1337;

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

    // use microseconds as the seed
    //seed = (unsigned int)tp.tv_usec;
    srand(seed);

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