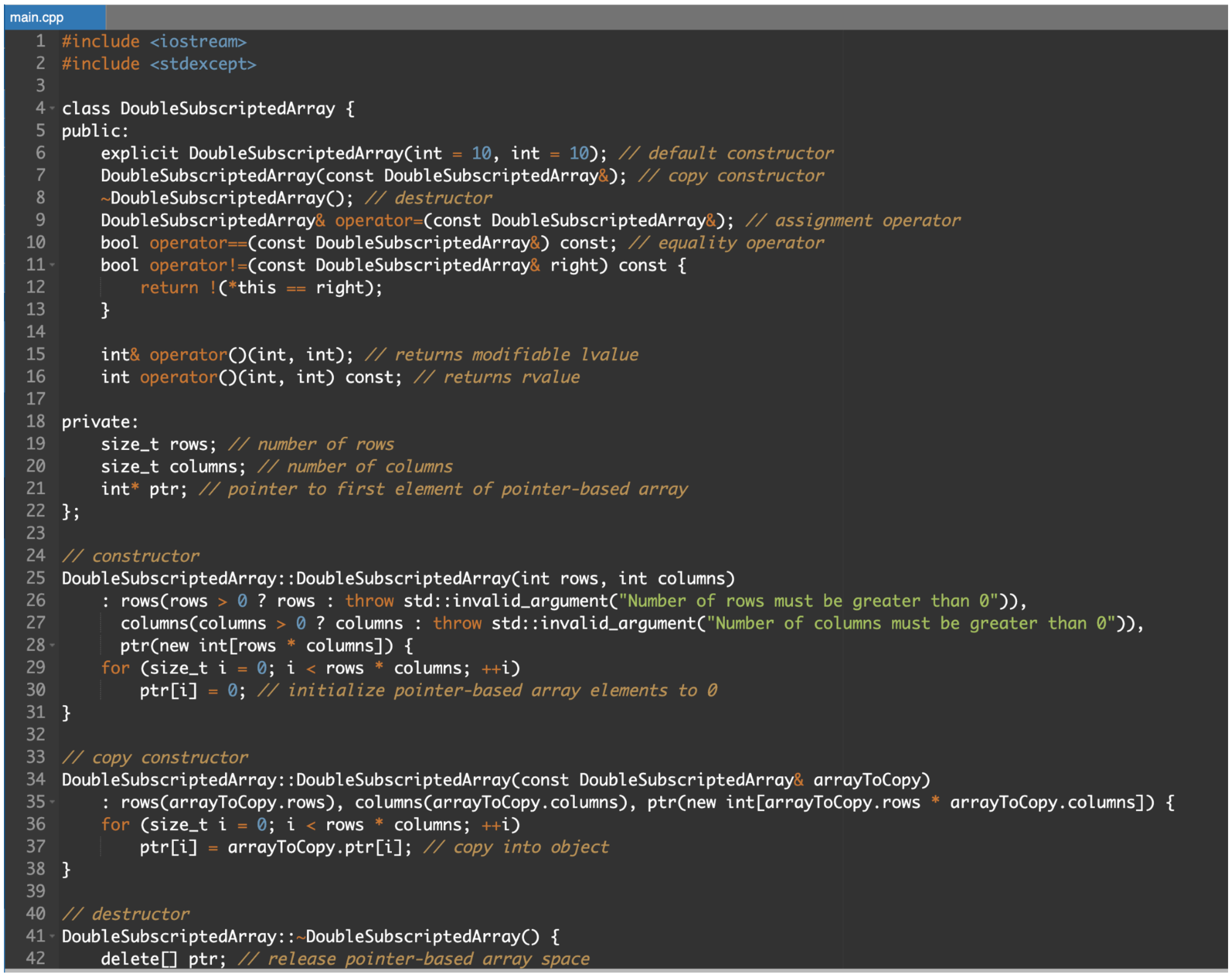
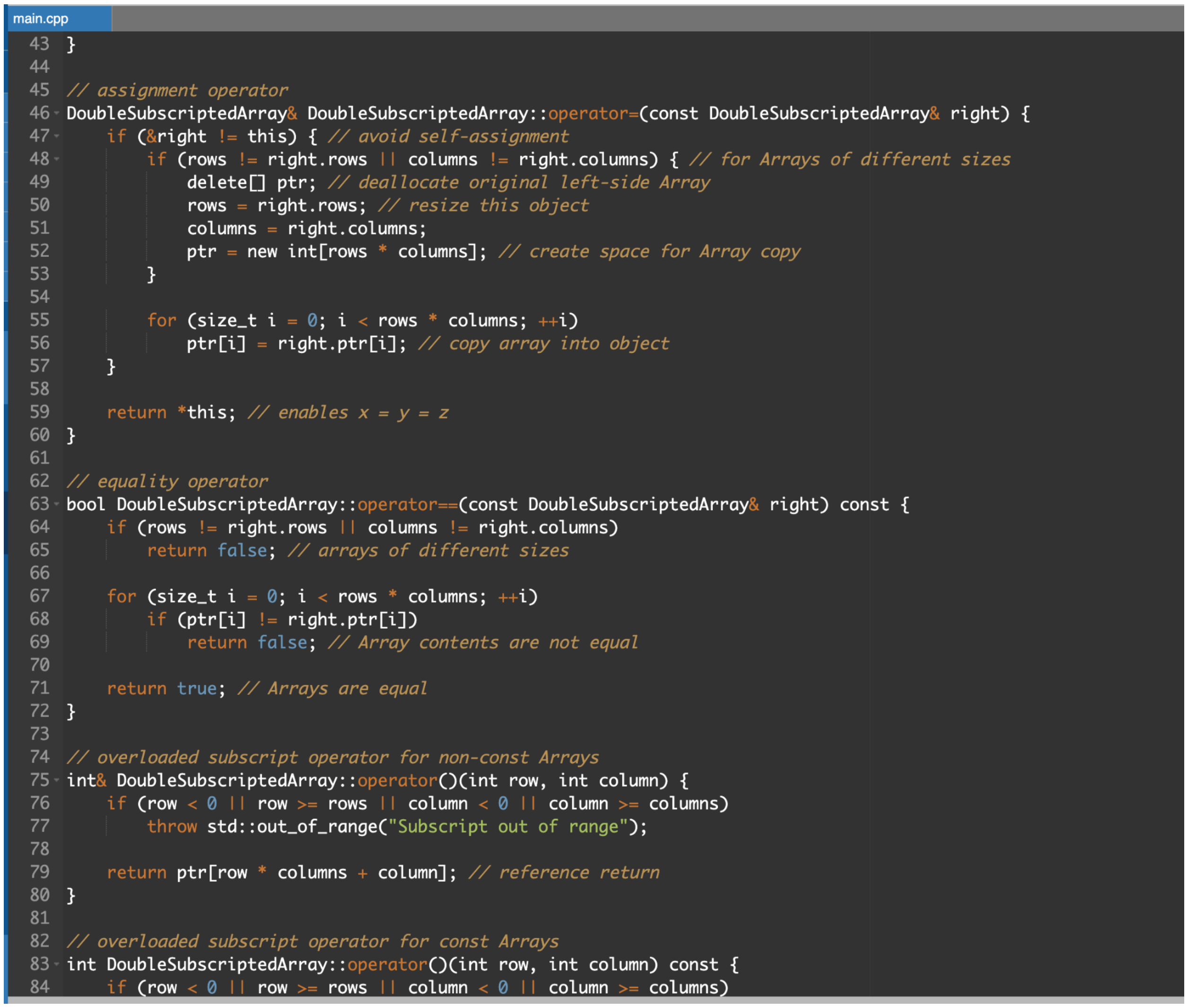
Samir Khadka
CS360 - Programming in C and C++
Homework Assignment #4

