

AI Written Code Assessment

A1_T2/T3 OOP Dr. Mohammed El Ramly

Loai Hataba

20230553

Abdallah Mohammed

20230231

Hossam Abdelaziz

20230121

Table of Contents

Introduction	2
Parameters	3
Correctness	4
Efficiency	6
Elegance	6
Cleanliness	7
Summary	9
References	10
Appendix	11
Code 1 (ChatGPT)	11
Code 2 (Claude)	22
Test Cases	30
Test Results (Failed)	53
ChatGPT	53
Claude	61
Gtest Code	72
Training Opportunities	111

Introduction

Since humans started walking on this earth they kept innovating, discovering, and utilizing they're surroundings. Just when you think that you've reached the peak of technological and industrial evolution you're surprised with more.

In 1829 when the sewing machine was made it was heavily attacked because a lot of people worked in sewing clothes by hand which meant it would run them out of jobs. And the machine did in fact was much faster and more efficient and cost less, so businesses replaced the manual sewers with them; Yet it opened countless opportunities for people to use them. The machine took out the tedious effort that was exerted in order to make only one piece of clothing.

Similarly, the new rise of Artificial Intelligence technology is becoming more rapid, and it will surely take a lot of people out of their jobs, yet it will open a lot of other jobs. AI's number one purpose is to get rid of tedious routine work.

In spite of what has been stated we're trying to answer the question "will AI replace programmers", we have asked two different ai models to implement a polynomial class in C++ to assess their code quality, results, elegance, and their user experience.

The test was done using a 100 different test cases being as inclusive as possible including polynomials with complex roots and degrees up to a 1000!

The testing was done assessing the models' ability to write codes implementing a class of a polynomial that had 15 different methods including:

- Assignment operator
- Arithmetic operators
- Equality operator
- Output operator
- Degree
- Evaluate
- Compose
- Derivative
- Indefinite Integral
- Definite Integral
- Get Root
- Get coefficient
- Set coefficient

Parameters

- **AI Models Used:** Chat-GPT (4o mini) / Claude (3.5 Sonnet)
- **Initial Prompt:** Both Ai models were prompted with the same prompt

“You’re a senior programmer with a lot of knowledge in C++, I want you to complete this class ... (provide the header file)”

One of the most important aspects of a good prompt is the persona, where you tell the ai model who it should be so it has a clearer path of where to search and which information to provide; so by telling the models that they’re programmers who are knowledgeable in C++ that would potentially help the results be better than just asking them to complete the class.

- **Modifications/Reprompting:** Both Models have been prompted more than once (with nearly the same prompts) to get somewhat of an acceptable result that matches the desired outcome.

1. Correctness

❖ Chat-GPT:

- Overall, the code was acceptable, the code works well in most normal cases in most operations, and the code seems to cover a lot of edge cases.
- Yet when met with cases such as polynomials with complex it fails.
- The get root function was a complete disaster even after a couple of tries to correct its code the model completely ruined the code making it unusable.
- The model didn't add any type of input checking or safeguarding the code, whenever an invalid input the program either runs forever and needs to be force closed or there's some kind of segmentation error or logical that forces the program to completely quit.
- The model could sometimes slip up and forget very basic and easy stuff that would seem not that important but every detail matters, and sometimes it ignores some of your requests.
 - Example: when provided the model to make the get root value give back multiple roots if exists and set default values for the parameters it failed to comply with the requests and yielded the code nearly as it is, even a bit worse, not adding the multiple roots feature nor setting the default parameters and removing the input asking the user for the guess and tolerance values, it had to be reminded again that it didn't do what it was asked to do.
- The user experience was a bit annoying having to correct the ai continuously to get something that works.
- The code after being tested on a 100 different test cases failed at 36 test cases (check appendix for details)
- Most of the failed tests were regarding the getRoot function, the method was very sloppy and didn't even yield every possible root let alone a single root not being correct either.

[FAILED] PolynomialTest.TestCase29	[FAILED] PolynomialTest.Test76	[FAILED] PolynomialTest.Test89
[FAILED] PolynomialTest.Test35	[FAILED] PolynomialTest.Test77	[FAILED] PolynomialTest.Test90
[FAILED] PolynomialTest.Test61	[FAILED] PolynomialTest.Test78	[FAILED] PolynomialTest.Test91
[FAILED] PolynomialTest.Test62	[FAILED] PolynomialTest.Test80	[FAILED] PolynomialTest.Test92
[FAILED] PolynomialTest.Test63	[FAILED] PolynomialTest.Test81	[FAILED] PolynomialTest.Test93
[FAILED] PolynomialTest.Test66	[FAILED] PolynomialTest.Test82	[FAILED] PolynomialTest.Test94
[FAILED] PolynomialTest.Test67	[FAILED] PolynomialTest.Test83	[FAILED] PolynomialTest.Test95
[FAILED] PolynomialTest.Test68	[FAILED] PolynomialTest.Test84	[FAILED] PolynomialTest.Test96
[FAILED] PolynomialTest.Test69	[FAILED] PolynomialTest.Test85	[FAILED] PolynomialTest.Test97
[FAILED] PolynomialTest.Test70	[FAILED] PolynomialTest.Test86	[FAILED] PolynomialTest.Test98
[FAILED] PolynomialTest.Test73	[FAILED] PolynomialTest.Test87	[FAILED] PolynomialTest.Test99
[FAILED] PolynomialTest.Test74	[FAILED] PolynomialTest.Test88	[FAILED] PolynomialTest.Test100

- **Rating:** 70%

❖ Claude:

- The code correctly implements polynomial operations and provides a comprehensive set of features for manipulation and evaluation
- As for edge cases, the code removes leading zeros when constructing polynomials, ensuring that the polynomial degree and operations don't get skewed by unnecessary terms. However, input validation for edge cases like empty coefficients and division by zero in Newton's method should be handled more thoroughly.
- Most functions handled the tests well, except for the getRoot function, which struggled. This is reasonable, as it uses Newton's method, an older technique that becomes less accurate as the polynomial's degree increases.
- **Error Handling:** The code lacks robust error handling in some cases (e.g., division by zero in Newton's method, input validation). Adding checks would improve reliability, especially for edge cases.
- While the model occasionally forgets to complete certain tasks, it has never ignored a prompt entirely
 - For example, when asked to create a menu for operations, it only generated one for 8 operations and left out the remaining ones. Additionally, when prompted to edit the behavior of a specific function—such as its nature or return type—the model successfully made the changes to the function itself but neglected to update the menu or the header file. As a result, I frequently had to remind it to synchronize the menu with the latest function updates.
- The code failed at 30 test cases (Check appendix for details)
- The code also failed mostly in the get Root function, it did yield different multiple roots; yet not correct in a lot of cases, but it overall did better than GPT's code.

