Cairo University

Faculty of Computers and Artificial Intelligence

CS213: object oriented programming

Assignment 1

Tasks 2&3

محمد علي حسن امين شكري 20230347 S21

يوسف فريد سيد حسنين حبلص 20230504 S21

اسلام وليد صلاح عبدالمطلب 20230062 S15

Marking standards:

In this report we have given the estimations of code quality following this methodology:

- 65 correctness
- 15 code being easy to read, understood and modified in the future.
- 20 efficiency.

Code has 19 functions, which means each function has ~5 marks:

- 3.25 on correctness.
- 0.75 on elegancy.
- 1 on efficiency.

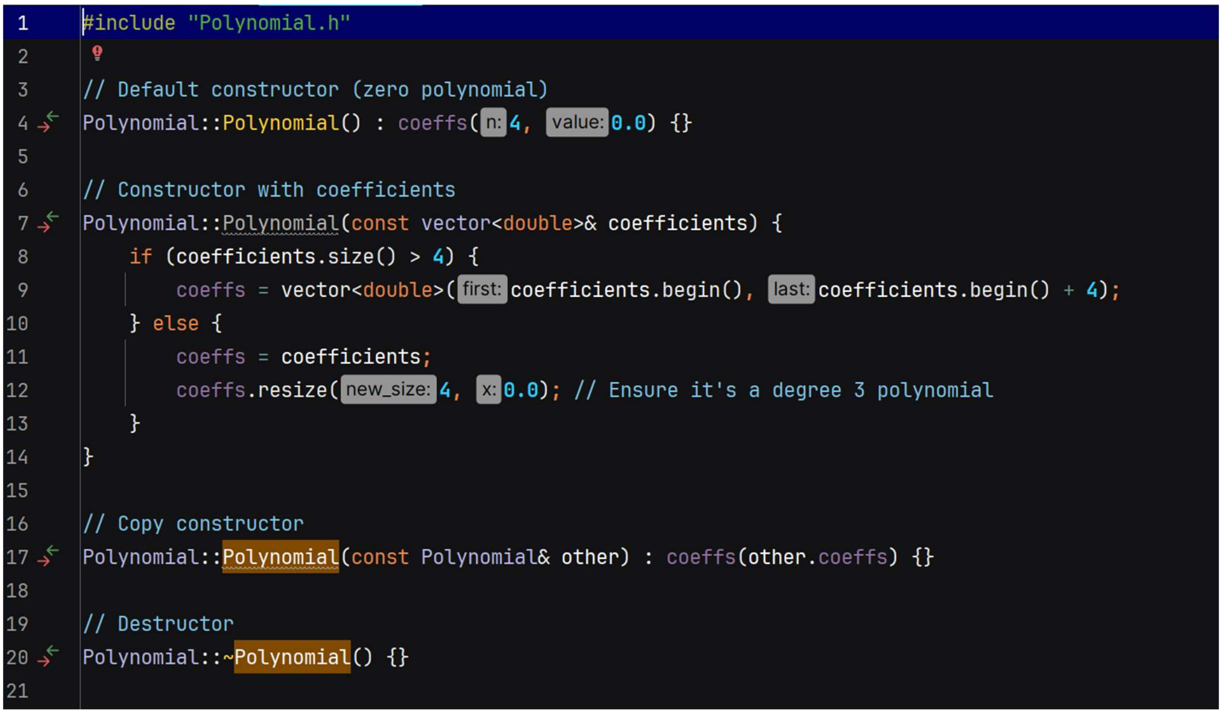
1. Chatgpt code:

   1.1. Prompt:

   I want to make a class for polynomial of degree 3, this is the header file, take the prototypes and give me the source file with all the prototypes implemented :

   And then provided the Ai with the header file.

   1.2. Constructors:

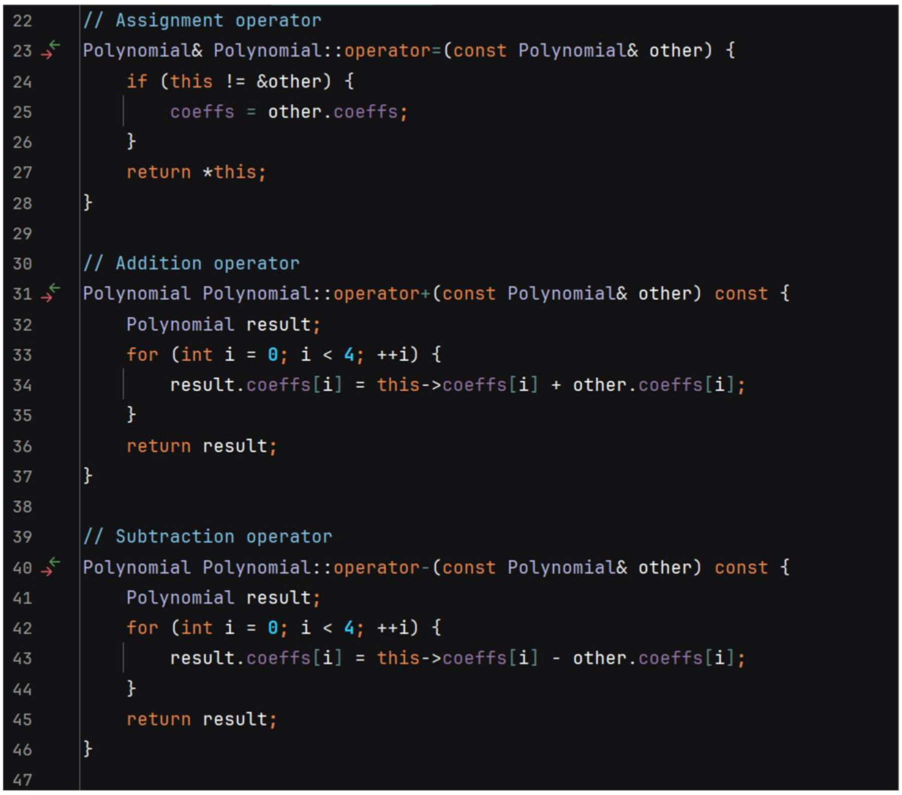


```cpp
#include "Polynomial.h"

// Default constructor (zero polynomial)
Polynomial::Polynomial() : coeffs(n: 4, value: 0.0) {}

// Constructor with coefficients
Polynomial::Polynomial(const vector<double>& coefficients) {
    if (coefficients.size() > 4) {
        coeffs = vector<double>(first: coefficients.begin(), last: coefficients.begin() + 4);
    } else {
        coeffs = coefficients;
        coeffs.resize(new_size: 4, x: 0.0); // Ensure it's a degree 3 polynomial
    }
}

// Copy constructor
Polynomial::Polynomial(const Polynomial& other) : coeffs(other.coeffs) {}

// Destructor
Polynomial::~Polynomial() {}
```

No issues found with these methods of construction.

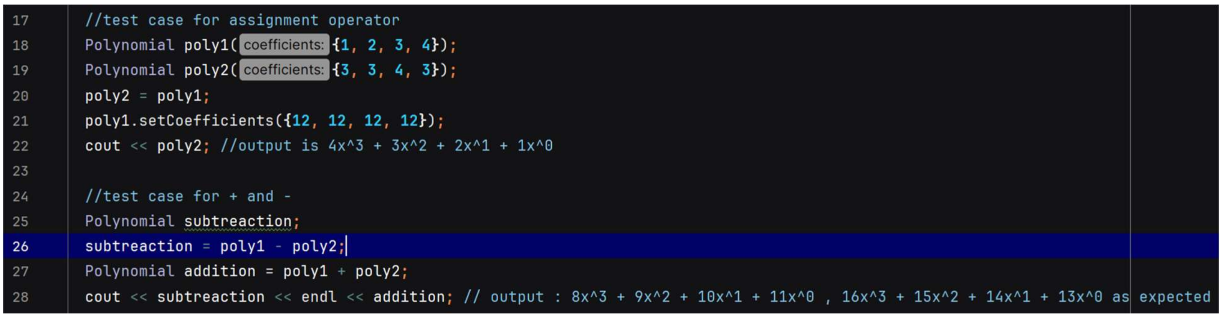
It’s a good practice to make a copy constructor Although cpp provide it by default.

1.3. Operator overloading



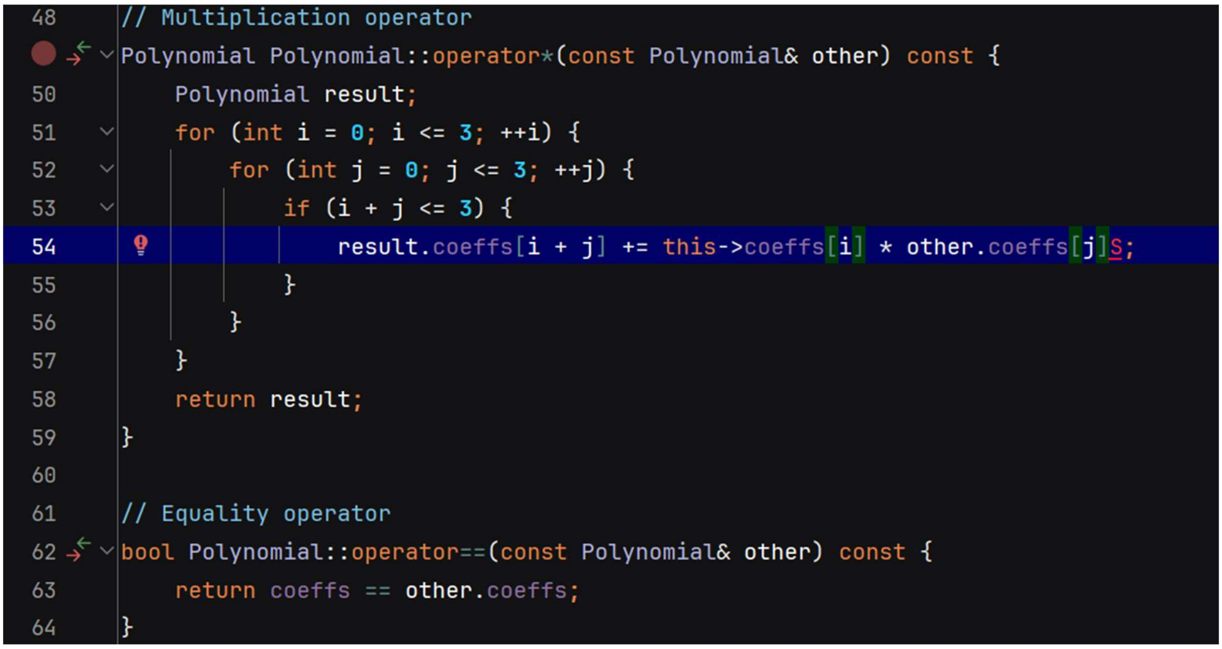
```cpp
// Assignment operator
Polynomial& Polynomial::operator=(const Polynomial& other) {
    if (this != &other) {
        coeffs = other.coeffs;
    }
    return *this;
}

// Addition operator
Polynomial Polynomial::operator+(const Polynomial& other) const {
    Polynomial result;
    for (int i = 0; i < 4; ++i) {
        result.coeffs[i] = this->coeffs[i] + other.coeffs[i];
    }
    return result;
}

// Subtraction operator
Polynomial Polynomial::operator-(const Polynomial& other) const {
    Polynomial result;
    for (int i = 0; i < 4; ++i) {
        result.coeffs[i] = this->coeffs[i] - other.coeffs[i];
    }
    return result;
}