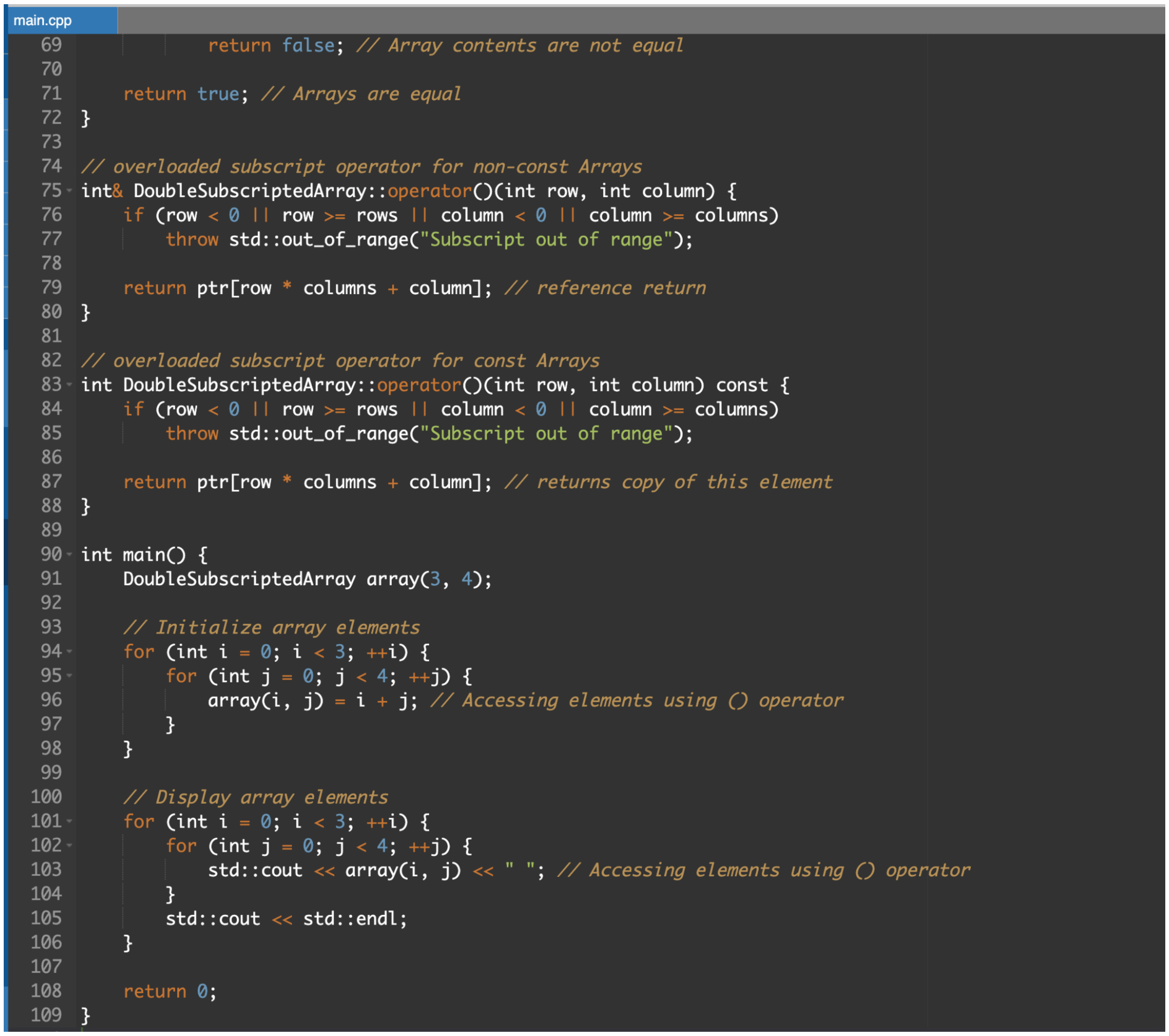
Question 1



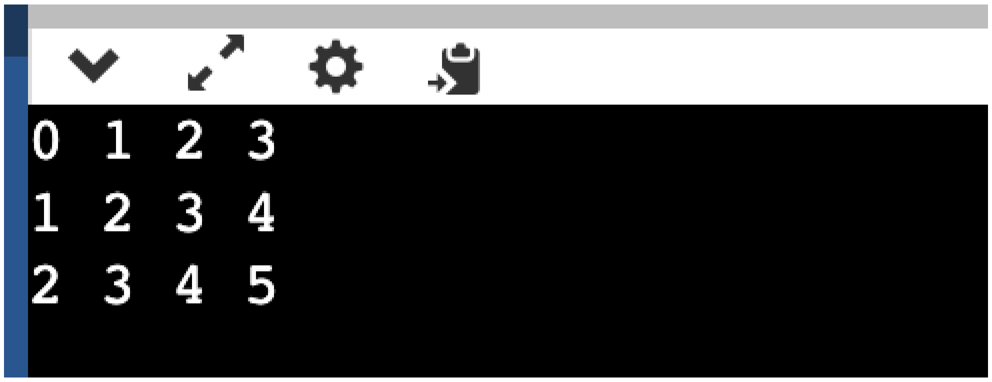
```cpp
#include <iostream>
#include <stdexcept>

class DoubleSubscriptedArray {
public:
    explicit DoubleSubscriptedArray(int = 10, int = 10); // default constructor
    DoubleSubscriptedArray(const DoubleSubscriptedArray&); // copy constructor
    ~DoubleSubscriptedArray(); // destructor
    DoubleSubscriptedArray& operator=(const DoubleSubscriptedArray&); // assignment operator
    bool operator==(const DoubleSubscriptedArray&) const; // equality operator
    bool operator!=(const DoubleSubscriptedArray& right) const {
        return !(*this == right);
    }

    int& operator()(int, int); // returns modifiable lvalue
    int operator()(int, int) const; // returns rvalue

private:
    size_t rows; // number of rows
    size_t columns; // number of columns
    int* ptr; // pointer to first element of pointer-based array
};

// constructor
DoubleSubscriptedArray::DoubleSubscriptedArray(int rows, int columns)
    : rows(rows > 0 ? rows : throw std::invalid_argument("Number of rows must be greater than 0")),
      columns(columns > 0 ? columns : throw std::invalid_argument("Number of columns must be greater than 0")),
      ptr(new int[rows * columns]) {
    for (size_t i = 0; i < rows * columns; ++i)
        ptr[i] = 0; // initialize pointer-based array elements to 0
}

// copy constructor
DoubleSubscriptedArray::DoubleSubscriptedArray(const DoubleSubscriptedArray& arrayToCopy)
    : rows(arrayToCopy.rows), columns(arrayToCopy.columns), ptr(new int[arrayToCopy.rows * arrayToCopy.columns]) {
    for (size_t i = 0; i < rows * columns; ++i)
        ptr[i] = arrayToCopy.ptr[i]; // copy into object
