Section Overview

Arrays and Vectors

- Arrays
  - What they are
  - Why we use arrays
  - Declaration and initialization
  - Accessing array elements
- Multi-dimensional arrays
- Vectors
  - What they are
  - Advantages vs. arrays
  - Declaration and initialization

Arrays

What is an array?

- Compound data type or data structure
  - Collection of elements
- All elements are of the same type
- Each element can be accessed directly

Arrays

Why do we need arrays?

```
int test_score_1 {0};
int test_score_2 {0};
int test_score_3 {0};
```

Arrays

Why do we need arrays?

```
int test_score_1 {0};
int test_score_2 {0};
int test_score_3 {0};
int test_score_4 {0};
int test_score_5 {0};
. . .
int test_score_100 {0};
```

Arrays

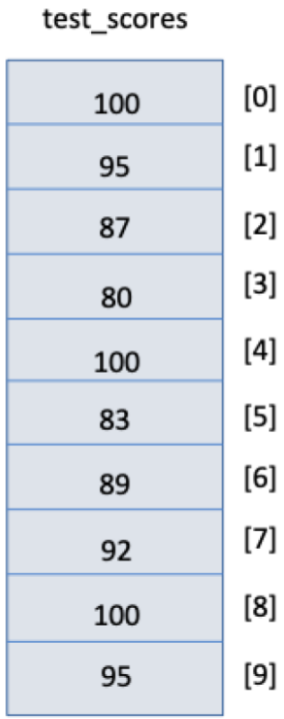
Characteristics

- Fixed size
- Elements are all the same type
- Stored contiguously in memory
- Individual elements can be accessed by
- their position or index

- First element is at index 0
- Last element is at index size-1

- No checking to see if you are out of bounds

- Always initialize arrays
- Very efficient
- Iteration (looping) is often used to process



Arrays

Declaring

```
Element_Type array_name [constant number of elements];
```

```
int test_scores [5];


int high_score_per_level [10];

const int days_in_year {365};
double hi_temperatures [days_in_year];
```

Arrays

Initialization

```
Element_Type array_name [number of elements] {init list}
```

```
int test_scores [5] {100,95,99,87,88};


int high_score_per_level [10] {3,5};        // init to 3,5 and remaining to 0

const double days_in_year {365};
double hi_temperatures [days_in_year] {0}; // init all to zero


int another_array [] {1,2,3,4,5};           // size automatically calculated
```

Arrays

Accessing array elements

```
array_name [element_index]

test_scores [1]
```

```
int test_scores [5] {100,95,99,87,88};

cout << "First score at index 0: "  << test_scores[0] << endl;
cout << "Second score at index 1: " << test_scores[1] << endl;
cout << "Third score at index 2:  " << test_scores[2] << endl;
cout << "Fourth score at index 3: " << test_scores[3] << endl;
cout << "Fifth score at index 4: "  << test_scores[4] << endl;
```

Arrays

Changing the contents of array elements

```
array_name [element_index]
```

```
int test_scores [5] {100,95,99,87,88};

cin >> test_scores[0];
cin >> test_scores[1];
cin >> test_scores[2];
cin >> test_scores[3];
cin >> test_scores[4];

test_scores[0] = 90;              // assignment statement
```

Arrays

How does it work?

- The name of the array represent the location of the first element in the array (index 0)
- The [index] represents the offset from the beginning of the array
- C++ simply performs a calculation to find the correct element
- Remember – no bounds checking!

Arrays

Declaring multi-dimensional arrays

```
Element_Type array_name [dim1_size][dim2_size]


int movie_rating [3][4];
```

Arrays

Multi-dimensional arrays

```
const int rows {3};
const int cols {4};
int movie_rating [rows][cols];
```

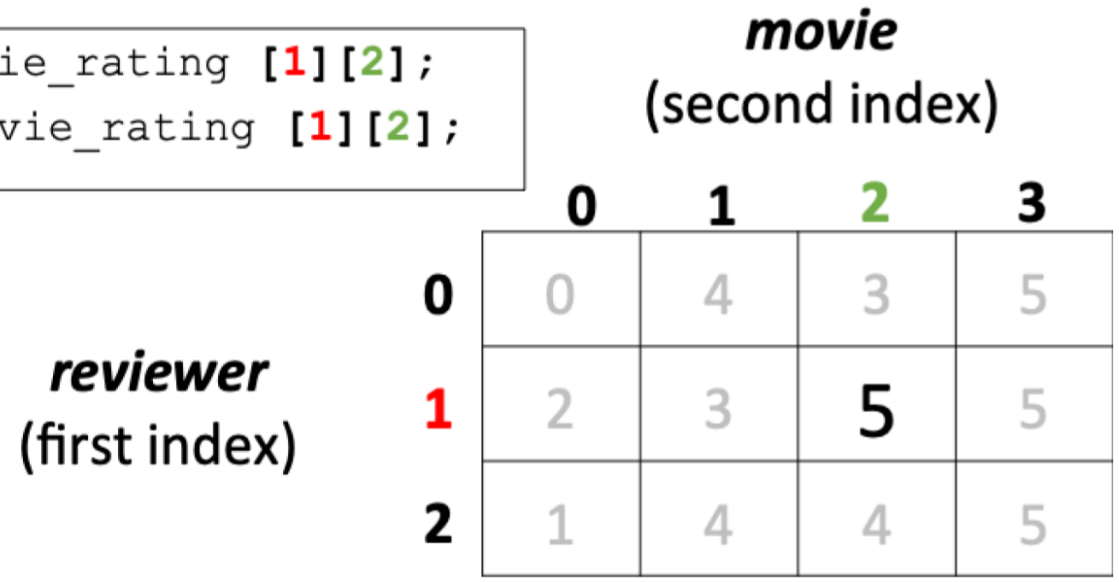
movie
(second index)

