Lecture 1A

The array and vector classes

From the last part of Savitch chapter 7 and the Standard Template Library.
Learning Objectives

• Introduce Arrays and Vectors
  – Introduction to *vector* class
  – Introduction to the array *class*

We do this now so that you can use them “blindly”. You won’t understand them fully until we have covered *classes* and *templates* but they will be immediately useful.
array container class

• Array Introduction
  – Recall: arrays are fixed size
  – Formed from Standard Template Library (STL)
    • Using template class

#include<array>

The type is defined in the namespace std:
namespace std {
    template<typename T, size_t N>
    class array;
}
The *template* idea.

```cpp
#include<array>

The type is defined in the namespace std:
namespace std {
    template<typename T, size_t N>
    class array;
}

We can substitute “anything” for T which means:
std::array<std::string, 10> as1, as2;
std::array<int, 5> c3 = {1, 2, 3, 4, 5};
std::array<double, 250> c5 = {};

Are all ok – the template thing takes care of the different types.
Hereafter treat as ”C” arrays:
C3[4] = 1234;

NB – code is not ”managed” – C3[99] = 0 will not raise an exception.
Vectors

• Vector Introduction
  – Recall: arrays are fixed size
  – Vectors: "arrays that grow and shrink"
    • During program execution
  – Formed from Standard Template Library (STL)
    • Using template class
Vector Basics

• Similar to array:
  – Has base type
  – Stores collection of base type values

• Declared differently:
  – Syntax: vector<Base_Type>
    • Indicates template class
    • Any type can be "plugged in" to Base_Type
    • Produces "new" class for vectors with that type

  – Example declaration:
    vector<int> v;
Vector Use

• `vector<int> v;`
  – "v is vector of type int"
  – Calls class default constructor
    • Empty vector object created

• Indexed like arrays for access

• But to add elements:
  – Must call member function `push_back`

• Member function `size()`
  – Returns current number of elements
Vector Example:

Display 7.7 Using a Vector (1 of 2)

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

int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
    << "Place a negative number at the end.\n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size() = " << v.size() << endl;
        cin >> next;
    }
}
```
Vector Example:

Display 7.7  Using a Vector (2 of 2)

18    cout << "You entered:\n";
19    for (unsigned int i = 0; i < v.size( ); i++)
20        cout << v[i] << " ";
21    cout << endl;
22    return 0;
23 }

**Sample Dialogue**

Enter a list of positive numbers.
Place a negative number at the end.
2 4 6 8 -1
2 added. v.size = 1
4 added. v.size = 2
6 added. v.size = 3
8 added. v.size = 4
You entered:
2 4 6 8
Vector Efficiency

• Member function capacity()
  – Returns memory currently allocated
  – Not same as size()
  – Capacity typically > size
    • Automatically increased as needed

• If efficiency critical:
  – Can set behaviors manually
    • v.reserve(32); //sets capacity to 32
    • v.reserve(v.size()+10); //sets capacity to 10 more than size
Summary 1

• Constructors: automatic initialization of class data
  – Called when objects are declared
  – Constructor has same name as class

• Default constructor has no parameters
  – Should always be defined

• Class member variables
  – Can be objects of other classes
    • Require initialization-section
Summary 2

- **Constant call-by-reference parameters**
  - More efficient than call-by-value

- **Can *inline* very short function definitions**
  - Can improve efficiency

- **Static member variables**
  - Shared by all objects of a class

- **Vector classes**
  - Like: "arrays that grow and shrink"