Iterator Creation

In this lesson, we'll observe how a map creates and handles its iterator.

Each container generates its suitable iterator on request. For example, an `std::unordered_map` generates constant and non-constant forward iterators.

```
std::unordered_map<std::string, int>::iterator unMapIt= unordMap.begin();
std::unordered_map<std::string, int>::iterator unMapIt= unordMap.end();

std::unordered_map<std::string, int>::const_iterator unMapIt= unordMap.cbegin();
std::unordered_map<std::string, int>::const_iterator unMapIt= unordMap.cend();
```

In addition, `std::map` supports backward iterators:

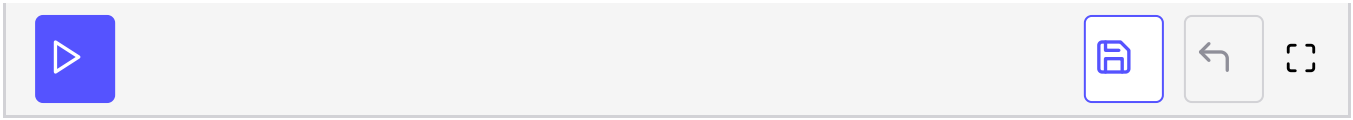
```
std::map<std::string, int>::reverse_iterator mapIt= map.rbegin();
std::map<std::string, int>::reverse_iterator mapIt= map.rend();

std::map<std::string, int>::const_reverse_iterator mapIt= map.rcbegin();
std::map<std::string, int>::const_reverse_iterator mapIt= map.rcend();
```

> 🔑 **Use `auto` for iterator definition**
> Iterator definition is very labour intensive. Automatic type deduction with `auto` reduces writing to the bare minimum.
>
> ```
> std::map<std::string, int>::const_reverse_iterator
> mapIt= map.rcbegin();
> auto mapIt2= map.rcbegin();
> ```

The final example:

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>

int main(){
```

```
int main(){

  std::cout <<  std::endl;

  std::unordered_map<std::string, int> unordMap{ {"Rainer", 1966}, {"Beatrix", 1966}, {"Julie

  std::unordered_map<std::string, int>::const_iterator endMapIt= unordMap.end();
  std::unordered_map<std::string, int>::iterator mapIt;

  for ( mapIt= unordMap.begin(); mapIt != endMapIt; ++mapIt ) std::cout << "{" << mapIt->firs

  std::cout << "\n\n";

  std::vector<int> myVec{1, 2, 3, 4, 5, 6, 7, 8, 9};

  std::vector<int>::const_iterator vecEndIt= myVec.end();
  std::vector<int>::iterator vecIt;
  for ( vecIt= myVec.begin(); vecIt != vecEndIt; ++vecIt ) std::cout << *vecIt << " ";

  std::cout << std::endl;

  for ( const auto v: myVec ) std::cout << v  << " ";

  std::cout << std::endl;

  std::vector<int>::const_reverse_iterator vecEndRevIt= myVec.rend();
  std::vector<int>::reverse_iterator vecRevIt;
  for ( vecRevIt= myVec.rbegin(); vecRevIt != vecEndRevIt; ++vecRevIt ) std::cout << *vecRevI

  std::cout << "\n\n";

}
```



In the next lesson, we'll discuss tools that make the iteration process simple.