```
// e12_1.cpp
// http://www.buzluca.info/oop
// Simple vector example

#include <iostream>
#include <vector>
using namespace std;

int main() {
	vector<int> nums1;
	nums1.insert(nums1.begin(), -999);		// insert to the beginning -999
	nums1.insert(nums1.begin(), 14);		// 14, -999
	nums1.insert(nums1.end(), 25);			// insert to the end 14, -999, 25

	int i;
	for (i=0; i<nums1.size(); i++)			// prints elements of nums1
		cout << nums1[i] << " ";
	cout << endl;

	nums1.erase( nums1.begin() );			// -999, 25

	for (i=0; i<nums1.size(); i++)
		cout << nums1[i] << " ";		// prints elements of nums1
	cout << endl;

	vector<int> nums2 = nums1;
	nums2.insert(nums2.begin(), 32);   // 32, -999, 25

	vector<int> nums3;
	nums3 = nums2;

	for (i=0; i<nums3.size(); i++)
		cout << nums3[i] << " ";

	return 0;
}
-------------------------------------------------------

// e12_2.cpp
// http://www.buzluca.info/oop
//    demonstrates push_back(), operator[], size() **/
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  vector<int> v;                // create a vector of ints

  v.push_back(10);              // put values at end of array
  v.push_back(11);
  v.push_back(12);
  v.push_back(13);

  v[0] = 20;                    // replace with new values
  v[3] = 23;

  for(int j=0; j<v.size(); j++)    // display vector contents
    cout << v[j] << ' ';
  return 0;
}
/*
I use the vectorâ€™s default (no-argument) constructor to create a vector v.
```

As with all STL containers, the template format is used to specify the type of variable the container will hold; in this case, type int.
I donâ€™t specify the containerâ€™s size, so it starts off at 0.
The push_back() member function inserts the value of its argument at the back of the vector.
(The back is where the element with the highest index number is.)
The front of a vector (where the element with index 0 is), unlike that of a list or queue,
is not accessible.
Here I push the values 10, 11, 12, and 13, so that v[0] contains 10, v[1] contains 11,
v[2] contains 12, and v[3] contains 13.

Once a vector has some data in it, this data can be accessed-both read and written to-using
the overloaded [] operator, just as if it were in an array. I use this operator to change
the first element from 10 to 20 and the last element from 13 to 23.
Hereâ€™s the output from VECTOR:
20 11 12 23
*/

```
// e12_3.cpp
// http://www.buzluca.info/oop
// demonstrates constructors, swap(), empty(), back(), pop_back() **/
#include <iostream>
#include <vector>
using namespace std;

int main()
{                                   // an array of floats
  float arr[] = { 1.1, 2.2, 3.3, 4.4 };
  vector<float> v1(arr, arr+4);           // initialize vector to array
  vector<float> v2(4);                    // empty vector of size 4

  v1.swap(v2);                          // swap contents of v1 and v2

  while( !v2.empty() )                  // until vector is empty,
    {
    cout << v2.back() << ' ';           // display the last element
    v2.pop_back();                      // remove the last element
    }                                 // output: 4.4 3.3 2.2 1.1
  return 0;
}
/*
```

Iâ€™ve used two new vector constructors in this program. The first initializes the vector v1
with the values of a normal C++ array passed to it as an argument.
The arguments to this constructor are pointers to the start of the array and
to the element one past the end. The second constructor sets v2 to an initial size of 4,
but does not supply any initial values. Both vectors hold type float.
The swap() member function exchanges all the data in one vector with all the data in another,
keeping the elements in the same order. In this program, only garbage data is in v2,
so itâ€™s swapped with the data in v1. I display v2 to show it now contains the data that was in v1.
The output is
4.4, 3.3, 2.2, 1.1

The back() member function returns the value of the last element in the vector.
I display this value with cout. The pop_back() member function removes the last element in the vector.
Thus, each time through the loop there is a different last element.
Itâ€™s a little surprising that pop_back() does not simultaneously return the value of the last element
and remove it from the vector, as youâ€™ve seen in previous examples with stacks,
but it doesnâ€™t, so back() must be used as well.)
*/

```
// e12_4.cpp
// http://www.buzluca.info/oop
//demonstrates insert(), erase() **/
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  int arr[] = { 100, 110, 120, 130 };  // an array of ints
  vector<int> v(arr, arr+4);           // initialize vector to array
  int j;
  cout << "\nBefore insertion: ";
  for(j=0; j<v.size(); j++)            // display all elements
    cout << v[j] << ' ';

  v.insert( v.begin()+2, 115);         // insert 115 at element 2
  cout << "\nAfter insertion:  ";
  for(j=0; j<v.size(); j++)            // display all elements
    cout << v[j] << ' ';

  v.erase( v.begin()+2 );              // erase element 2

  cout << "\nAfter erasure:    ";
  for(j=0; j<v.size(); j++)            // display all elements
    cout << v[j] << ' ';
  return 0;
}
/*
The insert() member function (this version of it) takes two arguments:
the place where an element will be inserted in a container and the value of the element.
I add 2 to the begin() member function to specify element 2 (the third element) in the vector.
The elements from the insertion point to the end of the container are moved upward
to make room and the size of the container is increased by 1.

The erase() member function removes the element at the specified location.
The elements above the deletion point are moved downward and the size of the container
is decreased by 1. Hereâ€™s the output from VECTINS:

Before insertion: 100 110 120 130
After insertion:  100 110 115 120 130
After erasure:    100 110 120 130
*/


-------------------------------------------
```