for_each

As the name suggests, this method picks up each value in our container and performs the desired action.

`std::for_each` applies a unary callable to each element of its range. The range is given by the input iterators.

```
UnFunc std::for_each(InpIt first, InpIt second, UnFunc func)
void std::for_each(ExePol pol, FwdIt first, FwdIt second, UnFunc func)
```

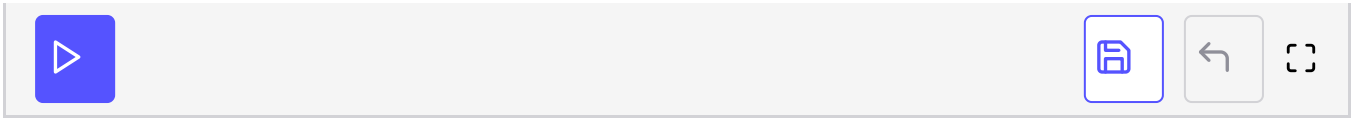
`std::for_each`, when used without an explicit execution policy is a special algorithm because it returns its callable argument. If we invoke `std::for_each` with a function object, we can store the result of the function call directly in the function object.

```
InpIt std::for_each_n(InpIt first, Size n, UnFunc func)
FwdIt std::for_each_n(ExePol pol, FwdIt first, Size n, UnFunc func)
```

`std::for_each_n` is new with C++17 and applies a unary callable to the first n elements of its range. The range is given by an input iterator and a size.

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

template <typename T>
class ContainerInfo{
public:

  void operator()(T t){
    num++;
    sum+= t;
  }

  int getSum() const{
    return sum;
  }

  int getSize() const{ return num; }
```

```
  double getMean() const{
      return static_cast<double>(sum) / static_cast<double>(num);
  }

private:
  T sum{0};
  int num{0};
};

int main(){

  std::cout << std::endl;

  std::vector<double> myVec{1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};

  auto vecInfo= std::for_each(myVec.begin(), myVec.end(), ContainerInfo<double>());
  std::cout << "vecInfo.getSum(): " << vecInfo.getSum() << std::endl;
  std::cout << "vecInfo.getSize(): " << vecInfo.getSize() << std::endl;
  std::cout << "vecInfo.getMean(): " << vecInfo.getMean() << std::endl;

  std::cout << std::endl;

  std::array<int, 100> myArr{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  auto arrInfo= std::for_each(myArr.begin(), myArr.end(), ContainerInfo<int>());
  std::cout << "arrInfo.getSum(): " << arrInfo.getSum() << std::endl;
  std::cout << "arrInfo.getSize(): " << arrInfo.getSize() << std::endl;
  std::cout << "arrInfo.getMean(): " << arrInfo.getMean() << std::endl;

  std::cout << std::endl;

}
```



To build upon our understanding of this topic, let’s solve an exercise in the next lesson.