Vectors

Luke Kreczko
(based on slides by Mark Slater)

University of BRISTOL

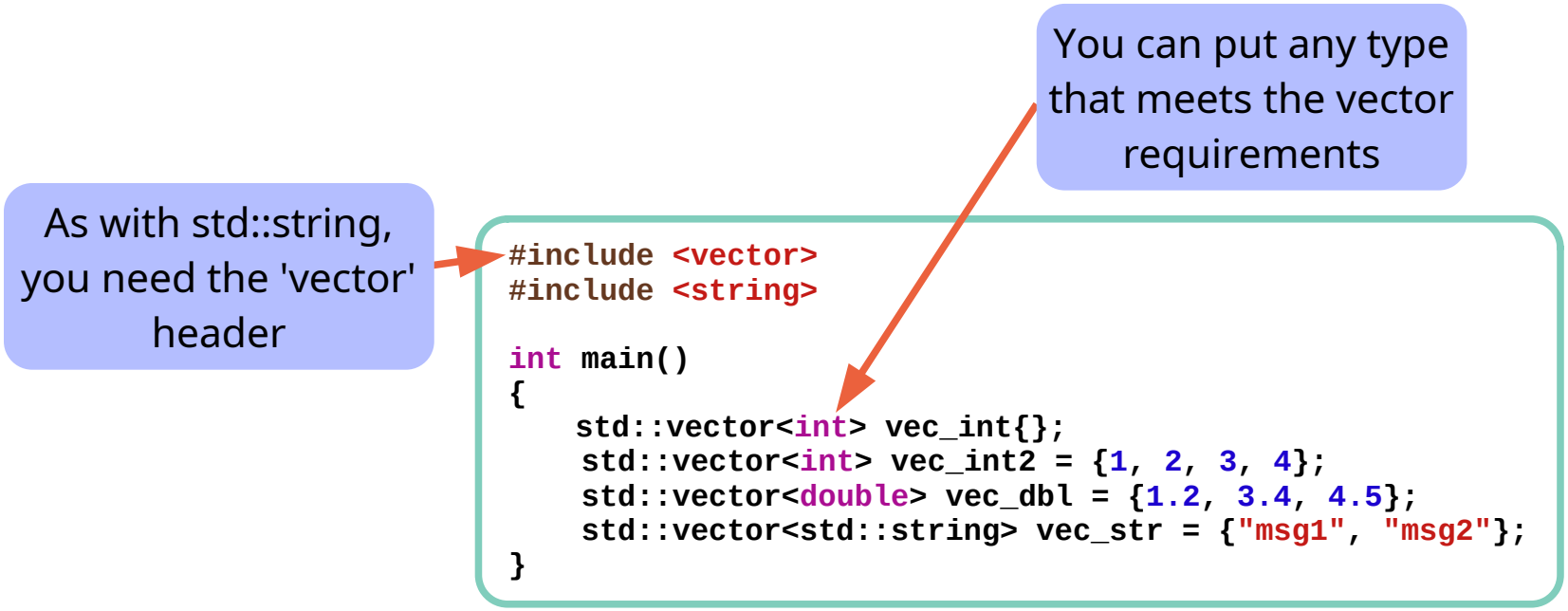
UNIVERSITY OF BIRMINGHAM

Overview

- For the next part of the cipher code we will need to start using collections of objects
- To deal with collections of objects **dynamically** we need to be able to:
  - Hold any type
  - Resize the collection based on runtime requirements
  - Ensure the memory is allocated and de-allocated correctly
  - Add and remove objects from the collection
  - Loop over the collection
  - Get basic information from it (e.g. size)
- There is another extended C++ type very similar to std::string that can do all of these things and more – **std::vector**

Declaring and Initialising

- In order for a **std::vector** to store any type you want, you need to specify the type at compile time
- You can create an empty vector or a vector with specified content
- There are more, less common ways to construct a vector → see docs



Useful Member Functions

- As std::vector is a more complex type than an integer or double type, it also has some functions associated with it that can be used to manipulate and get info from the object
- Some of the most useful are:
  - size() - return the number of elements in the vector
  - empty() - returns true or false depending on if the vector has zero elements
  - push_back(<object>) - Increase the size of the vector by one and add an object to the end
  - pop_back() - Remove the last object in the vectors
  - at(<index>) / [<index>] operator – Access element <index>
  - emplace_back(<args to construct object>) - a more efficient version of push_back that creates the object in place. More on that later!
- To call these functions, you use the '.' operator on the object
- We'll learn a lot more about this when we deal with classes!

std::vector Example 1: Manipulation

```
#include <vector>
#include <string>
#include <iostream>

int main()
{
    // Construct a vector
    std::vector<double> vec = {1.2, 3.4, 5.6};

    // print out the vector size (3)
    std::cout << vec.size() << std::endl;

    // add a few elements
    vec.push_back(7.8);
    vec.push_back(9.1);

    // vector size (5)
    std::cout << vec.size() << std::endl;

    // remove the last element (9.1)
    vec.pop_back();

     // vector size (4)
    std::cout << vec.size() << std::endl;

    // loop over the vector using an index counter
    for (size_t i{0}; i < vec.size(); i++)
    {
        std::cout << "Index: " << i << " " << vec[i] << std::endl;
    }
}
```

Use the '.' operator to call the member function 'on' the object

Access elements with a for loop and index. (Another way later!)