Reference Wrappers

C++ takes reference functionality one step higher by introducing reference wrappers!

WE'LL COVER THE FOLLOWING

A reference wrapper is a copy-constructible and copy-assignable wrapper for an object of type&, which is defined in the header `<functional>`. It is an object that behaves like a reference, but can be copied. Contrary to classic references, `std::reference_wrapper` objects support two additional use cases:

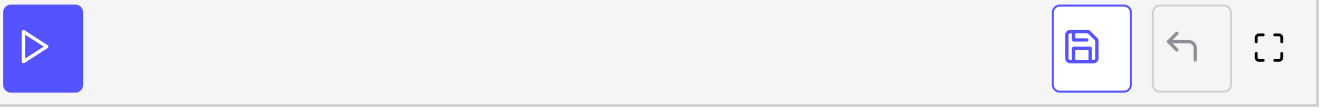
- They can be used in containers of the Standard Template Library. `std::vector<std::reference_wrapper<int>> myIntRefVector`
- They can be copy instances of classes, which have `std::reference_wrapper` objects. That is generally not possible with references.

To access the reference of a `std::reference_wrapper<int> myInt(1)`, the get method can be used: `myInt.get()`. We can also use a reference wrapper to encapsulate and invoke a callable.

Let's discuss three different examples for a better understanding of the concept:

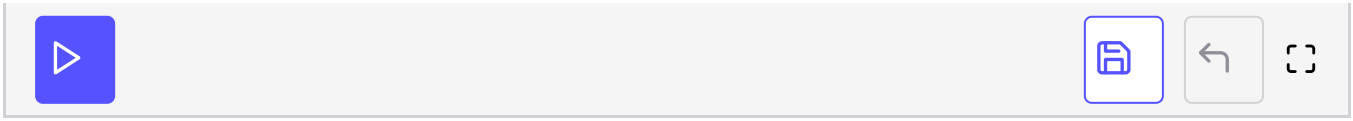
```
#include <functional>
#include <iostream>
#include <vector>

int main(){

  std::cout << std::endl;

  // will not compile
  //std::vector<int&> myIntRefVector;
```

```
  int a = 0;
  int b = 0;

  int c = 0;

  std::vector< std::reference_wrapper<int>> myIntRefVector= {std::ref(a), std::ref(b), std::r

  for (auto b: myIntRefVector) std::cout << b << " ";

  std::cout << std::endl;

  // modify b and also myIntRefVec[1] !!!!
  b = 2011;

  for (auto b: myIntRefVector) std::cout << b << " ";

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

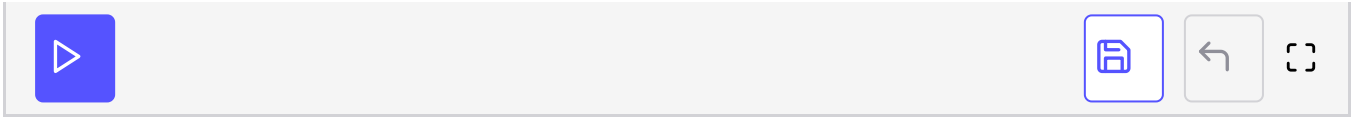


```
#include <functional>
#include <iostream>
#include <string>

class Bad{
public:
  Bad(std::string& s):message(s){}
private:
  std::string& message;
};

class Good{
public:
  Good(std::string& s):message(s){}
  std::string getMessage(){
    return message.get();
  }
  void changeMessage(std::string s){
    message.get()= s;
  }
private:
  std::reference_wrapper<std::string> message;
};

int main(){

  std::cout << std::endl;

  std::string bad1{"bad1"};
  std::string bad2{"bad2"};

  Bad b1(bad1);
  Bad b2(bad2);
  // will not compile, because of reference
  //b1= b2;

  std::string good1{"good1"};
```

```
  std::string good2{"good2"};

  Good g1(good1);

  Good g2(good2);
  std::cout << "g1.getMessage(): " << g1.getMessage() << std::endl;
  std::cout << "g2.getMessage(): " << g2.getMessage() << std::endl;

  std::cout << std::endl;

  std::cout << "g2= g1" << std::endl;
  g2= g1;
  std::cout << "g1.getMessage(): " << g1.getMessage() << std::endl;
  std::cout << "g2.getMessage(): " << g2.getMessage() << std::endl;

  std::cout << std::endl;

  g1.changeMessage("veryGood");
  std::cout << "g1.changeMessage(\"veryGood\")" << std::endl;
  std::cout << "g1.getMessage(): " << g1.getMessage() << std::endl;
  std::cout << "g2.getMessage(): " << g2.getMessage() << std::endl;

  std::cout << std::endl;

}
```



```
// referenceWrapperCallable.cpp
#include <iostream>
#include <functional>

void foo(){
  std::cout << "Invoked" << std::endl;
}

int main() {
  typedef void callableUnit();
  std::reference_wrapper<callableUnit> refWrap(foo);

  refWrap(); // Invoked
  return 0;
}
```

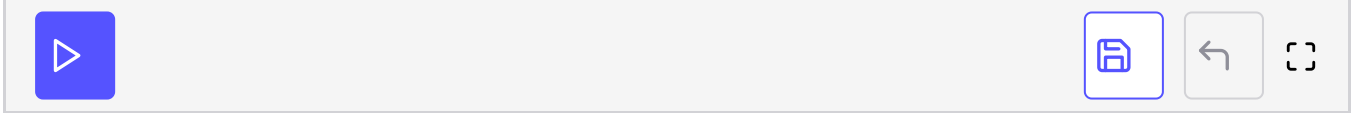


Reference wrappers

`std::ref` and `std::cref`

With the helper functions `std::ref` and `std::cref` we can easily create reference wrappers to variables. `std::ref` will create a non constant reference wrapper, while `std::cref` will create a constant one:

```
#include <functional>
#include <iostream>
#include <string>

void invokeMe(std::string& s){
  std::cout << s << ": not const " << std::endl;
}

void invokeMe(const std::string& s){
  std::cout << s << ": const " << std::endl;
}

template <typename T>
void doubleMe(T t){
  t *= 2;
}

int main(){

  std::cout << std::endl;

  std::string s{"string"};

  invokeMe(std::ref(s));
  invokeMe(std::cref(s));

  std::cout << std::endl;

  int i = 1;
  std::cout << "i: " << i << std::endl;

  doubleMe(i);
  std::cout << "doubleMe(i): " << i << std::endl;

  doubleMe(std::ref(i));
  std::cout << "doubleMe(std::ref(i)): " << i << std::endl;

  double a = 5;
  std::cout << "a= " << a << std::endl;
  doubleMe(std::ref(a));
  std::cout << "doubleMe(std::ref(a)): " << a << std::endl;

  std::cout << std::endl;

}
```



The helper functions `std::ref` and `std::cref`

So it's possible to invoke the function `invokeMe`, which gets a constant reference to an `std::string`, with a non-constant `std::string s`, which is wrapped in an `std::cref(s)`. If we wrap the variable `i` in the helper function

`std::ref`, the function template `doubleMe` will be invoked with a reference. So, the variable `i` will be doubled.

Further information

- copy-constructible
- copy-assignable

In the next lesson, let's talk about type-traits in C++.