Arrays

Arrays, C Style

- Array is an aggregate type
  - Multiple elements, all of the same type
- Number of elements defined when array is defined
  - Elements accessed using zero-based int value

```
int simpleArray[12];
simpleArray[0] = firstValue;
simpleArray[11] = lastValue;
```

Defines an array with 12 elements.

Indexes range from 0 to (number of elements)-1.

Example

- Read 10 int values and compute their average

```
int anValues[10];
for ( size_t i = 0; i < 10; i++ )
{
        cout << "Enter value " << i << ": ";
        cin >> anValues[i];
}
int sum = 0;
for ( size_t i = 0; i< 10; i++ )
        sum += anValues[i];
cout << "Sum is " << sum << ", average is " << sum / 10 << endl;
```

Defines an array of 10 integers with indexes from 0 to 9

The array is indexed by a counter in a for loop. Notice the for condition uses < rather than <=

Bounds Checking

- Size of an array is not stored as part of the type
  - Impossible to bounds check array accesses at run time
  - Common source of (often very dangerous) bugs

```
int simpleArray[12];
simpleArray[0] = firstValue;
simpleArray[111] = lastValue;
```

Programmer accidentally types 111 instead of 11. This will not directly cause any warnings or error messages at compile time or run time, and will cause memory beyond the end of the array to be overwritten with potentially disastrous consequences.

Sizing an Array

- Size of an array should be constant at the point of definition

```
int anLiteral[12];
```

A literal constant like this is fine, but rarely used as it is not obvious where 12 came from or when it may need to change.

```
const size_t size = 14;
int anConst[size];
```

A 'variable' may be used provided it is const. The name of the variable should be meaningful.

```
#define OLD_STYLE_CONSTANT 3
double adOldStyle[OLD_STYLE_CONSTANT];
```

C did not allow const to be used in this way, so the #define preprocessor directive could be used.

```
size_t num = 4;
float afFails[num];
```

The array size must be a constant, not a simple variable.

Multiple Dimensions

- C++ supports multi-dimensional arrays

```
int an2d[3][4];
an2d[0][0] = firstValue;
an2d[2][3] = lastValue;
```

This works as a 2 dimensional array, but C++ thinks of it as a 3 element array where each element is a 4 element array of ints.

Arrays defined in this way can have any number of dimensions.

- Any number of dimensions may be specified