[FAILED] PolynomialTest.Test35	[FAILED] PolynomialTest.Test81	[FAILED] PolynomialTest.Test91
[FAILED] PolynomialTest.Test67	[FAILED] PolynomialTest.Test82	[FAILED] PolynomialTest.Test92
[FAILED] PolynomialTest.Test68	[FAILED] PolynomialTest.Test83	[FAILED] PolynomialTest.Test93
[FAILED] PolynomialTest.Test69	[FAILED] PolynomialTest.Test84	[FAILED] PolynomialTest.Test94
[FAILED] PolynomialTest.Test70	[FAILED] PolynomialTest.Test85	[FAILED] PolynomialTest.Test95
[FAILED] PolynomialTest.Test73	[FAILED] PolynomialTest.Test86	[FAILED] PolynomialTest.Test96
[FAILED] PolynomialTest.Test74	[FAILED] PolynomialTest.Test87	[FAILED] PolynomialTest.Test97
[FAILED] PolynomialTest.Test76	[FAILED] PolynomialTest.Test88	[FAILED] PolynomialTest.Test98
[FAILED] PolynomialTest.Test77	[FAILED] PolynomialTest.Test89	[FAILED] PolynomialTest.Test99
[FAILED] PolynomialTest.Test78	[FAILED] PolynomialTest.Test90	[FAILED] PolynomialTest.Test100

- **Rating:** 85%

2. Efficiency

- **Time Complexity:** Both of the models had $O(n)$ time in most of the methods averaging for about 4-8 microseconds (using chrono library) with the longest being $O(n*m)$ where there's two inputs (composition)
 - **Space Complexity:** Again, both of them used minimal storage where they just used variables and vectors.
 - Both Models achieved very well in terms of speed and resource allocation in the best ways possible
 - **Rating: 100%** (Both)
-

3. Elegance

❖ Chat-GPT:

- Code elegance is definitely one of gpt's strengths where using its huge training sets it figures out the best most elegant solution for a problem.
- The code overall didn't take up a lot of coding space with a total of 470 lines of code (very acceptable) with some methods only containing one line of execution.
- The model uses brilliant and efficient algorithms and techniques to solve problems and in as little coding space as it can possibly can, it achieves its goal with minimal, clear lines of code.
- **Rating: 85%**

❖ Claude:

- The code balances functionality and simplicity. While operations like differentiation and integration are user-friendly, Newton's root-finding method can be somewhat overwhelming.
- Claude efficiently managed the code space, completing the task in just 356 lines of code.
- **Rating: 85%**

4. Cleanliness

❖ Chat-GPT:

- Despite the class being relatively not small, GPT wrote most of the code in clean and very human readable way with proper variable naming conventions and indentations.
- The model uses comments quite on point where it knows where a comment would be necessary to explain that line of code.
- A plus would be the color coding and code snapshots that the model uses to present the code, and also providing very detailed explanations under the code for what every part of the code does.
- Another thing we really liked about it also was that if something in the code was still unclear, we could easily ask the model, and it would walk us step by step on what every line does exactly.

Updated `syntheticDivide()` Function

```
cpp Copy code

// Synthetic division: Removes a found root by dividing the polynomial by (x - root)
Polynomial Polynomial::syntheticDivide(double root) {
    // Retrieve all coefficients of the polynomial
    const std::vector<double>& coeffs = getCoefficients(); // This method must return all

    std::vector<double> newCoeffs(coeffs.size() - 1); // Reduced polynomial will have one

    newCoeffs[0] = coeffs[0]; // Start with leading coefficient
    for (int i = 1; i < coeffs.size() - 1; ++i) {
        newCoeffs[i] = coeffs[i] + newCoeffs[i - 1] * root;
    }

    return Polynomial(newCoeffs); // Return reduced polynomial
}

// Method to get all coefficients (make sure this exists)
const std::vector<double>& Polynomial::getCoefficients() const {
    return coeffs; // Assuming 'coeffs' is the internal representation of the polynomial's
}
```

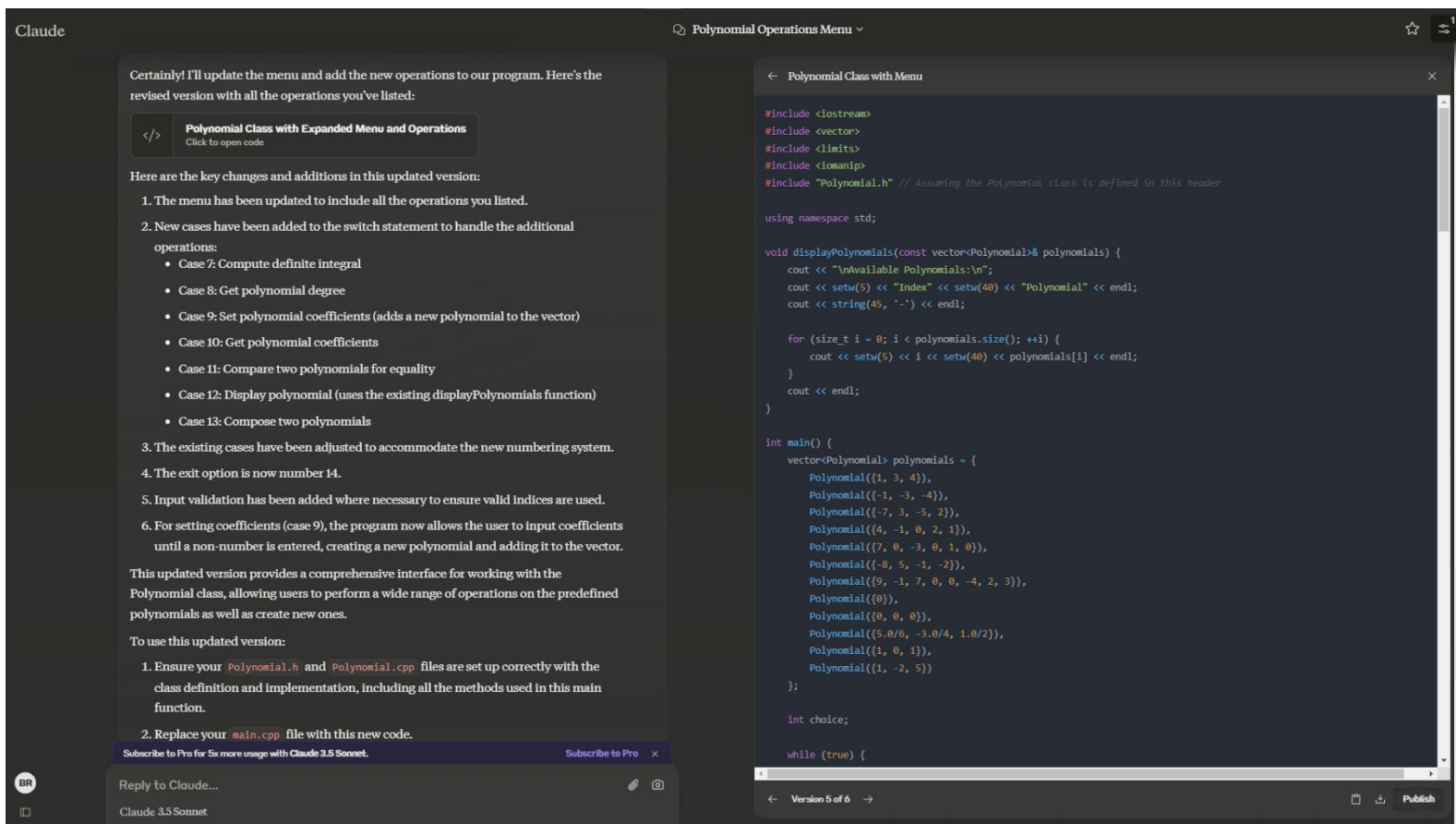
Key Points:

- `getCoefficients()`: You need to ensure that the `getCoefficients()` method returns a reference to the vector containing all the polynomial coefficients.
- **Indexing in the loop**: The loop runs from `i = 1` to `coeffs.size() - 1` because the reduced polynomial has one degree less than the original one.

