- Solution

We'll discuss the solution to the exercises covered in the previous lesson.

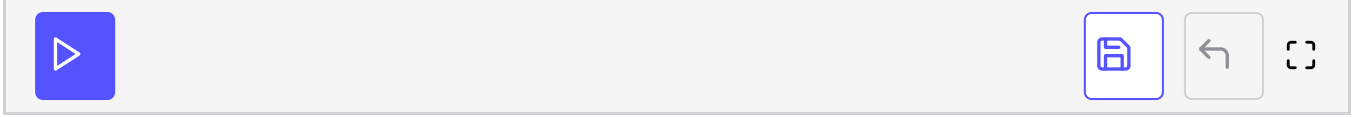
WE'LL COVER THE FOLLOWING

Solution

copy | move

```
//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

- In lines 37-40, we defined the move constructor for `BigArray`. Note that in line 40, we explicitly set `other.data_` to `nullptr` after the elements have been *moved* into the new object.

- In lines 43-55, we defined the move assignment operator `=` for `BigArray`. Note that in line 51, we explicitly set `other.data_` to `nullptr` after the elements have been *moved*.

- As you can see, move semantic is way quicker than the copy semantic since we are only redirecting the pointer to an already stored data and assigning the correct `size`. Thus, the costs of move semantics are independent of the size of the data structure. This does not hold for the copy semantic. Memory allocation becomes more and more expensive the bigger the size of the user-defined type.

For further information read Rvalue References Explained by Thomas Becker.

In the next lesson, we will discuss perfect forwarding in modern C++.