```

Test case:



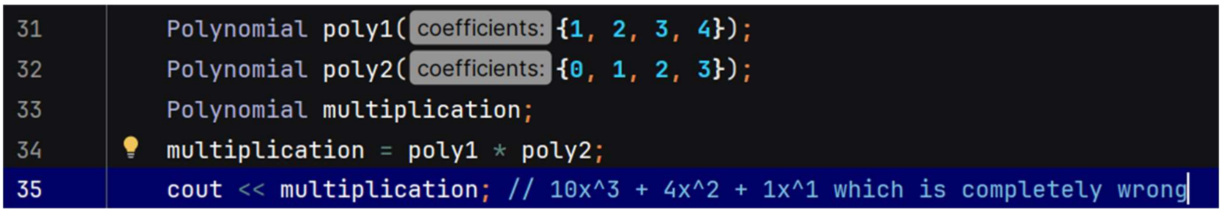
```cpp
//test case for assignment operator
Polynomial poly1(coefficients: {1, 2, 3, 4});
Polynomial poly2(coefficients: {3, 3, 4, 3});
poly2 = poly1;
poly1.setCoefficients({12, 12, 12, 12});
cout << poly2; //output is 4x^3 + 3x^2 + 2x^1 + 1x^0

//test case for + and -
Polynomial subtreaction;
subtreaction = poly1 - poly2;
Polynomial addition = poly1 + poly2;
cout << subtreaction << endl << addition; // output : 8x^3 + 9x^2 + 10x^1 + 11x^0 , 16x^3 + 15x^2 + 14x^1 + 13x^0 as expected
```

Assignment, addition, subtraction, and insertion operators work well.



```cpp
// Multiplication operator
Polynomial Polynomial::operator*(const Polynomial& other) const {
    Polynomial result;
    for (int i = 0; i <= 3; ++i) {
        for (int j = 0; j <= 3; ++j) {
            if (i + j <= 3) {
                result.coeffs[i + j] += this->coeffs[i] * other.coeffs[j]S;
            }
        }
    }
    return result;
}

// Equality operator
bool Polynomial::operator==(const Polynomial& other) const {
    return coeffs == other.coeffs;
}
```

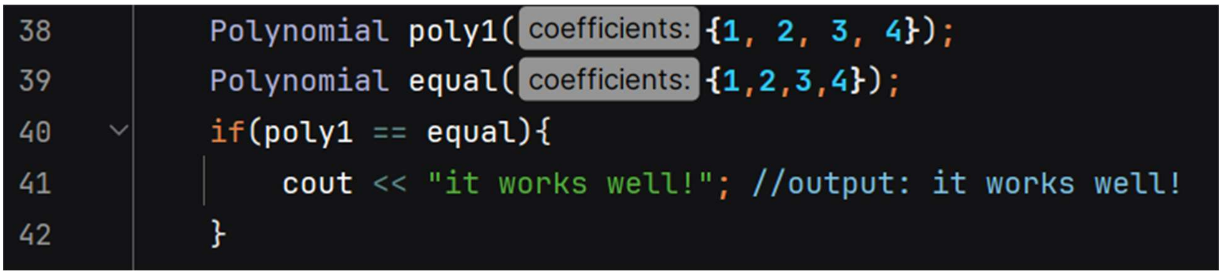
- Multiplying each coefficient and summing it in the result polynomial is , needless to say, vain and silly.
- Making variable named max_coefficients instead of the magical number 3, would have been a better practice, although code is clean enough.



```cpp
    Polynomial poly1(coefficients: {1, 2, 3, 4});
    Polynomial poly2(coefficients: {0, 1, 2, 3});
    Polynomial multiplication;
    multiplication = poly1 * poly2;
    cout << multiplication; // 10x^3 + 4x^2 + 1x^1 which is completely wrong
```

- As expected code doesn’t work with this test case.

So code looses 3.25 marks .

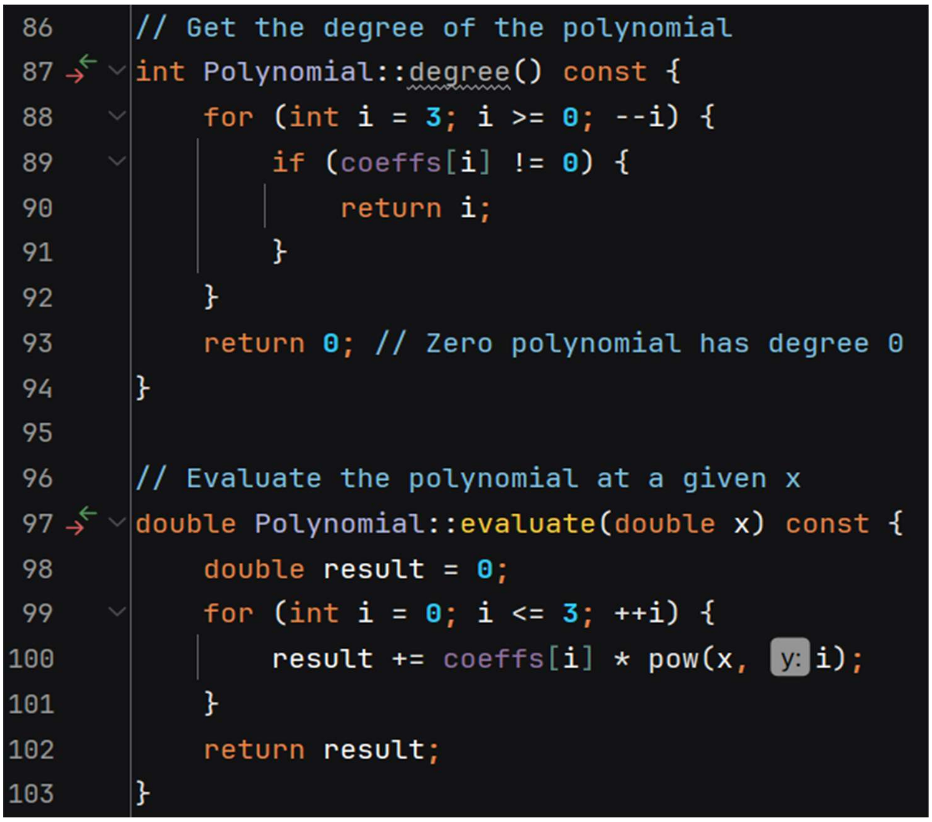


```cpp
    Polynomial poly1(coefficients: {1, 2, 3, 4});
    Polynomial equal(coefficients: {1,2,3,4});
    if(poly1 == equal){
        cout << "it works well!"; //output: it works well!
    }
```

Equal operator overloading works well.

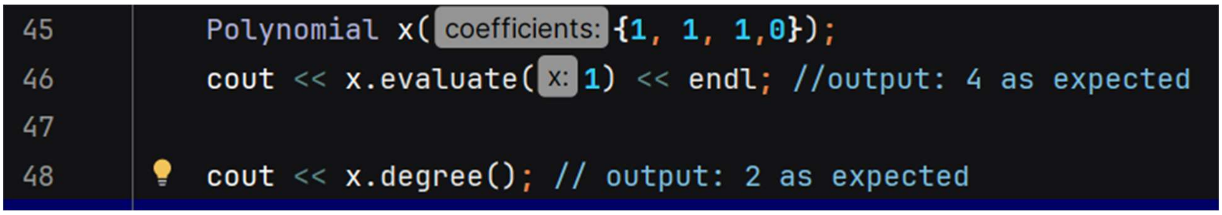
1.4. Utility functions:

1.4.1. Degree and evaluate



```cpp
// Get the degree of the polynomial
int Polynomial::degree() const {
    for (int i = 3; i >= 0; --i) {
        if (coeffs[i] != 0) {
            return i;
        }
    }
    return 0; // Zero polynomial has degree 0
}

// Evaluate the polynomial at a given x
double Polynomial::evaluate(double x) const {
    double result = 0;
    for (int i = 0; i <= 3; ++i) {
        result += coeffs[i] * pow(x, y: i);
    }
    return result;
}
```