- **Rating: 90%**

❖ Claude:

- **Code structure:** The code is structured well with clear separation of header and implementation. It follows common C++ practices with proper use of encapsulation and operator overloading.
- **Naming conventions:** Most function and variable names are descriptive (e.g., `degree()`, `evaluate()`, `derivative()`), making the code easier to follow.
- **Code comments:** The code could benefit from more comments, especially for more complex parts like the composition, root-finding, and evaluation algorithms. This would make it easier for future readers to understand the reasoning behind certain design choices.
- Additionally, it is important to note that Claude is designed for complex tasks, such as code generation, which is reflected in its user interface by separating the code from its explanations.



Rating: 70%

5. Summary:

❖ Chat-GPT:

- Overall, the model did a fairly decent job in completing roughly the main idea of having a polynomial class with several methods.
- Yet aside from simple subtraction, addition, and simple integration the code is vulnerable to bugs and total program shutdowns.
- The experience using the model and having a conversation with it was definitely not an easy task, perhaps a bit frustrating when you're literally asking the model for something and it completely ignores it and responds with "of course here's your updated code" and the code isn't changed a single bit; another point is when the model gives you a fix for a problem but it completely obliterates another part of the code, where sometimes maybe the model can fix it or it may be can't and then you have no choice but to pray that you had some backup for your previous code.
- We would give the model an overall rating of **75%**

❖ Claude:

- The implementation features a well-encapsulated, object-oriented design. It effectively handles most edge cases, particularly concerning polynomial degree and coefficients.
- There is room for improvement in error handling, especially related to root-finding and user input. Additionally, increasing the number of comments and documentation would help clarify the more complex sections of the code.
- During my interaction with the model, the only Aggravating aspect was when it made edits to a function's code without synchronizing those changes in the menu or header. Aside from this issue, the model demonstrated a strong understanding of the prompts and provided efficient responses. Overall, it was a great experience.
- Would give an overall rating of **85%**

It has been concluded that AI in abstraction won't replace humans (Programmers) that easily, for now at least. To use AI to program you would still need solid knowledge of what you're asking the model to do so you can notice any bugs or errors that may arise from the code, on the models themselves it's written that you should check the information that the model provides because it may not always be accurate.

ChatGPT can make mistakes. Check important info.

AI models excel well when it comes to explaining complex concepts and breaking them into easier terminologies and using simple analogies until you get the information which is quite impressive.

References:

- Chat-GPT conversation 1(class implementation): <https://chatgpt.com/share/670fd569-b568-8011-8501-c96948df8a49>
- Chat-GPT conversation 2(menu): <https://chatgpt.com/share/670fd5cd-ee64-8011-98e0-78e1dd90b984>
- Claude conversation 1: <https://claude.ai/chat/9eb78b11-cfd2-475a-8ed1-b93ae5c3155e>
- Claude conversation 2: <https://claude.ai/chat/70a67b7d-6719-4ba7-b29a-60b2665016fa>
- Claude conversation 3: <https://claude.ai/chat/4ddd0f76-8837-4db3-b5bc-3b30d3c1d2e0>
- Claude conversation 4(get root fix): <https://claude.ai/chat/80922f3e-7e1c-4e0b-8654-d6357522f834>

Appendix

Code 1 (ChatGPT)

```
#include "Polynomial.h"
#include <vector>
#include <iostream>
#include <chrono>
using namespace std;

Polynomial::Polynomial() : coeffs(1, 0.0) {}
// Constructor with coefficients
Polynomial::Polynomial(const vector<double>& coefficients) : coeffs(coefficients) {}
// Copy constructor
Polynomial::Polynomial(const Polynomial& other) : coeffs(other.coeffs) {}
// Destructor
Polynomial::~Polynomial() {}
// 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 {
    vector<double> result(max(coeffs.size(), other.coeffs.size()), 0.0);
    for (size_t i = 0; i < result.size(); ++i) {
        if (i < coeffs.size()) result[i] += coeffs[i];
        if (i < other.coeffs.size()) result[i] += other.coeffs[i];
    }
    return Polynomial(result);
}
// Subtraction operator
Polynomial Polynomial::operator-(const Polynomial& other) const {
    vector<double> result(max(coeffs.size(), other.coeffs.size()), 0.0);
    for (size_t i = 0; i < result.size(); ++i) {
        if (i < coeffs.size()) result[i] += coeffs[i];
        if (i < other.coeffs.size()) result[i] -= other.coeffs[i];
    }
}
```

```

    return Polynomial(result);
}

// Multiplication operator
Polynomial Polynomial::operator*(const Polynomial& other) const {
    vector<double> result(coeffs.size() + other.coeffs.size() - 1, 0.0);
    for (size_t i = 0; i < coeffs.size(); ++i) {
        for (size_t j = 0; j < other.coeffs.size(); ++j) {
            result[i + j] += coeffs[i] * other.coeffs[j];
        }
    }
    return Polynomial(result);
}

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

// Output operator (friend function)
ostream& operator<<(ostream& out, const Polynomial& poly) {
    for (int i = poly.coeffs.size() - 1; i >= 0; --i) {
        if (poly.coeffs[i] != 0) {
            if (i != poly.coeffs.size() - 1 && poly.coeffs[i] > 0) out << "+";
            out << poly.coeffs[i];
            if (i > 0) out << "x";
            if (i > 1) out << "^" << i;
            out << " ";
        }
    }
    return out;
}

// Utility functions
int Polynomial::degree() const {
    return coeffs.size() - 1;
}

// Evaluate the polynomial at x
double Polynomial::evaluate(double x) const {
    double result = 0;
    for (int i = coeffs.size() - 1; i >= 0; --i) {
        result = result * x + coeffs[i];
    }
    return result;
}

