Vectors

Lecture 3

Motivation for Vectors

- So far, when we wanted to store many values of the same type, we used an array
- However, we have seen that with arrays, we need to know the size ahead of time, and can't adjust later
- The C++ **`vector`** class can be thought of as an array that can grow and shrink while your program is running
  - Similar to the comparison between a C style string and the **`string`** class

Declaring a Vector

```
vector<type> name;
```

- Like an array, when you declare a vector, you provide a data type for all elements
- This **`type`** parameter can be any valid data type (e.g. **`int`**, **`double`**) as well as any class that has a default constructor
  - We will revisit later how *you* can write classes that can work with any type, called a **template class**.
- The vector class is defined within the library vector, and is part of the std namespace, so don't forget…

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

Vector Size

- The size of a vector is how many elements have been initialized
  - Access this via the **`size`** function
- The default constructor creates an empty vector (size = 0)
- You can create a vector with an initial size by using a different constructor

```
vector<type> name( int initial_size );
```

Example

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

int main()
{

    vector<int> v1;
    cout << v1.size() << endl; // 0

    vector<double> v2( 5 );
    cout << v2.size() << endl; // 5

    return 0;
}
```

Changing Vector Size

- The **`push_back`** function adds an element to the end of the vector, increasing its size by 1

```
vector<int> v;
v.push_back( 7 );
cout << v.size(); // 1
```

- The **`resize(int new_size)`** function will change the size of the vector
  - If the new size is bigger, new elements are added
  - If the new size is smaller, then all but the first **`new_size`** elements are lost

Example

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

int main()
{

	vector<int> v;
	cout << v.size() << endl; // 0

	v.push_back( 7 );
	cout << v.size() << endl; // 1

	v.resize( 10 );
	cout << v.size() << endl; // 10

	v.push_back( 11 );
	cout << v.size() << endl; // 11

	v.resize( 2 );
	cout << v.size() << endl; // 2

	return 0;
}
```

New Vector Elements

- When using the constructor with an integer argument, or the **`resize`** function to make the vector bigger, new elements are initialized automatically

- If the vector type is a base type (e.g. **`int`**, **`double`**), the initial value is 0

- If the vector type is a class, then the default constructor is utilized for initialization

Accessing Vector Elements (1)

- Like an array, you can use the `[]` operator to access any *initialized* element of a vector

```
vector<int> v(10);
cout << v[0]; // 0
cout << v[10]; // badness
```

- Like the string class, the `[]` operator does not perform *bounds checking*
  - You may or may not get an immediate error when accessing beyond the size of a vector, but your program will likely not perform as expected

Accessing Vector Elements (2)

- As with the string class, you can use the **`at`** function to access vector elements safely (i.e. if a bad index is attempted, you are guaranteed an error will result)

```
vector<int> v(10);
cout << v.at(0); // 0
cout << v.at(10); // error
```

Exercise

Write a program that asks the user for a list of positive integers. When the user ends the list (by entering a value ≤ 0), output the sequence of numbers that they entered in reverse, each number squared.

```
> Enter numbers: 1 2 3 5 10 -1
100 25 9 4 1
```

Answer

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

int main()
{
	vector<int> v;
	int next;

	cout << "Enter numbers: ";
	do
	{
		cin >> next;
		if ( next > 0 )
			v.push_back( next );
	} while ( next > 0 );

	for ( int i=( v.size()-1 ); i>=0; i-- )
		cout << v[i]*v[i] << " ";

	cout << endl;

	return 0;
}
```

Exercise

Write a program that asks the user for a sequence of words. When the user ends the list (by typing CTRL-D), output the sequence in reverse.

```
Enter words (end with CTRL-D): c++ is cool
cool is c++
```

Answer

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

int main()
{
	vector<string> v;
	string next;

	cout << "Enter words (end with CTRL-D): ";
	while ( cin >> next )
	{
		v.push_back( next );
	}

	for ( int i=( v.size()-1 ); i>=0; i-- )
	{
		cout << v[i] << " ";
	}
	cout << endl;

	return 0;
}
```

Vector Internals

- A vector keeps track of two pieces of related information: size and capacity
- The size is the number of *initialized* elements; these are the elements that can be accessed
- The capacity is the *amount of memory allocated* for elements; this is always ≥ size
  - Get the capacity via the **`capacity`** function, though usually you don't directly need this information
- When the vector needs to grow, it doubles in size (you can't rely upon this behavior, as it depends upon implementation)
  - A lot of time is spent allocating memory and copying old data
  - If you know you are going to add a lot of elements, you can speed up your program via the **`reserve`** function

Example

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

int main()
{

  vector<int> v;
  cout << v.capacity() << endl; // 0

  v.reserve( 1000 );
  cout << v.capacity() << endl; // 1000

  for ( int i=0; i<1000; i++ )
  {
    v.push_back( i );
    cout << v.capacity() << endl; // 1000
  }

  return 0;
}
```

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

int main()
{

  vector<int> v;
  cout << v.capacity() << endl; // 0

  // v.reserve( 1000 );
  cout << v.capacity() << endl; // 0

  for ( int i=0; i<1000; i++ )
  {
    v.push_back( i );
    cout << v.capacity() << endl; // 1, 2, 4, 8
                                  // 16, 32, …
                                  // 1024
  }

  return 0;
}
```

Wrap Up

- A vector class can be thought of as an array that grows/shrinks as necessary
- You can only access those elements that have been initialized

  index < **`vector.size()`**
- You typically add elements to a vector one at a time via **`push_back`**, and access them via the `[]` operator or the **`at`** function.