```
#include "List.H"

List::List()
        : _n(0), _size(10), _current(0)
{
        _list = new int [_size];
}

List::~List()
{
        if (_list != NULL)
                delete [] _list;

        _current = -1;
        _size = _n = 0;
}

List::List(const List & rhs)
        : _n(rhs._n), _size(rhs._size), _current(rhs._current
)
{
        _list = new int [_size];
        for (int i=0; i<_n; i++)
                _list[i] = rhs._list[i];
}
```

```
List &
List::operator =(const List &rhs)
{
        _n = rhs._n;
        _size = rhs._size;
        _current = rhs._current;

        _list = new int [_size];
        for (int i=0; i<_n; i++)
                _list[i] = rhs._list[i];

        return *this;
}

bool
List::insert(int x) {
        //  Since current is the position for
        //  the next insertion, it pays to know
        //  a little about it.
        if (_current < 0)
                return false;

        if (_current > _n)
                return false;

        //  If we're out of space for new elements,
```

```
        //  allocate some more...
        if (_n == _size) {
                int * tmp = _list;
                _list = new int [2*_size + 1];
                for (int i=0; i<_n; i++)
                        _list[i] = tmp[i];
                delete [] tmp;
                _size = 2*_size+1;
        }


        for (int i=_n-1; i>=_current; i--) {
                _list[i+1] = _list[i];
        }
        _list[_current] = x;
        _n++;

        return true;
}

void
List::remove()
{
        if (_current < 0 || _current >= _n)
                return;
```

```
        for (int i=_current; i<_n; i++)
                _list[i] = _list[i+1];
        _n--;
}

void
List::next()
{
        if (_current >= _n)
                _current = _n;
        else
                _current++;
}

void
List::previous()
{
        if (_current <= 0)
                _current = -1;
        else
                _current--;
}

int
List::current()
{
```

```
        if (_current < 0 || _current >= _n)
                cerr << "Uh oh." << endl;

        return _list[_current];
}

void
List::reset()
{
        _current = 0;
}

bool
List::isEmpty()
{
        return _n == 0;
}

bool
List::isPastEnd()
{
        return _current >= _n;
}

ostream &
operator <<(ostream & os, List & l)
```

```
{
        int old_current = l._current;

        if (l.isEmpty())
                return os << "()";

        os << "(";

        bool first_pass=true;
        for (l.reset() ; !l.isPastEnd(); l.next()) {
                if (first_pass) {
                        os << l.current();
                        first_pass = false;
                } else
                        os << "," << l.current();
        }

        l._current = old_current;
        os << ")";

        return os;
}
```

```
#ifndef CMSC202_LIST_H
#define CMSC202_LIST_H

#include <iostream.h>

class List {
public:
        List();
        ~List();
        List(const List &);
        List & operator =(const List &);

        bool insert(int);
        void remove();
        bool isEmpty();

        bool isPastEnd();
        void next();
        void previous();
        int  current();
        void reset();

private:
        int * _list;
        int _size, _n, _current;
```

```
        friend ostream & operator <<(ostream &, List &);
};

ostream & operator <<(ostream & os, List & l);

#endif
```