}

// destructor
DoubleSubscriptedArray::~DoubleSubscriptedArray() {
    delete[] ptr; // release pointer-based array space
```


main.cpp

```cpp
}

// assignment operator
DoubleSubscriptedArray& DoubleSubscriptedArray::operator=(const DoubleSubscriptedArray& right) {
    if (&right != this) { // avoid self-assignment
        if (rows != right.rows || columns != right.columns) { // for Arrays of different sizes
            delete[] ptr; // deallocate original left-side Array
            rows = right.rows; // resize this object
            columns = right.columns;
            ptr = new int[rows * columns]; // create space for Array copy
        }

        for (size_t i = 0; i < rows * columns; ++i)
            ptr[i] = right.ptr[i]; // copy array into object
    }

    return *this; // enables x = y = z
}

// equality operator
bool DoubleSubscriptedArray::operator==(const DoubleSubscriptedArray& right) const {
    if (rows != right.rows || columns != right.columns)
        return false; // arrays of different sizes

    for (size_t i = 0; i < rows * columns; ++i)
        if (ptr[i] != right.ptr[i])
            return false; // Array contents are not equal

    return true; // Arrays are equal
}

// overloaded subscript operator for non-const Arrays
int& DoubleSubscriptedArray::operator()(int row, int column) {
    if (row < 0 || row >= rows || column < 0 || column >= columns)
        throw std::out_of_range("Subscript out of range");

    return ptr[row * columns + column]; // reference return
}

// overloaded subscript operator for const Arrays
int DoubleSubscriptedArray::operator()(int row, int column) const {
    if (row < 0 || row >= rows || column < 0 || column >= columns)
```

