

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский университет ИТМО

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №2
По дисциплине «Программирование»
Лабораторная работа. Классы. Перегрузка операторов

Выполнил Кудашев И.Э
(Фамилия Имя Отчество)

Проверил Повышев В.В
(Фамилия Имя Отчество)

Санкт-Петербург, 2021г

УСЛОВИЕ ЛАБОРАТОРНОЙ

Спроектировать и реализовать класс для описания сущности многочлен (полином), раздела математики - Алгебра.

Реализовать конструктор(ы), конструктор копирования, деструктор, а также следующие операторы:

1. =
2. ==, !=
3. +, - (унарный и бинарный), +=, -=
4. *, / (на число), *=, /=
5. <<, >>
6. [] (для получения коэффициента i-го члена)

КОД

```
#include <iostream>
#include <vector>
#include <algorithm> // sort

using namespace std;

class ElementOfPolynomial {
public:
    ElementOfPolynomial() = default;

    ElementOfPolynomial(int multiplier_, int pow_) : multiplier(multiplier_),
pow(pow_) {}

    void setMultiplier(int multiplier_) {
        multiplier = multiplier_;
    }

    void setPow(int pow_) {
        pow = pow_;
    }

    void setElementOfPolynomial(int multiplier_, int pow_) {
        multiplier = multiplier_;
        pow = pow_;
    }

    int getMultiplier() const {
        return multiplier;
    }
}
```

```
int getPow() const {  
    return pow;  
}
```

```
void getElement() {  
    description();  
}
```

```
~ElementOfPolynomial() = default;;
```

protected:

```
void description() const {  
    if (pow != 0) {  
        if (multiplier != 0) {  
            if (pow == 1 && abs(multiplier) == 1)  
                cout << '+' << 'x';  
            else if (pow == 1 && abs(multiplier) != 1)  
                cout << ' ' << abs(multiplier) << 'x';  
            else if (pow != 1 && abs(multiplier) == 1)  
                cout << '+' << 'x^';  
            else if (pow != 1 && abs(multiplier) != 1)  
                cout << ' ' << abs(multiplier) << 'x^';  
        }  
    } else if (pow < 0) {  
        if (multiplier != 0)  
            if (pow == -1 && abs(multiplier) == -1)  
                cout << ' ' << abs(multiplier) << '/' << 'x';  
            else if (pow == -1 && abs(multiplier) != -1)  
                cout << '+' << abs(multiplier) << '/' << 'x';  
            else if (pow != -1 && abs(multiplier) == -1)
```

```

        cout << ' ' << abs(multiplier) << '/' << 'x^' << abs(pow);
    else if (pow != -1 && abs(multiplier) != -1)
        cout << '+' << abs(multiplier) << '/' << 'x' << abs(pow);
} else if (pow == 0 && abs(multiplier) != 0) {
    cout << ' ' << abs(multiplier);
} else {
    cout << 0 << endl;
}

}

```

private:

```

    int multiplier = 0;
    int pow = 0;
};

```

class Polynomial {
public:

```

    explicit Polynomial(vector<ElementOfPolynomial> &arr) {
        if (polynomialArr.empty()) {
            this->setPolynomial(arr);
        } else {
//            throw invalid_argument('Polynomial cant be created');
            cout << 'Error' << endl;
        }
    }

    Polynomial(const Polynomial &arr) {
        for (auto i : arr.polynomialArr) {
            this->polynomialArr.push_back(i);
        }
    }
}

```

```
Polynomial() = default;
```

```
~Polynomial() = default;
```

```
friend Polynomial operator+(Polynomial &arr1, Polynomial &arr2) {  
    if (!arr1.polynomialArr.empty() || !arr2.polynomialArr.empty()) {  
        vector<ElementOfPolynomial> arr3;  
        mergeArrs((vector<struct ElementOfPolynomial> &) arr1,  
(vector<struct ElementOfPolynomial> &) arr2, arr3);  
        return Polynomial(arr3);  
    }  
    return Polynomial();  
}
```

```
Polynomial &operator=(const Polynomial &arr) {  
    polynomialArr.resize(arr.polynomialArr.size());  
    polynomialArr = arr.polynomialArr;  
    this->setPolynomial(this->polynomialArr);  
    return *this;  
}
```

```
bool operator!=(const Polynomial &arr) {  
    bool result = (*this == arr);  
    return !result;  
}
```

```
bool operator==(const Polynomial &arr) {  
    bool result = false;  
    if (polynomialArr.size() == arr.polynomialArr.size()) {  
        for (int i = 0; i < arr.polynomialArr.size(); i++) {  
            if (polynomialArr[i].getMultiplier() ==  
arr.polynomialArr[i].getMultiplier())
```

```

        && polynomialArr[i].getPow() == arr.polynomialArr[i].getPow()) {
            result = true;
        }
    }
}
return result;
}

