

Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Полиномы

Выполнил:
студент ф-та ИИТММ гр. 381908-01

Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение	4
Постановка задачи	5
Руководство пользователя	6
Руководство программиста.....	7
Описание структуры программы	7
Описание структур данных	7
Описание алгоритмов.....	9
Эксперименты	10
Заключение.....	12
Литература	13
Приложения	14
Приложение 1 Monom.h	14
Приложение 2 Monom.cpp	15
Приложение 3 Polinom.h	18
Приложение 4 Polinom.cpp	19
Приложение 5 PolinomIterator.h	23
Приложение 6 PolinomIterator.cpp	24
Приложение 7 Main.cpp	25

Введение

Полином - это многочлен от N переменных, в себе он содержит мономы, которые являются одночленами. Разработка структуры данных полинома важна, так как она позволит работать с многочленами – совершать операции обработки с ними.

Постановка задачи

Цель данной работы - разработка структуры данных для хранения многочленов (полиномы) и одночленов (мономы), полином содержит в себе список мономов.

Выполнение работы предполагает решение следующих задач:

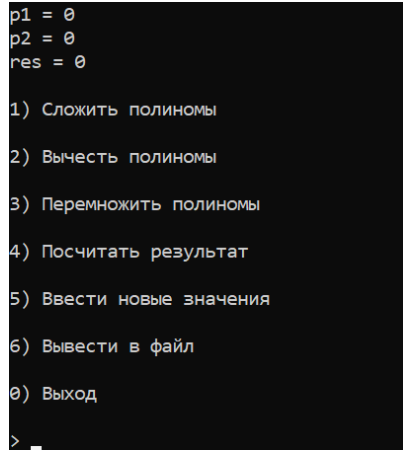
1. Реализация структуры Monom.
2. Реализация класса полиномов Polinom.
3. Реализация итератора для полинома PolinomIterator.
4. Реализация методов для ввода/вывода структуры данных в файл
5. Публикация исходных кодов в личном репозитории на GitHub.

Руководство пользователя

Пользователю нужно запустить файл Polinom_manom.exe.

Откроется консольное приложение для тестирования полиномов.

Программа покажет меню для удобного взаимодействия с программой.



```
p1 = 0
p2 = 0
res = 0

1) Сложить полиномы
2) Вычитать полиномы
3) Перемножить полиномы
4) Посчитать результат
5) Ввести новые значения
6) Вывести в файл
0) Выход
>
```

- 1) Складывает полином $p1$ и $p2$
- 2) Вычитает полином $p1$ из $p2$
- 3) Перемножает полиномы $p1$ и $p2$
- 4) Ввести значение коэффициентов a , b , c и вывести конечный ответ
- 5) Изменить полиномы $p1$ и $p2$
- 6) Вывести полином в файл
- 0) Закрытие консоли

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение Polinom_manom
- Статическая библиотека Monom:
 - Monom.h – описание одночлена для полинома
- Статическая библиотека Polinom:
 - Polinom.h – описание многочлена состоящий из одночленов.
- Статическая библиотека PolinomIterator:
 - PolinomIterator.h – описание класса итератора для полинома
- Приложение main:
 - main.cpp – тестирование работы программы

Описание структур данных

Класс Monom:

- 1) Monom() – конструктор по умолчанию;
- 2) Monom(int coef, int A, int B, int C) – конструктор с параметрами;
- 3) Monom(const Monom& m) – конструктор с параметрами;
- 4) int Get_A()const – получить значение A;
- 5) int Get_B()const – получить значение B;
- 6) int Get_C()const – получить значение C;
- 7) int Get_coef()const – получить значение коэффициента;
- 8) bool operator==(const Monom& m)const – перегрузка сравнения для мономов;
- 9) bool operator!=(const Monom& m)const – перегрузка неравенства для мономов;
- 10) bool operator>(const Monom& m)const – перегрузка знака больше для мономов;
- 11) bool operator<(const Monom& m)const – перегрузка знака меньше для мономов;
- 12) bool operator<=(const Monom& m)const – перегрузка знака меньше или равно для мономов;
- 13) bool operator>=(const Monom& m)const – перегрузка знака больше или равно для мономов;
- 14) Monom operator+(const Monom& m)const – сложение двух мономов;
- 15) Monom& operator+=(const Monom& m) – сложение двух мономов;
- 16) Monom operator-(const Monom& m)const – вычитание двух мономов;
- 17) Monom operator-()const – сделать моном отрицательным;
- 18) Monom& operator-=(const Monom& m) – вычитание двух мономов;
- 19) Monom operator*(const Monom& m)const – перемножение двух мономов;
- 20) Monom& operator*=(const Monom& m) – перемножение двух мономов;
- 21) Monom operator*(const int& num)const – умножение монома на число;
- 22) Monom& operator*=(const int& num) – умножение монома на число;
- 23) int Result(int a, int b, int c) – получить результат выражения при заданных коэффициентах;
- 24) std::string ConvertString()const – получить привычную запись многочлена из мономов;

