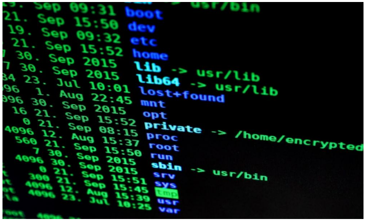
Specialized Numerical Methods for Transport Phenomena

Advanced C++ Programming - Additional Topics

Bruno Blais and Laura Prieto Saavedra

Department of Chemical Engineering
Polytechnique Montréal

January 20, 2025



Outline

Outline

Vectors, lists and maps

Functions in classes

STL containers

There are three categories:

- Sequence containers: maintain the ordering of the elements and you can choose where to insert your element by position, e.g., `std::array`, `std::vector`, `std::list`.
- Associative containers: automatically sort their inputs, e.g., `std::set`, `std::map`.
- Container adapters: adapted to specific uses, e.g., `std::stack`, `queue`.

std::vector

Similar to dynamic arrays but with the ability to resize itself automatically when an element is inserted or deleted:

```cpp
#include <iostream>
#include <vector>
void print_vector(const std::vector<int>& v)
{
    for (int n : v)
        std::cout << n << " ";
    std::cout << std::endl;
}
int main()
{
    std::vector<int> v = {8, 4, 5, 9};
    std::cout << "Size : " << v.size() << std::endl;
    std::cout << "Capacity : " << v.capacity() << std::endl;
    std::cout << "Max_Size : " << v.max_size() << std::endl;
    print_vector(v);
    return 0;
}
```

std::vector (Cont.)

What else can we do with these containers?

```
// Add two more integers
v.push_back(6);
v.push_back(2);
print_vector(v);

// Overwrite element at position 1
v[1] = -2;
print_vector(v);

// Resize vector (more space)
v.resize(7);
print_vector(v);
std::cout << "Size : " << v.size() << std::endl;
std::cout << "Capacity : " << v.capacity() << std::endl;
std::cout << "Max_Size : " << v.max_size() << std::endl;
```

Let's see the code!

std::list

Sequence containers that allow non-contiguous memory allocation:

```cpp
#include <iostream>
#include <list>
void print_list(const std::list<int>& l)
{
    for (auto i : l)
        std::cout << i << " ";
    std::cout << std::endl;
}
int main()
{
    std::list<int> l{12, 45, 8, 6};
    print_list(l);

    return 0;
}
```

`std::list` (Cont.)

What else can we do with these containers?

```
l.push_back(5);
l.push_front(50);
print_list(l);

l.pop_front();
l.pop_back();
print_list(l);

l.reverse();
print_list(l);

l.sort();
print_list(l);
```

Let's see the code!

std::map

Store elements in a specific order by a combination of two things:

- Key value: used to sort and uniquely identify the elements
- Map value: store content associated to this key

The types of the key and the value may differ.

```cpp
#include <iostream>
#include <map>
#include <string>
void print_map(const std::map<std::string, int>& m)
{
    for (const auto& [key, value] : m)
        std::cout << key << "," << value << std::endl;
}
int main()
{
    std::map<std::string,int> m {{"Laura", 15}, {"Bruno", 19}};
    print_map(m);
    return 0;
}
```

std::map (Cont.)

What else can we do with these containers?

```
// Change mapped value according to key
m["Laura"] = 17;
print_map(m);
// Add a new entry to the map
m["Olivier"] = 20;
print_map(m);
// Delete an entry in the map
m.erase("Bruno");
print_map(m);
// Clear map
m.clear();
print_map(m);
```

Let's see the code!

Outline

Creating a class

```
class Rectangle
{
public:
    Rectangle(int w, int h)// Constructor
    {
        width = w;
        height = h;
    }
    // Member functions
    int calculate_area()
    {
        return width*height;
    }
private:
    // Member variables
    int width;
    int height;
};
```

Using a class

Use the public function of the class in main:

```
int main()
{
    Rectangle rectangle_1(4,2);
    std::cout << "Area of rectangle = " << rectangle_1.calculate_area()
        << std::endl; // Area of rectangle = 8
}
```

What if I try to access the width or the height?

```
std::cout << "Width of rectangle = " << rectangle_1.width << std::endl;
rectangle_1.height = 2;
```

This will not compile! The member variables are `private`! This is why we have getter functions, e.g., `get_width()` or `get_height()` .