```

```

Polynomial operator-(Polynomial &arr) {
    for (auto &i : arr.polynomialArr) {
        i.setMultiplier((-1) * i.getMultiplier());
    }
    return arr;
}

```

```

friend Polynomial operator-(const Polynomial &arr1, const Polynomial &arr2)
{
    if (!arr1.polynomialArr.empty() || !arr2.polynomialArr.empty()) {
        vector<ElementOfPolynomial> a, b;
        b = arr2.polynomialArr;
        for (auto &i : b) {
            i.setMultiplier((-1) * i.getMultiplier());
        }
        mergeArrs((vector<struct ElementOfPolynomial> &) arr1.polynomialArr,
b, a);

        return Polynomial(a);
    } else {
        return Polynomial();
    }
}

```

```

friend Polynomial &operator++(Polynomial &arr) {
    arr.newPolynomialElem(1, 0);
}

```

```

        return arr;
    }

    friend Polynomial &operator--(Polynomial &arr) {
        arr.newPolynomialElem(-1, 0);
        return arr;
    }

    Polynomial &operator+=(Polynomial &arr) {
        arr = arr + *this;
        return arr;
    }

    Polynomial &operator-=(Polynomial &arr) {
        arr = arr - *this;
        return arr;
    }

    friend Polynomial &operator/(Polynomial &arr1, Polynomial &arr2) {
        polynomialDivision(arr1.polynomialArr, arr2.polynomialArr);
        arr1.setPolynomial(arr1.polynomialArr);
        return arr1;
    }

    friend Polynomial &operator*(Polynomial &arr1, Polynomial &arr2) {
        polynomialMultiplication(arr1.polynomialArr, arr2.polynomialArr);
        arr1.setPolynomial(arr1.polynomialArr);
        return arr1;
    }

    Polynomial &operator*=(Polynomial &arr) {
        arr = *this * arr;
    }

```



```

    return arr;
}

Polynomial &operator/=(Polynomial &arr) {
    arr = *this * arr;
    return arr;
}

ElementOfPolynomial &operator[](int i) {
    return this->polynomialArr[i];
}

friend istream &operator>>(istream &input, Polynomial &arr) {
    cout << 'Please, enter the Polynomial Index' << endl;
    int i = 0;
    while (i != arr.polynomialArr.size()) {
        cin >> i;
    }
    int multiplier, pow;
    cout << 'Please, enter the Polynomial multiplier and a pow' << endl;
    input >> multiplier >> pow;
    if (multiplier && pow) {
        arr.polynomialArr[i] = ElementOfPolynomial(multiplier, pow);
        arr.setPolynomial(arr.polynomialArr);
    } else {
        cout << 'Something wrong' << endl;
    }
    return input;
}

friend ostream &operator<<(ostream &output, Polynomial &arr) {
    cout << 'Please, enter the Polynomial Index' << endl;

```

```

    int i = 0;
    while (i != arr.polynomialArr.size()) {
        cin >> i;
    }
    output << arr.polynomialArr[i].getMultiplier() <<
arr.polynomialArr[i].getPow() << endl;
    return output;
}

```

protected:

```

static void countEqualPow(vector<ElementOfPolynomial> &arr) {
    if (!arr.empty()) {
        for (int i = 0; i < arr.size(); i++) {
            for (int j = i + 1; j < arr.size(); j++) {
                if (arr[i].getPow() == arr[j].getPow()) {
                    arr[i].setMultiplier(arr[i].getMultiplier() + arr[j].getMultiplier());
                    arr.erase(arr.begin() + j);
                    j--;
                }
            }
        }
    }
}

```

} //количество с одинаковыми степенями

```

static void sortByPow(vector<ElementOfPolynomial> &arr) {
    if (arr.size() > 2) {
        for (int i = 0; i < arr.size() - 1; i++) {
            for (int j = i + 1; j < arr.size(); j++) {
                if (arr[i].getPow() > arr[j].getPow())
                    swap(arr[i], arr[j]);
            }
        }
    }
}

