МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

*Системы Сбора И Обработки Данных*

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название кафедры)

**Курсовая работа**

по дисциплине: Программирование

наименование дисциплины в соответствии с УП

Тема: Разработка класса Палиндром

Рецензия: (краткое обоснование оценки):

Выполнил:

Студент:Жуков И.А.

Группа: АТ-34

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

Проверил:

Преподаватель: Воронов В.В.

Балл:\_\_\_\_\_\_\_\_\_\_, ECTS\_\_\_\_\_\_\_\_\_\_\_\_,

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Новосибирск 2024

Содержание  
  
Технической задание — 1стр  
Описание разработанного класса — 1 стр

* **Техническое задание**

Разработать класс, представляющий заданный тип данных (ТД). Класс должен включать в себя (если это необходимо и подходит по смыслу к ТД):

* Конструкторы (по умолчанию, с параметрами, копии);
* Деструктор;
* Функцию вывода на экран;
* Перегруженные операторы:
  + Оператор присваивания;
  + Бинарные арифметические (+, -, \*, /) с операндами ТД и ТД, ТД и базовый тип, базовый тип и ТД;
  + Арифметика с накоплением (+=,-=);
  + Унарные (++,--) в префиксной и постфиксной форме;
  + Логические (<, >, ==, != );
  + Операторы взятия элемента ( [] или () ) по номеру или ключу;
  + Операторы преобразования типа к любому базовому.

Написать головную программу, обеспечивающую проверку всех реализованных свойств и методов класса.

В варианте 10 нужно реализовать полином

* **Описание разработанного класса**

Polynomial — это класс, представляющий полином с целочисленными коэффициентами, который поддерживает базовые арифметические операции, операции сравнения, индексный доступ к коэффициентам и различные дополнительные операции. Класс позволяет создавать полиномы из списка коэффициентов, выполнять математические операции между полиномами, получать значение полинома при заданном значении переменной и производить преобразования типа.

### Члены класса

#### Конструкторы и деструкторы

* Polynomial() — Конструктор по умолчанию. Создаёт полином, представляющий собой ноль.
* Polynomial(const std::vector<int>& coeffs) — Конструктор, принимающий вектор коэффициентов coeffs, где каждый элемент представляет коэффициент при соответствующей степени переменной.
* Polynomial(const Polynomial& other) — Конструктор копирования. Создаёт полином, копируя коэффициенты из other.
* ~Polynomial() — Деструктор. Очищает ресурсы, если это необходимо (в данном случае ресурсы std::vector очищаются автоматически).

#### Операторы присваивания и преобразования

* Polynomial& operator=(const Polynomial& other) — Оператор присваивания, копирующий содержимое other в текущий полином.
* operator int() const — Преобразование в int, возвращающее старший коэффициент полинома.
* operator double() const — Преобразование в double, возвращающее сумму всех коэффициентов (эквивалентно значению полинома при x = 1).

#### Арифметические операторы

* Polynomial operator+(const Polynomial& other) const — Оператор сложения. Возвращает сумму текущего полинома и other.
* Polynomial operator-(const Polynomial& other) const — Оператор вычитания. Возвращает разность текущего полинома и other.
* Polynomial operator\*(const Polynomial& other) const — Оператор умножения. Возвращает результат умножения текущего полинома на other.
* Polynomial operator\*(int scalar) const — Умножение на скаляр. Возвращает полином, полученный умножением текущего полинома на целое число scalar.
* Polynomial operator/(int scalar) const — Деление на скаляр. Возвращает полином, полученный делением текущего полинома на целое число scalar.

#### Операторы с накоплением

* Polynomial& operator+=(const Polynomial& other) — Оператор +=. Складывает other с текущим полиномом.
* Polynomial& operator-=(const Polynomial& other) — Оператор -=. Вычитает other из текущего полинома.

#### Унарные операторы

* Polynomial& operator++() — Префиксный инкремент. Увеличивает свободный член полинома на 1.
* Polynomial operator++(int) — Постфиксный инкремент. Увеличивает свободный член полинома на 1 и возвращает состояние полинома до увеличения.
* Polynomial& operator--() — Префиксный декремент. Уменьшает свободный член полинома на 1.
* Polynomial operator--(int) — Постфиксный декремент. Уменьшает свободный член полинома на 1 и возвращает состояние полинома до уменьшения.

#### Логические операторы

* bool operator==(const Polynomial& other) const — Возвращает true, если текущий полином равен other.
* bool operator!=(const Polynomial& other) const — Возвращает true, если текущий полином не равен other.
* bool operator<(const Polynomial& other) const — Возвращает true, если текущий полином имеет степень меньше, чем у other.
* bool operator>(const Polynomial& other) const — Возвращает true, если текущий полином имеет степень больше, чем у other.

