C++ Workshop 3 Algorithms

Panagiotis Petridis

Brought to you by HackSoc 

std::sort

- Sorts elements in container
- Uses a combination of quicksort and insertion sort.
- Takes 2 iterators to a container and it will sort it from first to last
- There's also std::stable_sort that uses mergesort.

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    sort(begin(v), end(v));
    for(int e : v)
        cout << e << endl;
}
```

Link to reference

std::find

- Searched a container for a specific element
- Returns iterator to the position the element was found
- Takes 3 parameters. 2 Iterators, first and last, and the value to search

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    vector<int>::iterator it =
                  find(begin(v),end(v), 4);
    vector<int>::iterator ik =
                  find(begin(v), end(v), 10);
    cout << boolalpha;
    cout << "4 is in the list: " << (it != end(v))
         << endl;
    cout << "10 is in the list: " << (ik != end(v))
    << endl;
}
```

Link to reference

std::remove

- Removes all instances of an element in the container
- Takes 2 iterators, first and last, and the value to remove
- Returns iterator to the end of the new list of values

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    remove(begin(v), end(v), 4);
    vector<int>::iterator it =
                    find(begin(v),end(v), 4);
    cout << boolalpha;
    cout << "4 is in the list: " << (it != end(v))
         << endl;
}
```

Link to reference

std::remove_if

- Takes predicate P (can be lambda expression) and returns iterator to the elements to be removed
- Need to use the erase function of the container for this one

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    v.erase(remove_if(begin(v), end(v), [&](const int& i){
        return i%2==0;
    }), end(v));
    vector<int>::iterator it = find(begin(v), end(v), 4);
    vector<int>::iterator ik = find(begin(v), end(v), 2);
    cout << boolalpha;
    cout << "4 is in the list: " << (it != end(v))
         << endl;
    cout << "2 is in the list: " << (ik != end(v))
         << endl;
}
```

Link to reference

std::for_each

- Applies function to all elements in container.
- Uses lambdas so function can even take parameters by reference and thus change the elements
- Takes 3 parameters. 2 Iterators, first and last, and the function to be applied (usually a lambda)

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    for_each(begin(v), end(v), [&](const int& i){
        cout << i << " ";
    });
    cout << endl;
}
```

Link to reference

std::min_element / std::max_element

- Returns iterator to the maximum/minimum element in array
- Can take custom predicate to use as a compare function (usually a lambda)
- Takes 2 iterators first and last and finds the element in that range

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

using namespace std;

int main() {
    vector<int> v = {1,4,2,1,4,6,8,-10};
    vector<int>::iterator it =
                    min_element(begin(v), end(v));
    cout << "Minimum value is: " << *it << endl;
}
```

Link to reference(min)

Link to reference(max)

Writing our own algorithms

- We will write a map-reduce algorithm that takes 2 iterators, first and last and 2 functions map and reduce and performs the map-reduce operation.

- We will make it work with any kind of STL container and with any data type.

- The point of the algorithms is to be as generic as possible and that’s what we will focus on.

```cpp
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>
#include <numeric>
#include <iterator>

using namespace std;

template<typename Iterator, typename Map, typename Reduce>
int map_reduce(Iterator first, Iterator last, Map m, Reduce r) {
    Iterator tmp = first;
    while(tmp != last) {
        *tmp = m(*tmp);
        tmp++;
    }
    return r(first, last);
}

template<typename Iterator,
         typename r_type = typename std::iterator_traits<Iterator>::value_type>
r_type sum(Iterator first, Iterator last) {
    r_type res = *first;
    while(++first != last)
        res += *first;
    return res;
}

int main() {
    vector<int> v = {1,2,3,4,5};
    int res = map_reduce(begin(v), end(v), [](const int& i){
        return i*i;
    }, sum<vector<int>::iterator>);
    cout << res << endl;
}
```

Our Map-Reduce function

Gives value type of iterator

Lambda expression

Need to pass template parameters!

Thank you for attending the workshops!

Practice!
https://www.hackerrank.com/

Resources
https://github.com/PanagiotisPtr/cpp_workshop