CSCE 121
Introduction to Program Design & Concepts
Vectors

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

Vectors

- A data type defined in the Standard Template Library (discussed later)
- Vectors are like arrays that can change size automatically as your program runs
- Vectors, like arrays, have a base type
- To declare an empty vector with base type int:

  vector<int> v;

  - <int> identifies vector as a template class
  - You can use any base type in a template class:

    vector<string> v;

Declaring Vectors

- You must `#include<vector>`
- Declare a vector to hold `int` element:

```
vector<int> scores;
```

- Declare a vector with initial size 30:

```
vector<int> scores(30);
```

- Declare a vector and initialize all elements to 0:

```
vector<int> scores(30, 0);
```

- Declare a vector initialized to size and contents of another vector:

```
vector<int> finals(scores);
```

Accessing vector Elements

- Vectors elements are indexed starting with 0
  - [ ]'s are used to read or change the value of an item:

```
v[i] = 42
cout << v[i];
```

  - [ ]'s cannot be used to initialize a vector element

Adding Elements to a Vector

- You can initialize a vector with a list of values:

```
vector<int> numbers { 10, 20, 30, 40 };
```

- Use `push_back` member function to add element to a full array or to an array that had no defined size:

```
scores.push_back(75);
```

- Use `size` member function to determine size of a vector:

```
howbig = scores.size();
```

Initializing vector Elements

- **`push_back`** adds an element in the next available position
  - Example:

```
vector<double> sample;
    sample.push_back(0.0);
    sample.push_back(1.1);
    sample.push_back(2.2);
```

Removing Vector Elements

- Use `pop_back` member function to remove last element from vector:

```
sample.pop_back();
```

- To remove all contents of vector, use `clear` member function:

```
sample.clear();
```

- To determine if vector is empty, use `empty` member function:

```
while (!sample.empty()) ...
```

The **`size`** of a `vector`

- The member function size() returns the number of elements in a vector
  - Example: To print each element of a vector given the previous vector initialization:

```
for (int i= 0; i < sample.size( ); i++)
        cout << sample[i] << endl;
```

The Type `unsigned int`

- The vector class member function size returns an **`unsigned int`**
  - Unsigned `int`'s are nonnegative integers
  - Some compilers will give a warning if the previous `for`-loop is not changed to:

```
for (unsigned int i= 0; i < sample.size( ); i++)
      cout << sample[i] << endl;
```

Alternate `vector` Initialization

- A vector constructor exists that takes an integer argument and initializes that number of elements
  - Example: vector<int> v(10);

    initializes the first 10 elements to 0
    v.size( ) would return 10
    - [ ]'s can now be used to assign elements 0 through 9
    - push_back is used to assign elements greater than 9

Vector Initialization With Classes

- The vector constructor with an integer argument
  - Initializes elements of number types to zero
  - Initializes elements of class types using the default constructor for the class

The `vector` Library

- To use the vector class
  - Include the vector library

```
#include <vector>
```

  - Vector names are placed in the standard namespace so the usual using directive is needed:

```
using namespace std;
```

`vector` Issues

- Attempting to use [ ] to set a value beyond the size of a vector may not generate an error
  - The program will probably misbehave

- The assignment operator with vectors does an element by element copy of the right hand vector
  - For class types, the assignment operator must make independent copies

Accessing Elements with the `at()` Member Function

- You can use the `at()` member function to retrieve a `vector` element by its index with bounds checking:

```
vector<string> names = {"Joe", "Karen", "Lisa"};
cout << names.at(0) << endl;
cout << names.at(1) << endl;
cout << names.at(2) << endl;
cout << names.at(3) << endl; // Throws an exception
```

Throws an `out_of_bounds` exception when given an invalid index

`vector` Efficiency

- A vector's capacity is the number of elements allocated in memory
  - Accessible using the capacity( ) member function
- Size is the number of elements initialized
- When a vector runs out of space, the capacity is automatically increased
  - A common scheme is to double the size of a vector
    - More efficient than allocating smaller chunks of memory

Controlling `vector` Capacity

- When efficiency is an issue
  - Member function `reserve` can increase the capacity of a vector
    - Example:

```
v.reserve(32); // at least 32 elements
v.reserve(v.size( ) + 10);  // at least 10 more
```

  - `resize` can be used to shrink a vector
    - Example: v.resize(24);
      //elements beyond 24 are lost

Useful Member Functions

Member Function	Description	Example
`at(i)`	Returns the value of the element at position `i` in the vector	`cout << vec1.at(i);`
`capacity()`	Returns the maximum number of elements a vector can store without allocating more memory	`maxElements = vec1.capacity();`
`reverse()`	Reverse the order of the elements in a vector	`vec1.reverse();`
`resize (n, val)`	Resizes the vector so it contains *n* elements. If new elements are added, they are initialized to *val*.	`vec1.resize(5, 0);`
`swap(vec2)`	Exchange the contents of two vectors	`vec1.swap(vec2);`