```

```

}
// Derivative of the polynomial
Polynomial Polynomial::derivative() const {
    if (coeffs.size() <= 1) return Polynomial();
    vector<double> deriv(coeffs.size() - 1);
    for (size_t i = 1; i < coeffs.size(); ++i) {
        deriv[i - 1] = coeffs[i] * i;
    }
    return Polynomial(deriv);
}
// Integral of the polynomial (indefinite integral, constant term is 0)
Polynomial Polynomial::integral() const {
    vector<double> integ(coeffs.size() + 1, 0.0);
    for (size_t i = 0; i < coeffs.size(); ++i) {
        integ[i + 1] = coeffs[i] / (i + 1);
    }
    return Polynomial(integ);
}
// Definite integral from x1 to x2
double Polynomial::integral(double x1, double x2) const {
    Polynomial integ = integral();
    return integ.evaluate(x2) - integ.evaluate(x1);
}
// Get Root
double Polynomial::getRoot(double guess, double tolerance, int maxIter) {

    Polynomial deriv = derivative();
    for (int i = 0; i < maxIter; ++i) {
        double f_val = evaluate(guess);
        double f_deriv = deriv.evaluate(guess);
        if (abs(f_val) < tolerance) return guess;
        guess -= f_val / f_deriv;
    }
    return guess;
}
// Get coefficient of a specific degree
double Polynomial::getCoefficient(int degree) const {
    if (degree < 0 || degree >= coeffs.size()) return 0.0;
    return coeffs[degree];
}

```

```

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

// Composition of polynomials
Polynomial Polynomial::compose(const Polynomial& q) const {
    Polynomial result({0});
    Polynomial temp({1});
    for (size_t i = 0; i < coeffs.size(); ++i) {
        result = result + temp * Polynomial({coeffs[i]});
        temp = temp * q;
    }
    return result;
}

void displayPolynomialMenu() {
    cout << "\n--- Polynomial Menu ---\n";
    cout << "1. Add two polynomials\n";
    cout << "2. Subtract two polynomials\n";
    cout << "3. Multiply two polynomials\n";
    cout << "4. Evaluate polynomial\n";
    cout << "5. Get polynomial derivative\n";
    cout << "6. Get polynomial integral\n";
    cout << "7. Compute definite integral\n";
    cout << "8. Get polynomial degree\n";
    cout << "9. Set polynomial coefficients\n";
    cout << "10. Get polynomial coefficients\n";
    cout << "11. Compare two polynomials for equality\n";
    cout << "12. Display polynomial\n";
    cout << "13. Compose two polynomials\n";
    cout << "14. Get root using Newton's method\n"; // Added menu option
    cout << "15. Exit\n"; // Updated exit option
}

vector<double> inputCoefficients() {
    int degree;
    cout << "Enter the degree of the polynomial: ";
    cin >> degree;

    vector<double> coefficients(degree + 1);
    cout << "Enter coefficients (constant term first): ";
    for (int i = 0; i <= degree; ++i) {

```

```

        cin >> coefficients[i];
    }
    return coefficients;
}

int main() {
    int choice;
    do {
        displayPolynomialMenu();
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                // Addition
                cout << "Input coefficients for the first polynomial:\n";
                Polynomial p1(inputCoefficients());
                cout << "Input coefficients for the second polynomial:\n";
                Polynomial p2(inputCoefficients());
                // Start timing
                auto start = chrono::high_resolution_clock::now();
                // Perform the addition
                Polynomial result = p1 + p2;
                // End timing
                auto end = chrono::high_resolution_clock::now();
                auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
                cout << "Result of addition: " << result << endl;
                cout << "Time taken for addition: " << duration.count() << " microseconds" <<
endl;
                break;
            }
            case 2: {
                // Subtraction
                cout << "Input coefficients for the first polynomial:\n";
                Polynomial p1(inputCoefficients());
                cout << "Input coefficients for the second polynomial:\n";
                Polynomial p2(inputCoefficients());
                // Start timing
                auto start = chrono::high_resolution_clock::now();
                // Perform the subtraction
                Polynomial result = p1 - p2;

```



```

        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Result of subtraction: " << result << endl;
        cout << "Time taken for subtraction: " << duration.count() << " microseconds"
<< endl;

        break;
    }
    case 3: {
        // Multiplication
        cout << "Input coefficients for the first polynomial:\n";
        Polynomial p1(inputCoefficients());
        cout << "Input coefficients for the second polynomial:\n";
        Polynomial p2(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Perform the multiplication
        Polynomial result = p1 * p2;
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Result of multiplication: " << result << endl;
        cout << "Time taken for multiplication: " << duration.count() << "
microseconds" << endl;

        break;
    }
    case 4: {
        // Evaluation
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        double x;
        cout << "Enter value of x: ";
        cin >> x;
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Perform the evaluation
        double result = p.evaluate(x);
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Polynomial evaluated at " << x << ": " << result << endl;
    }
}

```

```

        cout << "Time taken for evaluation: " << duration.count() << " microseconds"
<< endl;
        break;
    }
    case 5: {
        // Derivative
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Compute the derivative
        Polynomial deriv = p.derivative();
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Derivative: " << deriv << endl;
        cout << "Time taken for derivative: " << duration.count() << " microseconds"
<< endl;
        break;
    }
    case 6: {
        // Indefinite integral
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Compute the integral
        Polynomial integ = p.integral();
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Integral: " << integ << endl;
        cout << "Time taken for integral: " << duration.count() << " microseconds" <<
endl;
        break;
    }
    case 7: {
        // Definite integral
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        double x1, x2;
        cout << "Enter lower limit of integration: ";

```

```

        cin >> x1;
        cout << "Enter upper limit of integration: ";
        cin >> x2;
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Compute the definite integral
        double result = p.integral(x1, x2);
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Definite integral from " << x1 << " to " << x2 << ": " << result <<
endl;
        cout << "Time taken for definite integral: " << duration.count() << "
microseconds" << endl;
        break;
    }
    case 8: {
        // Degree
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Get the degree
        int degree = p.degree();
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Degree of polynomial: " << degree << endl;
        cout << "Time taken to get degree: " << duration.count() << " microseconds" <<
endl;
        break;
    }
    case 9: {
        // Set coefficients
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        cout << "Setting new coefficients for this polynomial.\n";
        p.setCoefficients(inputCoefficients());
        cout << "Coefficients set successfully.\n";
        break;
    }
    case 10: {

```

```

        // Get coefficient
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        int degree;
        cout << "Enter degree to get coefficient: ";
        cin >> degree;
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Get coefficient
        double coeff = p.getCoefficient(degree);
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Coefficient of degree " << degree << ": " << coeff << endl;
        cout << "Time taken to get coefficient: " << duration.count() << "
microseconds" << endl;
        break;
    }
    case 11: {
        // Compare equality
        cout << "Input coefficients for the first polynomial:\n";
        Polynomial p1(inputCoefficients());
        cout << "Input coefficients for the second polynomial:\n";
        Polynomial p2(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Compare equality
        bool equal = (p1 == p2);
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Polynomials are " << (equal ? "equal" : "not equal") << endl;
        cout << "Time taken for comparison: " << duration.count() << " microseconds"
<< endl;
        break;
    }
    case 12: {
        // Display polynomial
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        // Start timing

