CSCI 262

3 – Vectors

Arrays and Vectors

Arrays:
```c
int foo[10];
for (int j = 0; j < 10; j++)
    foo[j] = j;
```

Vectors (Stanford CPP Lib):
```c
#include "vector.h"

Vector<int> foo(10);
for (int j = 0; j < 10; j++)
    foo[j] = j;
```

Declares that this Vector will hold int values.
Gives an initial size to the Vector.

Advantage: Vectors

What if size of array is unknown at compile time?
```c
int main() {
    int n;
    ...
    // read n from file, etc.
    int foo[n]; // COMPILE ERROR
}
```

However, this is no problem:
```c
Vector<int> foo(n);
```

Do More with Vectors

If you have a Vector, you can append to it – it automatically resizes:
```c
Vector<int> foo;
for (int j = 0; j < 10; j++) {
    foo.add(j);
}
```

Output:
```c
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Initially size 0 without optional argument.

Other things you can do with Vector (but not arrays):
```c
Vector<int> foo = {0, 1, 2, 3, 4};
cout << "foo is size " << foo.size() << endl;
cout << foo << endl; // print a representation
foo.remove(3); // remove element 3
foo.insert(3, 123456); // insert an element before index 3
foo.clear(); // remove all elements
```

And more – see StanfordCPPLib docs!

Empty Vectors

Note that this:
```c
Vector<int> foo;
```
creates an empty Vector – one with no elements.

You should not do this:
```c
Vector<int> foo;
foo[5] = 0;
```
or
```c
cout << foo[5];
```
There is no element 5!

In the StanfordCPPLib, this will "throw an exception". That's a friendly kind of error message, but it will still halt your program.

If you try this with a standard library vector object, your program will crash with a memory exception error (segfault, segmentation fault, etc.)
Vector is a Template

We haven’t studied templates yet, but basically:

```cpp
Vector<T> x;
```
decares x as a Vector containing values of type T.

You can make a Vector to store any one type, e.g.

```cpp
Vector<string> svec;
svec.add("Hello");
```
```cpp
Vector<double> dvec = {3.14, 2.72, 1.62};
```
etc.

Initializing Vectors

You can use the Vector constructors to initialize a Vector to a particular size:

```cpp
Vector<int> foo(5); // fills with zeroes
foo[4] = 42;
cout << foo << endl;
```
Output:
```
{0,0,0,0,42}
```

You can also fill with something other than zero:

```cpp
Vector<int> foo(10, 17); // fills with 17s
cout << foo << endl;
```
Output:
```
{17,17,17,17,17,17,17,17,17,17}
```

2-D using Vectors

Vectors can even hold Vectors!

Here’s code to make a multiplication table:

```cpp
Vector<Vector<int>> table;
for (int i = 0; i <= 10; i++) {
    Vector<int> one_row;
    for (int j = 0; j <=10; j++) {
        one_row.add(i*j);
    }
    table.add(one_row);
}
```
You can now do this:

```cpp
// ask user for x, y (require x, y <= 10)
cout << x << " times " << y << " is " << table[x][y] << endl;
```

2-D with Initialization

```cpp
int rows = 11; int(columns = 11;
Vector<Vector<int>> table(rows, Vector<int>(columns));
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        table[i][j] = i*j;
    }
}
```

More on Vectors

We’ll return to Vectors when we look at collections (chapter 5 – you’re welcome to read ahead!)

Note there is also a C++ Standard Library class named vector which is almost identical to Vector. You can read about it in the standard library docs.