



UNIVERSITY OF
RICHMOND

CMSC 240 Lecture 4

CMSC 240 Software Systems Development
Fall 2023

Today

- Collections
- Arrays
- Vectors
- File Input/Output
- In-class coding exercise



An aerial photograph of a university campus. In the center, a tall, ornate brick clock tower with Gothic-style architecture stands out. The tower has multiple levels of arched windows and a pointed top. Surrounding the tower are various other brick buildings and a large, well-maintained lawn. Numerous trees, some with vibrant yellow and green foliage, are scattered throughout the campus. Several people can be seen walking along paved paths that wind through the greenery. The sky is a clear, bright blue.

Collections

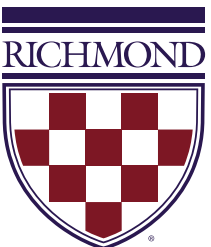


Collections

- Almost all interesting programs process **data**
- Data comes from many sources



- **Collections** are objects that store data, a.k.a. data structures
 - The stored data objects are called **elements**
 - Some collections maintain ordering of elements
 - Some allow for duplicate elements
 - Typical operations: `add`, `remove`, `clear`, `find`, `size`

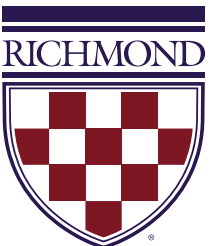


Standard Template Library (STL)

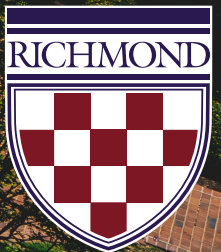
- The C++ Standard Template Library ([STL](#)) contains a powerful library of collections for you to use in your programs
- We will learn about collections from the STL library

Containers library

```
array (C++11)
vector – deque
list – forward_list (C++11)
set – multiset
map – multimap
unordered_map (C++11)
unordered_multimap (C++11)
unordered_set (C++11)
unordered_multiset (C++11)
stack – queue – priority_queue
```



Arrays



C-Style Arrays

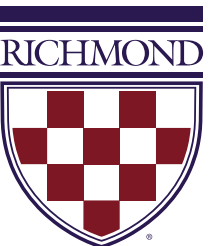
```
int main()
{
    // Create a c-style integer array of size 10.
    → int numbers[10];

    // Insert elements into the array.
    → for (int i = 0; i < 10; i++)
    {
        → numbers[i] = i + 1;
    }

    // Create another c-style integer array of size 10.
    → int otherNumbers[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    return 0;
}
```

C-style arrays
do not have a
length method



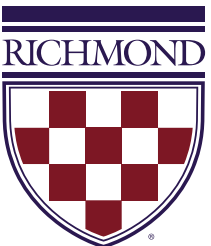
C-Style Arrays

- A C-style array is
 - fixed-sized collection
 - data elements of the same type
 - stored in contiguous memory locations

```
double numbers[4] = {3.1, 2.7, 4.2, 9.9};
```

Index	0	1	2	3
Element	3.1	2.7	4.2	9.9

- Problems with C-style arrays
 - Doesn't know its own size
 - No methods available
 - Converts to a pointer to its first element



C++ Arrays

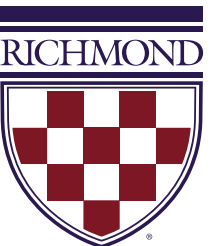
```
→ #include <array>
   using namespace std;

   int main()
   {
       // A new integer array object called numbers of size 10.
       → array<int, 10> numbers;

       → for (int i = 0; i < numbers.size(); i++)
       → {
           numbers[i] = i + 1;
       }

       → array<int, 10> moreNumbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

       return 0;
   }
```



C++ Arrays

Array Member Functions	Description
<code>a.at(index)</code>	Return the element at the given index
<code>a.front()</code>	Return the first element
<code>a.back()</code>	Return the last element
<code>a.empty()</code>	Checks whether the array is empty
<code>a.size()</code>	Returns the number of elements
<code>a.fill(value)</code>	Fill the array with specified value
<code>a.swap(array)</code>	Exchanges the contents of the array with those of the given array



C++ Array Limitations

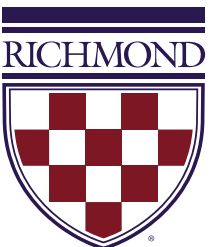
- They have a **fixed size** and cannot be easily resized after creation
- If you index **out of bounds** of the array, it lets you do it, and you access random garbage memory (yuck!)

```
array<int, 10> numbers;
```

