operator overloading-2(13-08-24)

Author: ThanhTH10
Date: 13/08/2024

1.wap to overload multiplication operator in matrix multiplication the overloading function should be a member function.

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

class Matrix
{
    int rows;
    int cols;
    int **data;
public:
    Matrix(int _rows, int _cols) : rows(_rows), cols(_cols)
    {
        data = new int *[rows];
        for (int i = 0; i < rows; i++)
        {
            data[i] = new int[cols];
        }
    }
    ~Matrix()
    {
        for (int i = 0; i < rows; i++)
        {
            delete[] data[i];
        }
        delete[] data;
    }
    void setElement(int r, int c, int val)
    {
        data[r][c] = val;
    }
    int getElement(int r, int c)
    {
        return data[r][c];
    }
    Matrix operator*(const Matrix &other);
    void display()
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                cout << data[i][j] << "\t";
            }
            cout << endl;
        }
    }
};
Matrix Matrix::operator*(const Matrix &other)
{
    if (cols != other.rows)
    {
        cerr << "Matrix dimensions are incompatible for multiplication." << endl;
```

```
        exit(1);
    }
    Matrix result(rows, other.cols);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < other.cols; j++)
        {
            int sum = 0;
            for (int k = 0; k < cols; k++)
            {
                sum += data[i][k] * other.data[k][j];
            }
            result.data[i][j] = sum;
        }
    }
    return result;
}
int main(int argc, char const *argv[])
{
    Matrix m1(2, 3);
    m1.setElement(0, 0, 1);
    m1.setElement(0, 1, 2);
    m1.setElement(0, 2, 3);
    m1.setElement(1, 0, 4);
    m1.setElement(1, 1, 5);
    m1.setElement(1, 2, 6);
    Matrix m2(3, 2);
    m2.setElement(0, 0, 7);
    m2.setElement(0, 1, 8);
    m2.setElement(1, 0, 9);
    m2.setElement(1, 1, 10);
    m2.setElement(2, 0, 11);
    m2.setElement(2, 1, 12);
    Matrix result = m1 * m2;
    std::cout << "Matrix 1:" << std::endl;
    m1.display();
    std::cout << "Matrix 2:" << std::endl;
    m2.display();
    std::cout << "Result:" << std::endl;
    result.display();
    return 0;
}
```

Output:

```
Matrix 1:
1       2       3
4       5       6
Matrix 2:
7       8
9       10
11      12
Result:
58      64
139     154
```

2. wap to overload addition operator in matrix addition the overloading function should be a friend function

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

class Matrix
{
```

```
    int rows;
    int cols;
    int **data;
public:
    Matrix(int _rows, int _cols) : rows(_rows), cols(_cols)
    {
        data = new int *[rows];
        for (int i = 0; i < rows; i++)
        {
            data[i] = new int[cols];
        }
    }
    ~Matrix()
    {
        for (int i = 0; i < rows; i++)
        {
            delete[] data[i];
        }
        delete[] data;
    }
    void setElement(int r, int c, int val)
    {
        data[r][c] = val;
    }
    int getElement(int r, int c)
    {
        return data[r][c];
    }
    friend Matrix operator+(const Matrix &m1, const Matrix &m2);
    void display()
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                cout << data[i][j] << "\t";
            }
            cout << endl;
        }
    }
};
Matrix operator+(const Matrix &m1, const Matrix &m2)
{
    if (m1.rows != m2.rows || m1.cols != m2.cols)
    {
        cerr << "Matrix dimensions are incompatible for addition." << std::endl;
        exit(1);
    }
    Matrix result(m1.rows, m1.cols);
    for (int i = 0; i < m1.rows; i++)
    {
        for (int j = 0; j < m1.cols; j++)
        {
            result.data[i][j] = m1.data[i][j] + m2.data[i][j];
        }
    }
    return result;
}
int main(int argc, char const *argv[])
{
    Matrix m1(2, 2);
```

```
    m1.setElement(0, 0, 1);
    m1.setElement(0, 1, 2);
    m1.setElement(1, 0, 3);
    m1.setElement(1, 1, 4);
    Matrix m2(2, 2);
    m2.setElement(0, 0, 5);
    m2.setElement(0, 1, 6);
    m2.setElement(1, 0, 7);
    m2.setElement(1, 1, 8);
    Matrix result = m1 + m2;
    std::cout << "Matrix 1:" << std::endl;
    m1.display();
    std::cout << "Matrix 2:" << std::endl;
    m2.display();
    std::cout << "Result:" << std::endl;
    result.display();
    return 0;
}
```

Output:

```
Matrix 1:
1       2
3       4
Matrix 2:
5       6
7       8
Result:
6       8
10      12
```

3. wap to implement a class student which is having members as name, percentage and age. The comparision operator using friend function should be overloaded in such a way that it should compare objects based on percenatages .if percentages are same then it should comapare age.

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

class Student
{
    string name;
    double percentage;
    int age;
public:
    Student(string _name, double _percentage, int _age) : name(_name), percentage(_percentage),
age(_age) {}
    void display()
    {
        cout << "Name: " << name << "\t|Percentage: " << percentage << "\t\t|Age: " << age <<
endl;
    }
    friend bool operator==(const Student &s1, const Student &s2);
    friend bool operator!=(const Student &s1, const Student &s2);
    friend bool operator<(const Student &s1, const Student &s2);
    friend bool operator<=(const Student &s1, const Student &s2);
    friend bool operator>(const Student &s1, const Student &s2);
    friend bool operator>=(const Student &s1, const Student &s2);
};
bool operator==(const Student &s1, const Student &s2)
```

```
{
    return (s1.percentage == s2.percentage) && (s1.age == s2.age);
}
bool operator!=(const Student &s1, const Student &s2)
{
    return !(s1 == s2);
}
bool operator<(const Student &s1, const Student &s2)
{
    if (s1.percentage == s2.percentage)
    {
        return s1.age < s2.age;
    }
    else
    {
        return s1.percentage < s2.percentage;
    }
}
bool operator<=(const Student &s1, const Student &s2)
{
    return (s1 < s2) || (s1 == s2);
}
bool operator>(const Student &s1, const Student &s2)
{
    return !(s1 <= s2);
}
bool operator>=(const Student &s1, const Student &s2)
{
    return !(s1 < s2);
}
int main(int argc, char const *argv[])
{
    Student s1("John", 85.0, 20);
    Student s2("Jane", 85.0, 21);
    Student s3("Bob", 90.0, 22);
    std::cout << "Comparing students:" << std::endl;
    if (s1 == s2)
    {
        std::cout << "s1 and s2 are equal" << std::endl;
    }
    else
    {
        std::cout << "s1 and s2 are not equal" << std::endl;
    }
    if (s1 < s2)
    {
        std::cout << "s1 is less than s2" << std::endl;
    }
    else
    {
        std::cout << "s1 is not less than s2" << std::endl;
    }
    if (s1 > s3)
    {
        std::cout << "s1 is greater than s3" << std::endl;
    }
    else
    {
        std::cout << "s1 is not greater than s3" << std::endl;
    }
    std::cout << "Displaying students:" << std::endl;
```

```
    s1.display();
    s2.display();
    s3.display();
    return 0;
}
```

Output:

```
Comparing students:
s1 and s2 are not equal
s1 is less than s2
s1 is not greater than s3
Displaying students:
Name: John      |Percentage: 85         |Age: 20
Name: Jane      |Percentage: 85         |Age: 21
Name: Bob       |Percentage: 90         |Age: 22
```