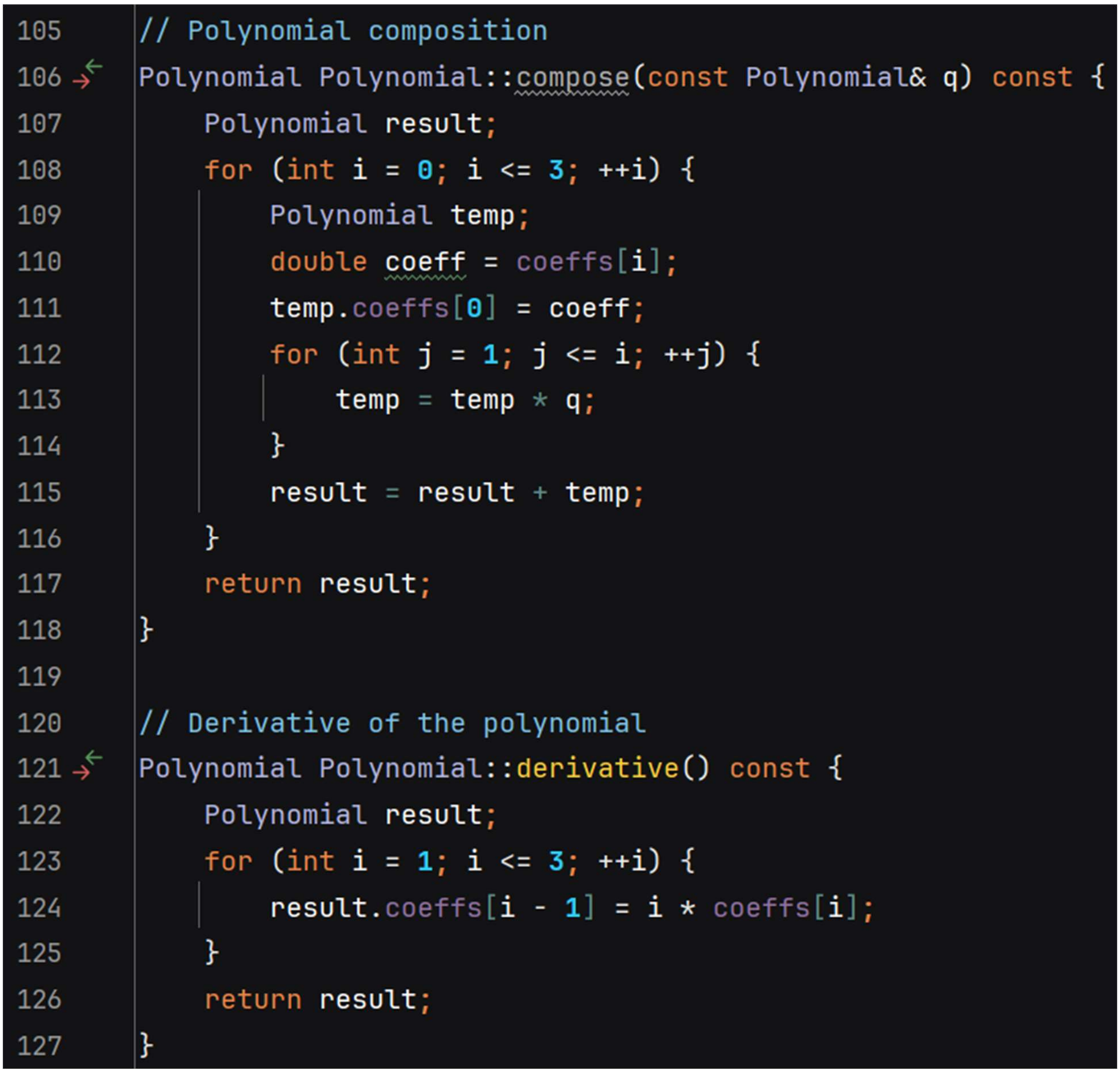
Using pow function is not the best idea, since using fast power would have been more efficient, code looses 1 mark.



```cpp
    Polynomial x( coefficients: {1, 1, 1,0});
    cout << x.evaluate( x: 1) << endl; //output: 4 as expected

    cout << x.degree(); // output: 2 as expected
```

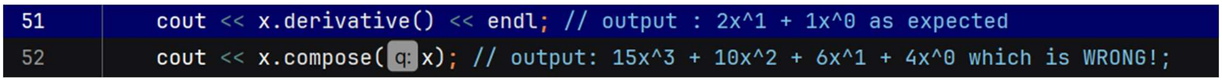
Incorrect evaluation, code losses 3.25.

1.4.2. Compose and derivative



```cpp
// Polynomial composition
Polynomial Polynomial::compose(const Polynomial& q) const {
    Polynomial result;
    for (int i = 0; i <= 3; ++i) {
        Polynomial temp;
        double coeff = coeffs[i];
        temp.coeffs[0] = coeff;
        for (int j = 1; j <= i; ++j) {
            temp = temp * q;
        }
        result = result + temp;
    }
    return result;
}

// Derivative of the polynomial
Polynomial Polynomial::derivative() const {
    Polynomial result;
    for (int i = 1; i <= 3; ++i) {
        result.coeffs[i - 1] = i * coeffs[i];
    }
    return result;
}
```

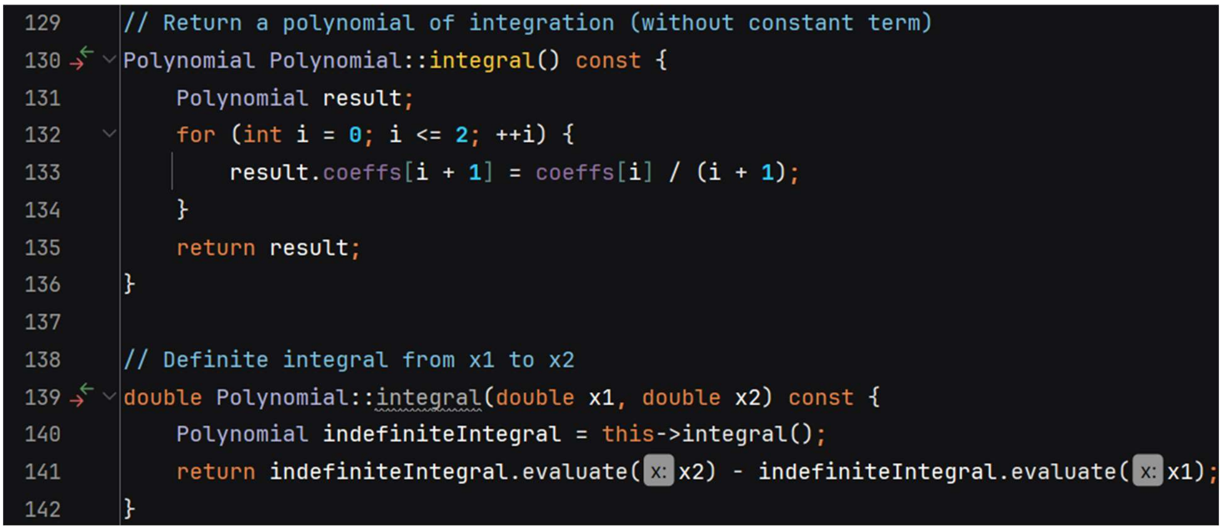
Function compose is poorly coded, as variable name ‘q’ doesn’t really tell anything about what it is, and same thing goes with variable name ‘temp’, code looses 0.75 mark.



```cpp
    cout << x.derivative() << endl; // output : 2x^1 + 1x^0 as expected
    cout << x.compose(q: x); // output: 15x^3 + 10x^2 + 6x^1 + 4x^0 which is WRONG!;
```

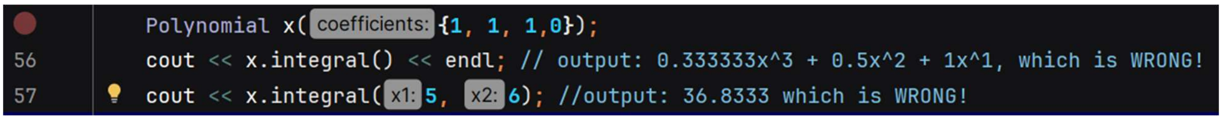
Code looses 3.25 for function compose for being not correct.

1.4.3. Integral and definite integral :



```
// Return a polynomial of integration (without constant term)
Polynomial Polynomial::integral() const {
    Polynomial result;
    for (int i = 0; i <= 2; ++i) {
        result.coeffs[i + 1] = coeffs[i] / (i + 1);
    }
    return result;
}

// Definite integral from x1 to x2
double Polynomial::integral(double x1, double x2) const {
    Polynomial indefiniteIntegral = this->integral();
    return indefiniteIntegral.evaluate(x: x2) - indefiniteIntegral.evaluate(x: x1);
}
```

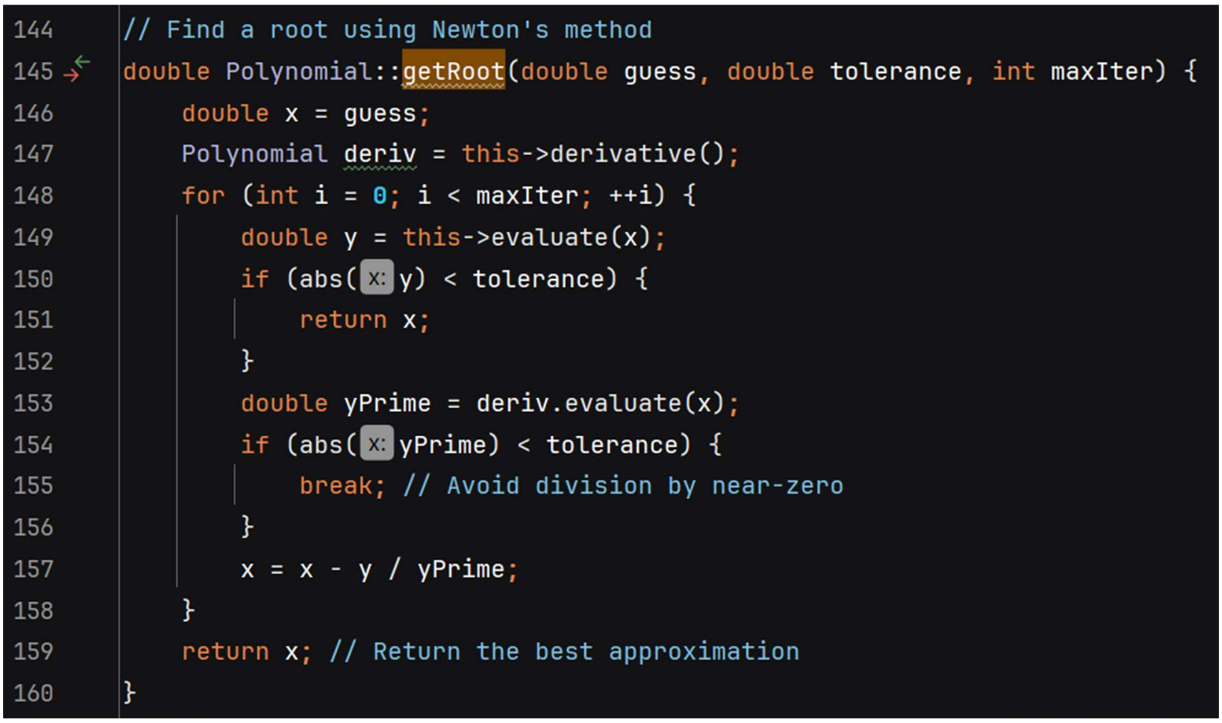
Test case :



```
    Polynomial x(coefficients: {1, 1, 1,0});
    cout << x.integral() << endl; // output: 0.333333x^3 + 0.5x^2 + 1x^1, which is WRONG!
    cout << x.integral(x1: 5, x2: 6); //output: 36.8333 which is WRONG!
```

Code fails!, consequently looses 3.25 * 2 for being incorrect.

