STL Algorithms

C++ for Developers

Content

- Functions Refreshment
- Algorithms
  - Copying
  - Looping
  - Searching
  - Counting
  - Deleting
  - Sorting

Functions

C++ for Developers

Function pointers

```
void doCalculation(int x, int y, void (*callback)(int)) {
    callback(addTwoNumbers(x,y));
}
```

```
void (*callback)(int)
```

Function pointer

Return type

Function name

Parameters

std::function is defined i
<functional>

```
void doDivision(double x, double y, std::function<void(double)> callback) {
    callback(x / y);
}
```

```
std::function<void(double)> callback
```

Return type

Parameters

Function name

lafftale
for developers

Lambda functions

A lambda is a tiny, nameless function you write inline.
It's defined right where you use it, and it can "capture" variables from the surrounding scope so it can use them.

```cpp
// USING A LAMBDA FUNCTION TO SORT A VECTOR
std::sort(v.begin(), v.end(), [](int a, int b) {
    return a < b;
});

for (auto number : v) {
    std::cout << number << std::endl;
}

// USING A LAMBDA FUNCTION TO ADD NUMBERS
int number5 = [=]() {
    return number1 + number2;
}();

// DECLARING A LAMBDA FUNCTION
auto localAdder = [](int x, int y) {
    return x + y;
};
int number6 = localAdder(number3, number4);
std::cout << "Lambda localAdder result: " << number6 << std::endl;

// PASSING LAMBDA FUNCTION AS A PARAMETER
doCalculation(number1, number2, [](int value) {
    std::cout << "Lambda callback function value : " << value << std::endl;
});
```

lafftale for developers

Lambda syntax

Within [ ] you define the scope you capture with the lambda function:

[ ] = capture nothing

[=] = any used variable from outside is captured by copy

[&] = any used variable from outside is captured by reference

[a, &b] = capture variable a by copy and b by reference

[a = aNew] = captures the variable by copy and gives it a new name. It works the same for reference.

Algorithms

C++ for Developers

Copying

STL provides a generic way to copy elements from one container to another. By providing the *range* you want to copy and a way to insert it into the container.

std::copy(container.begin(), container.end(), std::inserter_type(targetContainer);

```
MovieList movies = MovieList(50);
auto& r_movies = movies.getMovies();

std::deque<Movie> movieDeque;
std::copy(r_movies.begin(), r_movies.end(), std::back_inserter(movieDeque));
```

Looping

You can loop through containers using the std::for_each(). Here you specify the range and the function to be performed.

std::for_each(container.begin(), container.end(), function());

```
std::for_each(r_movies.begin(), r_movies.end(), [](const auto& e) {
    std::cout << e.getTitle() << std::endl;
});
```

std::for_each_n(container.begin(), numberOfIterations, function());

```
std::for_each_n(r_movies.begin(), 5, [](const auto& e) {
    std::cout << e.getTitle() << std::endl;
});
```

Searching

You can search for certain elements inside your container using different functions

std::all_of(container.begin(), container.end(), function());
Returns true if all of the function calls return true.

std::any_of(container.begin(), container.end(), function());
Returns true if any of function calls return true.

std::none_of(container.begin(), container.end(), function());
Returns true if none of the function calls return true.

Searching

You can find elements and get their positions by using std::find. This function returns an iterator to where the element was found - or the end of container if it wasn't found.

std::find(container.begin(), container.end(), function());
Returns an iterator of where the element is found (or container.end() if not found).

```
const auto find_it = std::find_if(movieList.begin(), movieList.end(), searchFunction);
if (find_it != movieList.end()) {
    std::cout << "The list contains the movie: " << find_it->getTitle() << std::endl;
}
```

Counting

You can count occurrences in your containers using counting functions.

std::count_if(container.begin(), container.end(), function());
Returns the amount of times the function returned true.

```
std::string yearTarget = "1994";
auto count = std::count_if(movieForwardList.begin(), movieForwardList.end(), [&yearTarget](const auto& movie) {
    return movie.getYearOfRelease() == yearTarget;
});
std::cout << "There where " << count << " movies released in " << yearTarget << std::endl;
```

Removing

You can remove elements that matches the criteria in the function by using std::remove_if(). This will rearrange the elements and place them in a range and return an iterator to the start of that range.

std::remove_if(container.begin(), container.end(), function());
Rearranges the elements matching the function and returns an iterator to the beginning of that range.

```
auto remove_it = std::remove_if(movieVector.begin(), movieVector.end(), [&yearTarget](const Movie& movie) {
    return movie.getYearOfRelease() == yearTarget;
});

// then we use this iterator to create a range of elements we want to remove
movieVector.erase(remove_it, movieVector.end());
```

Sorting

Sort your container using std::sort. It either needs the element to have assigned the less than operator or pass in a comparative function as in the example.

std::sort(container.begin(), container.end(), function());
Sorts the container based on your comparison. (less or more than).

```
std::sort(movieVector.begin(), movieVector.end(), [](const auto& a, const auto& b) {
    return a.getTitle() < b.getTitle();
});
```

There are a bunch more...

Check out the C++ Reference website for more algorithms:

https://en.cppreference.com/w/cpp/algorithm.html


lafftale
for developers