// ell_1.cpp
// 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;
}
// ell_2.cpp
//    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 vector’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 don’t specify the container’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

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.
Here’s the output from VECTOR:
20 11 12 23
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    float arr[] = { 1.1, 2.2, 3.3, 4.4 }; // an array of floats
    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
    }
    return 0;
}
```

I’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 it’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.

It’s a little surprising that pop_back() does not simultaneously return the value of the last element and remove it from the vector, as you’ve seen in previous examples with stacks, but it doesn’t, so back() must be used as well.)
// e11_4.cpp
// 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. Here’s the output:

Before insertion: 100 110 120 130
After insertion: 100 110 115 120 130
After erasure: 100 110 120 130
*/
// e11_5.cpp
// demonstrates push_front(), front(), pop_front()
#include <iostream>
#include <list>
using namespace std;

int main()
{
    list<int> ilist;
    ilist.push_back(30); // push items on back
    ilist.push_back(40);
    ilist.push_front(20); // push items on front
    ilist.push_front(10);

    int size = ilist.size(); // number of items

    for(int j=0; j<size; j++)
    {
        cout << ilist.front() << ' '; // read item from front
        ilist.pop_front(); // pop item off front
    }
    return 0;
}

/*
The push_front(), pop_front(), and front() member functions are similar to
push_back(), pop_back(), and back(), which you’ve already seen at work with vectors.
Note that you can’t use random access for list elements because such access is too
slow; that is, the [] operator is not defined for lists. If it were, this operator would
need to traverse along the list, counting elements as it went, until it reached the correct
one, a time-consuming operation. If you need random access, you should use a vector or a
deque.
*/
// e11_6.cpp
// demonstrates reverse(), merge(), and unique() **/
#include <iostream>
#include <list>
using namespace std;

int main()
{
    int arr1[] = { 40, 30, 20, 10 };
    int arr2[] = { 15, 20, 25, 30, 35 };

    list<int> list1(arr1, arr1+4);
    list<int> list2(arr2, arr2+5);

    list1.reverse();                     // reverse list1: 10 20 30 40
    list1.merge(list2);                  // merge list2 into list1
    list1.unique();                      // remove duplicate 20 and 30
    int size = list1.size();
    for(int j=0; j<size; j++)            // for every item
    {
        cout << list1.front() << ' ';     // read item from front
        list1.pop_front();                // pop item off front
    }
    return 0;
}

/*
The first list is in backward order, so I return it to normal sorted order using the
reverse() member function. (It’s quick to reverse a list container because both ends are
accessible.)
This is necessary because the second member function, merge(), operates on two lists and
requires both of them to be in sorted order. Following the reversal, the two lists are
10, 20, 30, 40
15, 20, 25, 30, 35

Now the merge() function merges list2 into list1, keeping everything sorted and expanding
list1 to hold the new items. The resulting content of list1 is
10, 15, 20, 20, 25, 30, 30, 35, 40

Finally, I apply the unique() member function to list1. This function finds adjacent pairs
of elements with the same value and removes all but the first. The contents of list1 are then displayed. The output of LISTPLUS is
10, 15, 20, 25, 30, 35, 40

To display the contents of the list, I use the front() and pop_front() member functions
in a for loop. Each element, from front to back, is displayed and then popped off the list.
The process of displaying the list destroys it.
*/
```cpp
// ell_7.cpp
// demonstrates push_back(), push_front(), front() /
#include <iostream>
#include <deque>
using namespace std;

int main()
{
    deque<int> deq;
    deq.push_back(30); // push items on back
    deq.push_back(40);
    deq.push_back(50);
    deq.push_front(20); // push items on front
    deq.push_front(10);
    deq[2] = 33; // change middle item

    for(int j=0; j<deq.size(); j++)
        cout << deq[j] << ' '; // display items
    return 0;
}

