- Examples

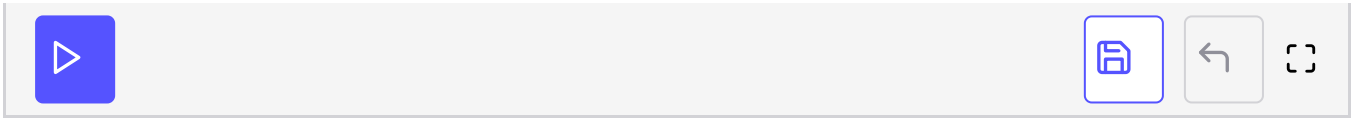
Some examples of copy and move semantics will be discussed in this lesson.

WE'LL COVER THE FOLLOWING

Example 1

```
// copyMoveSemantic.cpp
#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

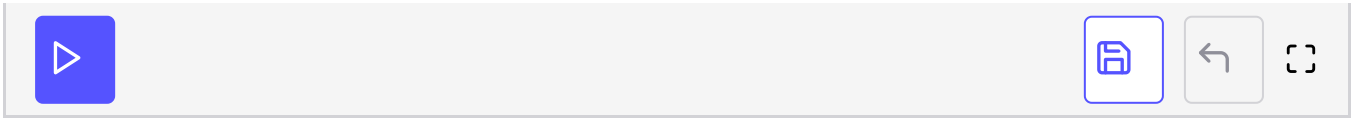
In the above example, we demonstrated how the value of `str1` can be transferred to strings using two different methods: copy semantic and move semantic.

- In line 18, we used the copy semantic, and the string `"ABCDEF"` is present in both `str1` and `str2`. We can, therefore, say that the value has been *copied* from `str1` to `str2`.
- In line 32, we used the move semantic, and now the string `"ABCDEF"` is present only in `str3` but not in `str1`. We can therefore say the value has *moved* from `str1` to `str3`.

Example 2

```
//swap.cpp
#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 the workings of a simple `swap` function that uses the move semantic internally. `MyData` does not support move semantic.
- Line 9 invokes the move constructor in line 20.
- Lines 10 and 11 invoke the move assignment operator defined in line 24.
- When you invoke move on a copyable type, copy-semantic will begin. This is due to the fact that an rvalue is first bound to an rvalue reference, and the second is bound to a const lvalue reference.
- Copy semantic is a fallback for move semantic.

Example 3

```
//bigArray.cpp
#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 54. The containers of the standard template library have copy-semantic.
- This means that the containers want to copy all elements. If `BigData` had implemented move semantic implemented, it would have been used automatically in line 54 since the constructor call `BigArray(1000000000)` creates an rvalue.

> Copy semantic is a fallback for move semantic.

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