

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

Question 1

```
main.cpp
1 #include <iostream>
2 #include <stdexcept>
3
4 class DoubleSubscriptedArray {
5 public:
6     explicit DoubleSubscriptedArray(int = 10, int = 10); // default constructor
7     DoubleSubscriptedArray(const DoubleSubscriptedArray&); // copy constructor
8     ~DoubleSubscriptedArray(); // destructor
9     DoubleSubscriptedArray& operator=(const DoubleSubscriptedArray&); // assignment operator
10    bool operator==(const DoubleSubscriptedArray&) const; // equality operator
11    bool operator!=(const DoubleSubscriptedArray& right) const {
12        return !(*this == right);
13    }
14
15    int& operator()(int, int); // returns modifiable lvalue
16    int operator()(int, int) const; // returns rvalue
17
18 private:
19     size_t rows; // number of rows
20     size_t columns; // number of columns
21     int* ptr; // pointer to first element of pointer-based array
22 };
23
24 // constructor
25 DoubleSubscriptedArray::DoubleSubscriptedArray(int rows, int columns)
26 : rows(rows > 0 ? rows : throw std::invalid_argument("Number of rows must be greater than 0")),
27   columns(columns > 0 ? columns : throw std::invalid_argument("Number of columns must be greater than 0")),
28   ptr(new int[rows * columns]) {
29     for (size_t i = 0; i < rows * columns; ++i)
30         ptr[i] = 0; // initialize pointer-based array elements to 0
31 }
32
33 // copy constructor
34 DoubleSubscriptedArray::DoubleSubscriptedArray(const DoubleSubscriptedArray& arrayToCopy)
35 : rows(arrayToCopy.rows), columns(arrayToCopy.columns), ptr(new int[arrayToCopy.rows * arrayToCopy.columns]) {
36     for (size_t i = 0; i < rows * columns; ++i)
37         ptr[i] = arrayToCopy.ptr[i]; // copy into object
38 }
39
40 // destructor
41 DoubleSubscriptedArray::~DoubleSubscriptedArray() {
42     delete[] ptr; // release pointer-based array space
43 }
```

main.cpp

```
43 }
44
45 // assignment operator
46 DoubleSubscriptedArray& DoubleSubscriptedArray::operator=(const DoubleSubscriptedArray& right) {
47     if (&right != this) { // avoid self-assignment
48         if (rows != right.rows || columns != right.columns) { // for Arrays of different sizes
49             delete[] ptr; // deallocate original left-side Array
50             rows = right.rows; // resize this object
51             columns = right.columns;
52             ptr = new int[rows * columns]; // create space for Array copy
53         }
54
55         for (size_t i = 0; i < rows * columns; ++i)
56             ptr[i] = right.ptr[i]; // copy array into object
57     }
58
59     return *this; // enables x = y = z
60 }
61
62 // equality operator
63 bool DoubleSubscriptedArray::operator==(const DoubleSubscriptedArray& right) const {
64     if (rows != right.rows || columns != right.columns)
65         return false; // arrays of different sizes
66
67     for (size_t i = 0; i < rows * columns; ++i)
68         if (ptr[i] != right.ptr[i])
69             return false; // Array contents are not equal
70
71     return true; // Arrays are equal
72 }
73
74 // overloaded subscript operator for non-const Arrays
75 int& DoubleSubscriptedArray::operator()(int row, int column) {
76     if (row < 0 || row >= rows || column < 0 || column >= columns)
77         throw std::out_of_range("Subscript out of range");
78
79     return ptr[row * columns + column]; // reference return
80 }
81
82 // overloaded subscript operator for const Arrays
83 int DoubleSubscriptedArray::operator()(int row, int column) const {
84     if (row < 0 || row >= rows || column < 0 || column >= columns)
```

```

main.cpp
69         return false; // Array contents are not equal
70
71     return true; // Arrays are equal
72 }
73
74 // overloaded subscript operator for non-const Arrays
75 int& DoubleSubscriptedArray::operator()(int row, int column) {
76     if (row < 0 || row >= rows || column < 0 || column >= columns)
77         throw std::out_of_range("Subscript out of range");
78
79     return ptr[row * columns + column]; // reference return
80 }
81
82 // overloaded subscript operator for const Arrays
83 int DoubleSubscriptedArray::operator()(int row, int column) const {
84     if (row < 0 || row >= rows || column < 0 || column >= columns)
85         throw std::out_of_range("Subscript out of range");
86
87     return ptr[row * columns + column]; // returns copy of this element
88 }
89
90 int main() {
91     DoubleSubscriptedArray array(3, 4);
92
93     // Initialize array elements
94     for (int i = 0; i < 3; ++i) {
95         for (int j = 0; j < 4; ++j) {
96             array(i, j) = i + j; // Accessing elements using () operator
97         }
98     }
99
100    // Display array elements
101    for (int i = 0; i < 3; ++i) {
102        for (int j = 0; j < 4; ++j) {
103            std::cout << array(i, j) << " "; // Accessing elements using () operator
104        }
105        std::cout << std::endl;
106    }
107
108    return 0;
109 }