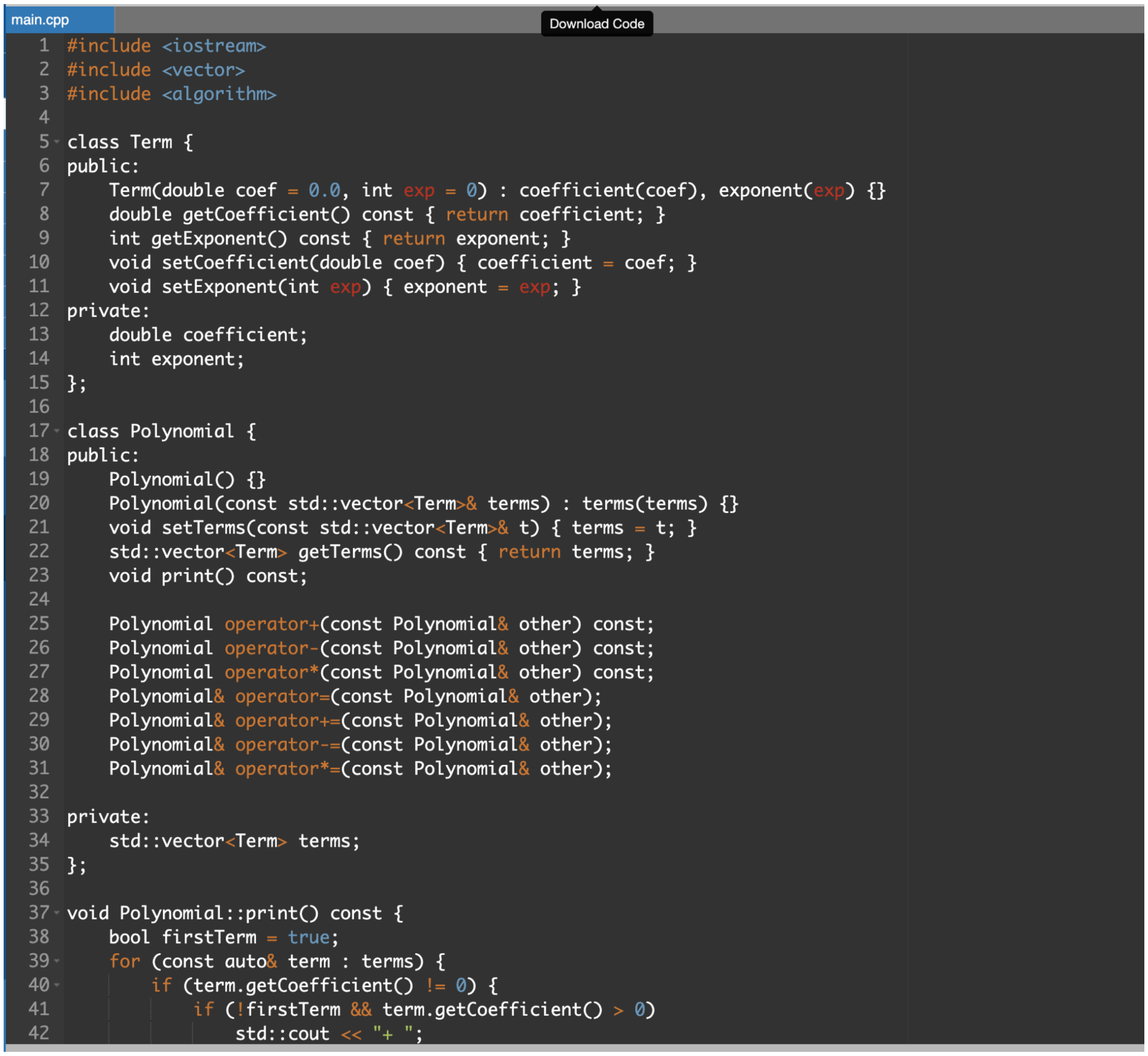


```cpp
                return false; // Array contents are not equal

    return true; // Arrays are equal
}

// overloaded subscript operator for non-const Arrays
int& DoubleSubscriptedArray::operator()(int row, int column) {
    if (row < 0 || row >= rows || column < 0 || column >= columns)
        throw std::out_of_range("Subscript out of range");

    return ptr[row * columns + column]; // reference return
}

// overloaded subscript operator for const Arrays
int DoubleSubscriptedArray::operator()(int row, int column) const {
    if (row < 0 || row >= rows || column < 0 || column >= columns)
        throw std::out_of_range("Subscript out of range");

    return ptr[row * columns + column]; // returns copy of this element
}

int main() {
    DoubleSubscriptedArray array(3, 4);

    // Initialize array elements
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 4; ++j) {
            array(i, j) = i + j; // Accessing elements using () operator
        }
    }

    // Display array elements
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 4; ++j) {
            std::cout << array(i, j) << " "; // Accessing elements using () operator
        }
        std::cout << std::endl;
    }

    return 0;
}
```