reviewer (first index)	0	1	2	3
0	0	4	3	5
1	2	3	3	5
2	1	4	4	5

Arrays

Accessing array elements in multi-dimensional arrays

```
cin >> movie_rating [1][2];
cout << movie_rating [1][2];
```



Arrays

Initializing multi-dimensional arrays

```
int movie_rating [3][4]
{
      { 0, 4 ,3, 5},
      { 2, 3, 3, 5},
      { 1, 4, 4, 5}
};
```

	0	1	2	3
0	0	4	3	5
1	2	3	3	5
2	1	4	4	5

Vectors

- Suppose we want to store test scores for my school

- I have no way of knowing how many students will register next year
- Options:
  - Pick a size that you are not likely to exceed and use static arrays
  - Use a dynamic array such as vector

Vectors

What is a vector?

- Container in the C++ Standard Template Library
- An array that can grow and shrink in size at execution time
- Provides similar semantics and syntax as arrays
- Very efficient
- Can provide bounds checking
- Can use lots of cool functions like sort, reverse, find, and more.

Vectors

Declaring

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

vector <char> vowels;

vector <int> test_scores;
```

Vectors

Declaring

```
vector <char> vowels (5);

vector <int> test_scores (10);
```

Vectors

Initializing

```
vector <char> vowels {'a' ,'e', 'i', 'o', 'u' };

vector <int> test_scores {100, 98, 89, 85, 93};

vector <double> hi_temperatures (365, 80.0);
```

Vectors

Characteristics

- Dynamic size
- Elements are all the same type
- Stored contiguously in memory
- Individual elements can be accessed by
- their position or index

- First element is at index 0
- Last element is at index size-1

- [ ] - no checking to see if you are out of bounds
- Provides many useful function that do bounds check

- Elements initialized to zero
- Very efficient
- Iteration (looping) is often used to process

Vectors

Accessing vector elements – array syntax

```
vector_name [element_index]

test_scores [1]
```

```
vector <int> test_scores {100,95,99,87,88};

cout << "First score at index 0: "  << test_scores[0] << endl;
cout << "Second score at index 1: " << test_scores[1] << endl;
cout << "Third score at index 2:  " << test_scores[2] << endl;
cout << "Fourth score at index 3: " << test_scores[3] << endl;
cout << "Fifth score at index 4: "  << test_scores[4] << endl;
```

Vectors

Accessing vector elements – vector syntax

```
vector_name.at(element_index)

test_scores.at(1)
```

```
vector <int> test_scores {100,95,99,87,88};

cout << "First score at index 0: "  << test_scores.at(0) << endl;
cout << "Second score at index 1: " << test_scores.at(1) << endl;
cout << "Third score at index 2:  " << test_scores.at(2) << endl;
cout << "Fourth score at index 3: " << test_scores.at(3) << endl;
cout << "Fifth score at index 4: "  << test_scores.at(4) << endl;
```

Vectors

Changing the contents of vector elements – vector syntax

```
vector_name.at(element_index)
```

```
vector <int> test_scores {100,95,99,87,88};

cin >> test_scores.at(0);
cin >> test_scores.at(1);
cin >> test_scores.at(2);
cin >> test_scores.at(3);
cin >> test_scores.at(4);

test_scores.at(0) = 90;          // assignment statement
```

Vectors

So, when do they grow as needed?

```
vector_name.push_back(element)
```

```
vector <int> test_scores {100,95,99};    // size is 3

test_scores.push_back(80);               // 100, 95, 99, 80
test_scores.push_back(90);               // 100, 95, 99, 80, 90
```

Vector will automatically allocate the required space!

Vectors

What if you are out of bounds?

- Arrays never do bounds checking
- Many vector methods provide bounds checking
- An exception and error message is generated

```
vector <int> test_scores { 100,95 };

cin >> test_scores.at(5);

terminate called after throwing an instance of 'std::out_of_range'
what():  vector::_M_range_check: __n (which is 5) >= this->size() (which is
2)
This application has requested the Runtime to terminate it in an unusual
way.
Please contact the application's support team for more information.
```