```



```

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

```

Question 2

main.cpp

[Download Code](#)

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 class Term {
6 public:
7     Term(double coef = 0.0, int exp = 0) : coefficient(coef), exponent(exp) {}
8     double getCoefficient() const { return coefficient; }
9     int getExponent() const { return exponent; }
10    void setCoefficient(double coef) { coefficient = coef; }
11    void setExponent(int exp) { exponent = exp; }
12 private:
13     double coefficient;
14     int exponent;
15 };
16
17 class Polynomial {
18 public:
19     Polynomial() {}
20     Polynomial(const std::vector<Term>& terms) : terms(terms) {}
21     void setTerms(const std::vector<Term>& t) { terms = t; }
22     std::vector<Term> getTerms() const { return terms; }
23     void print() const;
24
25     Polynomial operator+(const Polynomial& other) const;
26     Polynomial operator-(const Polynomial& other) const;
27     Polynomial operator*(const Polynomial& other) const;
28     Polynomial& operator=(const Polynomial& other);
29     Polynomial& operator+=(const Polynomial& other);
30     Polynomial& operator-=(const Polynomial& other);
31     Polynomial& operator*=(const Polynomial& other);
32
33 private:
34     std::vector<Term> terms;
35 };
36
37 void Polynomial::print() const {
38     bool firstTerm = true;
39     for (const auto& term : terms) {
40         if (term.getCoefficient() != 0) {
41             if (!firstTerm && term.getCoefficient() > 0)
42                 std::cout << "+ ";
```

```

main.cpp
43         std::cout << term.getCoefficient() << "x^" << term.getExponent() << " ";
44         firstTerm = false;
45     }
46 }
47 std::cout << std::endl;
48 }
49
50 Polynomial Polynomial::operator+(const Polynomial& other) const {
51     Polynomial result;
52     std::vector<Term> resultTerms;
53     std::merge(terms.begin(), terms.end(), other.terms.begin(), other.terms.end(), std::back_inserter(resultTerms),
54               [](const Term& a, const Term& b) { return a.getExponent() > b.getExponent(); });
55     result.setTerms(resultTerms);
56     return result;
57 }
58
59 Polynomial Polynomial::operator-(const Polynomial& other) const {
60     Polynomial result;
61     std::vector<Term> negatedTerms;
62     for (const auto& term : other.getTerms()) {
63         negatedTerms.push_back(Term(-term.getCoefficient(), term.getExponent()));
64     }
65     std::vector<Term> resultTerms;
66     std::merge(terms.begin(), terms.end(), negatedTerms.begin(), negatedTerms.end(), std::back_inserter(resultTerms),
67               [](const Term& a, const Term& b) { return a.getExponent() > b.getExponent(); });
68     result.setTerms(resultTerms);
69     return result;
70 }
71
72 Polynomial Polynomial::operator*(const Polynomial& other) const {
73     Polynomial result;
74     std::vector<Term> resultTerms;
75     for (const auto& term1 : terms) {
76         for (const auto& term2 : other.terms) {
77             double coef = term1.getCoefficient() * term2.getCoefficient();
78             int exp = term1.getExponent() + term2.getExponent();
79             resultTerms.push_back(Term(coef, exp));
80         }
81     }
82     result.setTerms(resultTerms);
83     return result;
84 }

```

main.cpp

```
85
86 Polynomial& Polynomial::operator=(const Polynomial& other) {
87     if (this != &other) {
88         terms = other.terms;
89     }
90     return *this;
91 }
92
93 Polynomial& Polynomial::operator+=(const Polynomial& other) {
94     *this = *this + other;
95     return *this;
96 }
97
98 Polynomial& Polynomial::operator-=(const Polynomial& other) {
99     *this = *this - other;
100    return *this;
101 }
102
103 Polynomial& Polynomial::operator*=(const Polynomial& other) {
104     *this = *this * other;
105     return *this;
106 }
107
108 int main() {
109     Polynomial p1({Term(2, 4), Term(-3, 2), Term(5, 0)});
110     Polynomial p2({Term(3, 3), Term(1, 2), Term(2, 0)});
111
112     Polynomial sum = p1 + p2;
113     std::cout << "Sum: ";
114     sum.print();
115
116     Polynomial diff = p1 - p2;
117     std::cout << "Difference: ";
118     diff.print();
119
120     Polynomial prod = p1 * p2;
121     std::cout << "Product: ";
122     prod.print();
123
124     return 0;
125 }
126
```

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: $6x^7 + 2x^6 + 4x^4 - 9x^5 - 3x^4 - 6x^2 + 15x^3 + 5x^2 + 10x^0$