# STL

#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main(int argc, char\*\* argv) {

// Vectors are dynamic arrays. Fast random access but takes a little

// time to grow/shrink.

vector<int> ages; // Vector of ints

vector<string> names; // Vector of strings

//vector<Point> points; // Vector of a user-defined type

names.push\_back("Hello"); // Add to end

names.push\_back("There");

names.push\_back("World");

int s = names.size();

bool empt = names.empty(); // Why not "size==0" ?

cout << names.at(0) << endl; // Get element at

cout << names[0] << endl; // Overloaded [] returns reference

string a = names.front(); // First element

a = names.back(); // Last element

names[0] = "HELLO"; // Use reference to change the value

for(int x=0;x<names.size();++x) {

cout << x << ":" << names[x] << endl;

}

system("pause");

}

Iterators are cursors. They point to elements in a collection. You can quickly access the element pointed to. You can advance the iterator to the next element. Changing the collection might invalidate the iterator and you have to start it again.

Containers provide “.start()” to get a pointer to the first and “.end()” to get ONE-PAST the end.

int main(int argc, char\*\* argv) {

vector<string> names; // Vector of strings

names.push\_back("Hello"); // Add to end

names.push\_back("There");

names.push\_back("World");

vector<string>::iterator i = names.begin(); // Point to first element

cout << \*i << endl; // Use pointer style \* to access (operator overload)

++i; // Next element

cout << \*i << endl;

--i;

cout << \*i << endl;

i = i + 2;

cout << \*i << endl;

i = names.begin();

++i;

names.insert(i,"NEW STRING");

// Typical for-loop over all elements;

for(vector<string>::iterator j=names.begin();j!=names.end();++j) {

cout << \*j << endl;

}

system("pause");

}

# List

#include <iostream>

#include <list>

#include <string>

using namespace std;

int main(int argc, char\*\* argv) {

// List maintains a linked-list making it super fast to move things around

// and add to end/beginning. But getting to a particular element takes

// time.

list<string> names;

names.push\_back("Hello");

names.push\_back("There");

names.push\_back("World");

// front(), back(), insert(), erase()

// No "at" or [] operator. You have to iterate to the desired element.

// Typical for-loop over all elements;

for(list<string>::iterator j=names.begin();j!=names.end();++j) {

cout << \*j << endl;

}

system("pause");

}

names.push\_back("ZZZ");

names.push\_back("AAA");

names.push\_back("BBB");

names.sort(); // Uses the "operator<" you define for a class or built-in "<".

// You can also define a function that takes two elements and returns bool true

// if first goes before second.

# map

int main(int argc, char\*\* argv) {

map<string,string> names;

// Put/Get ... using STL "pair".

names.insert(pair<string,string>("123","Bob"));

cout << names.at("123");

// Use the overloaded []

names["123-45-6789"] = "Chris";

names["000-11-2222"] = "John";

cout << names["000-11-2222"] << endl;

cout << names["99"] << endl;

// Must return something if not found. If it were pointers we could test

// for a null pointer to know the find failed.

// The thing returned is a new object of the type using default constructor.

// In this case ... an empty string

map<string,string>::iterator i = names.find("888");

if(i==names.end()) {

cout << "NOT FOUND" << endl;

}

system("pause");

}

int main(int argc, char\*\* argv) {

map<string,string> names;

names["123-45-6789"] = "Chris";

names["000-11-2222"] = "John";

map<string,string>::iterator i = names.begin();

// The iterator moves over pair<string,string>. The "pair" class

// has a "first" and a "second":

cout << i->first << endl;

cout << i->second << endl;

system("pause");

}