```
//Class: CSE 330
// Term: Spring 2014
// Instructor: George M. Georgiou
// Name: Seth Lemanek
// Lab 7
// Title: Deque.h
// Note: I have been able to compile this program but it refuses to work with test.cpp by giving zero
output
# include <vector>

using namespace std;

template <class T> class DequeIterator;

template <class T> class Deque {
public:
        typedef DequeIterator<T> iterator;
        typedef T value_type;

                // constructors
        Deque(): vecOne(), vecTwo() { }
        Deque(const unsigned int size, const T& initial): vecOne(size/2, initial), vecTwo(size-(size/2),
initial) { }
        Deque(const Deque<T> & d): vecOne(d.vecOne), vecTwo(d.vecTwo) { }
     ~Deque() { } // destructors for vecOne and vecTwo are automatically called
               // never call a destructor explicitly

                // operations
        T &     operator [ ] (unsigned int);
        T &     front ();
        T &     back ();
        bool    empty () { return vecOne.empty () && vecTwo.empty (); }
        iterator begin () { return iterator(this, 0); }
        iterator end () { return iterator(this, size ()); }
        void    erase (iterator);
        void    erase (iterator &, iterator &);
        void    insert (iterator &, T &);
        int     size () { return vecOne.size () + vecTwo.size (); }
        void    push_front (T  value) { vecOne.push_back(value); }
        void    push_back (T  value) { vecTwo.push_back(value); }
        void    pop_front ();
        void    pop_back ();

protected:
        vector<T> vecOne;
        vector<T> vecTwo;
};

template <class T> T & Deque<T>::front ()
```

```
        // return first element in deque
{
        if (vecOne.empty ())
                return vecTwo.front ();
        else
                return vecOne.back ();
}
template <class T> T & Deque<T>::back ()
        // return last element in deque
{
        if (vecTwo.empty ())
                return vecOne.front ();
        else
                return vecTwo.back ();
}

template <class T> void Deque<T>::insert(DequeIterator<T> & pos, T & value) {
   int index = pos.index;
   int n = vecOne.size();

   if(index < n)
      vecOne.insert(vecOne.begin() + ((n)-index), value);
   else
      vecTwo.insert((vecTwo.begin() + (index-n)),value);
}

template <class T> void Deque<T>::erase (DequeIterator<T> & start, DequeIterator<T> & stop) {
   const int vonesize = vecOne.size();
   iterator pos;

   if(start.index < vecOne.size() && stop.index > (vecOne.size()-1)) {
      vecOne.erase(vecOne.begin(), (vecOne.begin() + (vonesize- start.index)));
      vecTwo.erase(vecTwo.begin(), (vecTwo.begin() + (stop.index - vonesize)));
   }
   else if(start.index < vecOne.size() && stop.index < vecOne.size()) {
      for(pos=start; pos != stop; pos++)
         erase(pos);
   }
   else if(start.index >= vecOne.size() && stop.index > vecOne.size()) {
      for(pos=start; pos != stop; pos++)
         erase(pos);
   }
}

template <class T> void Deque<T>::pop_back()
{
        if (vecTwo.empty())
                vecOne.erase(vecOne.begin());
        else
```

```
		vecTwo.pop_back();
}
template <class T> void Deque<T>::pop_front ()
	// remove first element in deque
{
	if (vecOne.empty ())
		vecTwo.erase(vecTwo.begin ());
	else
		vecOne.pop_back ();
}

template <class T> T & Deque<T>::operator [ ] (unsigned int index)
	// return given element from deque
{
	int n = vecOne.size ();
	if (index <= n)
		return vecOne [ (n-1) - index ];
	else
		return vecTwo [ index - n ];
}

template <class T> class DequeIterator {
	friend class Deque<T>;
	typedef DequeIterator<T> iterator;
public:
		// constructors
	DequeIterator(): theDeque(0), index(0) { }
	DequeIterator(Deque<T> * d, int i): theDeque(d), index(i) { }
	DequeIterator(const iterator & d): theDeque(d.theDeque), index(d.index) { }

		// iterator operations
	T & operator * () { return (*theDeque)[index]; }
	iterator & operator ++ () { ++index; return * this; }
	iterator operator ++ (int)
	{			// clone, update, return clone
	iterator clone(theDeque, index);
	index++;
	return clone;
	}// prefix change
	iterator & operator -- (int) { --index; return * this; }
	iterator operator -- (); // postfix change
	bool operator == (iterator & r)
		{ return theDeque == r.theDeque && index == r.index; }
	bool operator < (iterator & r)
		{ return theDeque == r.theDeque && index < r.index; }
	bool operator != (iterator  r)
		{ return theDeque != r.theDeque && index != r.index; }
	T & operator [ ] (unsigned int i)
		{ return (*theDeque) [index + i]; }
	iterator operator = (iterator r)
		{ theDeque = r.theDeque;  index = r.index; return * this;}
	iterator operator + (int i)
		{ return iterator(theDeque, index + i); }
	iterator operator - (int i)
		{ return iterator(theDeque, index - i); }

protected:
	Deque<T> * theDeque;
	int index;
};


//template <class T> DequeIterator<T> DequeIterator<T>::operator ++ (int)
	// postfix form of increment


template <class T> void Deque<T>::erase (DequeIterator<T> itr)
	// erase value from deque
{
	int index = itr.index;
	int n = vecOne.size ();
	if (index < n)
		vecOne.erase (vecOne.begin () + ((n-1) - index));
	else
		vecTwo.erase (vecTwo.begin () + (n - index));
}


//test.cpp


#include <iostream>
#include <cassert>
//#include <queue>
#include "deque.h"

using namespace std;

int main()
{
	Deque<int> d;

	d.push_back(10);
	d.push_back(20);
	d.push_front(1);
	d.push_front(2);
	d.push_front(3);
	assert (d.back() == 20);
```

```
        Deque<int> c;
        c = d;

        Deque<int>::iterator i;
        for (i = d.begin(); i != c.end(); ++i)
                cout << *i << " ";
        cout << endl;
}
```