#### Операторы доступа

* int operator[](size\_t index) const — Возвращает коэффициент при степени index. Бросает исключение std::out\_of\_range, если индекс выходит за границы.
* int& operator[](size\_t index) — Возвращает ссылку на коэффициент при степени index. Бросает исключение std::out\_of\_range, если индекс выходит за границы.

#### Вспомогательные функции

* void print() const — Выводит полином в удобочитаемом виде.

int evaluate(int x) const — Возвращает значение полинома при заданном x.

Приложения

// оператор присваивания  
Polynomial &Polynomial::operator=(const Polynomial &other) {  
 if (this != &other) {  
 coefficients = other.coefficients;  
 }  
 return \*this;  
}  
  
// Арифметические операторы  
Polynomial Polynomial::operator+(const Polynomial &other) const {  
 size\_t maxSize = std::max(coefficients.size(), other.coefficients.size());  
 std::vector<int> resultCoeffs(maxSize, 0);  
  
 for (size\_t i = 0; i < maxSize; ++i) {  
 int a = i < coefficients.size() ? coefficients[i] : 0;  
 int b = i < other.coefficients.size() ? other.coefficients[i] : 0;  
 resultCoeffs[i] = a + b;  
 }  
 Polynomial ret = Polynomial(resultCoeffs);  
 return ret;  
}  
  
Polynomial Polynomial::operator-(const Polynomial &other) const {  
 size\_t maxSize = std::max(coefficients.size(), other.coefficients.size());  
 std::vector<int> resultCoeffs(maxSize, 0.0);  
  
 for (size\_t i = 0; i < maxSize; ++i) {  
 int a = i < coefficients.size() ? coefficients[i] : 0.0;  
 int b = i < other.coefficients.size() ? other.coefficients[i] : 0.0;  
 resultCoeffs[i] = a - b;  
 }  
 Polynomial ret = Polynomial(resultCoeffs);  
 return ret;  
}  
  
Polynomial Polynomial::operator\*(const Polynomial &other) const {  
 std::vector<int> resultCoeffs(  
 coefficients.size() + other.coefficients.size() - 1, 0.0);  
  
 for (size\_t i = 0; i < coefficients.size(); ++i) {  
 for (size\_t j = 0; j < other.coefficients.size(); ++j) {  
 resultCoeffs[i + j] += coefficients[i] \* other.coefficients[j];  
 }  
 }  
 Polynomial ret = Polynomial(resultCoeffs);  
 return ret;  
}  
  
Polynomial Polynomial::operator\*(int scalar) const {  
 if (scalar == 0) {  
 Polynomial a;  
 return a;  
 }  
 std::vector<int> resultCoeffs(coefficients.size());  
  
 for (size\_t i = 0; i < coefficients.size(); ++i) {  
 resultCoeffs[i] = coefficients[i] \* scalar;  
 }  
 Polynomial ret = Polynomial(resultCoeffs);  
 return ret;  
}  
  
Polynomial Polynomial::operator/(int scalar) const {  
 if (scalar == 0)  
 throw std::runtime\_error("Division by zero");  
 std::vector<int> resultCoeffs(coefficients.size());  
  
 for (size\_t i = 0; i < coefficients.size(); ++i) {  
 resultCoeffs[i] = coefficients[i] / scalar;  
 }  
 Polynomial ret = Polynomial(resultCoeffs);  
 return ret;  
}  
Polynomial operator+(int scalar, const Polynomial& poly) {  
 // Складываем скаляр только с нулевым коэффициентом  
 std::vector<int> resultCoeffs = poly.coefficients;  
 if (!resultCoeffs.empty()) {  
 resultCoeffs[0] += scalar;  
 } else {  
 resultCoeffs.push\_back(scalar);  
 }  
 return Polynomial(resultCoeffs);  
}  
  
Polynomial operator-(int scalar, const Polynomial& poly) {  
 // Вычитаем полином из скаляра, инвертируем знак всех коэффициентов полинома  
 std::vector<int> resultCoeffs(poly.coefficients.size());  
 for (size\_t i = 0; i < poly.coefficients.size(); ++i) {  
 resultCoeffs[i] = -poly.coefficients[i];  
 }  
 if (!resultCoeffs.empty()) {  
 resultCoeffs[0] += scalar;  
 } else {  
 resultCoeffs.push\_back(scalar);  
 }  
 return Polynomial(resultCoeffs);  
}  
  