Класс Polinom:

1. Polinom() - конструктор;
2. Polinom(const std::vector<Monom>& m) - конструктор;
3. Polinom(const Polinom& p) - конструктор;
4. Polinom& operator+=(const Monom& m) - прибавить моном;
5. Polinom operator+(const Monom& m) const - прибавить моном от полинома;
6. Polinom operator-(const Monom& m) const - отнять моном от полинома;
7. Polinom& operator-=(const Monom& m) - отнять моном от полинома;
8. Polinom operator*(const Monom& m) - умножить полином на моном;
9. Polinom& operator*=(const Monom& m) - умножить полином на моном;
10. Polinom operator+(Polinom& p) - сложить полиномы;
11. Polinom& operator+=(Polinom& p) - сложить полиномы;
12. Polinom operator-(Polinom& p) - вычесть полином;
13. Polinom& operator-=(Polinom& p) - вычесть полином;
14. Polinom operator*(Polinom& p) - перемножить полиномы;
15. Polinom& operator*=(Polinom& p) - перемножить полиномы;
16. Polinom operator*(const int& num) const - умножить полином на число;
17. Polinom& operator*=(const int& num) - умножить полином на число;
18. Polinom operator-(const int& num) const - отнять от полинома число;
19. Polinom& operator-=(const int& num) - отнять от полинома число;
20. Polinom operator+(const int& num) const - прибавить число к полиному;
21. Polinom& operator+=(const int& num) - прибавить число к полиному;
22. bool operator==(const Polinom& p) const - сравнить два полинома;
23. Polinom& operator=(const Polinom& p) - присвоить полином полиному;
24. int Result(int x, int y, int z) - получить ответ для многочлена;
25. void OutputFile(std::string path) - вывести полином в файл;
26. std::string to_str() - получить строку с многочленом;
27. void del_nulls() - удалить нули;
28. void set_null() - сделать многочлен нулевым;
29. void DeletePolinom() - удалить полином;

Класс PolinomIterator:

1. PolinomIterator() - конструктор;
2. PolinomIterator(std::list<Monom>& l) - конструктор;
3. void Set_list(std::list<Monom>& l) - установить список;
4. void init() - инициализация;
5. bool check_next() - проверка не кончился ли список;
6. void go_next() - установить указатель на следующий;
7. Monom& get_value() - получить значение по текущему указателю;
8. void del_cur() - удалить текущий указатель;
9. void insert_before(const Monom& m) - вставить моном;

Описание алгоритмов

Принцип работы полиномов

Инициализация полиномов происходит заполнением мономами, при вводе монома, вводится коэффициент и значение степени у переменных тут же высчитывается мощность множества по формуле $A * 20^2 + B * 20 + C$. При умножении двух полиномов, перемножаются мономы, то есть коэффициент с коэффициентом и степень со степенью, при сложении мономы записываются в порядке убывания степеней, при вычитании производится аналогичная операция, но с учётом минуса в операции.

Эксперименты

```
p1 = 0
p2 = 0
res = 0

1) Сложить полиномы
2) Вычесть полиномы
3) Перемножить полиномы
4) Посчитать результат
5) Ввести новые значения
6) Вывести в файл
0) Выход
```

После ввода новых значений (5 пункт меню).