1.4.4. getRoot function:

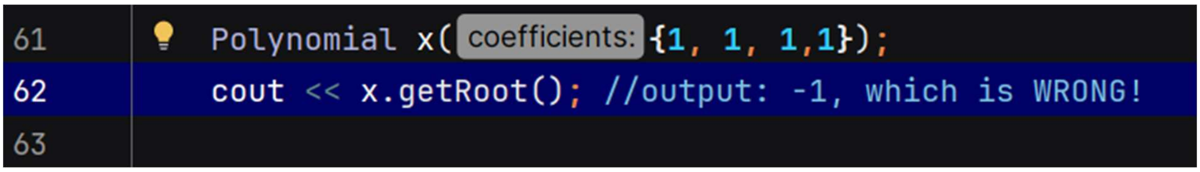


```cpp
// Find a root using Newton's method
double Polynomial::getRoot(double guess, double tolerance, int maxIter) {
    double x = guess;
    Polynomial deriv = this->derivative();
    for (int i = 0; i < maxIter; ++i) {
        double y = this->evaluate(x);
        if (abs( x: y) < tolerance) {
            return x;
        }
        double yPrime = deriv.evaluate(x);
        if (abs( x: yPrime) < tolerance) {
            break; // Avoid division by near-zero
        }
        x = x - y / yPrime;
    }
    return x; // Return the best approximation
}
```

‘x’ and ‘y’ variable names are not meaningful, ‘maxIter’ as well as ‘deriv’ can not be pronounced, consequently can not be discussed between programmers, code looses 0.75.

Using newton’s method instead of simply using sqrt function or any other specialist function makes code more messy and problematic, code looses 0.5 marks.

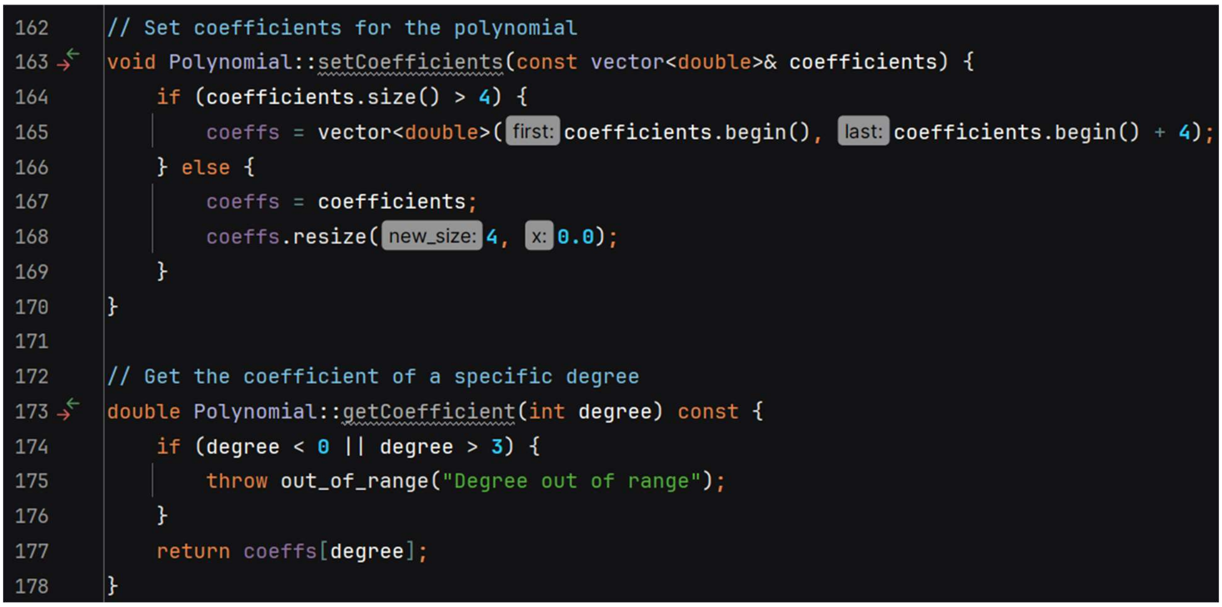
Test case :



```cpp
    Polynomial x( coefficients: {1, 1, 1,1});
    cout << x.getRoot(); //output: -1, which is WRONG!
```

Code looses 3.25 for being incorrect.

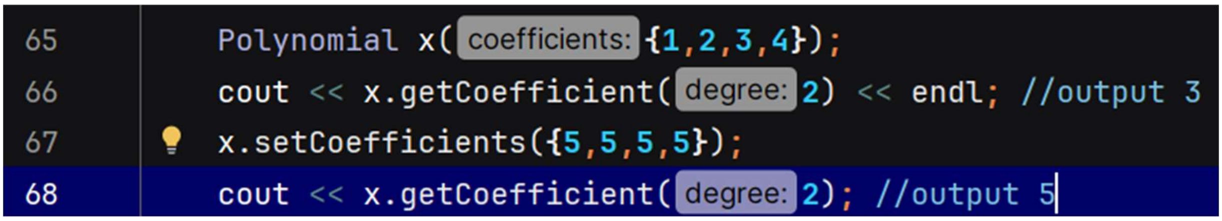
1.4.5. Getters and setters:



```
// Set coefficients for the polynomial
void Polynomial::setCoefficients(const vector<double>& coefficients) {
    if (coefficients.size() > 4) {
        coeffs = vector<double>(first: coefficients.begin(), last: coefficients.begin() + 4);
    } else {
        coeffs = coefficients;
        coeffs.resize( new_size: 4, x: 0.0);
    }
}

// Get the coefficient of a specific degree
double Polynomial::getCoefficient(int degree) const {
    if (degree < 0 || degree > 3) {
        throw out_of_range("Degree out of range");
    }
    return coeffs[degree];
}
```

setCoefficients function does not throw any errors when given more than 4 items in the vector, code looses 1 mark for being incorrect.

Test case:



```
    Polynomial x( coefficients: {1,2,3,4});
    cout << x.getCoefficient( degree: 2) << endl; //output 3
    x.setCoefficients({5,5,5,5});
    cout << x.getCoefficient( degree: 2); //output 5
```

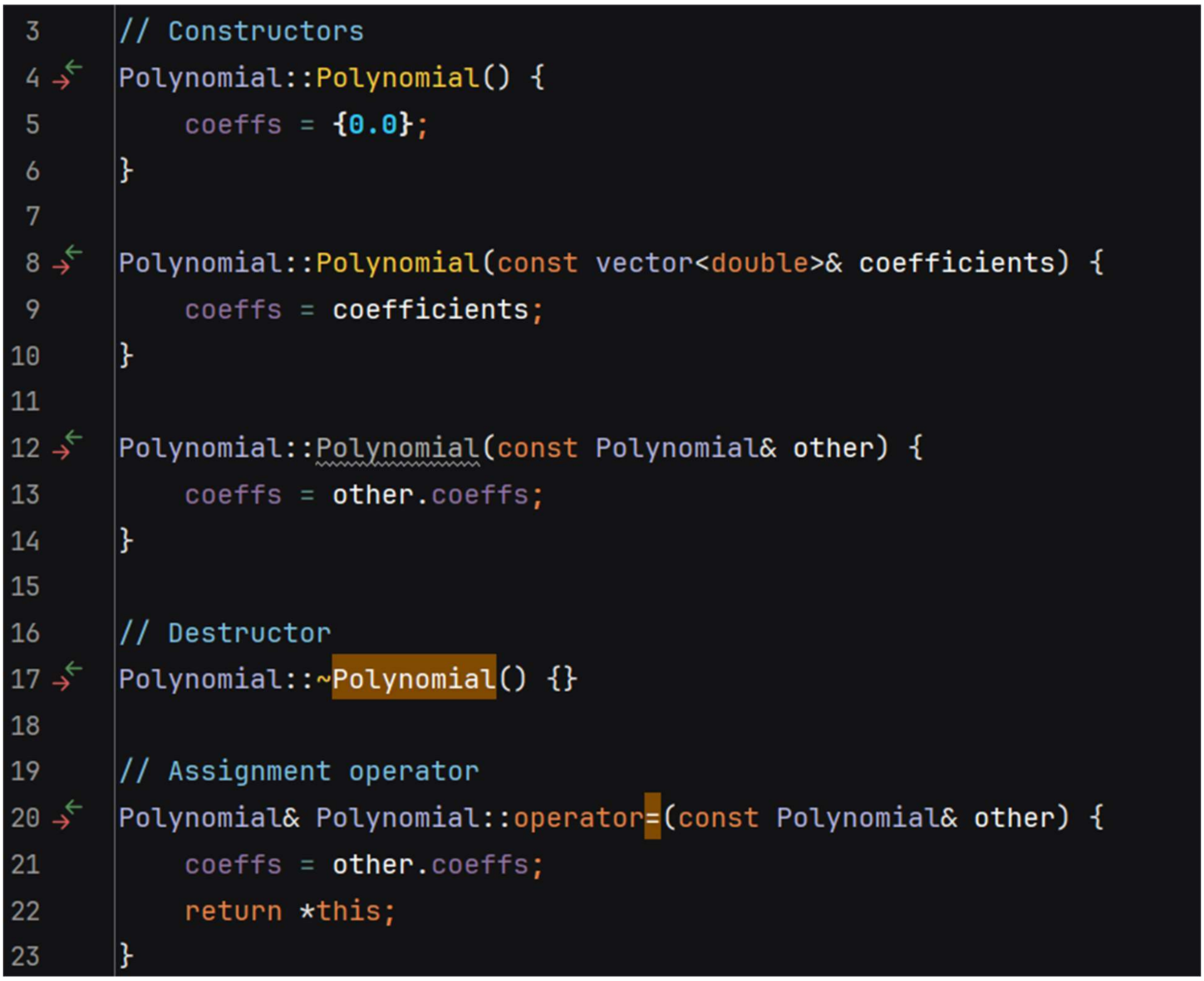
1.5. Final score : 68.25/100

2. Gimini:

   2.1. Prompt:

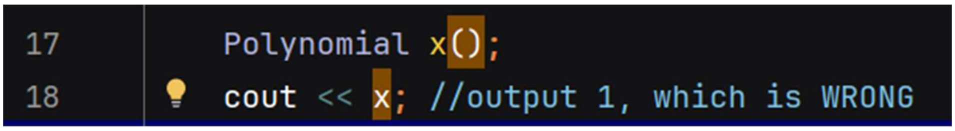
   Same as chat gpt.

