- Examples

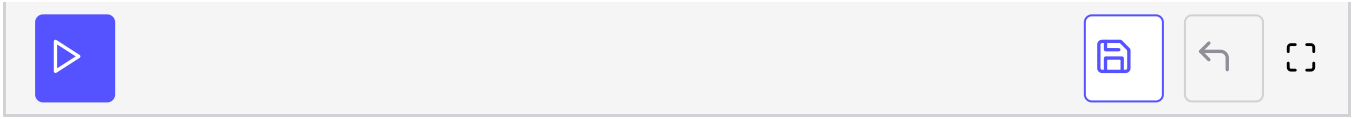
Below, we can find some examples of copy and move semantics in action.

WE'LL COVER THE FOLLOWING

Copying and moving strings

```
#include <iostream>
#include <string>
#include <utility>

int main(){

  std::string str1{"ABCDEF"};
  std::string str2;

  std::cout << "\n";

  // initial value
  std::cout << "str1 = " << str1 << std::endl;
  std::cout << "str2 = " << str2 << std::endl;

  // copy semantic
  str2= str1;
  std::cout << "str2 = str1;\n";
  std::cout << "str1 = " << str1 << std::endl;
  std::cout << "str2 = " << str2 << std::endl;

  std::cout << "\n";

  std::string str3;

  // initial value
  std::cout << "str1 = " << str1 << std::endl;
```

```
  std::cout << "str3 = " << str3 << std::endl;

  // move semantic

  str3= std::move(str1);
  std::cout << "str3 = std::move(str1);\n";
  std::cout << "str1 = " << str1 << std::endl;
  std::cout << "str3 = " << str3 << std::endl;

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



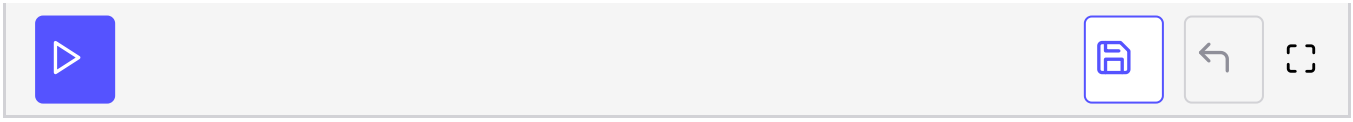
Explanation

- In the example above, we are demonstrating how the value of `str1` can be transferred to strings using the copy semantic and the move semantic.
- In line 17, we have used the copy semantic and the string `"ABCDEF"` is present in both `str1` and `str2`. We can say the value has been copied from `str1` to `str3`.
- In line 31, we have used the move semantic and now the string `"ABCDEF"` is present only in `str3` and not in `str1`. We can say the value has moved from `str1` to `str3`.

Swap

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

template <typename T>
void swap(T& a, T& b){
    T tmp(std::move(a));
    a = std::move(b);
    b = std::move(tmp);
}

struct MyData{
  std::vector<int> myData;

  MyData():myData({1, 2, 3, 4, 5}){}

  // copy semantic
  MyData(const MyData& m):myData(m.myData){
    std::cout << "copy constructor"  << std::endl;
  }

  MyData& operator=(const MyData& m){
    myData = m.myData;
```

```
    std::cout << "copy assignment operator"  << std::endl;
    return *this;
  }

};

int main(){

  std::cout << std::endl;

  MyData a, b;
  swap(a, b);

  std::cout << std::endl;

};
```



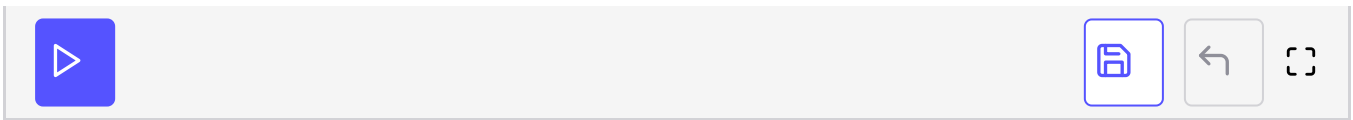
Explanation

- The example shows a simple swap function that uses the move semantic internally. `MyData` doesn't support move semantics.
- Line 7 invokes the copy constructor in line 18.
- Lines 8 and 9 invoke the copy assignment operator defined in line 22.
- When we invoke move on an only copyable type, copy-semantic will kick in as fallback to move-semantic. The reason is that an rvalue is first bound to an rvalue reference and second to a const lvalue reference. The copy constructor and the copy assignment operator take constant lvalue references.

BigArray

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

using std::cout;
using std::endl;

using std::chrono::system_clock;
using std::chrono::duration;

using std::vector;

class BigArray{
```

```
public:
  BigArray(size_t len): len_(len), data_(new int[len]){}


  BigArray(const BigArray& other): len_(other.len_), data_(new int[other.len_]){
    cout << "Copy construction of " << other.len_ << " elements "<< endl;
    std::copy(other.data_, other.data_ + len_, data_);
   }

  BigArray& operator = (const BigArray& other){
     cout << "Copy assignment of " << other.len_ << " elements "<< endl;
     if (this != &other){
        delete[] data_;

        len_ = other.len_;
        data_ = new int[len_];
        std::copy(other.data_, other.data_ + len_, data_);
     }
     return *this;
  }

  ~BigArray(){
     if (data_ != nullptr) delete[] data_;
  }

private:
  size_t len_;
  int* data_;
};

int main(){

  cout << endl;

  vector<BigArray> myVec;

  auto begin = system_clock::now();

  myVec.push_back(BigArray(1000000000));

  auto end = system_clock::now() - begin;
  auto timeInSeconds = duration<double>(end).count();

  cout << endl;
  cout << "time in seconds: " << timeInSeconds << endl;
  cout << endl;

}
```



Explanation

- `BigArray` only supports copy semantic. This is a performance issue in line 53. The containers of the standard template library have copy-semantic.

- This means, that they want to copy all elements. If `BigArray` would have

move-semantic implemented, move-semantic would have been used

automatically in line 53, because the constructor call, `BigArray(1000000000)`, creates an rvalue.

Let's test our understanding of this topic with an exercise in the next lesson.