- Solution

This is the solution to Exercise 2 from the previous lesson.

WE'LL COVER THE FOLLOWING

Solution

Move | Copy

```
//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]){}

  explicit 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(BigArray&& other): len_(other.len_), data_(other.data_){
     cout << "Move construction of " << other.len_ << " elements "<< endl;
     other.len_= 0;
     other.data_ = nullptr;
   }

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

       len_= other.len_;
       data_= other.data_;

       other.data_ = nullptr;
       other.len_= 0;
     }
     return *this;
   }

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

private:
  size_t len_;
  int* data_;
};

int main(){

  cout << endl;

  vector<BigArray> myVec;
  myVec.reserve(2);

  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 – 41, we have defined the move constructor for `BigArray`. Note that in line 40, we are explicitly setting `other.data_` to `nullptr` after the elements have been *moved* into the new object.
- In lines 43 – 55, we have defined the move assignment operator, `=`, for `BigArray`. Note that in line 51, we are explicitly setting `other.data_` to `nullptr` after the elements have been *moved*.
- As we can see, move semantic is significantly faster than the copy semantic since we are only redirecting the pointer to another vector and assigning the correct `size`. To be more precise, the costs of move semantics are independent of the size of the data structure. This does not hold true for the copy semantic. The bigger the user-defined data structure, the more expensive the memory allocation.

Further information

- rvalue references explained by Thomas Becker