```
// Illustrate simple behaviors of both the "vector" and "deque"
// objects.

#include <iostream>
// To use vector and deque, you must include the appropriate header
#include <vector>
#include <deque>

// namespace not required, but saves typing
using namespace std;

int main()
{
  vector<int> v;  // This creates an empty vector of integers
  // vectors have a "size" member function to report the number
  // of elements in the vector (should be zero in this case)
  cout << "v has " << v.size() << " elements" << endl;
  // Vectors can be "extended" by adding new elements at the end
  // with the "push_back(int)" member function.  The below adds
  // 10 elements to the vector v
  for (int i = 0; i < 10; ++i)
    {
      v.push_back(i);
    }
  // Size should now be 10
  cout << "v now has " << v.size() << " elements" << endl;
  // Vectors have an indexing "[]" operator
  for (int i = 0; i < v.size(); ++i)
    {
      cout << "element " << i << " is " << v[i] << endl;
    }
  // You can get a copy of either the first or last element in the
  // vector using "front()" and "back()" member functions.
  cout << "v.front() is " << v.front() << " v.back() is " << v.back() << endl;
  // NOte that front() and back() do not remove the elements.
  // For a vector, you can only remove from the back using "pop_back()",
  // removing the most recently added element.
  //The following code loops getting the back() element and removing it.
  // Also notice the use of the empty() member function.
  // Also be aware the neither front() nor back() can legally be called
  // on an empty vector.
  while(!v.empty())
    {
      int b = v.back();
      v.pop_back();
      cout << "back element is " << b << " new size " << v.size() << endl;
    }
  // There is another vector constructor that is useful.  The following
  // declaration creates a new vector v1 that initially contains 10
  // elements, all set to the value 100
  vector<int>v1(10, 100);
  cout << "Size of v1 is " << v1.size() << endl;
  cout << "v1[0] is " << v1[0] << endl;
  // Finally note the "clear()" member function that removes all
  // elements from the vector.
  v1.clear();
```

Program vector-deque.cc

```
  cout << "Size of V1 after clear is " << v1.size() << endl;

  // The limitation of a vector is that you can only add and remove
  // elements from the end, so it essentially acts like a LIFO
  // stack.  In many cases we want a FIFO queue where we can add
  // and remove elements from either the front or back.  This is
  // accomplished using a "double-ended queue" (deque).  It has all
  // the functionality of the vector described above, and also has
  // "push_front()" and "pop_front()" member functions.
  deque<int> d1;
  for (int i = 0; i < 10; ++i)
    { // Add to back, just like vector
      d1.push_back(i);
    }
  for (int i = 0; i < 10; ++i)
    { // Add to front
      d1.push_front(i * 100);
    }
  // And print out (and remove) from front to back
  while(!d1.empty())
    {
      int v = d1.front();
      d1.pop_front();
      cout << "v is " << v << endl;
    }
  // Finally clear the elements.  This is technically not neeed
  // as the destructor for both the vector and deque clear the
  // elements as the vector/deque is destroyed.
  d1.clear();
  cout << "Final size of d1 is " << d1.size() << endl;
}
```

Program vector-deque.cc (continued)