Введите p1 и p2: Количество мономов: 2 Введите моном 1 Коэффициент: 2 Введите A: 2 Введите B: 3 Введите C: 1 Введите моном 2 Коэффициент: 3 Введите A: 1 Введите B: 2 Введите C: 1 Количество мономов: 1 Введите моном 1 Коэффициент: 2 Введите A: 3 Введите B: 4 Введите C: 5_	$p1 = 2 * a^2 * b^3 * c + 3 * a * b^2 * c$ $p2 = 2 * a^3 * b^4 * c^5$ $res = 0$ 1) Сложить полиномы 2) Вычесть полиномы 3) Перемножить полиномы 4) Посчитать результат 5) Ввести новые значения 6) Вывести в файл 0) Выход >
--	--

Сложим полиномы:

```
p1 = 2 * a^2 * b^3 * c + 3 * a * b^2 * c
p2 = 2 * a^3 * b^4 * c^5
res = 0

1) Сложить полиномы
2) Вычесть полиномы
3) Перемножить полиномы
4) Посчитать результат
5) Ввести новые значения
6) Вывести в файл
0) Выход

> 1
2 * a^2 * b^3 * c + 3 * a * b^2 * c + 2 * a^3 * b^4 * c^5 = 2 * a^3 * b^4 * c^5 + 2 * a^2 * b^3 * c + 3 * a * b^2 * c
Для продолжения нажмите любую клавишу . . .
```

Результат вычисления запишется в переменную "res":

```
p1 = 2 * a^2 * b^3 * c + 3 * a * b^2 * c  
p2 = 2 * a^3 * b^4 * c^5  
res = 2 * a^3 * b^4 * c^5 + 2 * a^2 * b^3 * c + 3 * a * b^2 * c
```

- 1) Сложить полиномы
 - 2) Вычесть полиномы
 - 3) Перемножить полиномы
 - 4) Посчитать результат
 - 5) Ввести новые значения
 - 6) Вывести в файл
 - 0) Выход
- >

Перемножим полиномы:

```
p1 = 2 * a^2 * b^3 * c + 3 * a * b^2 * c  
p2 = 2 * a^3 * b^4 * c^5  
res = 4 * a^5 * b^7 * c^6 + 6 * a^4 * b^6 * c^6
```

- 1) Сложить полиномы
 - 2) Вычесть полиномы
 - 3) Перемножить полиномы
 - 4) Посчитать результат
 - 5) Ввести новые значения
 - 6) Вывести в файл
 - 0) Выход
- > ☐

Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных полином.

Литература

1. <https://fb-ru.turbopages.org/fb.ru/s/article/449193/что-такое-полином-и-чем-он-полезен>
2. <https://ru.strephonsays.com/polynomial-and-vs-monomial-8816>
3. <https://ru.wikipedia.org/wiki/Многочлен>

Приложения

Приложение 1

Monom.h

```
class Monom
{
private:
    short int coef;
    int power;

    void Set_power(int A, int B, int C);
public:
    Monom();
    Monom(int coef, int A, int B, int C);
    Monom(const Monom& m);

    int Get_A()const;
    int Get_B()const;
    int Get_C()const;
    int Get_coef()const;

    bool operator==(const Monom& m)const;
    bool operator!=(const Monom& m)const;

    bool operator>(const Monom& m)const;
    bool operator<(const Monom& m)const;

    bool operator<=(const Monom& m)const;
    bool operator>=(const Monom& m)const;

    Monom operator+(const Monom& m)const;
    Monom& operator+=(const Monom& m);

    Monom operator-(const Monom& m)const;
    Monom operator-()const;
    Monom& operator--(const Monom& m);

    Monom operator*(const Monom& m)const;
    Monom& operator*=(const Monom& m);

    Monom operator*(const int& num)const;
    Monom& operator*=(const int& num);

    int Result(int a, int b, int c);
    std::string ConvertString()const;
};
```