```

```

    }
} else if (arr.size() == 2) {
    if (arr[0].getPow() < arr[1].getPow())
        swap(arr[0], arr[1]);
}
} //сортировка элементов по убыванию степени

static void invMultiplier(vector<ElementOfPolynomial> &arr) {
    for (auto &i : arr) {
        i.setMultiplier((-1) * i.getMultiplier());
    }
} // все элементы домножаем на -1

static void mergeArrs(vector<ElementOfPolynomial> &arr1,
vector<ElementOfPolynomial> &arr2,
vector<ElementOfPolynomial> &mergedArr) {
    mergedArr.reserve(arr1.size() + arr2.size());
    mergedArr.insert(mergedArr.end(), arr1.begin(), arr1.end());
    mergedArr.insert(mergedArr.end(), arr2.begin(), arr2.end());
} //слияние векторов

static void polynomialMultiplication(vector<ElementOfPolynomial> &arr1,
vector<ElementOfPolynomial> &arr2) {
    vector<ElementOfPolynomial> multiplicationArr;
    for (auto &i : arr1) {
        for (auto &j : arr2) {
            ElementOfPolynomial value = ElementOfPolynomial(i.getMultiplier()
* j.getMultiplier(),
i.getPow() + j.getPow());
            multiplicationArr.push_back(value);
        }
    }
}

```

```

    arr1 = multiplicationArr;
} // перемножение коэффициентов полинома

static void polynomialDivision(vector<ElementOfPolynomial> &arr1,
vector<ElementOfPolynomial> &arr2) {
    vector<ElementOfPolynomial> divisionArr;
    int divisionC = 0;
    int maxPow = 0;

    for (int i = 0; i < arr2.size(); i++) {
        if (arr2[i].getPow() == 0) {
            divisionC = -1 * arr2[i].getMultiplier();
        }
        if (arr2[i].getPow() != maxPow) {
            divisionC = arr1[i].getMultiplier();
        }
    }
    if (divisionC != 0) {
        divisionArr.emplace_back(arr1[0].getMultiplier(), arr1[0].getPow() - 1);
        int C = arr1[0].getMultiplier();
        for (int i = 1; i < arr1.size(); i++) {
            C = C * divisionC + arr1[i].getMultiplier();
            divisionArr.emplace_back(ElementOfPolynomial(C, arr1[i].getPow()
- 1));
        }

        divisionArr[arr1.size() - 1].setPow(-1 * arr2[0].getPow());
        divisionArr.emplace_back(divisionArr[arr1.size() - 1].getMultiplier() /
            arr2[arr2.size() - 1].getMultiplier(), 0);
        arr1 = divisionArr;
    } else {

```

```

        if (arr2[0].getPow() != 0) {
            for (auto &i : arr1) {
                i.setMultiplier(i.getMultiplier() / arr2[0].getMultiplier());
                i.setPow(i.getPow() - arr2[0].getPow());
            }
        } else {
            cout << "Division failed" << endl;
            arr1.resize(0);
        }
    }
}

```

```

void setPolynomial(vector<ElementOfPolynomial> &arr) {
    polynomialArr = arr;
    countEqualPow(this->polynomialArr);
    sortByPow(this->polynomialArr);
}

```

```

void newPolynomialElem(int multiplier, int pow) {
    ElementOfPolynomial element;
    element.setElementOfPolynomial(multiplier, pow);
    this->setPolynomial(this->polynomialArr);
}

```

```

static void inversion(vector<ElementOfPolynomial> &arr) {
    for (auto &i : arr) {
        i.setMultiplier((-1) * i.getMultiplier());
    }
}

```

private:

```

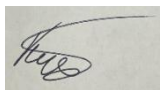
vector<ElementOfPolynomial> polynomialArr;
};

int main() {
    ElementOfPolynomial p11(1, 1), p12(2, 2), p13(3, 3);
    ElementOfPolynomial p21(3, 1), p22(2, 2), p23(1, 3);
    ElementOfPolynomial p31(0, 1), p32(0, 2), p33(0, 3);
    vector<ElementOfPolynomial> polynomialArr1(3);
    vector<ElementOfPolynomial> polynomialArr2(3);
    vector<ElementOfPolynomial> polynomialArr3(3);
    polynomialArr1 = {p11, p12, p13};
    polynomialArr2 = {p21, p22, p23};
    polynomialArr3 = {p31, p32, p33};
    Polynomial pol1(polynomialArr1);
    Polynomial pol2(polynomialArr2);
    Polynomial pol3(polynomialArr3);

    pol1 = pol2;
    if (pol1 == pol2) {
        cout << 'pol1 == pol2' << endl;
    }
    if (pol1 != pol3) {
        cout << 'pol1 != pol3' << endl;
    }
    pol3 = pol1 + pol2;
    pol3 = pol1 - pol2;
    pol1 += pol2;
    pol1 -= pol3;
}

```

Кудашев И.Э

A small, square, light-colored image containing a handwritten signature in dark ink. The signature is stylized and appears to be the initials or full name of the person mentioned in the adjacent text block.