```

```

        auto start = chrono::high_resolution_clock::now();
        // Display the polynomial
        cout << p << endl;
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Time taken to display polynomial: " << duration.count() << "
microseconds" << endl;
        break;
    }
    case 13: {
        // Compose two polynomials
        cout << "Input coefficients for the first polynomial:\n";
        Polynomial p1(inputCoefficients())
        cout << "Input coefficients for the second polynomial:\n";
        Polynomial p2(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Compose the polynomials
        Polynomial result = p1.compose(p2);
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Composition result: " << result << endl;
        cout << "Time taken for composition: " << duration.count() << " microseconds"
<< endl;
        break;
    }
    case 14: { // Case for getting the root using Newton's method
        cout << "Input coefficients for the polynomial:\n";
        Polynomial p(inputCoefficients());
        // Start timing
        auto start = chrono::high_resolution_clock::now();
        // Find the root using Newton's method
        double root = p.getRoot();
        // End timing
        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        cout << "Root found: " << root << endl;
        cout << "Time taken to find root: " << duration.count() << " microseconds" <<
endl;
        break;
    }

```

```
    }  
    case 15: { // Update exit case number  
        // Exit  
        cout << "Exiting...\n";  
        break;  
    }  
    default:  
        cout << "Invalid choice, please try again.\n";  
    }  
} while (choice != 15); // Update loop condition to match exit case number  
return 0;  
}
```

Code 2(Claude)

```
#include "Polynomial.h"
#include <numeric>
#include <stdexcept>
#include <cmath>
#include <iostream>
#include <limits>

Polynomial::Polynomial() {}
Polynomial::Polynomial(const vector<double>& coefficients) : coeffs(coefficients) {
    // Remove leading zeros
    while (!coeffs.empty() && coeffs.back() == 0) {
        coeffs.pop_back();
    }
}
Polynomial::Polynomial(const Polynomial& other) : coeffs(other.coeffs) {}
Polynomial::~~Polynomial() {}
Polynomial& Polynomial::operator=(const Polynomial& other) {
    if (this != &other) {
        coeffs = other.coeffs;
    }
    return *this;
}
Polynomial Polynomial::operator+(const Polynomial& other) const {
    vector<double> result_coeffs(max(coeffs.size(), other.coeffs.size()), 0);
    for (size_t i = 0; i < coeffs.size(); ++i) {
        result_coeffs[i] += coeffs[i];
    }
    for (size_t i = 0; i < other.coeffs.size(); ++i) {
        result_coeffs[i] += other.coeffs[i];
    }
    return Polynomial(result_coeffs);
}
Polynomial Polynomial::operator-(const Polynomial& other) const {
    vector<double> result_coeffs(max(coeffs.size(), other.coeffs.size()), 0);
    for (size_t i = 0; i < coeffs.size(); ++i) {
        result_coeffs[i] += coeffs[i];
    }
    for (size_t i = 0; i < other.coeffs.size(); ++i) {
        result_coeffs[i] -= other.coeffs[i];
    }
}
```

```

    }
    return Polynomial(result_coeffs);
}

Polynomial Polynomial::operator*(const Polynomial& other) const {
    vector<double> result_coeffs(coeffs.size() + other.coeffs.size() - 1, 0);
    for (size_t i = 0; i < coeffs.size(); ++i) {
        for (size_t j = 0; j < other.coeffs.size(); ++j) {
            result_coeffs[i + j] += coeffs[i] * other.coeffs[j];
        }
    }
    return Polynomial(result_coeffs);
}

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

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

```



```

}
int Polynomial::degree() const {
    return coeffs.size() - 1;
}
double Polynomial::evaluate(double x) const {
    return accumulate(coeffs.rbegin(), coeffs.rend(), 0.0,
        [x](double sum, double coeff) { return sum * x + coeff; });
}
Polynomial Polynomial::compose(const Polynomial& q) const {
    Polynomial result;
    for (int i = coeffs.size() - 1; i >= 0; --i) {
        result = result * q + Polynomial({coeffs[i]});
    }
    return result;
}
Polynomial Polynomial::derivative() const {
    if (coeffs.size() <= 1) {
        return Polynomial({0});
    }
    vector<double> deriv_coeffs(coeffs.size() - 1);
    for (size_t i = 1; i < coeffs.size(); ++i) {
        deriv_coeffs[i - 1] = i * coeffs[i];
    }
    return Polynomial(deriv_coeffs);
}
Polynomial Polynomial::integral() const {
    vector<double> int_coeffs(coeffs.size() + 1);
    int_coeffs[0] = 0; // Constant of integration
    for (size_t i = 0; i < coeffs.size(); ++i) {
        int_coeffs[i + 1] = coeffs[i] / (i + 1);
    }
    return Polynomial(int_coeffs);
}
double Polynomial::integral(double x1, double x2) const {
    Polynomial antiderivative = integral();
    return antiderivative.evaluate(x2) - antiderivative.evaluate(x1);
}
vector<double> Polynomial::getRoot(double tolerance, int maxIter) const {
    vector<double> roots;

```

```

Polynomial derivative = this->derivative();
// Helper function to check if a root is unique (not already in the roots vector)
auto isUniqueRoot = [&](double root) {
    for (double r : roots) {
        if (abs(r - root) < tolerance) {
            return false;
        }
    }
    return true;
};
// Helper function for Newton's method
auto newtonMethod = [&](double guess) {
    double x = guess;
    for (int i = 0; i < maxIter; ++i) {
        double fx = this->evaluate(x);
        if (abs(fx) < tolerance) {
            return make_pair(true, x);
        }
        double dfx = derivative.evaluate(x);
        if (dfx == 0) {
            return make_pair(false, x);
        }
        x = x - fx / dfx;
    }
    return make_pair(false, x);
};
// Try to find roots starting from different initial guesses
vector<double> initialGuesses = {-10, -1, 0, 1, 10};
for (double guess : initialGuesses) {
    auto [found, root] = newtonMethod(guess);
    if (found && isUniqueRoot(root)) {
        roots.push_back(root);
    }
}
// If no roots found, try random guesses
if (roots.empty()) {
    srand(time(nullptr));
    for (int i = 0; i < 10; ++i) {
        double randomGuess = (rand() / double(RAND_MAX)) * 20 - 10; // Random number
between -10 and 10
        auto [found, root] = newtonMethod(randomGuess);
    }
}

```

```

        if (found && isUniqueRoot(root)) {
            roots.push_back(root);
        }
    }
}

return roots;
}

void Polynomial::setCoefficients(const vector<double>& coefficients) {
    coeffs = coefficients;
    while (!coeffs.empty() && coeffs.back() == 0) {
        coeffs.pop_back();
    }
}

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

Polynomial inputPolynomial() {
    vector<double> coeffs;
    double coeff;
    cout << "Enter coefficients (starting from constant term, enter non-number to finish):\n";
    while (cin >> coeff) {
        coeffs.push_back(coeff);
    }
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    return Polynomial(coeffs);
}

int main() {
    int choice;
    Polynomial p1, p2, result;
    while (true) {
        cout << "\nPolynomial Operations Menu:\n";
        cout << "1. Add two polynomials\n";
        cout << "2. Subtract two polynomials\n";
        cout << "3. Multiply two polynomials\n";
        cout << "4. Evaluate polynomial\n";
    }
}