/*
You’ve already seen examples of push_back(), push_front(), and operator [].
They work the same for deques as for other containers. The output of this program is
10 20 33 40 50
*/
```
// e11_8.cpp
// Using the find algorithm with user defined-types (Complex Numbers)

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

class ComplexT{
    float re, im;
public:
    set(float r, float i){re=r; im=i;}
    bool operator==(const ComplexT &c) const{
        return re==c.re && im==c.im;
    }
};

int main()
{
    ComplexT z[3];
    z[0].set(1.1, 1.2);
    z[1].set(2.1, 2.2);
    z[2].set(3.1, 3.2);

    ComplexT zSearch;
    zSearch.set(2.1, 2.2);

    ComplexT *result;

    result=find(z, z+3, zSearch);
    if (result == z+3) cout << "Not found";
    else cout << "Found";
    return 0;
}
// ell_9.cpp
// Sort example
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

int main()
{
    string words[] =
        { "november", "kilo", "mike", "lima",
        "oscar", "quebec", "papa"};
    sort(words, words +7);
    for(int i =0 ; i<7; i++) cout << words[i] << endl;
    return 0;
}
// e11_10.cpp
// The sort algorithm uses the after function

#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

bool after(const string &left, const string &right)
{
    return left > right;
}

int main()
{
    string words[] =
    {"november", "kilo", "mike", "lima",
     "oscar", "quebec", "papa"};
    sort(words, words +7, after);
    for(int i =0 ; i<7; i++) cout << words[i] << endl;
    return 0;
}
// e11_11.cpp
// uses for_each() to output inches array elements as centimeters **/
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

void in_to_cm(float in)         // convert and display as centimeters
{                           
    cout << (in * 2.54) << ' ';  
}

int main()
{                                      // array of inches values
    float array[] = { 3.5, 6.2, 1.0, 12.75, 4.33 };  // vector of inches values
    vector<float> inches (array, array+5);  // output as centimeters
    for_each(inches.begin(), inches.end(), in_to_cm);
    return 0;
}
#include <iostream>
#include <set>
#include <string>
using namespace std;

void main()
{
    string names[] = {"Katie", "Robert", "Mary", "Amanda", "Marie"};
    set<string> nameSet(names, names+5); // initialize set to array

    set<string>::const_iterator iter; // iterator to set
    nameSet.insert("Jack"); // insert some more names
    nameSet.insert("Larry"); // no effect; already in set
    nameSet.insert("Barry");
    nameSet.erase("Mary"); // erase a name

    cout << "\nSize=" << nameSet.size() << endl;
    iter = nameSet.begin(); // display members of set
    while( iter != nameSet.end() )
        cout << *iter++ << 'n';

    string searchName; // get name from user
    cout << "\nEnter name to search for: ";
    cin >> searchName; // find matching name in set

    iter = nameSet.find(searchName);
    if( iter == nameSet.end() )
        cout << "The name " << searchName << " is NOT in the set.";
    else
        cout << "The name " << *iter << " IS in the set.";
}
// e11_13.cpp
// tests ranges within a set
#include <iostream>
#include <set>
#include <string>
using namespace std;

int main()
{
    // set of string objects
    set<string> city;
    // iterator to set
    set<string>::iterator iter;
    city.insert("New York");  // insert city names
    city.insert("Boston");
    city.insert("Norfolk");
    city.insert("Los Angeles");
    city.insert("Las Vegas");
    city.insert("Danbury");
    city.insert("Washington");
    city.insert("San Diego");
    city.insert("Albany");
    city.insert("Miami");
    city.insert("Chicago");

    iter = city.begin();  // display set
    while( iter != city.end() )
        cout << *iter++ << endl;

    string lower, upper;  // display entries in range
    cout << "Enter range (example A Azz): ";
    cin >> lower >> upper;
    iter = city.lower_bound(lower);
    while( iter != city.upper_bound(upper) )
        cout << *iter++ << endl;
    return 0;
}
// ell_14.cpp
// map: cities and plate numbers **/
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
    // set of string objects
    map< string, int > city_num;
    city_num["New York"] = 61;  // insert city names and numbers
    city_num["Washington"] = 01;
    city_num["Chicago"] = 22;
    city_num["Albany"] = 16;
    city_num["Boston"] = 34;
    city_num["Danbury"] = 53;
    city_num["Norfolk"] = 07;
    city_num["Miami"] = 35;
    city_num["Los Angeles"] = 31;
    city_num["San Diego"] = 06;
    city_num["Las Vegas"] = 67;

    string city_name;
    cout << "Enter a city: ";
    cin >> city_name;
    if (city_num.end() == city_num.find(city_name))
        cout << city_name << " is not in the database" << endl;
    else
        cout << "Number of " << city_name << ": " << city_num[city_name];
    return 0;
}