Приложение 2

Monom.cpp

```
void Monom::Set_power(int A, int B, int C)
{
    power = A * POWER_MAX * POWER_MAX + B * POWER_MAX + C;
}

Monom::Monom()
{
    coef = 1;
    Set_power(1, 1, 1);
}

Monom::Monom(int coef, int A, int B, int C)
{
    if (A >= POWER_MAX || B >= POWER_MAX || C >= POWER_MAX)
    {
        throw std::exception("Too many numbers!");
    }
    this->coef = coef;
    Set_power(A, B, C);
}

int Monom::Get_A()const
{
    return power / POWER_MAX / POWER_MAX;
}

int Monom::Get_B()const
{
    return (power - Get_A() * POWER_MAX * POWER_MAX) / POWER_MAX;
}

int Monom::Get_C()const
{
    return power % POWER_MAX;
}

Monom::Monom(const Monom& m)
{
    coef = m.coef;
    power = m.power;
}

int Monom::Get_coef() const { return coef; }

bool Monom::operator==(const Monom& m) const { return power == m.power; }
bool Monom::operator!=(const Monom& m) const { return power != m.power; }
bool Monom::operator<=(const Monom& m) const { return power <= m.power; }
bool Monom::operator>=(const Monom& m) const { return power >= m.power; }
bool Monom::operator>(const Monom& m) const { return power > m.power; }
bool Monom::operator<(const Monom& m) const { return power < m.power; }

Monom Monom::operator+(const Monom& m) const
{
    if (*this != m)
    {
        throw std::logic_error("Monoms are different!");
    }
    Monom res(*this);
    res.coef += m.coef;
    return res;
}
```

```

Monom& Monom::operator+=(const Monom& m)
{
    if (*this != m)
    {
        throw std::exception("Monoms are different!");
    }
    coef += m.coef;
    return *this;
}

Monom Monom::operator-(const Monom& m) const
{
    if (*this != m)
    {
        throw std::exception("Monoms are different!");
    }
    Monom res(*this);
    res.coef += m.coef;
    return res;
}

Monom Monom::operator-() const
{
    Monom res(*this);
    res.coef *= -1;
    return res;
}

Monom& Monom::operator-=(const Monom& m)
{
    if (*this != m)
    {
        throw std::exception("Monoms are different!");
    }
    coef -= m.coef;
    return *this;
}

Monom Monom::operator*(const Monom& m) const
{
    return Monom(coef * m.coef, Get_A() + m.Get_A(), Get_B() + m.Get_B(), Get_C() + m.Get_C());
}

Monom& Monom::operator*=(const Monom& m)
{
    coef *= m.coef;
    Set_power(Get_A() + m.Get_A(), Get_B() + m.Get_B(), Get_C() + m.Get_C());
    return *this;
}

Monom Monom::operator*(const int& num) const
{
    Monom res(*this);
    res.coef *= num;
    return res;
}

Monom& Monom::operator*=(const int& num)
{
    coef *= num;
    return *this;
}

```



```

Monom Monom::operator*(const int& num) const
{
    Monom res(*this);
    res.coef *= num;
    return res;
}

Monom& Monom::operator*=(const int& num)
{
    coef *= num;
    return *this;
}

int Monom::Result(int x, int y, int z)
{
    return coef * pow(x, Get_A()) * pow(y, Get_B()) * pow(z, Get_C());
}

std::string Monom::ConvertString()const
{
    std::string res;
    const int A = Get_A();
    const int B = Get_B();
    const int C = Get_C();

    switch (coef)
    {
        case -1:
            res += "- ";
            break;
        case 0:
            res += std::to_string(coef);
            break;
        case 1:
            // +
            break;
        default:
            res += std::to_string(coef) + " * ";
            break;
    }

    if (A)
    {
        if (A == 1)
        {
            res += "a * ";
        }
        else
        {
            res += "a^" + std::to_string(A) + " * ";
        }
    }

    if (B)
    {
        if (B == 1)
        {
            res += "b * ";
        }
        else
        {
            res += "b^" + std::to_string(B) + " * ";
        }
    }

    if (C)
    {
        if (C == 1)
        {
            res += "c ";
        }
        else
        {
            res += "c^" + std::to_string(C) + " ";
        }
    }

    return res;
}

```

Приложение 3

Polinom.h

```
class Polinom
{
private:
    std::list<Monom> polinom;
    PolinomIterator it;

public:
    Polinom();
    Polinom(const std::vector<Monom>& m);
    Polinom(const Polinom& p);

    Polinom& operator+=(const Monom& m);
    Polinom operator+(const Monom& m)const;

    Polinom operator-(const Monom& m)const;
    Polinom& operator-=(const Monom& m);

    Polinom operator*(const Monom& m);
    Polinom& operator*=(const Monom& m);

    Polinom operator+(Polinom& p);
    Polinom& operator+=(Polinom& p);

    Polinom operator-(Polinom& p);
    Polinom& operator-=(Polinom& p);
    Polinom operator*(Polinom& p);
    Polinom& operator*=(Polinom& p);

    Polinom operator*(const int& num)const;
    Polinom& operator*=(const int& num);

    Polinom operator-(const int& num)const;
    Polinom& operator-=(const int& num);

    Polinom operator+(const int& num)const;
    Polinom& operator+=(const int& num);

    bool operator==(const Polinom& p)const;
    Polinom& operator=(const Polinom& p);
    int Result(int x, int y, int z);

    void OutputFile(std::string path);

    std::string to_str();
    void del_nulls();
    void set_null();
    void DeletPolinom();

    friend PolinomIterator;
};
```