   2.2. Constructor, destructor, copy and assignment operator :



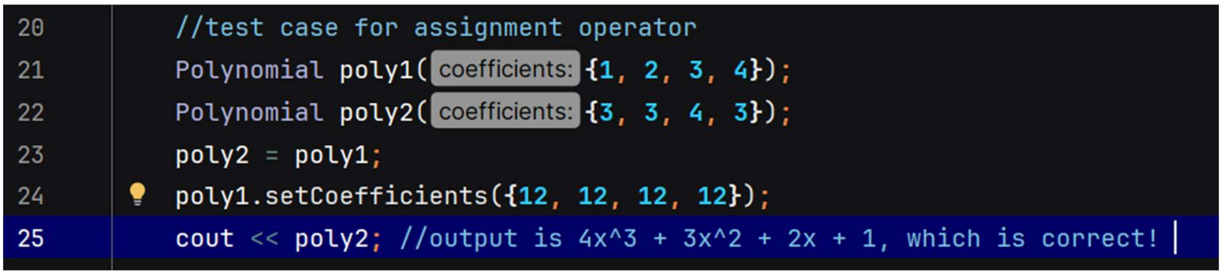
```
// Constructors
Polynomial::Polynomial() {
    coeffs = {0.0};
}

Polynomial::Polynomial(const vector<double>& coefficients) {
    coeffs = coefficients;
}

Polynomial::Polynomial(const Polynomial& other) {
    coeffs = other.coeffs;
}

// Destructor
Polynomial::~Polynomial() {}

// Assignment operator
Polynomial& Polynomial::operator=(const Polynomial& other) {
    coeffs = other.coeffs;
    return *this;
}
```

Test cases:



```
    Polynomial x();
    cout << x; //output 1, which is WRONG
```

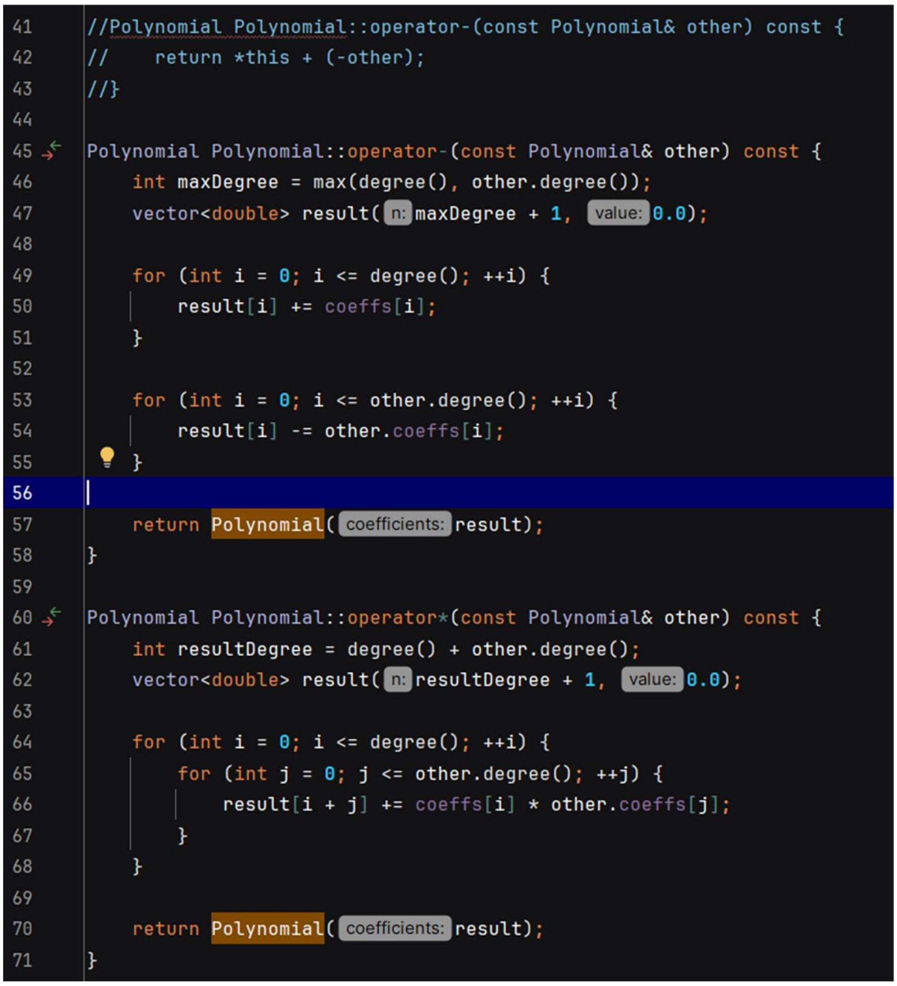
Code looses 3.25 marks for incorrectness.



```cpp
    //test case for assignment operator
    Polynomial poly1(coefficients: {1, 2, 3, 4});
    Polynomial poly2(coefficients: {3, 3, 4, 3});
    poly2 = poly1;
    poly1.setCoefficients({12, 12, 12, 12});
    cout << poly2; //output is 4x^3 + 3x^2 + 2x + 1, which is correct!
```

Applies deep copy ok.

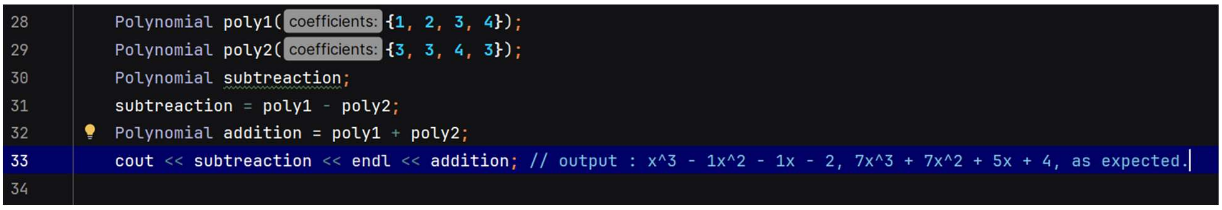
2.3. Addition and subtraction :



```cpp
//Polynomial Polynomial::operator-(const Polynomial& other) const {
//    return *this + (-other);
//}

Polynomial Polynomial::operator-(const Polynomial& other) const {
    int maxDegree = max(degree(), other.degree());
    vector<double> result(n: maxDegree + 1, value: 0.0);

    for (int i = 0; i <= degree(); ++i) {
        result[i] += coeffs[i];
    }

    for (int i = 0; i <= other.degree(); ++i) {
        result[i] -= other.coeffs[i];
    }

    return Polynomial(coefficients: result);
}

Polynomial Polynomial::operator*(const Polynomial& other) const {
    int resultDegree = degree() + other.degree();
    vector<double> result(n: resultDegree + 1, value: 0.0);

    for (int i = 0; i <= degree(); ++i) {
        for (int j = 0; j <= other.degree(); ++j) {
            result[i + j] += coeffs[i] * other.coeffs[j];
        }
    }

    return Polynomial(coefficients: result);
}
```

At first '-' Gimini did the overloading wrongly, and whe I prompted it to fix it, it did. Code looses 4 marks for not even compiling at the beginning.

