CS 3460

Introduction to C++ arrays

Raw Arrays

- `type name[size];`
  - Declares and creates storage for the array.
  - Storage is on the stack, not the heap.
  - Similar statement in Java only creates an array reference, not even the array elements.

Raw Arrays

- `type name[size];`
  - Declares and creates storage for the array.
  - Storage is on the stack, not the heap.
  - Similar statement in Java only creates an array reference, not even the array elements.

```
int primes[4];

primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
```

Raw Arrays

```
int primes[] = { 2, 3, 5, 7 };
```

- Declares, creates storage for 4 elements, and initializes the elements to the values.
-

```
Person employees[10];
```

- Declares, creates storage for 10 elements.
- Storage is on the stack, not the heap.
- Similar statement in Java only creates an array reference, not even the array of references.

Raw Arrays

- Multi-dimensional arrays are declared as…

```
int table[3][3];

table[0][0] = 0;
table[0][1] = 1;
table[0][2] = 2;
table[1][0] = 3;

int table[3][3] = { {0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8} };
```

Raw Arrays

- Multi-dimensional arrays are declared as…

```
int table[3][3];

table[0][0] = 0;
table[0][1] = 1;
table[0][2] = 2;
table[1][0] = 3;

int table[3][3] = { {0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8} };
```

- With all this said, I don't recommend the use of raw arrays in C++! We'll talk about a couple more array types…
  - `std::array`
  - `std::vector`

std::array

- Part of the standard library `<array>`
- A fixed size container
- Knows its size: `.size()`
- Templated type
  - C++ allows primitives in templates, unlike Java Generics

```
std::array<int, 4> primes;

primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
```

```
std::array<int, 4> primes { 2, 3, 5, 7 };
```

std::array

- Single dimensioned but can do multi-dimensional like...

```
std::array<std::array<int, 3>, 3> table3
    { {
        { 0, 1, 2 },
        { 3, 4, 5 },
        { 6, 7, 8 }
    } };
```

`std::array` – Code Demo

std::vector

- Part of the standard library `<vector>`
- A dynamically sized container
  - (discuss how it grows)
- Knows its size: `.size()`
- Templated type

```
std::vector<int> primes;

primes.push_back(2);
primes.push_back(3);
primes.push_back(5);
primes.push_back(7);
```

```
std::vector<int> primes(4);

primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
```

```
std::vector<int> primes { 2, 3, 5, 7 };
```

`std::vector` – Code Demo

Automatic Type Inference – Inferring Arrays

```
auto primes = { 2, 3, 5, 7 };
```

- You might expect `primes` is inferred as an array
- Instead it is inferred as an `std::initializer_list`
- Because `{ 2, 3, 5, 7 }` is an `std::initializer_list`

```
auto primes = { 2, 3, 5, 7 };
std::vector primes1 = primes;
```

- To initialize, but not infer an `std::vector` or `std::array`

```
std::vector primes{ 2, 3, 5, 7 };
std::array primes{ 2, 3, 5, 7 };
```