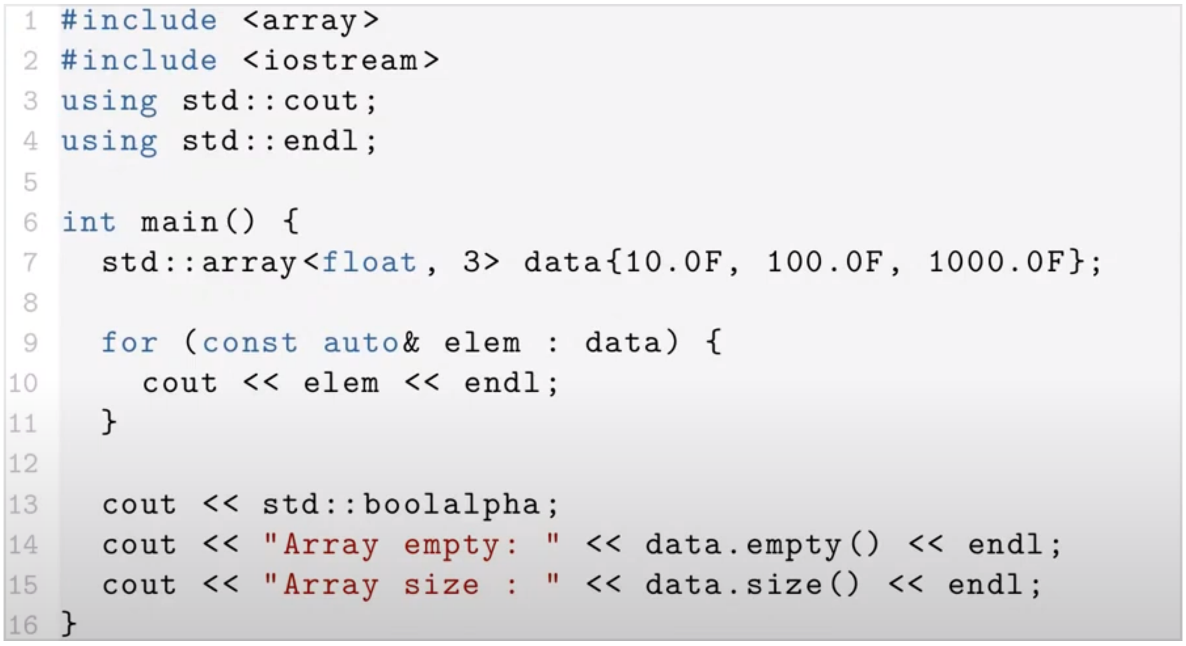
C++ STL Containers

STL: Standard Template Library

Static Containers

Containers whose size is known at compile time.

std::array

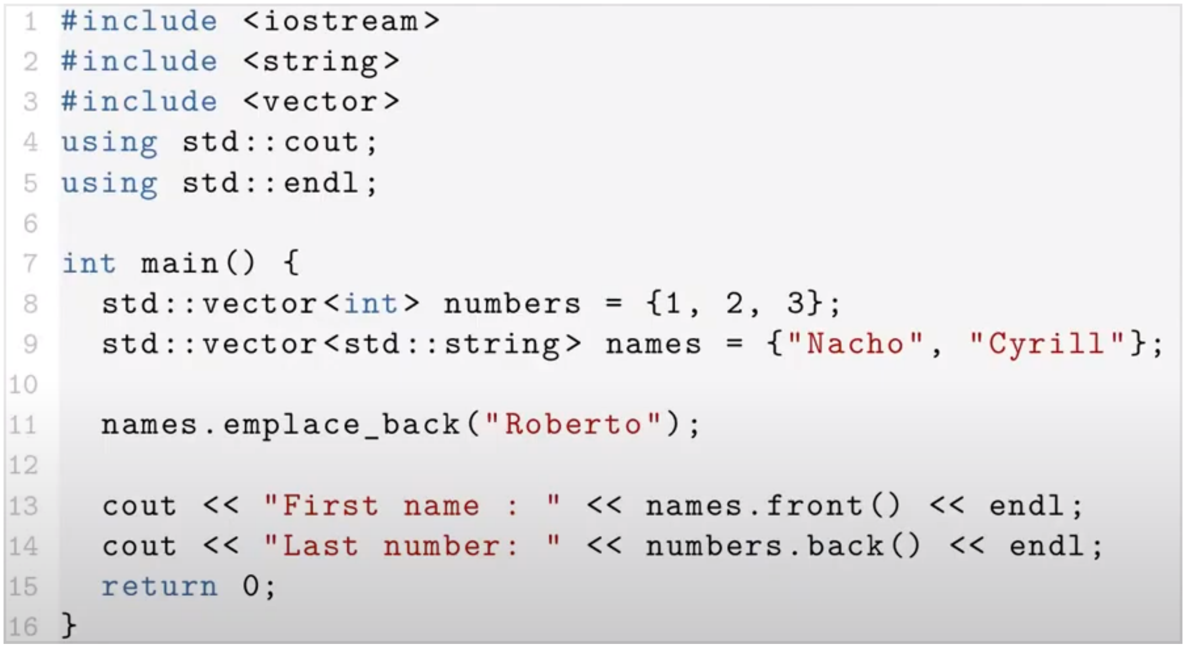


```cpp
#include <array>
#include <iostream>
using std::cout;
using std::endl;

int main() {
  std::array<float, 3> data{10.0F, 100.0F, 1000.0F};

  for (const auto& elem : data) {
    cout << elem << endl;
  }

  cout << std::boolalpha;
  cout << "Array empty: " << data.empty() << endl;
  cout << "Array size : " << data.size() << endl;
}
```