```

```

cout << "5. Find derivative of a polynomial\n";
cout << "5. Find integral of a polynomial\n";
cout << "7. Compute definite integral\n";
cout << "8. Get polynomial degree\n";
cout << "9. Set polynomial coefficients\n";
cout << "10. Get polynomial coefficients\n";
cout << "11. Compare two polynomials for equality\n";
cout << "12. Display polynomial\n";
cout << "13. Compose two polynomials\n";
cout << "14. Find a root of polynomial\n";
cout << "15. Exit\n";
cout << "Enter your choice: ";
cin >> choice
if (choice == 15) break;
switch(choice) {
    case 1: case 2: case 3: case 13: { // Addition, Subtraction, Multiplication,
Composition
        cout << "Enter first polynomial:\n";
        p1 = inputPolynomial();
        cout << "Enter second polynomial:\n";
        p2 = inputPolynomial();
        if (choice == 1) result = p1 + p2;
        else if (choice == 2) result = p1 - p2;
        else if (choice == 3) result = p1 * p2;
        else result = p1.compose(p2);
        cout << "Result: " << result << endl;
        break;
    }
    case 4: { // Evaluation
        cout << "Enter polynomial to evaluate:\n";
        p1 = inputPolynomial();
        double x;
        cout << "Enter x value: ";
        cin >> x;
        cout << "Result: " << p1.evaluate(x) << endl;
        break;
    }
    case 5: case 6: { // Derivative, Integral
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        result = (choice == 5) ? p1.derivative() : p1.integral();
    }
}

```

```

        cout << ((choice == 5) ? "Derivative: " : "Integral: ") << result << endl;
        break;
    }
    case 7: { // Definite Integral
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        double a, b;
        cout << "Enter lower bound a: ";
        cin >> a;
        cout << "Enter upper bound b: ";
        cin >> b;
        cout << "Definite integral from " << a << " to " << b << ": " <<
p1.integral(a, b) << endl;
        break;
    }
    case 8: { // Get degree
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        cout << "Degree: " << p1.degree() << endl;
        break;
    }
    case 9: { // Set coefficients
        p1 = inputPolynomial();
        cout << "Polynomial set: " << p1 << endl;
        break;
    }
    case 10: { // Get coefficients
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        cout << "Coefficients: ";
        for (int i = 0; i <= p1.degree(); ++i) {
            cout << p1.getCoefficient(i) << " ";
        }
        cout << endl;
        break;
    }
    case 11: { // Compare for equality
        cout << "Enter first polynomial:\n";
        p1 = inputPolynomial();
        cout << "Enter second polynomial:\n";
        p2 = inputPolynomial();

```

```

        cout << "Polynomials are " << (p1 == p2 ? "equal" : "not equal") << endl;
        break;
    }
    case 12: { // Display polynomial
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        cout << "Polynomial: " << p1 << endl;
        break;
    }
    case 14: { // Find roots of polynomial
        cout << "Enter polynomial:\n";
        p1 = inputPolynomial();
        vector<double> roots = p1.getRoot();
        if (roots.empty()) {
            cout << "No roots found.\n";
        } else {
            cout << "Roots found:\n";
            for (double root : roots) {
                cout << root << "\n";
            }
        }
        break;
    }
    default:
        cout << "Invalid choice. Please try again.\n";
}

}

cout << "Thank you for using the Polynomial Calculator!\n";
return 0;
}

```

Test Cases

Test Cases for Polynomial Class

Test Case 1

- Description: Composing polynomials.
- Polynomials:
 - $p_1(x) = x^2 - 1$
 - $p_2(x) = x^3 - 1$
- Operation: $p_2(p_1)$
- Expected Result: $x^6 - 3x^4 + 3x^2 - 2$

Test Case 2

- Description: Adding polynomials.
- Polynomials:
 - $p_1(x) = x$
 - $p_2(x) = x^3 - 1$
- Operation: $p_1 + p_2$
- Expected Result: $x^3 + x - 1$

Test Case 3

- Description: Subtracting polynomials.
- Polynomials:
 - $p_1(x) = x^2 + x$
 - $p_2(x) = x^3 - 2$
- Operation: $p_1 - p_2$
- Expected Result: $-x^3 + x^2 + x + 2$

Test Case 4

- Description: Multiplying polynomials.
- Polynomials:
 - $p_1(x) = x^3 + 1$
 - $p_2(x) = x^5 - 1$
- Operation: $p_1 * p_2$
- Expected Result: $x^8 - x^3 - 1$

Test Case 5

- Description: Evaluating a polynomial at a given point.
- Polynomial: $p(x) = x^2 + 2x - 1$
- Point: $x = 3$
- Expected Result: 14

Test Case 6

- Description: Finding the indefinite integral of a polynomial.

- Polynomial: $p(x) = x^2 + 2x - 1$
- Expected Result: $(1/3)x^3 + x^2 - x + C$

Test Case 7

- Description: Finding the definite integral of a polynomial.
- Polynomial: $p(x) = x^2 + 2x - 1$
- Integral Bounds: $[0, 9]$
- Expected Result: 315

Test Case 8

- Description: Finding the derivative of a polynomial.
- Polynomial: $p(x) = x$
- Expected Result: 0

Test Case 9

- Description: Finding the indefinite integral of a polynomial.
- Polynomial: $p(x) = x$
- Expected Result: $(1/2)x^2 + C$

Test Case 10

- Description: Evaluating a polynomial at a given point.
- Polynomial: $p(x) = x^5 + x^3 - x$
- Point: $x = 2$
- Expected Result: 38

Test Case 11

- Description: Finding the derivative of a polynomial.
- Polynomial: $p(x) = x^5 + x^3 - x$
- Expected Result: $5x^4 + 3x^2 - 1$

Test Case 12

- Description: Finding the indefinite integral of a polynomial.
- Polynomial: $p(x) = x^5 + x^3 - x$
- Expected Result: $(1/6)x^6 + (1/4)x^4 - (1/2)x^2 + C$

Test Case 13

- Description: Evaluating a polynomial at a given point.
- Polynomial: $p(x) = x^8 - x^6 + x^4 - x^2$
- Point: $x = 3$
- Expected Result: 5904

Test Case 14

- Description: Finding the derivative of a polynomial.
- Polynomial: $p(x) = x^8 - x^6 + x^4 - x^2$
- Expected Result: $8x^7 - 6x^5 + 4x^3 - 2x$

Test Case 15

- Description: Finding the second derivative of a polynomial.
- Polynomial: $p(x) = x^8 - x^6 + x^4 - x^2$
- Expected Result: $56x^6 - 30x^4 + 12x^2 - 2$

Test Case 16

- Description: Evaluating a polynomial at a given point.
- Polynomial: $p(x) = x^8 - x^6 + x^4 - x^2$
- Point: $x = 10$
- Expected Result: 99009900

Test Case 17

- Description: Evaluating a polynomial at a given point.
- Polynomial: $p(x) = 5x^3 - 3x^2 + x - 1$
- Point: $x = 0$
- Expected Result: -1

Test Case 18

- Description: Finding the derivative of a polynomial.
- Polynomial: $p(x) = 5x^4 - 3x^2 + x - 1$
- Expected Result: $20x^3 - 6x + 1$

Test Case 19

- Description: Finding the indefinite integral of a polynomial.
- Polynomial: $p(x) = 5x^4 - 3x^2 + x - 1$
- Expected Result: $x^5 - x^3 + (1/2)x^2 - x + C$