Polynomial operator\*(int scalar, const Polynomial& poly) {  
 // Просто умножаем все коэффициенты на скаляр  
 std::vector<int> resultCoeffs(poly.coefficients.size());  
 for (size\_t i = 0; i < poly.coefficients.size(); ++i) {  
 resultCoeffs[i] = poly.coefficients[i] \* scalar;  
 }  
 return Polynomial(resultCoeffs);  
}  
  
Polynomial operator/(int scal, const Polynomial &poly) {  
 if (scal == 0) {  
 throw std::runtime\_error(  
 "Division by zero scalar is not allowed"); // Проверка на деление на 0  
 }  
  
 std::vector<int> resCoef(poly.coefficients.size());  
 for (size\_t i = 0; i < poly.coefficients.size(); ++i) {  
 if (poly.coefficients[i] == 0) {  
 throw std::runtime\_error(  
 "Division by zero coefficient in polynomial"); // Проверка  
 // коэффициентов  
 }  
 resCoef[i] =  
 scal / poly.coefficients[i]; // Делим скаляр на каждый коэффициент  
 }  
  
 return Polynomial(resCoef);  
}

// Арифметика с накоплением  
Polynomial &Polynomial::operator+=(const Polynomial &other) {  
 \*this = \*this + other;  
 return \*this;  
}  
  
Polynomial &Polynomial::operator-=(const Polynomial &other) {  
 \*this = \*this - other;  
 return \*this;  
}  
  
// Унарные операторы  
Polynomial &Polynomial::operator++() {  
 coefficients[0] += 1;  
 return \*this;  
}  
  
Polynomial Polynomial::operator++(int) {  
 Polynomial temp = \*this;  
 ++(\*this);  
 return temp;  
}  
  
Polynomial &Polynomial::operator--() {  
 coefficients[0] -= 1;  
 return \*this;  
}  
  
Polynomial Polynomial::operator--(int) {  
 Polynomial temp = \*this;  
 --(\*this);  
 return temp;  
}  
  
// Логические операторы  
bool Polynomial::operator==(const Polynomial &other) const {  
 return coefficients == other.coefficients;  
}  
  
bool Polynomial::operator!=(const Polynomial &other) const {  
 return !(\*this == other);  
}  
  
bool Polynomial::operator<(const Polynomial &other) const {  
 return coefficients.size() < other.coefficients.size();  
}  
  
bool Polynomial::operator>(const Polynomial &other) const {  
 return coefficients.size() > other.coefficients.size();  
}  
  
// Оператор доступа к элементу  
int Polynomial::operator[](size\_t index) const {  
 if (index >= coefficients.size())  
 throw std::out\_of\_range("Index out of range");  
 return coefficients[index];  
}  
  
int &Polynomial::operator[](size\_t index) {  
 if (index >= coefficients.size())  
 throw std::out\_of\_range("Index out of range");  
 return coefficients[index];  
}

void Polynomial::print() const {  
 bool first = true;  
  
 if (coefficients.size() == 1){  
 std::cout<<coefficients[0]<<std::endl;  
 return;  
 }  
 for (int deg\_x = coefficients.size() - 1; deg\_x >= 0; --deg\_x) {  
 int coeff = coefficients[deg\_x];  
  
 if (coeff == 0) continue;  
  
 if (coeff < 0) {  
 if (!first) {  
 std::cout << " - ";  
 } else {  
 std::cout << "-";  
 }  
 coeff = -coeff;  
 } else {  
 if (!first) {  
 std::cout << " + ";  
 }  
 }  
 first = false;  
  
 if (coeff != 1 || deg\_x == 0) {  
 std::cout << coeff;  
 }  
  
 if (deg\_x > 0) {  
 std::cout << "x";  
 if (deg\_x > 1) {  
 std::cout << "^" << deg\_x;  
 }  
 }  
 }  
 std::cout << std::endl;  
}  
  
// Оператор преобразования в int  
// возвращает старший коэффициент  
Polynomial::operator int() const {  
  
 if (coefficients.empty()) {  
 return 0;  
 }  
 return coefficients.back();  
}  
  
// Оператор преобразования в double  
// Вычисляет значение полинома при x = 1  
Polynomial::operator double() const {  
 double result = 0.0;  
 for (int coeff : coefficients) {  
 result += coeff;  
 }  
 return result;  
}  
  
// Вспомогательная функция для вычисления значения полинома при x  
int Polynomial::evaluate(int x) const {  
 int result = 0;  
 for (size\_t i = 0; i < coefficients.size(); ++i) {  
 result += coefficients[i] \* std::pow(x, i);  
 }  
 return result;  
}