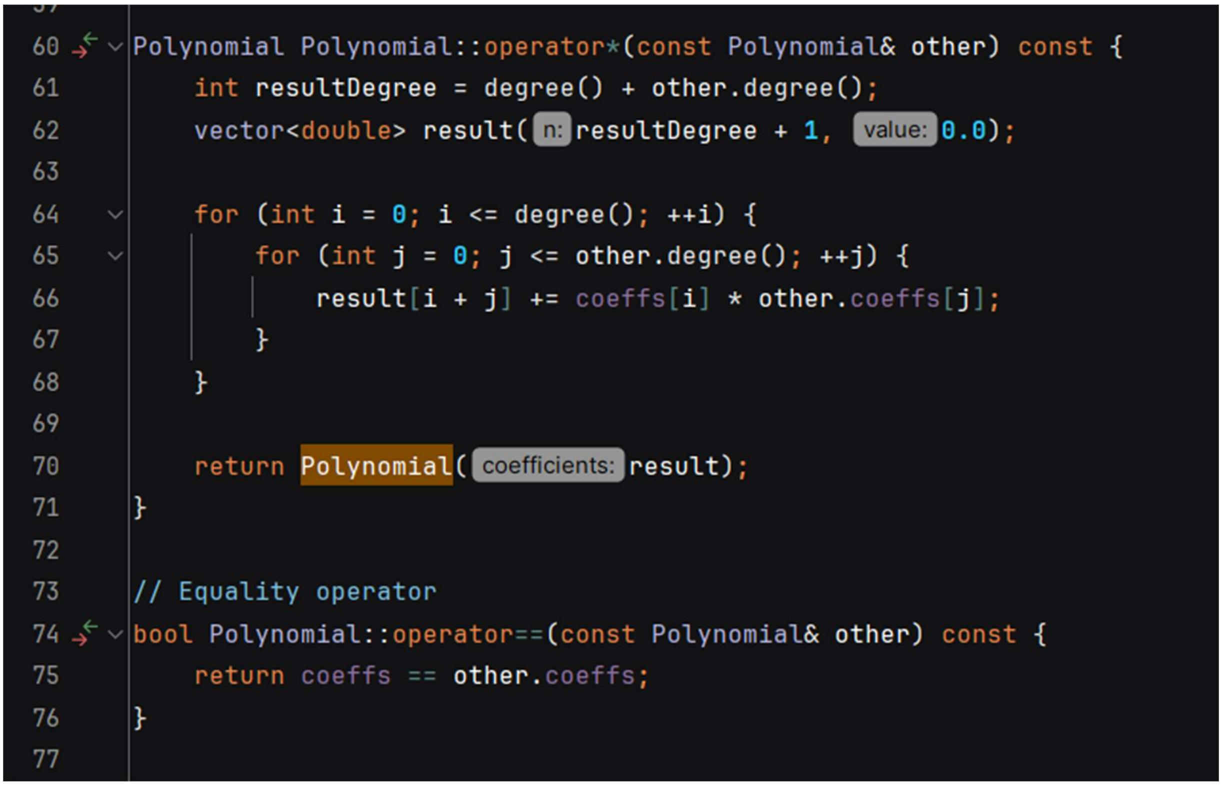
Test cases :



```
    Polynomial poly1(coefficients: {1, 2, 3, 4});
    Polynomial poly2(coefficients: {3, 3, 4, 3});
    Polynomial subtreaction;
    subtreaction = poly1 - poly2;
    Polynomial addition = poly1 + poly2;
    cout << subtreaction << endl << addition; // output : x^3 - 1x^2 - 1x - 2, 7x^3 + 7x^2 + 5x + 4, as expected.
```

Works great.

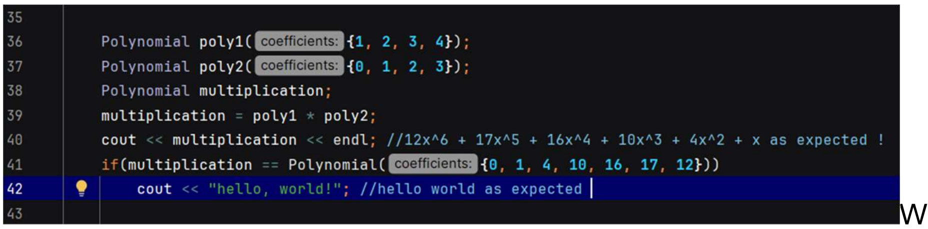
2.4. Multiplication and equal operators :



```
Polynomial Polynomial::operator*(const Polynomial& other) const {
    int resultDegree = degree() + other.degree();
    vector<double> result( n: resultDegree + 1,  value: 0.0);

    for (int i = 0; i <= degree(); ++i) {
        for (int j = 0; j <= other.degree(); ++j) {
            result[i + j] += coeffs[i] * other.coeffs[j];
        }
    }

    return Polynomial( coefficients: result);
}

// Equality operator
bool Polynomial::operator==(const Polynomial& other) const {
    return coeffs == other.coeffs;
}
```

Great coding style!

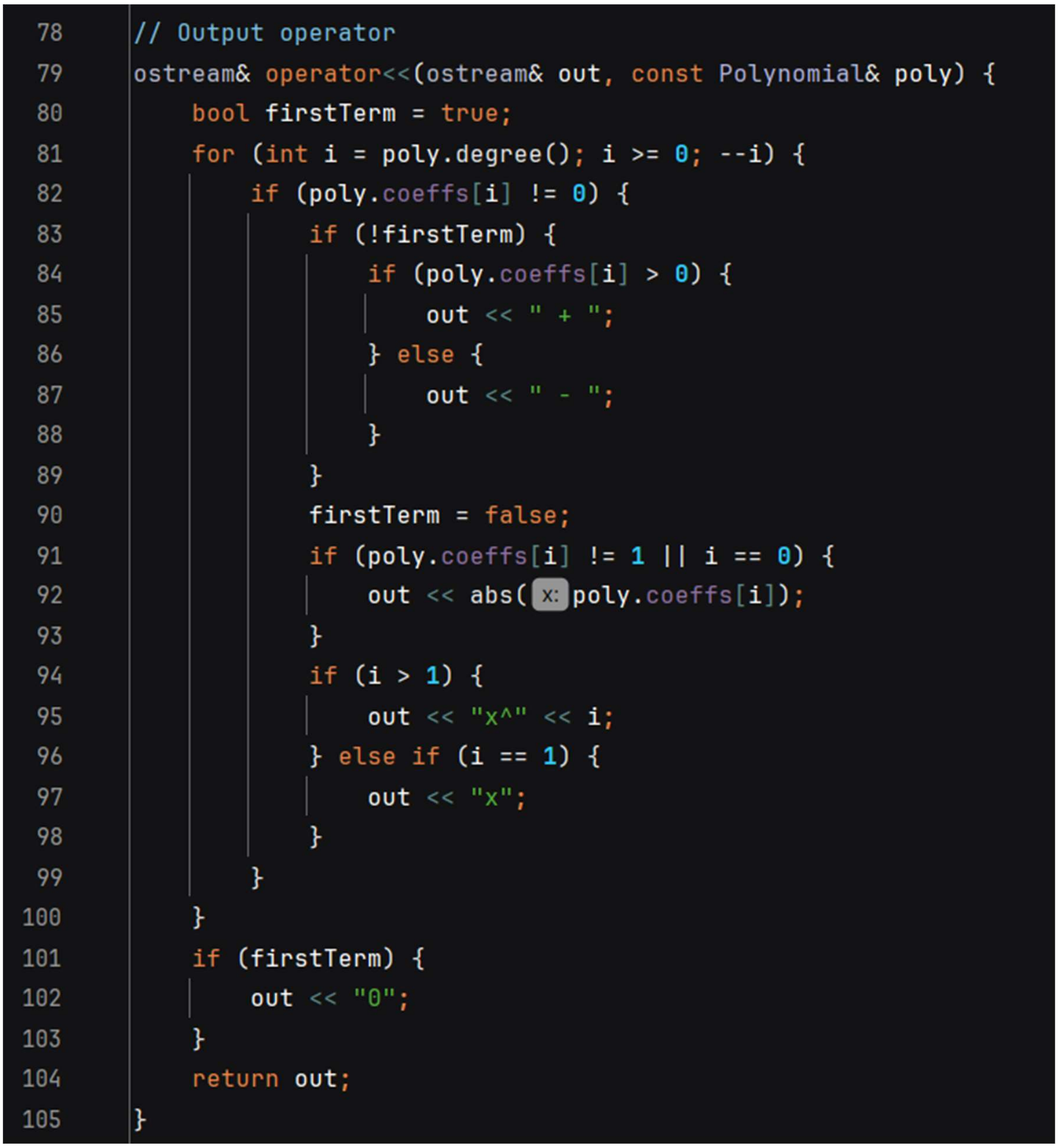
Test cases :



```
    Polynomial poly1(coefficients: {1, 2, 3, 4});
    Polynomial poly2(coefficients: {0, 1, 2, 3});
    Polynomial multiplication;
    multiplication = poly1 * poly2;
    cout << multiplication << endl; //12x^6 + 17x^5 + 16x^4 + 10x^3 + 4x^2 + x as expected !
    if(multiplication == Polynomial(coefficients: {0, 1, 4, 10, 16, 17, 12}))
        cout << "hello, world!"; //hello world as expected 

```

Works great!

2.5. Output operator :

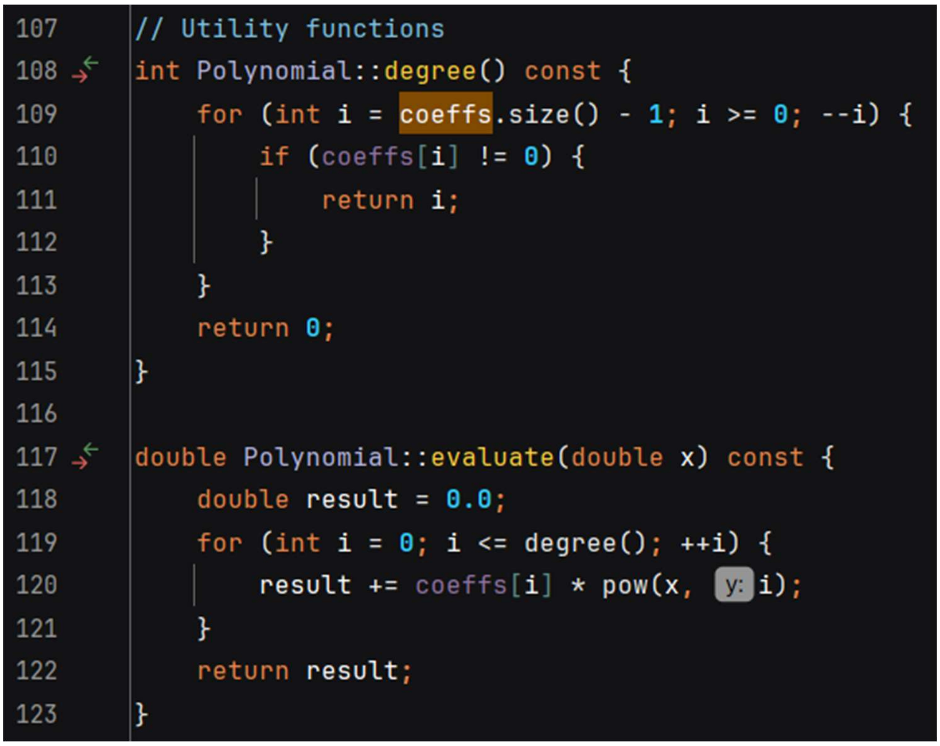


```
// Output operator
ostream& operator<<(ostream& out, const Polynomial& poly) {
    bool firstTerm = true;
    for (int i = poly.degree(); i >= 0; --i) {
        if (poly.coeffs[i] != 0) {
            if (!firstTerm) {
                if (poly.coeffs[i] > 0) {
                    out << " + ";
                } else {
                    out << " - ";
                }
            }
            firstTerm = false;
            if (poly.coeffs[i] != 1 || i == 0) {
                out << abs(x: poly.coeffs[i]);
            }
            if (i > 1) {
                out << "x^" << i;
            } else if (i == 1) {
                out << "x";
            }
        }
    }
    if (firstTerm) {
        out << "0";
    }
    return out;
}
```

Making no comments at all is considered a bad coding practice, code looses 0.25.