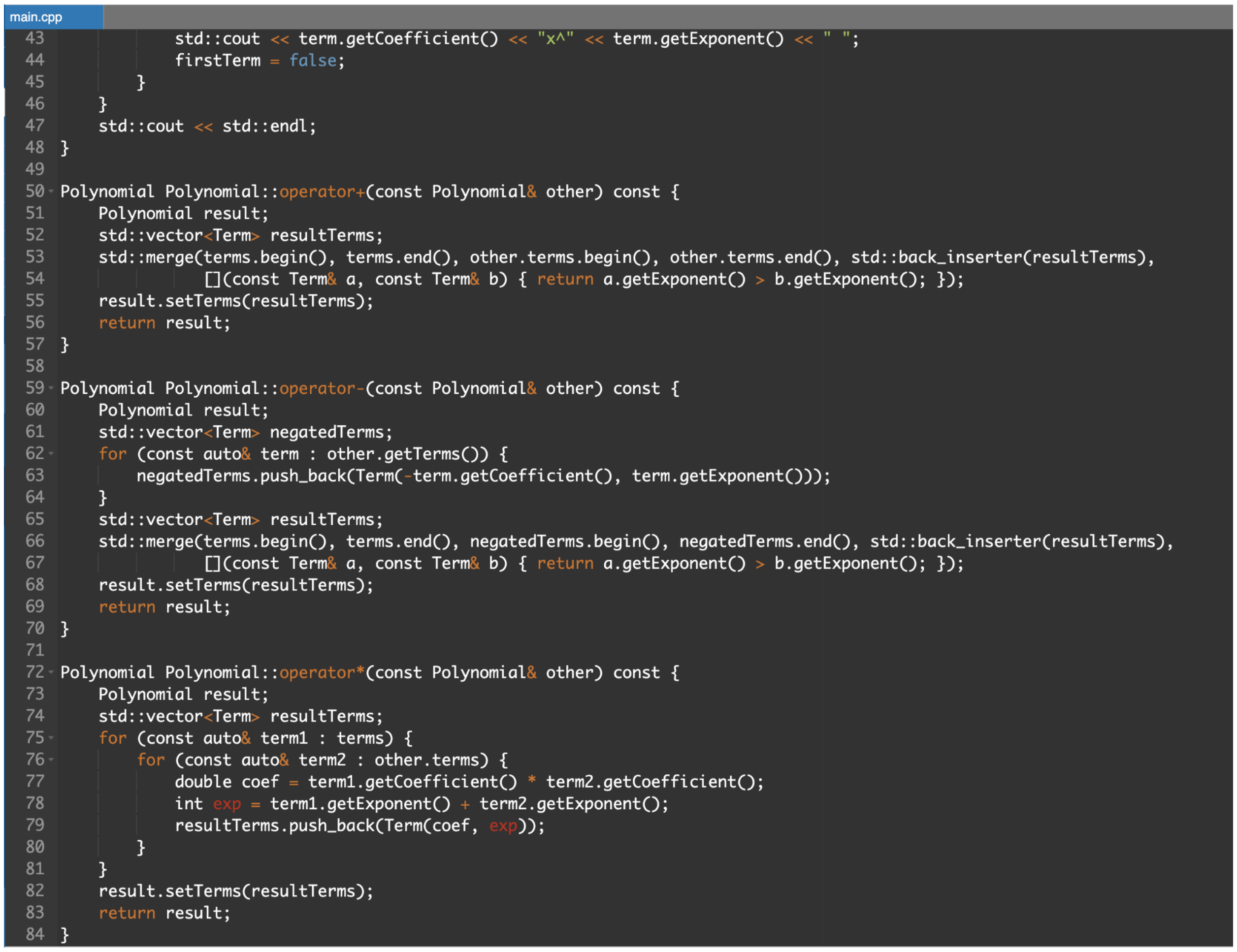


```
0 1 2 3
1 2 3 4
2 3 4 5
```

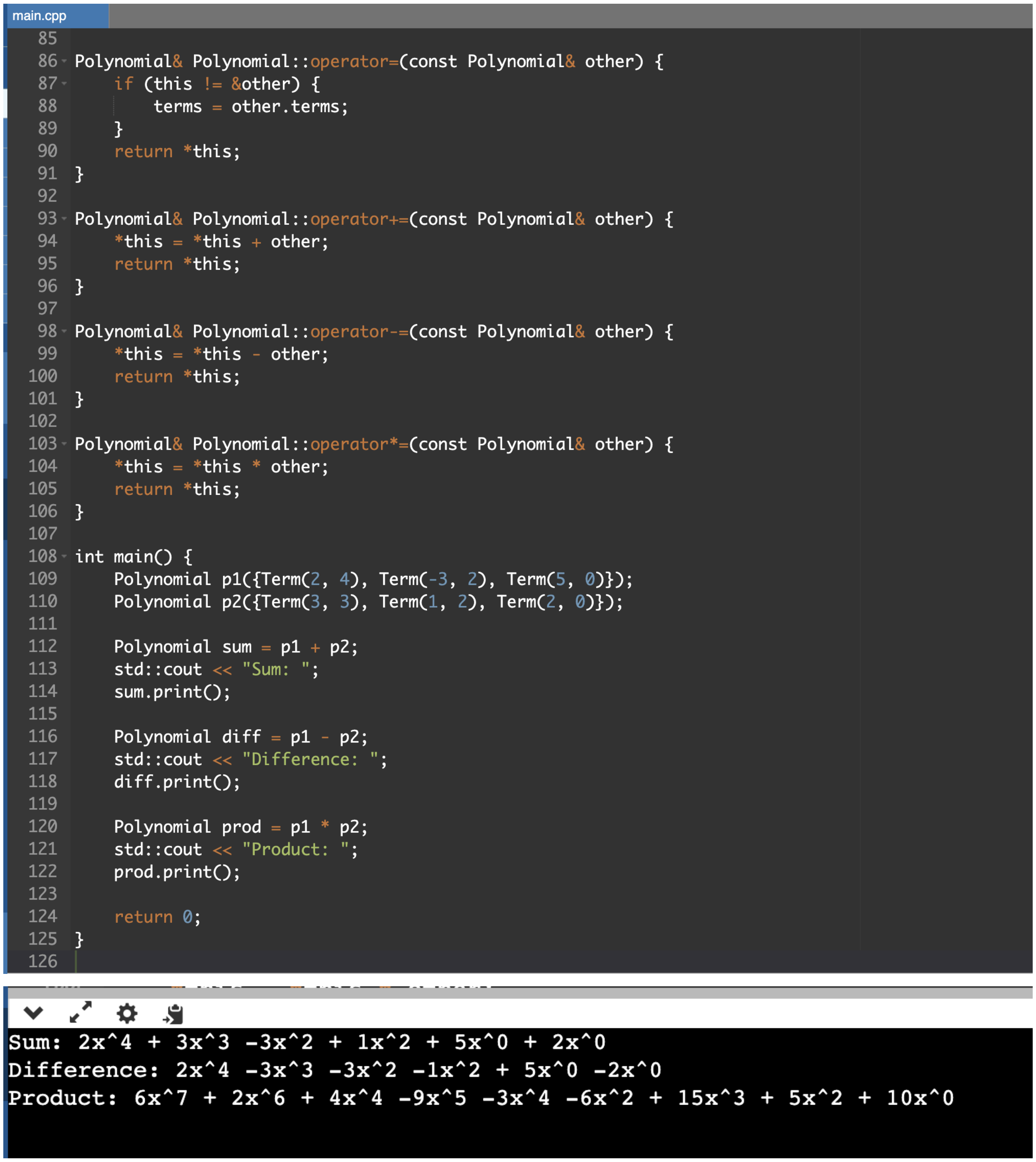
Question 2



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

class Term {
public:
    Term(double coef = 0.0, int exp = 0) : coefficient(coef), exponent(exp) {}
    double getCoefficient() const { return coefficient; }
    int getExponent() const { return exponent; }
    void setCoefficient(double coef) { coefficient = coef; }
    void setExponent(int exp) { exponent = exp; }
private:
    double coefficient;
    int exponent;
};

class Polynomial {
public:
    Polynomial() {}
    Polynomial(const std::vector<Term>& terms) : terms(terms) {}
    void setTerms(const std::vector<Term>& t) { terms = t; }
    std::vector<Term> getTerms() const { return terms; }
    void print() const;