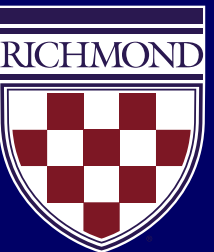
```
cout << numbers[323] << endl; // Clearly out of array bounds!!
```

795897921

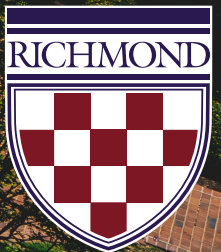
- An array **does not support** many operations that you may want
 - Inserting/deleting elements into the front/middle/back
 - Reversing the order, or sorting the elements
 - Searching the array for a given element



Ask me questions



Vectors



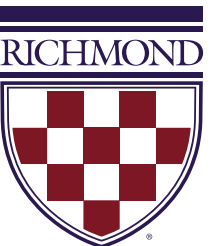
Vectors

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

int main()
{
    // Create a new vector of integers.
→ vector<int> numbers;

    cout << "Enter an integer, or Ctrl-D to quit: ";
    int num;
→ while (cin >> num)
    {
→         numbers.push_back(num);
        cout << "Enter an integer, or Ctrl-D to quit: ";
    }

    return 0;
}
```

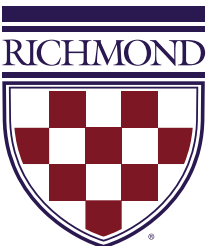


Vector Type Parameters

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

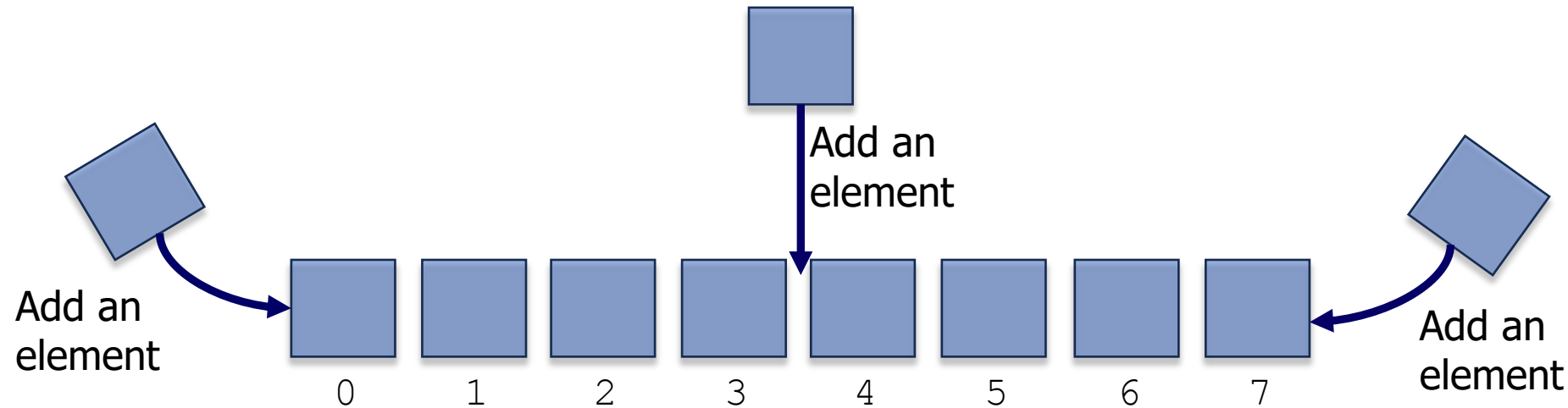
- When constructing a vector, you must specify the **type** of its elements in < >
 - This is called a **type parameter**
 - A vector is a **parameterized class**, aka. **template** classes
- The above constructs a vector object
 - You can use any type of elements, even primitive types like `int`