2.6. Utility functions

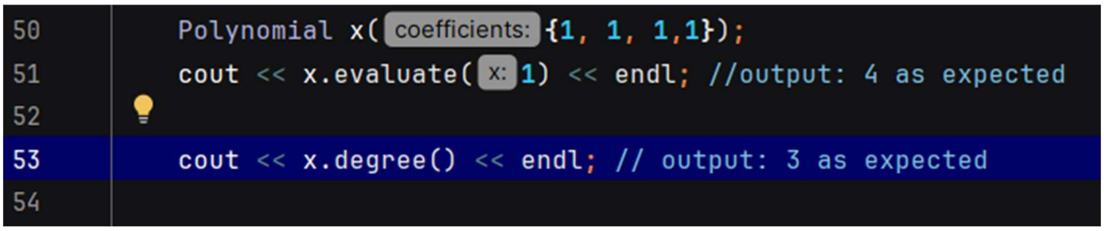
2.6.1. Degree and evaluate :



```
// Utility functions
int Polynomial::degree() const {
    for (int i = coeffs.size() - 1; i >= 0; --i) {
        if (coeffs[i] != 0) {
            return i;
        }
    }
    return 0;
}

double Polynomial::evaluate(double x) const {
    double result = 0.0;
    for (int i = 0; i <= degree(); ++i) {
        result += coeffs[i] * pow(x, y: i);
    }
    return result;
}
```

Using pow function is not the best idea, since using fast power would have been more efficient, code looses 1 mark.

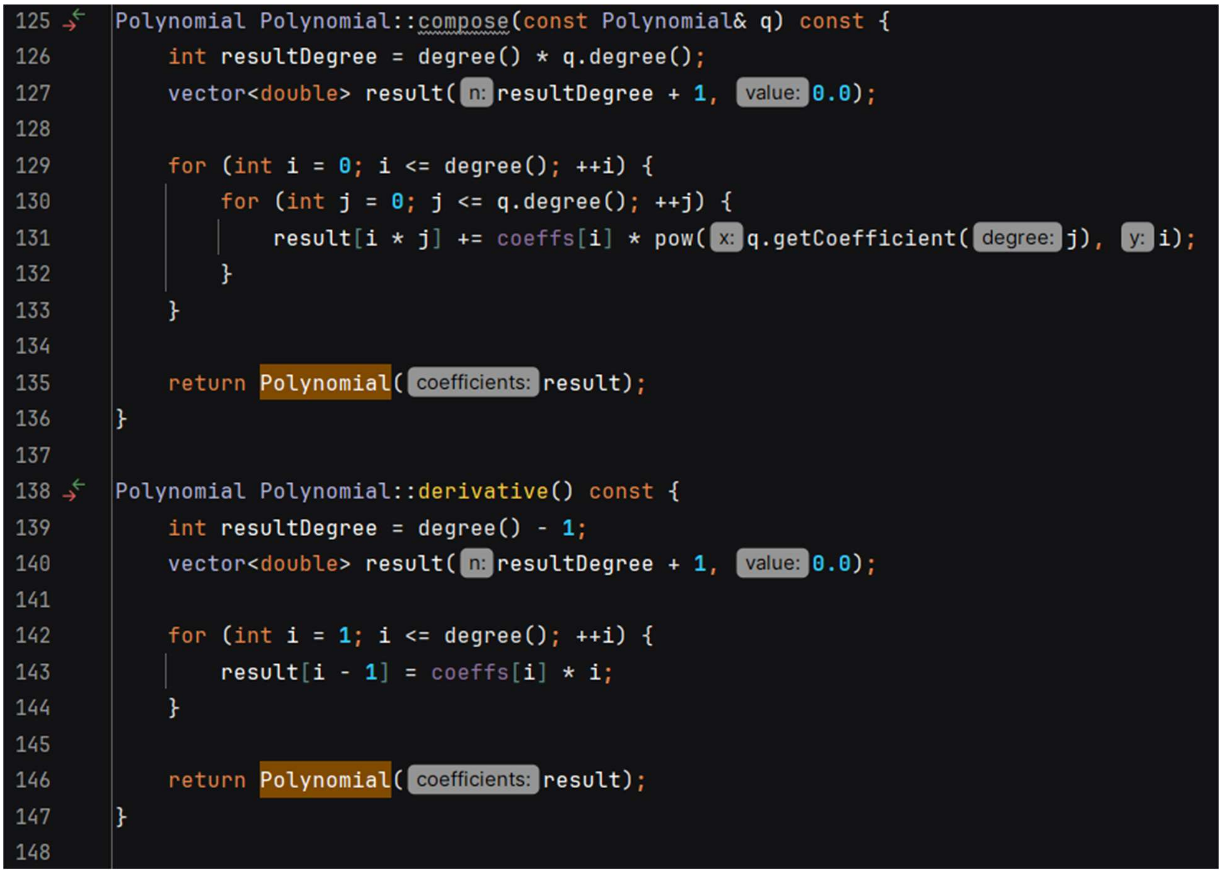
Test cases :



```
    Polynomial x( coefficients: {1, 1, 1,1});
    cout << x.evaluate( x: 1) << endl; //output: 4 as expected

    cout << x.degree() << endl; // output: 3 as expected

```

Works well!

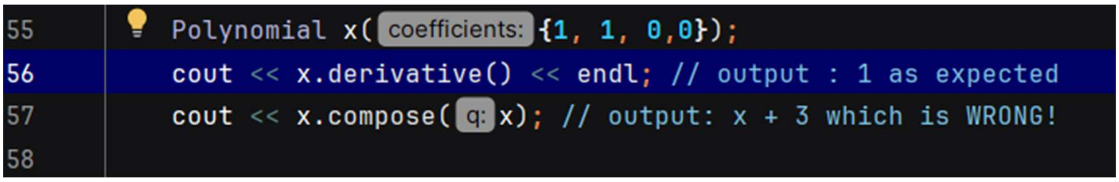
2.6.2. Compose and derivative:



```
Polynomial Polynomial::compose(const Polynomial& q) const {
    int resultDegree = degree() * q.degree();
    vector<double> result( n: resultDegree + 1,  value: 0.0);

    for (int i = 0; i <= degree(); ++i) {
        for (int j = 0; j <= q.degree(); ++j) {
            result[i * j] += coeffs[i] * pow( x: q.getCoefficient( degree: j),  y: i);
        }
    }

    return Polynomial( coefficients: result);
}

Polynomial Polynomial::derivative() const {
    int resultDegree = degree() - 1;
    vector<double> result( n: resultDegree + 1,  value: 0.0);

    for (int i = 1; i <= degree(); ++i) {
        result[i - 1] = coeffs[i] * i;
    }

    return Polynomial( coefficients: result);
}
```

variable name ‘q’ doesn’t really tell anything about what it is, code looses 0.1 marks.

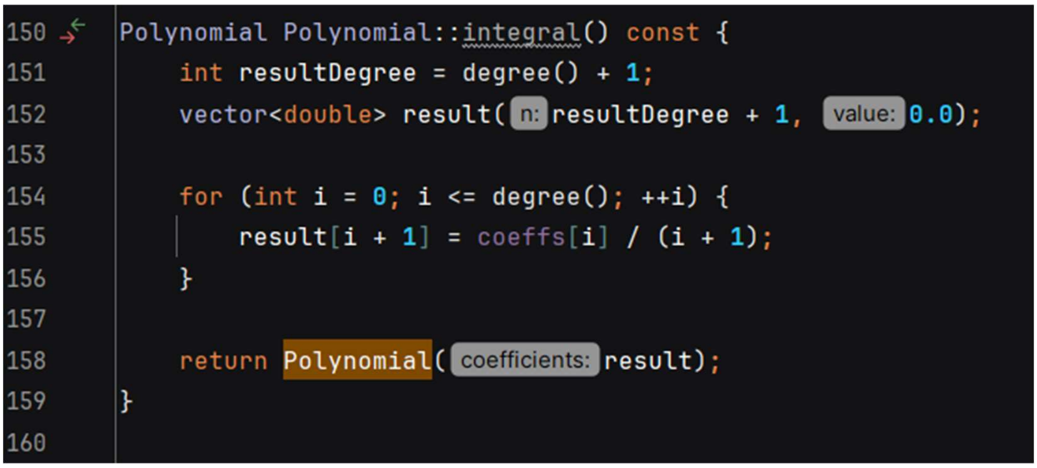
Test cases:



```
Polynomial x( coefficients: {1, 1, 0,0});
cout << x.derivative() << endl; // output : 1 as expected
cout << x.compose( q: x); // output: x + 3 which is WRONG!
```

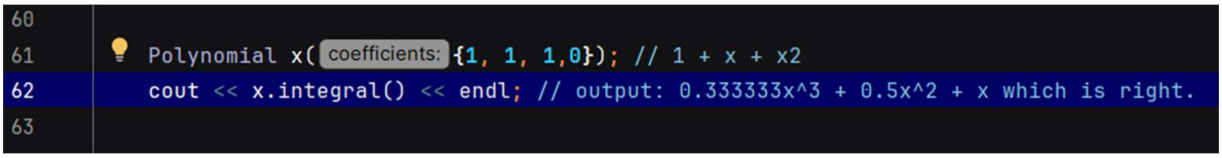
code looses 3.25 marks for incorrectness.