    Polynomial operator+(const Polynomial& other) const;
    Polynomial operator-(const Polynomial& other) const;
    Polynomial operator*(const Polynomial& other) const;
    Polynomial& operator=(const Polynomial& other);
    Polynomial& operator+=(const Polynomial& other);
    Polynomial& operator-=(const Polynomial& other);
    Polynomial& operator*=(const Polynomial& other);

private:
    std::vector<Term> terms;
};

void Polynomial::print() const {
    bool firstTerm = true;
    for (const auto& term : terms) {
        if (term.getCoefficient() != 0) {
            if (!firstTerm && term.getCoefficient() > 0)
                std::cout << "+ ";
```


main.cpp

```cpp
            std::cout << term.getCoefficient() << "x^" << term.getExponent() << " ";
            firstTerm = false;
        }
    }
    std::cout << std::endl;
}

Polynomial Polynomial::operator+(const Polynomial& other) const {
    Polynomial result;
    std::vector<Term> resultTerms;
    std::merge(terms.begin(), terms.end(), other.terms.begin(), other.terms.end(), std::back_inserter(resultTerms),
               [](const Term& a, const Term& b) { return a.getExponent() > b.getExponent(); });
    result.setTerms(resultTerms);
    return result;
}

Polynomial Polynomial::operator-(const Polynomial& other) const {
    Polynomial result;
    std::vector<Term> negatedTerms;
    for (const auto& term : other.getTerms()) {
        negatedTerms.push_back(Term(-term.getCoefficient(), term.getExponent()));
    }
    std::vector<Term> resultTerms;
    std::merge(terms.begin(), terms.end(), negatedTerms.begin(), negatedTerms.end(), std::back_inserter(resultTerms),
               [](const Term& a, const Term& b) { return a.getExponent() > b.getExponent(); });
    result.setTerms(resultTerms);
    return result;
}

Polynomial Polynomial::operator*(const Polynomial& other) const {
    Polynomial result;
    std::vector<Term> resultTerms;
    for (const auto& term1 : terms) {
        for (const auto& term2 : other.terms) {
            double coef = term1.getCoefficient() * term2.getCoefficient();
            int exp = term1.getExponent() + term2.getExponent();
            resultTerms.push_back(Term(coef, exp));
        }
    }
    result.setTerms(resultTerms);
    return result;
}
```



main.cpp

```cpp

Polynomial& Polynomial::operator=(const Polynomial& other) {
    if (this != &other) {
        terms = other.terms;
    }
    return *this;
}

Polynomial& Polynomial::operator+=(const Polynomial& other) {
    *this = *this + other;
    return *this;
}

Polynomial& Polynomial::operator-=(const Polynomial& other) {
    *this = *this - other;
    return *this;
}

Polynomial& Polynomial::operator*=(const Polynomial& other) {
    *this = *this * other;
    return *this;
}

int main() {
    Polynomial p1({Term(2, 4), Term(-3, 2), Term(5, 0)});
    Polynomial p2({Term(3, 3), Term(1, 2), Term(2, 0)});

    Polynomial sum = p1 + p2;
    std::cout << "Sum: ";
    sum.print();

    Polynomial diff = p1 - p2;
    std::cout << "Difference: ";
    diff.print();

    Polynomial prod = p1 * p2;
    std::cout << "Product: ";
    prod.print();

    return 0;
}

```

```
Sum: 2x^4 + 3x^3 -3x^2 + 1x^2 + 5x^0 + 2x^0
Difference: 2x^4 -3x^3 -3x^2 -1x^2 + 5x^0 -2x^0
Product: 6x^7 + 2x^6 + 4x^4 -9x^5 -3x^4 -6x^2 + 15x^3 + 5x^2 + 10x^0
```