Test Case 20

- Description: Definite integral of a polynomial from 0 to 2.
- Polynomial: $p(x) = 5x^4 - 3x^2 + x - 1$
- Integral Bounds: $[0, 2]$
- Expected Result: 24

Test Case 21

1. Description: Evaluate a polynomial at $x = -1$.
 - Polynomial: $P(x) = 3x^2 + 4x + 5$
 - Expected Result: 4
2. Description: Find the derivative of $P(x)$.
 - Polynomial: $P(x) = 3x^2 + 4x + 5$
 - Expected Result: $6x + 4$
3. Description: Compute the definite integral from 0 to 2.
 - Polynomial: $P(x) = 3x^2 + 4x + 5$
 - Expected Result: 26

Test Case 22

1. Description: Evaluate a polynomial at $x = 30$.

- Polynomial: $P(x) = x^6 + 3x^4 - 2x^2 + 1$
- Expected Result: 731428201

2. Description: Find the derivative of $P(x)$.

- Polynomial: $P(x) = x^6 + 3x^4 - 2x^2 + 1$
- Expected Result: $6x^5 + 12x^3 - 4x$

Test Case 23

1. Description: Evaluate a polynomial with square roots at $x = \sqrt{2}$.

- Polynomial: $P(x) = (\sqrt{10}/3)x^4 + (\sqrt{6}/3)x^3 + (\sqrt{3}/2)$
- Expected Result: $(\sqrt{10} + 10\sqrt{3})/3$

2. Description: Find the second derivative of $P(x)$.

- Expected Result: $6\sqrt{3}x + 2\sqrt{6}x^2$

Test Case 24

- Description: Compute the indefinite and definite integral of a polynomial.
- Polynomial: $P(x) = (\sqrt{10}/3)x^4 + (\sqrt{6}/3)x^3 + (\sqrt{3}/2)$
- Definite Integral Bounds: $[\sqrt{3}, \sqrt{10}]$
- Expected Result: Indefinite integral is $(1/5)x^5 + (1/4)x^4 + C$, definite integral ≈ 72.15514444

Test Case 25

- Description: Indefinite and definite integral of a fractional polynomial.
- Polynomial: $P(x) = (2/3)x^4 + (5/4)x^3 + (1/2)x^2 + 1$
- Definite Integral Bounds: $[0.5, 2.5]$
- Expected Result: $10337/640$

Test Case 26

- Description: Product and integral of polynomials with square roots.
- Polynomial 1: $p1(x) = (\sqrt{3}/3)x^8 + (\sqrt{10}/5)x^6 + 3x$
- Polynomial 2: $p2(x) = (\sqrt{6}/4)x^4 - (\sqrt{3}/2)x$
- Expected Result: Definite integral ≈ 117614.7595

Test Case 27

- Description: Find root of a polynomial using Newton's method.
- Polynomial: $P(x) = x^2 - 1$
- Root approximation near $x = 2$
- Expected Result: Root near 1, $P(\text{root}) \approx 0$

Test Case 28

- Description: Find the root of a polynomial with complex roots ($x^2 + 1$).
- Polynomial: $P(x) = x^2 + 1$
- Expected Result: No real root, return NaN

Test Case 29

- Description: Find the root of a polynomial with no real roots.

- Polynomial: $P(x) = x^2 - 2x + 5$
- Expected Result: No real root, return NaN

Test Case 30

- Description: Basic operations on constant polynomials and evaluation.
- Polynomials:
 - $p1(x) = -9$
 - $p2(x) = 12$
- Operations:
 1. ****Indefinite Integral of $p1(x)$ ****:
 - Expected Result: $P(x) = -9x + C$ (Integral of constant is the constant multiplied by x)
 2. ****Subtraction of $p1$ and $p2$ ****:
 - Operation: $p1 - p2$
 - Expected Result: $-9 - 12 = -21$
 3. ****Multiplication of $p1$ and $p2$ ****:
 - Operation: $p1 * p2$
 - Expected Result: $-9 * 12 = -108$
 4. ****Derivative of $p1(x)$ ****:
 - Expected Result: Derivative of a constant is 0
 5. ****Evaluate $p1$ at $x = 2$ ****:
 - Operation: $p1(2) = -9$
 - Expected Result: -9 (since $p1(x)$ is constant)
 6. ****Definite integral of $p1$ from 0 to 1****:
 - Operation: Integral of $p1(x)$ from 0 to 1
 - Expected Result: $\text{Integral}(-9) \text{ from } 0 \text{ to } 1 = -9 * (1 - 0) = -9$

Test Case 31

- Description: Basic operations on constant polynomials.
- Polynomials:
 - $p1(x) = 5$
 - $p2(x) = 1$
- Operations:
 1. ****Sum of $p1$ and $p2$ ****:
 - Expected Result: 6
 2. ****Subtraction of $p2$ from $p1$ ****:
 - Expected Result: 4
 3. ****Product of $p1$ and $p2$ ****:
 - Expected Result: 5
 4. ****Degree of $p1$ ****:

```

- Expected Result: 0
5. **Evaluation of p1 at x = 2**:
- Expected Result: 5
6. **Derivative of p1**:
- Expected Result: 0
7. **Indefinite integral of p1**:
- Expected Result: 5x
8. **Definite integral of p1 from 0 to 1**:
- Expected Result: 5
9. **Root of p1 using Newton's method**:
- Expected Result: NaN (no real root)
#### Test Case 32
- Description: Derivative and evaluation of a large polynomial.
- Polynomial:  $p_1(x) = -1x^{10} - 10x^9 + 10x^8 - x^7 + 2x^6 - 3x^5 + 13x^4 + 5x^3 - 4x^2 + 13x + 13$ 
- Operations:
1. **Derivative of p1**:
- Expected Result:  $-10x^9 + 20x^8 - 7x^7 + 12x^6 - 15x^5 + 52x^4 + 35x^3 - 32x^2 + 117x + 130$ 
2. **Evaluation of p1 at x = 2**:
- Expected Result: 20363
#### Test Case 33
- Description: Integral of a large polynomial and definite integral from 0 to 1.
- Polynomial:  $p_1(x) = -1x^{10} - 10x^9 + 10x^8 - x^7 + 2x^6 - 3x^5 + 13x^4 + 5x^3 - 4x^2 + 13x + 13$ 
- Operations:
1. **Indefinite integral of p1**:
- Expected Result:  $0 - x^{11}/11 - x^{10}/10 - (5/3)x^9 + (10/4)x^8 - (1/5)x^7 + (2/6)x^6 - (3/7)x^5 + (13/8)x^4 + (5/9)x^3 - (4/10)x^2 + (13/11)x + C$ 
2. **Definite integral of p1 from 0 to 1**:
- Expected Result: 41659/27720
#### Test Case 34
- Description: Subtraction of polynomials and definite integral from 0 to 1.
- Polynomial 1:  $p_1(x) = 12x^8 - 10x^7 + x^6 + x^5 + 10x^4 - 12x^3 - 6x^2 - 9x - 6$ 
- Polynomial 2:  $p_2(x) = -2x^8 + 3x^7 + 2x^6 - x^5 - 6x^4 + 8x^3 + 12x^2 - x - 6$ 
- Operations:
1. **Subtraction of p2 from p1**:
- Expected Result:  $14x^8 - 13x^7 - x^6 + 2x^5 + 16x^4 - 20x^3 - 18x^2 - 8x + 0$ 