Dynamic Containers

Containers whose size is unknown at compile time.

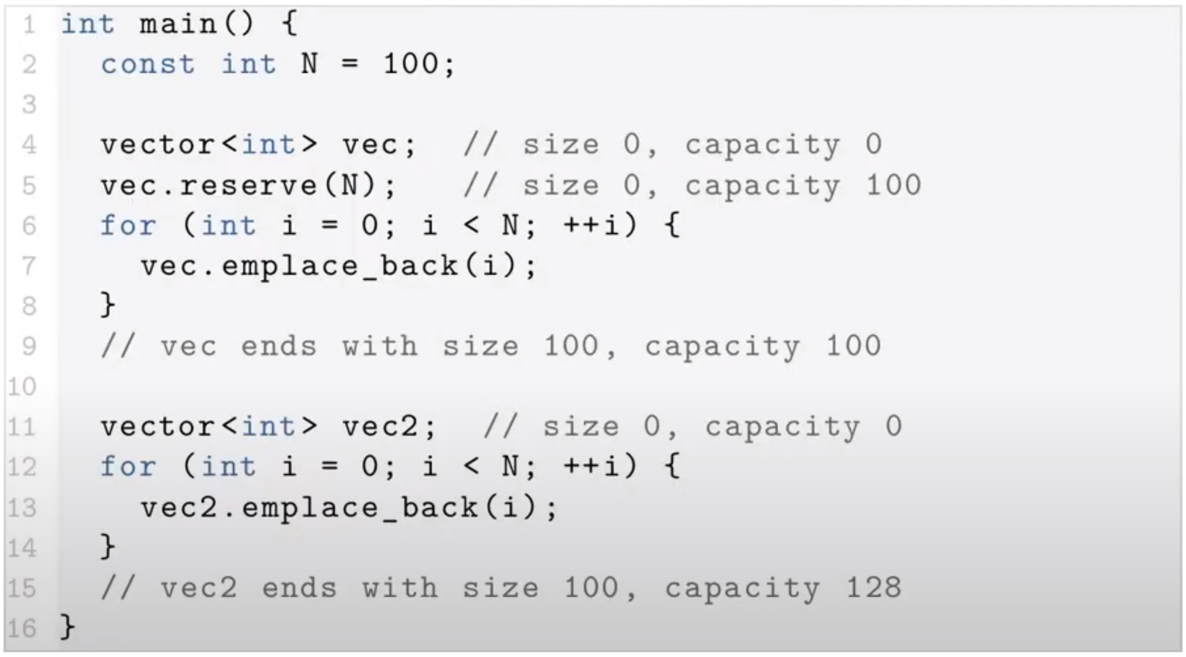
std::vector



```cpp
#include <iostream>
#include <string>
#include <vector>
using std::cout;
using std::endl;

int main() {
  std::vector<int> numbers = {1, 2, 3};
  std::vector<std::string> names = {"Nacho", "Cyrill"};

  names.emplace_back("Roberto");

  cout << "First name : " << names.front() << endl;
  cout << "Last number: " << numbers.back() << endl;
  return 0;
}
```

- vec.clear(): remove all elements
- vector size is unknown. Therefore, a capacity is defined.
- size != capacity
- Many push_back/emplace_back operations force vector to change its capacity many times.
- reserve(n) ensures that the vector has enough memory to store n items.
- This is very important optimization.

Optimizing vector resizing



```
int main() {
  const int N = 100;

  vector<int> vec;  // size 0, capacity 0
  vec.reserve(N);   // size 0, capacity 100
  for (int i = 0; i < N; ++i) {
    vec.emplace_back(i);
  }
  // vec ends with size 100, capacity 100

  vector<int> vec2;  // size 0, capacity 0
  for (int i = 0; i < N; ++i) {
    vec2.emplace_back(i);
  }
  // vec2 ends with size 100, capacity 128
}
```

Functions to work with containers:

- size(): number of elements in container
- empty(): check if container is empty
- front(): first element of container
- back(): last element of container
- clear(): clear the container completely