C++ Workshop 2
STL Containers

Panagiotis Petridis

Brought to you by HackSoc

std::array<class T, std::size_t N>

- Simple Array
- Fixed Size
- Size has to be constant expression
- Takes 2 parameters as templates
  - The stored type (int, float etc.)
  - The size of the array
- Access is O(1)
- Search is O(n)

```cpp
#include <iostream>
#include <array>

int main() {
    std::array<int, 5> arr = {1,2,3,4,5};
    for(int e : arr)
        std::cout << e << std::endl;
    return 0;
}
```

Link to reference

std::list<class T>

- Doubly Linked List
- Variable size
- Takes 1 template parameter*
    - The type to be stored
- You can add items to the front and back with O(1) complexity
- Search and Access is O(n)

```
#include <iostream>
#include <list>

int main() {
    std::list<int> l = {3};
    l.push_front(2);
    l.push_front(1);
    l.push_back(4);
    l.push_back(5);
    for(int e : l)
        std::cout << e << std::endl;
    return 0;
}
```

Note that singly linked list implementation also exists. See std::forward_list

Link to reference

std::vector<class T>

- Dynamic Array
- Variable size
- Size doesn’t have to be constant expression
- Takes 1 template parameter*
  - Type to be stored
- Insertion is O(n)*
- Search is O(n)
- Index is O(1)

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

int main() {
    int a = 0;
    a += 5;
    std::vector<int> v(a);
    for(int i = 0; i < 5; i++)
        v[i] = i+1;
    for(int e : v)
        std::cout << e << std::endl;
    return 0;
}
```

Link to reference

std::queue<class T>

- Basically a singly linked list
- Uses FIFO (First In First Out) approach
- Takes 1 parameter*
    - Type to be stored.
- O(1) insertion
- O(n) access
- O(1) to pop top element

```cpp
#include <iostream>
#include <queue>

int main() {
    std::queue<std::string> q;
    q.push("Hello");
    q.push(" ");
    q.push("World");
    q.push("!");
    while(!q.empty()) {
        std::cout << q.front();
        q.pop();
    }
    std::cout << std::endl;
    return 0;
}
```

Link to reference

std::stack<class T>

- Basically a singly linked list
- Uses LIFO (Last In First Out) approach
- Takes 1 parameter*
  - Type to be stored.
- O(1) insertion
- O(n) access
- O(1) to pop top element

```
#include <iostream>
#include <stack>

int main() {
    std::stack<std::string> s;
    s.push("!");
    s.push("World");
    s.push(" ");
    s.push("Hello");
    while(!q.empty()) {
        std::cout << s.top();
        q.pop();
    }
    std::cout << std::endl;
    return 0;
}
```

Link to reference

std::map<class T, class T>

- Ordered Hash Map
- Can store key-value pairs
- Takes 2 template parameters
  - Key type
  - Value type
- Items will have an ordering on them
  - you can also add a custom ordering method
- Uses Red-Black Tree for ordering elements
- Insertion O(lg(n))
- Access O(lg(n))
- Search O(lg(n))

Where lg(n) is $\log_2(n)$

```
#include <iostream>
#include <utility>
#include <map>

int main() {
    std::map<std::string, int> m;
    m["one"] = 1;
    m["two"] = 2;
    m["three"] = 3;
    m["four"] = 4;
    m["five"] = 5;
    for(std::pair<std::string, int> p : m)
        std::cout << p.first << ": " << p.second
                                    << std::endl;

    return 0;
}
```

Link to reference

std::unordered_map<class T, class T>

- Simple hash map
- Stores key-value pairs
- Takes 2 template parameters
  - Key type
  - Value type
- Access is O(1)
- Insertion is O(1)
- Search is O(1)
  - Checking if an element exists

```cpp
#include <iostream>
#include <utility>
#include <unordered_map>

int main() {
    std::unordered_map<std::string, int> m;
    m["one"] = 1;
    m["two"] = 2;
    m["three"] = 3;
    m["four"] = 4;
    m["five"] = 5;
    for(std::pair<std::string, int> p : m)
        std::cout << p.first << ": " << p.second
                                     << std::endl;
    return 0;
}
```

Link to reference

std::set<class T>

- Ordered Set
- Can store a set of values
- Takes 1 template parameter
  - Value type
- Items will have an ordering on them
  - you can also add a custom ordering method
- Uses Red-Black Tree for ordering elements
- Insertion O(lg(n))
- Access O(lg(n))
- Search O(lg(n))

```
#include <iostream>
#include <utility>
#include <set>

int main() {
    std::set<int> s;
    s.insert(1);
    s.insert(2);
    s.insert(3);
    s.insert(4);
    s.insert(5);
    for(int e : s)
        std::cout << e << std::endl;
    return 0;
}
```

Link to reference

std::unordered_set<class T>

- Simple hash set
- Stores a set of values
- Takes 1 template parameter
  - Value type
- Access is O(1)
- Insertion is O(1)
- Search is O(1)
  - Checking if an element exists

