Lecture 9 Functions and Algorithms

#CS106L

- **Concept Lifting Part 2**
- **Lambda Functions**
- **Algorithms**
- **Iterator Adaptors**

---

basic algorithm:

- `std::sort`
- `std::nth_element`
- `std::stable_partition`
- `std::copy_if`
- `std::remove_if`

why use algorithms?

• Abstraction: perform algorithms without looking at elements.

• Generic: operations are based on ranges, not containers.

• Correct: heavily tested, most definitely correct.

• Heavily optimized: performs optimizations using features we haven't/won't even learn about.

`std::remove` does not change the size of the container!The algorithm is not a member of any collection , so it can’t change its size member.

Q1:how does `std::copy_if` work?

`std::copy_if` is a function template in the C++ Standard Template Library (STL) that copies elements from a range to a new location only if they satisfy a certain condition. It takes three arguments:

- The first argument is an input iterator specifying the beginning of the source range.
- The second argument is an input iterator specifying the end of the source range.
- The third argument is an output iterator specifying the beginning of the destination range.

It also takes an additional unary predicate, a function object that takes an element from the source range as argument and returns a bool indicating whether the element should be copied.

Here is an example of using `std::copy_if` to copy all elements greater than 5 from an array to a vector:

```
#include <algorithm>

#include <vector>

using namespace std;

int main() {

    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    vector<int> vec;

    copy_if(begin(arr), end(arr), back_inserter(vec), [](int x){ return x > 5; });

    for (auto i : vec)  cout << i << " ";

    return 0;

}
```

The output will be 6,7,8,9,10

It works by iterating over the source range and for each element, it calls the unary predicate to check if it should be copied. If the predicate returns true, the element is copied to the destination range.

Q2:how does `std::remove_if` work?

`std::remove_if` is a standard algorithm in C++ that removes elements from a container (such as a vector or list) based on a given predicate. The function takes three arguments:

- the first is the start of the range of elements to be removed,
- the second is the end of the range of elements to be removed,
- the third is the predicate that will be used to determine which elements should be removed.

Here is an example of how to use `std::remove_if` to remove all negative values from a vector:

```
std::vector<int> v = {1, -2, 3, -4, 5};

auto new_end = std::remove_if(v.begin(), v.end(), [](int x){ return x < 0; });

v.erase(new_end, v.end());
```

`std::remove_if` works by moving all the elements that do not satisfy the predicate to the beginning of the range, and returning an iterator to the new end of the valid range. The elements that satisfy the predicate are still present in the container, but they are now in an unspecified order. To remove them, you need to call the `erase` function on the container, passing the returned iterator as the first argument, and the original end of the range as the second argument.

Note that `std::remove_if` does not work with associative containers like `std::map` and `std::set`, as it does not preserve the order of the elements.

namely, erase-remove idiom:

```
        v.erase(std::remove_if(v.begin(), v.end(), pred),v.end());
```