Effective and Modern C++ Programming

Lab 2 – Smart Pointers

Exercise 1. Unique Pointers

Change the implementation of the class Matrix (from Lab 1) so that it stores pointer to dynamically allocated one dimensional array (data) in a `unique_ptr` not as a raw pointer.

Exercise 2. Shared Pointers

Implement a class String that uses copy-on-change semantics. Use `shared_ptr` to store pointer to dynamically allocated std::string.

```
#include <iostream>
#include <algorithm>
using namespace std;

class String{
public:
      String();  // creates an empty string
      String(const char * str); // copy C-string
      String(const String & );  // no copy
      String operator=(const String &); // no copy
       // makes a copy of a string if it has more than one reference.
      char & operator[](int i);
      // no copy
      char operator[](int i) const;
      // concatenation creates a new string only if both strings are non empty
      friend String operator+(String a, String b);
      // no copy
      friend std::ostream & operator<< (std::ostream & out, String s);
};

int main(){
    String a("hi");
    String b;
    const String c = a;      // no copy
    String d = a+b;          // no copy
    a[0] = 'l';              // copy
    a[1] = 'l';              // no copy
    d = c+a;                 // copy elision
    cout << c << “ “ << d << endl;    // hi hill   (no copy)
    cout << c[0] << endl;             // h         (no copy)
    return 0;
}
```

Exercise 3. Default and deleted constructors

Create an aggregate type MatrixPair with two public fields left and right of type Matrix.
Then disable making a copy of MatrixPair but allow move operation (using default and delete).

```
Matix m1({{1,2},{4,4}}),  m2(4,5);
PairOfMatrices p1 {m1, std::move(m2)};
// PairOfMatrices p2 = p1;   // ERROR!
PairOfMatrices p3 = std::move(p1);
Matrix a = p3.left, b=p3.right;
PairOfMatrices p4{a,b};
// p1 = p4;  // ERROR!
p1 = std::move(p4);
```