UNIVERSITY OF CAMBRIDGE

Department of Engineering
IT Services

C++ vector memory and 2D vectors

Vector constructors

C++ vectors can grow dynamically, but not in all situations. For example, if you have a vector v of size 2, then v[10]=5; won't make the vector extend. So you still need to ensure that memory will be created for your vectors. Constructors will often create the memory for you. Here's an example that uses various constructors

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

int main()
{
// An empty vector
   vector <int> v;

// A vector with 5 elements, each an integer
   vector<int> v1(5);

// An array with 5 elements, each of which being an
// empty vector of integers. Don't confuse this with
// the previous line
   vector<int> v1array[5];

// A vector with 5 elements each having the value 99
   vector<int> v2(5, 99);

// A vector with the size and values of v2
   vector<int> v3(v2);

// Another, more flexible way to create a vector
// with the size and values of v2
   vector<int> v4(v2.begin(),v2.end());
}
```

2D vectors

You can create a 2D array by doing int a[5][7], but vector<int> a(5,7) won't create a 2D vector - it will create a 5-element vector full of 7s. Instead, you need to create a vector of vectors. Here are some ways to do it.

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

int main()
{
// An empty vector of vectors. The space
// between the 2 '>' signs is necessary
   vector<vector<int> > v2d;

// If you intend creating many vectors
// of vectors, the following's tidier
   typedef vector<vector<int> > IntMatrix;
   IntMatrix m;

// Now we'll try to create a 3 by 5 "matrix".
// First, create a vector with 5 elements
   vector<int> v2(5, 99);

// Now create a vector of 3 elements.
// Each element is a copy of v2
   vector<vector<int> > v2d2(3,v2);

// Print out the elements
   for(int i=0;i<v2d2.size(); i++) {
      for (int j=0;j<v2d2[i].size(); j++)
        cout << v2d2[i][j] << " ";
      cout << endl;
   }
}
```

Expanding vectors

The push_back member function will extend the vector. It would be inefficient for C++ to extend the vector a byte at a time on demand, so more space is often reserved internally for the vector than is requested. You can find out this reserved size by using the capacity member function. If you ask for the vector to grow beyond this size, it will, but there are performance penalities. Try this code on your machine. It asks for extra elements one at a time, printing the capacity out as it goes. Watch how the capacity increases.

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

int main()
{
vector <int> v;

 for (int i=0; i<50; i++) {
     cout << "size=" << v.size()
     << " and capacity=" << v.capacity() << endl;
     v.push_back(i);
 }
}
```

If you want to set the capacity yourself, you can use the reserve() member function.

A consequence of this is that the address of the elements can change, so it's unwise to store an iterator to an element if the vector might change. Instead, store the index. The following example shows how to get the index of an element given an iterator to it.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int main()
{
vector <string> v(3);
  v[0]="apple";
  v[1]="banana";
  v[2]="carrot";
  vector<string>::iterator it=find(v.begin(),v.end(),"apple");
  cout << "Index of apple=" << std::distance(v.begin(), it) << endl;
  it=find(v.begin(),v.end(),"carrot");
  cout << "Index of carrot="  << std::distance(v.begin(), it) << endl;
}
```