Приложение 4

Polinom.cpp

```
bool Polinom::operator==(const Polinom& p)const
{
    return polinom == p.polinom;
}

Polinom& Polinom::operator=(const Polinom& p)
{
    polinom = p.polinom;
    it.init();
    return *this;
}

int Polinom::Resault(int x, int y, int z)
{
    int res = 0;
    it.init();
    while (it.check_next())
    {
        res += it.get_value().Resault(x, y, z);
        it.go_next();
    }
    return res;
}

void Polinom::OutputFile(std::string path)
{
    std::fstream fs;
    fs.open(path, std::ios::out | std::ios::app);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        fs << to_str();
        std::cout << "File opened and rewritten!" << std::endl;
    }
    fs.close();
}

std::string Polinom::to_str()
{
    std::string res;
    it.init();
    bool add_first_flag = false;
    Monom tmp = it.get_value();
    while (it.check_next())
    {
        if (tmp.Get_coef() > 0 && add_first_flag)
        {
            res += "+" + tmp.ConvertString();
        }
        else
        {
            res += tmp.ConvertString();
        }
        it.go_next();
        tmp = it.get_value();
        add_first_flag = true;
    }

    return res;
}

void Polinom::del_nulls()
{
    it.init();
    while (it.check_next())
    {
        if (it.get_value().Get_coef() == 0)
        {
            it.del_cur();
            it.init();
        }
        if (!it.check_next())
        {
            break;
        }
        it.go_next();
    }
    if (polinom.size() == 0)
    {
        set_null();
    }
}
```

```

void Polinom::set_null()
{
    Monom null_monom(0, 0, 0, 0);
    if (polinom.size() == 0)
    {
        polinom.push_back(null_monom);
    }
}

void Polinom::DeletPolinom()
{
    polinom.clear();
    Monom m(0, 0, 0, 0);
    polinom.push_back(m);
}

Polinom::Polinom()
{
    Monom m(0, 0, 0, 0);
    polinom.push_back(m);
    it.Set_list(polinom);
}

Polinom::Polinom(const std::vector<Monom>& m)
{
    if (m.size() == 0)
    {
        throw std::exception("Initial vector is empty!");
    }
    for (int i = 0; i < m.size(); i++)
    {
        it.Set_list(polinom);
        *this += m[i];
    }
    del_nulls();
}

Polinom::Polinom(const Polinom& p)
{
    polinom = p.polinom;
    it.Set_list(polinom);
    it.init();
}

Polinom Polinom::operator+(const Monom& m) const
{
    Polinom res(*this);
    res += m;
    return res;
}

Polinom Polinom::operator-(const Monom& m) const
{
    Polinom res(*this);
    res -= m;
    return res;
}

Polinom& Polinom::operator-=(const Monom& m)
{
    if (polinom.size() == 0)
    {
        polinom.push_back(-m);
        del_nulls();
        return *this;
    }
    it.init();
    while (it.check_next())
    {
        if (it.get_value() == m)
        {
            it.get_value() -= m;
            del_nulls();
            return *this;
        }
        else if (it.get_value() < m)
        {
            it.insert_before(-m);
            del_nulls();
            return *this;
        }
    }
    it.go_next();
    polinom.push_back(m);
    del_nulls();
    return *this;
}