```
vector<string> mascots;  
mascots.push_back("WebstUR");  
mascots.push_back("Ram");  
mascots.push_back("Turtle");
```



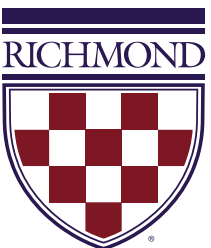
Vectors Are Dynamic

- A **vector** is a dynamic collection of elements with 0-based indexes
 - Elements can be added to the front, back, or elsewhere
 - A vector has a **size** (number of elements that have been added)



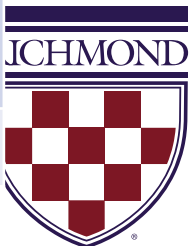
Vectors

- A vector is like an array that **resizes** to fit its contents
 - Similar to an `ArrayList` in Java
- When a vector is created, it is **initially empty**
`{ }`
- You can add items to the vector, by default it adds at the end
`{ 3.5, 6.8, 4.2, 0.9 }`
- Vector objects keep track of the element values that have been added to it, their order, indexes, and its total size
- You can **add**, **remove**, **get**, and **set** any index at any time

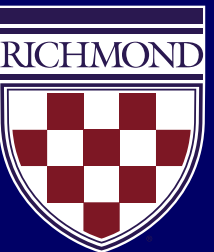


Vector Functionality

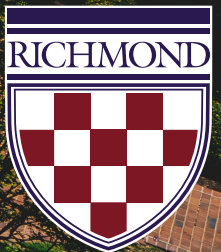
Vector Member Functions	Description
<code>v.at(index)</code>	Return the element at the given index
<code>v.front()</code>	Return the first element
<code>v.back()</code>	Return the last element
<code>v.empty()</code>	Checks whether the vector is empty
<code>v.size()</code>	Returns the number of elements
<code>v.clear()</code>	Clears the contents of the vector
<code>v.insert(position, value)</code>	Inserts value before position in this vector
<code>v.erase(position)</code>	Removes the element at position
<code>v.push_back(value)</code>	Appends the given element value to the end
<code>v.pop_back()</code>	Removes the last element
<code>v.swap(vector)</code>	Exchanges the contents of the vector with those of the given vector
<code>v.emplace(position)</code>	Inserts a new element directly before position



Ask me questions



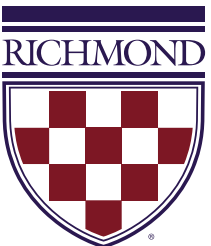
File Input/Output



Reading From Files

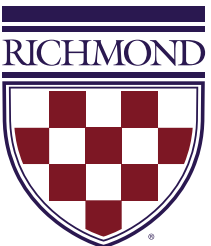
- `#include <fstream>`
 - Imports `ifstream`, `ofstream` classes for input/output files
 - Common pattern: open a file; read each line from it; close it

```
// Read and print every line of a file.
ifstream inputFileStream;
inputFileStream.open("filename.txt");
string line;
while(getline(inputFileStream, line))
{
    cout << line << endl;
}
inputFileStream.close(); // Close stream when done.
```



ifstream Member Functions

ifstream Member Functions	Description
<code>f.fail()</code>	Returns true if the last read call failed (e.g. EOF)
<code>f.open(filename)</code>	Opens the file represented by given string
<code>f.close()</code>	Stops reading the file
<code>f.get()</code>	Reads and returns 1 character
<code>getline(file, line)</code>	Reads line of input into a string; returns a true/false indicator of success
<code>f >> var</code>	Reads data from input file into variable (like cin)



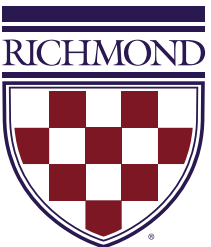
Reading From Files

1 one
2 two
3 three
4 four
5 five
6 six
7 seven

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

int main()
{
    ifstream inputFile;
    inputFile.open("numbers.txt"); // open the file
    int number;
    string numberText;
    for (int i = 1; i < 8; i++)
    {
        inputFile >> number >> numberText; // use file stream like cin

        cout << "number = " << number << " " << numberText << endl;
    }
    inputFile.close(); // Close the file.
    return 0;
}
```



Reading From Files

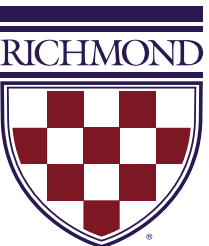
// Read and print every character of a file.

→ `ifstream inputFileStream;`
`inputFileStream.open(filename);`

→ `if (!inputFileStream)`
`{`
 `cerr << "Could not open file: " << filename << endl;`
 `exit(1);`
`}`

→ `char character;`
`while(inputFileStream >> character)`
`{`
 `cout << character << endl;`
`}`

→ `inputFileStream.close();` *// Close stream when done.*



Writing to Files

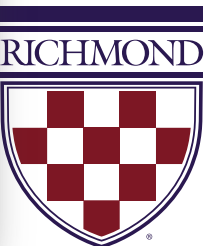
```
→ #include <fstream>
   using namespace std;

   int main()
   {
       ofstream outputFile;
       → outputFile.open("countdown.txt"); // Open the file for writing.

       for (int i = 10; i > 0; i--)
       {
           → outputFile << i << endl; // Write to the file like cout.
       }

       → outputFile.close(); // Close the file.

       return 0;
   }
```



In-Class Coding Exercise