```cpp
#include <iostream>
#include <unordered_set>

int main() {
    std::unordered_set<int> s;
    s.insert(1);
    s.insert(2);
    s.insert(3);
    s.insert(4);
    s.insert(5);
    std::cout << std::boolalpha << "5 is in set: "
              << (s.find(5)!=s.end()) << std::endl;
    std::cout << std::boolalpha << "6 is in set: "
              << (s.find(6)!=s.end()) << std::endl;
    return 0;
}
```

Link to reference

Building your own containers

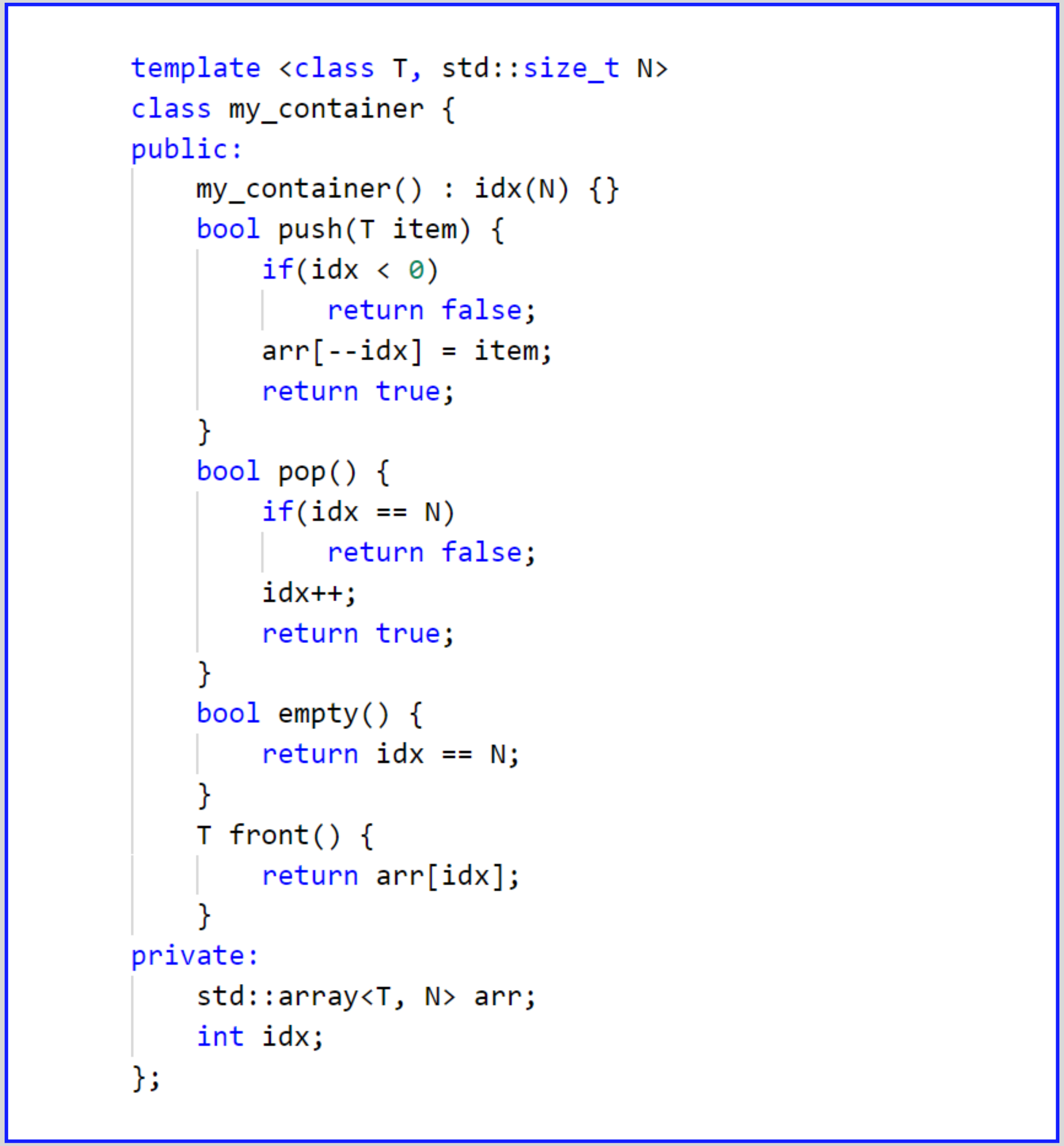
We will need

- A template type
- Private members to store data
- Get/set methods

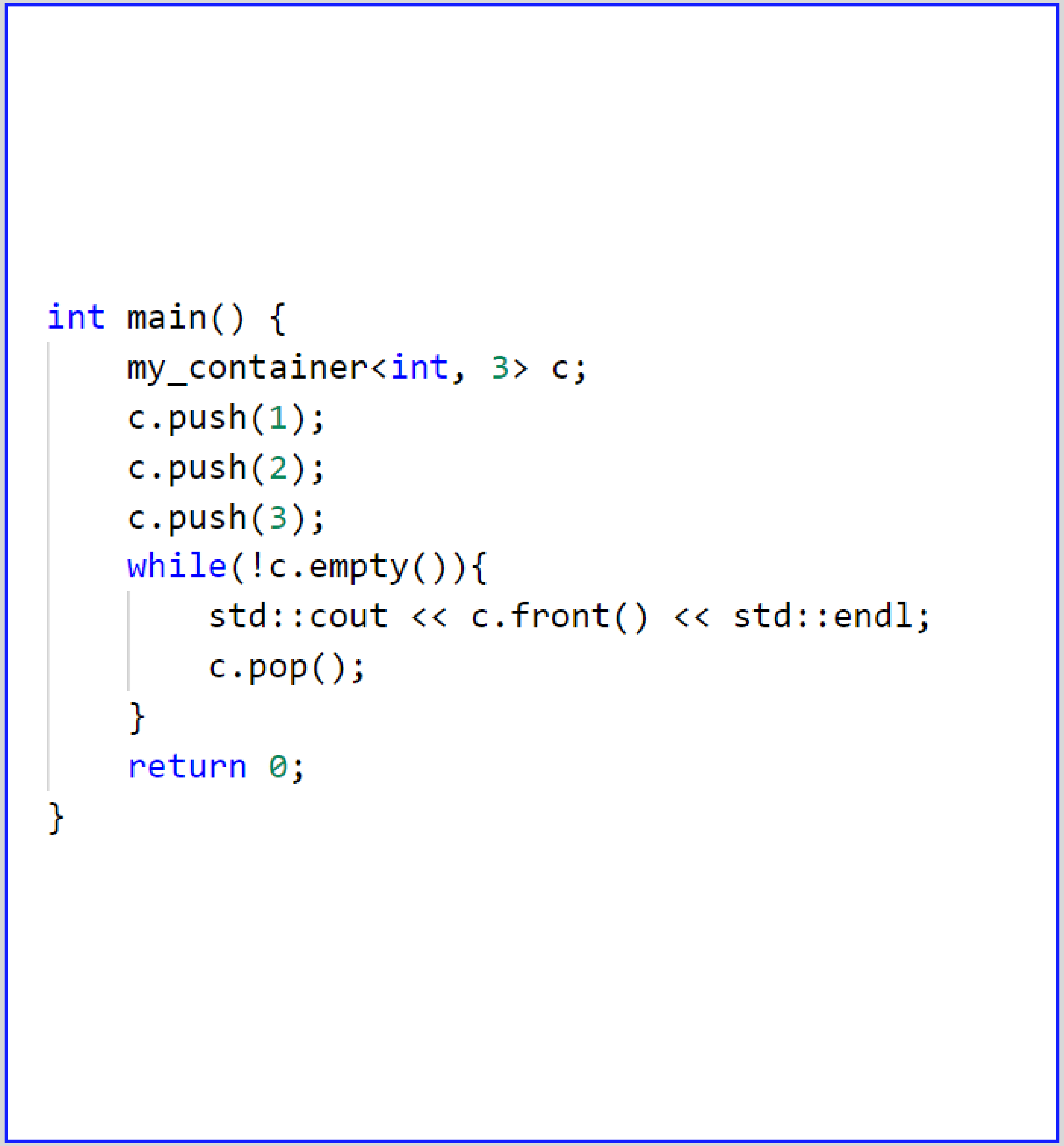
What we will write

A simple stack that allows for only a certain amount of elements.

Great talk by Marc Gregoire at CppCon on how to write STL compliant Data Structures and Algorithms



```cpp
template <class T, std::size_t N>
class my_container {
public:
    my_container() : idx(N) {}
    bool push(T item) {
        if(idx < 0)
            return false;
        arr[--idx] = item;
        return true;
    }
    bool pop() {
        if(idx == N)
            return false;
        idx++;
        return true;
    }
    bool empty() {
        return idx == N;
    }
    T front() {
        return arr[idx];
    }
private:
    std::array<T, N> arr;
    int idx;
};
```



```cpp
int main() {
    my_container<int, 3> c;
    c.push(1);
    c.push(2);
    c.push(3);
    while(!c.empty()){
        std::cout << c.front() << std::endl;
        c.pop();
    }
    return 0;
}
```

Practice!

https://www.hackerrank.com/

Resources

https://github.com/PanagiotisPtr/cpp_workshop