```

```

Polinom Polinom::operator*(const Monom& m)
{
    Polinom p;
    it.init();
    while (it.check_next())
    {
        p += it.get_value() * m;
        it.go_next();
    }
    return p;
}

Polinom& Polinom::operator*=(const Monom& m)
{
    Polinom p;
    it.init();
    while (it.check_next())
    {
        p += it.get_value() * m;
        it.go_next();
    }
    *this = p;
    return *this;
}

Polinom Polinom::operator+(Polinom& p)
{
    Polinom res(*this);
    p.it.init();
    while (p.it.check_next())
    {
        res += p.it.get_value();
        p.it.go_next();
    }
    return res;
}

Polinom& Polinom::operator+=(Polinom& p)
{
    p.it.init();
    while (p.it.check_next())
    {
        *this += p.it.get_value();
        p.it.go_next();
    }
    return *this;
}

Polinom Polinom::operator-(Polinom& p)
{
    Polinom res(*this);
    p.it.init();
    while (p.it.check_next())
    {
        res -= p.it.get_value();
        p.it.go_next();
    }
    return res;
}

Polinom& Polinom::operator-=(Polinom& p)
{
    p.it.init();
    while (p.it.check_next())
    {
        *this -= p.it.get_value();
        p.it.go_next();
    }
    return *this;
}

Polinom Polinom::operator*(Polinom& p)
{
    Polinom res;
    p.it.init();
    it.init();
    while (it.check_next())
    {
        while (p.it.check_next())
        {
            res += it.get_value() * p.it.get_value();
            p.it.go_next();
        }
        p.it.init();
        it.go_next();
    }
    return res;
}

```

```

Polinom& Polinom::operator*=(Polinom& p)
{
    Polinom res;
    p.it.init();
    it.init();
    while (it.check_next())
    {
        while (p.it.check_next())
        {
            res += it.get_value() * p.it.get_value();
            p.it.go_next();
        }
        p.it.init();
        it.go_next();
    }
    *this = res;
    return *this;
}

```

```

Polinom Polinom::operator*(const int& num) const
{
    Monom m(num, 0, 0, 0);
    Polinom res(*this);
    res *= m;
    return res;
}

```

```

Polinom& Polinom::operator*=(const int& num)
{
    Monom m(num, 0, 0, 0);
    *this *= m;
    return *this;
}

```

```

Polinom Polinom::operator-(const int& num) const
{
    Monom m(num, 0, 0, 0);
    Polinom res(*this);
    res -= m;
    return res;
}

```

```

Polinom& Polinom::operator-=(const int& num)
{
    Monom m(num, 0, 0, 0);
    *this -= m;
    return *this;
}

```

```

Polinom Polinom::operator+(const int& num) const
{
    Monom m(num, 0, 0, 0);
    Polinom res(*this);
    res += m;
    return res;
}

```

```

Polinom& Polinom::operator+=(const int& num)
{
    Monom m(num, 0, 0, 0);
    *this += m;
    return *this;
}

```

```

Polinom& Polinom::operator+=(const Monom& m)
{
    if (polinom.size() == 0)
    {
        polinom.push_back(m);
        del_nulls();
        return *this;
    }
    it.init();
    while (it.check_next())
    {
        if (it.get_value() == m)
        {
            it.get_value() += m;
            del_nulls();
            return *this;
        }
        else if (it.get_value() < m)
        {
            it.insert_before(m);
            del_nulls();
            return *this;
        }
        it.go_next();
    }
    polinom.push_back(m);
    del_nulls();
    return *this;
}

```

Приложение 5

PolinomIterator.h

```
class PolinomIterator {
private:
    std::list<Monom>* MonomList;
    std::_List_node<Monom, void*>* currentNode;
public:
    PolinomIterator();
    PolinomIterator(std::list<Monom>& l);

    void Set_list(std::list<Monom>& l);
    void init();
    bool check_next();
    void go_next();
    Monom& get_value();
    void del_cur();
    void insert_before(const Monom& m);
};
```

Приложение 6

PolinomIterator.cpp

```
PolinomIterator::PolinomIterator()
{
    MonomList = nullptr;
    currentNode = nullptr;
}

PolinomIterator::PolinomIterator(std::list<Monom>& l)
{
    MonomList = &l;
    init();
}

void PolinomIterator::Set_list(std::list<Monom>& l)
{
    MonomList = &l;
    currentNode = MonomList->begin()._Ptr;
}

void PolinomIterator::init()
{
    currentNode = MonomList->begin()._Ptr;
}

bool PolinomIterator::check_next()
{
    if (currentNode != MonomList->end()._Ptr)
        return true;
    else return false;
}

void PolinomIterator::go_next()
{
    if (!check_next()) throw std::exception("End of list!");
    currentNode = currentNode->_Next;
}

Monom& PolinomIterator::get_value()
{
    return currentNode->_Myval;
}

void PolinomIterator::del_cur()
{
    auto it = MonomList->begin();
    while (it._Ptr != currentNode) it++;
    MonomList->erase(it);
}

void PolinomIterator::insert_before(const Monom& m)
{
    auto it = MonomList->begin();
    while (it._Ptr != currentNode) it++;
    MonomList->insert(it, m);
}
```