2.6.3. Integral:



```
Polynomial Polynomial::integral() const {
    int resultDegree = degree() + 1;
    vector<double> result(n: resultDegree + 1, value: 0.0);

    for (int i = 0; i <= degree(); ++i) {
        result[i + 1] = coeffs[i] / (i + 1);
    }

    return Polynomial(coefficients: result);
}

```

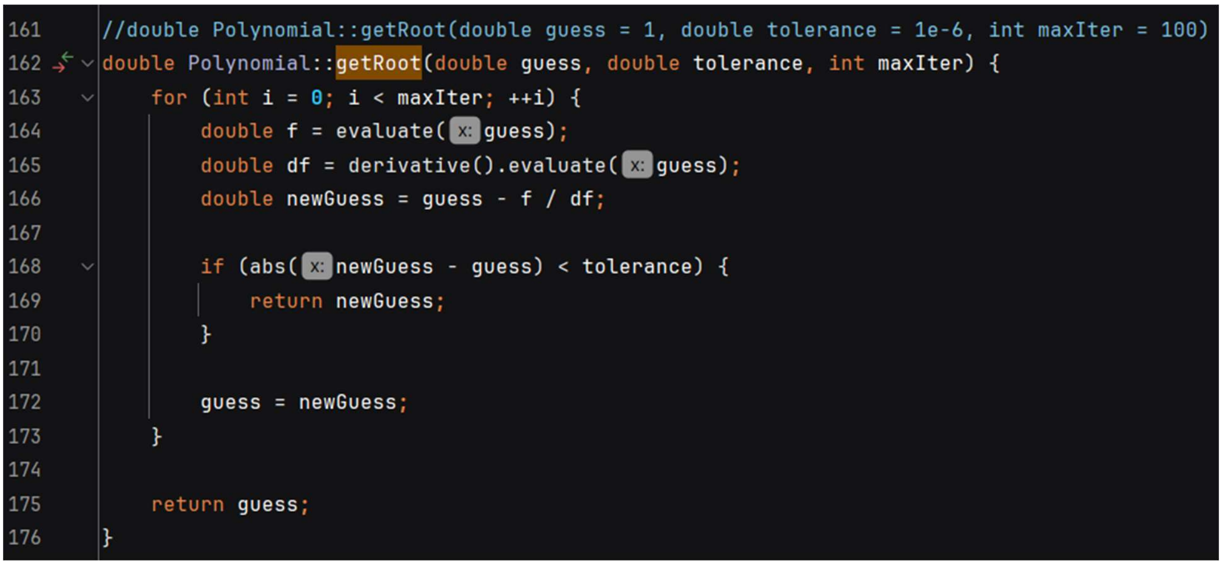
Test cases :



```

    Polynomial x(coefficients: {1, 1, 1,0}); // 1 + x + x2
    cout << x.integral() << endl; // output: 0.333333x^3 + 0.5x^2 + x which is right.

```

Forgetting about the constant at the end makes code loose 0.25 marks.

2.6.4. getRoot:



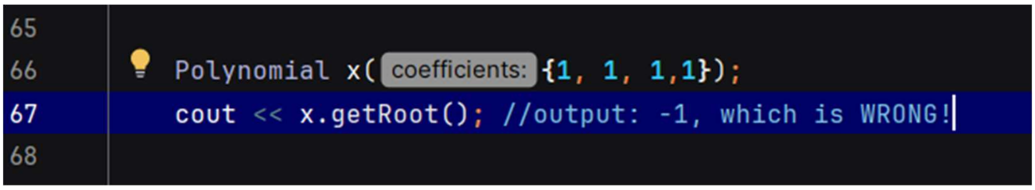
```
161  //double Polynomial::getRoot(double guess = 1, double tolerance = 1e-6, int maxIter = 100)
162  double Polynomial::getRoot(double guess, double tolerance, int maxIter) {
163      for (int i = 0; i < maxIter; ++i) {
164          double f = evaluate( x: guess);
165          double df = derivative().evaluate( x: guess);
166          double newGuess = guess - f / df;
167
168          if (abs( x: newGuess - guess) < tolerance) {
169              return newGuess;
170          }
171
172          guess = newGuess;
173      }
174
175      return guess;
176  }
```

Names like f and df are not meaningful names, maxIter is not pronounceable name, parantathese should have been used in line 166, code looses 0.6

Gimini did a fatal mistake by redefining default argument causing a compilation error.

Code looses 2 mark.

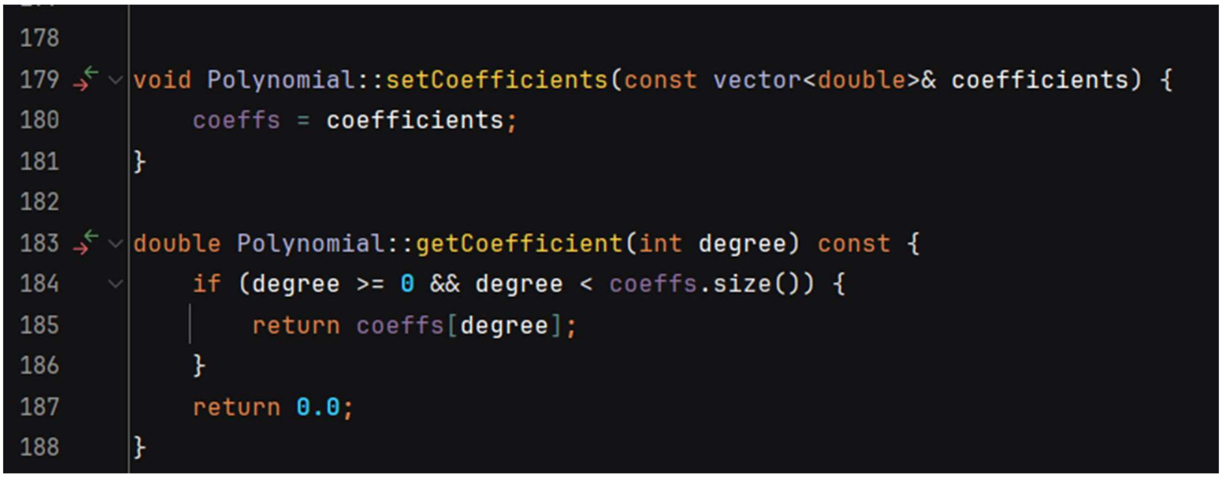
Test case :



```

    Polynomial x( coefficients: {1, 1, 1,1});
    cout << x.getRoot(); //output: -1, which is WRONG!

```

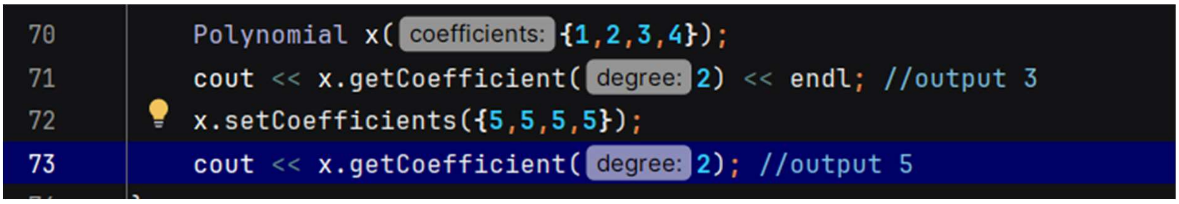
Code looses 2.25 marks for incorrectness.

2.6.5. Getters and setters :



```cpp

void Polynomial::setCoefficients(const vector<double>& coefficients) {
    coeffs = coefficients;
}

double Polynomial::getCoefficient(int degree) const {
    if (degree >= 0 && degree < coeffs.size()) {
        return coeffs[degree];
    }
    return 0.0;
}
```

Test cases :



```cpp
    Polynomial x( coefficients: {1,2,3,4});
    cout << x.getCoefficient( degree: 2) << endl; //output 3
    x.setCoefficients({5,5,5,5});
    cout << x.getCoefficient( degree: 2); //output 5
```

2.7. Final score : 83.05/100

Finally, we can confidently say Gimini has done better in this assignment than chatgpt, 83.05/100 for Gimini and 68.25/100 for chat gpt, Although it wouldn't be fair to say Gimini is better coder than chatgpt from 1 task.

Task 3

Route Academy:

An independent Egyptian education provider whose CEO is Ahmed Bahnasy. It is located in Cairo, but courses are available online as well. It offers learning programs for : (Web Development, Mobile App Development, Data Science, UI/UX Design.)

The duration of each program varies, but it's safe to assume that it won't take more than months.

Requirements to enroll in Route Academy programs typically include a basic understanding of programming, though beginners are also welcome, as some bootcamps are designed for students with little or no experience. Additionally, students should have access to a computer and reliable internet, particularly if taking online courses.

AlMakinah :

is a coding bootcamp based in Cairo, Egypt, aimed at providing practical tech education and preparing students for careers in the tech industry:

Full-time and part-time bootcamps with in-person and online options. And it's located in Cairo, Egypt (Nasr City)

Full-Stack Web Development Bootcamp:

- **Duration**: 12 weeks (full-time).
- **Curriculum**: Covers front-end and back-end web development, HTML, CSS, JavaScript, Node.js, databases, and version control systems like Git.
- **Projects**: Students work on individual and group projects to build a portfolio, which is crucial for job applications.

Fees and conditions are not mentioned but it's known to be affordable.

Misk Academy :

is an educational initiative launched by the Misk Foundation, a non-profit organization founded by Saudi Crown Prince Mohammed bin Salman. The academy is part of Misk's broader efforts to empower Saudi youth and develop the kingdom's workforce by providing cutting-edge education and training in key sectors like technology, media, leadership, and culture.

Location: Primarily based in Riyadh, Saudi Arabia, but offers online learning for students worldwide.

Partnered with institutions like General Assembly to offer coding bootcamps and courses in data science, UX/UI design, and cybersecurity. The programs are designed to be intensive and project-based, preparing students for real-world employment in the tech sector.

Conditions: Many programs target Saudi youth, but online programs and global partnerships also make it accessible to international students. Programs are available for both beginners and experienced professionals looking to upskill.

Fees: Misk Academy often provides scholarships or fully-funded programs, especially for Saudi citizens. Some international courses might have associated fees, but Misk frequently subsidizes them to make education more accessible.

As the students :

Youssef Farid 20230504: chose cybersecurity as it's a reliable job that's hard to replace.

Islam Waleed 20230062: chose the backend track because of its availability and how easy to find a job there.

Mohamed Ali 20230347: chose AI as it's the future of the world and he likes to push it to its maximum.