Приложение 7

Main.cpp

```
#define PATH_FOR_FILE "C:\\out.txt"

void Menu(Polinom& p1, Polinom& p2, Polinom& res)
{
    std::cout << "p1 = " << p1.to_str() << std::endl;
    std::cout << "p2 = " << p2.to_str() << std::endl;
    std::cout << "res = " << res.to_str() << std::endl << std::endl;
    std::cout << "1) Сложить полиномы" << std::endl << std::endl;
    std::cout << "2) Вычесть полиномы" << std::endl << std::endl;
    std::cout << "3) Перемножить полиномы" << std::endl << std::endl;
    std::cout << "4) Посчитать результат" << std::endl << std::endl;
    std::cout << "5) Ввести новые значения" << std::endl << std::endl;
    std::cout << "6) Вывести в файл" << std::endl << std::endl;
    std::cout << "0) Выход" << std::endl << std::endl;
    std::cout << "> ";
}

void calculate_polinom(Polinom& res)
{
    if (res.to_str() == "0") { std::cout << "Выполните 1,2 или 3 пункт меню!\n"; return; }
    int x, y, z;
    std::cout << "Введите a: ";
    std::cin >> x;
    std::cout << "Введите b: ";
    std::cin >> y;
    std::cout << "Введите c: ";
    std::cin >> z;
    std::cout << res.to_str() << " = " << res.Result(x, y, z) << std::endl;
}

void input_monom(Monom& m)
{
    int A, B, C, coef;
    std::cout << "Коэффициент: ";
    std::cin >> coef;
    std::cout << "Введите A: ";
    std::cin >> A;
    std::cout << "Введите B: ";
    std::cin >> B;
    std::cout << "Введите C: ";
    std::cin >> C;
    Monom monom(coef, A, B, C);
    m = monom;
}

void input_polinom(Polinom& p)
{
    Monom m;

    int num_of_monoms;
    std::cout << "Количество мономов: ";
    std::cin >> num_of_monoms;
    for (int i = 0; i < num_of_monoms; i++)
    {
        std::cout << "Введите моном " << i + 1 << std::endl;
        input_monom(m);
        p += m;
    }
}
```

```
int main()
{
    setlocale(LC_ALL, "RUS");
    Polinom p1;
    Polinom p2;
    Polinom res;
    int menuSelection = 0;
    while (true)
    {
        system("cls");
        Menu(p1, p2, res);
        std::cin >> menuSelection;
        switch (menuSelection)
        {
            case 1:
                res = p1 + p2;
                std::cout << p1.to_str() << " + " << p2.to_str() << " = " << res.to_str() << std::endl;
                system("pause");
                break;
            case 2:
                res = p1 - p2;
                std::cout << p1.to_str() << " - " << p2.to_str() << " = " << res.to_str() << std::endl;
                system("pause");
                break;
            case 3:
                res = p1 * p2;
                std::cout << p1.to_str() << " * " << p2.to_str() << " = " << res.to_str() << std::endl;
                system("pause");
                break;
            case 4:
                calculate_polinom(res);
                system("pause");
                break;
            case 5:
                system("cls");
                std::cout << "Введите p1 и p2:\n";
                p1.DeletPolinom();
                p2.DeletPolinom();
                res.DeletPolinom();
                input_polinom(p1);
                input_polinom(p2);
                break;
            case 6:
                if (res.to_str() == "0")
                {
                    std::cout << "Выполните 1,2 или 3 пункт меню!\n";
                    system("pause");
                    break;
                }

                std::cout << "Вывести в файл результат: ";
                res.OutputFile(PATH_FOR_FILE);
                system("pause");
                break;
            case 0:
                return 0;
        }
    }
    return 0;
}
```