```

```

2. **Definite integral of p1 from 0 to 1**:
    - Expected Result: -367/28
#### Test Case 35
- Description: Finding roots of a high-degree polynomial.
- Polynomial:  $p_1(x) = -4x^{10} - 4x^9 + 9x^8 - 6x^7 + 7x^6 - 9x^5 + 6x^4 - 11x^3 + 8x^2 - x - 11$ 
- Operations:
    1. **Find root near  $x = -2.41489$  using Newton's method**:
        - Expected Result: Root  $\approx -2.41489316821433$ 
    2. **Find root near  $x = -0.66301$  using Newton's method**:
        - Expected Result: Root  $\approx -0.663011179796566$ 
#### Test Case 36
- Description: Product of polynomials and evaluation at  $x = 2$ .
- Polynomial 1:  $p_1(x) = -x^7 + 6x^6 - 8x^5 + 10x^4 - 2x^3 + 6x^2 + 9x - 4$ 
- Polynomial 2:  $p_2(x) = -14x^7 + 7x^6 + 4x^5 + 7x^4 - 10x^3 + 8x^2 + 15x$ 
- Operations:
    1. **Product of  $p_1$  and  $p_2$ **:
        - Expected Result:  $14x^{14} + 63x^{13} - 60x^{12} + 47x^{11} + 90x^{10} - 84x^9 - 310x^8 - 29x^7 - 73x^6 + 52x^5 - 37x^4 - 72x^3 + 222x^2 + 90x + 135$ 
    2. **Evaluation of  $p_1$  at  $x = 2$ **:
        - Expected Result: 1775
#### Test Case 37
- Description: Subtraction of polynomials and definite integral.
- Polynomial 1:  $p_1(x) = -x^7 + 6x^6 - 8x^5 + 10x^4 - 2x^3 + 6x^2 + 9x - 4$ 
- Polynomial 2:  $p_2(x) = -14x^7 + 7x^6 + 4x^5 + 7x^4 - 10x^3 + 8x^2 + 15x$ 
- Operations:
    1. **Subtraction of  $p_2$  from  $p_1$ **:
        - Expected Result:  $13x^7 - x^6 - 12x^5 + 15x^4 - 8x^3 - 9x - 6$ 
    2. **Definite integral of  $p_1$  from 0 to 1**:
        - Expected Result: 209/840
#### Test Case 38
- Description: Derivative of a polynomial and checking the degree.
- Polynomial:  $p_1(x) = -x^7 + 6x^6 - 8x^5 + 10x^4 - 2x^3 + 6x^2 + 9x - 4$ 
- Operations:
    1. **Derivative of  $p_1$ **:
        - Expected Result:  $-7x^6 + 36x^5 - 40x^4 + 40x^3 - 6x^2 + 12x + 9$ 
    2. **Degree of  $p_1$ **:
        - Expected Result: 7
#### Test Case 39

```

```

- Description: Sum of two polynomials and checking the degree.
- Polynomial 1: p1(x) = -13x^10 + 12x^9 + 15x^8 + 5x^7 + 12x^6 - 2x^5 + 13x^4
- 10x^3 + 2x^2 + 12x + 15
- Polynomial 2: p2(x) = -12x^10 - 14x^9 - 10x^8 - 2x^7 + x^6 - 3x^5 + 14x^4 +
x^3 - 8x^2 - 10x - 11
- Operations:
  1. **Sum of p1 and p2**:
    - Expected Result: -25x^10 - 2x^9 + 5x^8 + 3x^7 + 13x^6 - 5x^5 + 27x^4 -
9x^3 - 6x^2 + 2x + 4
  2. **Degree of p1**:
    - Expected Result: 10
#### Test Case 40
- Description: Product of two polynomials, definite integral, and evaluation
at x = 2.
- Polynomial 1: p1(x) = -13x^10 + 12x^9 + 15x^8 + 5x^7 + 12x^6 - 2x^5 + 13x^4
- 10x^3 + 2x^2 + 12x + 15
- Polynomial 2: p2(x) = -12x^10 - 14x^9 - 10x^8 - 2x^7 + x^6 - 3x^5 + 14x^4 +
x^3 - 8x^2 - 10x - 11
- Operations:
  1. **Product of p1 and p2**:
    - Expected Result: 156x^20 + 38x^19 - 218x^18 - 364x^17 - 401x^16 -
173x^15 - 461x^14 + 49x^13 + 313x^12 - 17x^11 - 253x^10 - 721x^9 - 273x^8 -
310x^7 - 219x^6 + 97x^5 + 163x^4 + 9x^3 - 262x^2 - 282x - 165
  2. **Definite integral of p1 from 0 to 1**:
    - Expected Result: 16319/3465
  3. **Evaluation of p1 at x = 2**:
    - Expected Result: 21807
#### Test Case 41
- Description: Derivative and integral of a large polynomial.
- Polynomial: p1(x) = 15x^10 + 12x^9 + 2x^8 - 10x^7 + 13x^6 - 2x^5 + 12x^4 +
5x^3 - 11x^2 + 12x - 13
- Operations:
  1. **Indefinite integral of p1**:
    - Expected Result: (1/11)x^11 + (6)x^10 + (5/4)x^9 + (12/5)x^8 -
(1/3)x^7 + (13/7)x^6 - (5/4)x^5 + (2/9)x^4 + (6/5)x^3 + 15/11*x^2 + C
  2. **Derivative of p1**:
    - Expected Result: 150x^9 + 108x^8 + 16x^7 - 70x^6 + 78x^5 - 10x^4 +
48x^3 + 30x^2 + 12x + 12
#### Test Case 42
- Description: Polynomial addition, subtraction, multiplication, evaluation,
derivative, and definite integral.
- Polynomials:
  - poly1(x) = -8x^5 + 4x^4 + 10x^3 + 4x + 10

```

```

- poly2(x) = -14x^5 + 11x^4 + x^3 + 8x^2 + 15x + 12
- Operations:
  1. **Addition**:
    - Expected Result: -22x^5 + 15x^4 + 11x^3 + 8x^2 + 19x + 22
  2. **Subtraction**:
    - Expected Result: 6x^5 - 7x^4 + 9x^3 - 8x^2 - 11x - 2
  3. **Multiplication**:
    - Expected Result: 112x^10 - 144x^9 - 104x^8 + 50x^7 - 134x^6 - 52x^5 +
312x^4 + 162x^3 + 140x^2 + 198x + 120
  4. **Evaluation at x = 2**:
    - Expected Result: -94
  5. **Derivative**:
    - Expected Result: -40x^4 + 16x^3 + 30x^2 + 16
  6. **Indefinite integral**:
    - Expected Result: 5x^6 + (4/5)x^5 + (5/2)x^4 + 2x^3 + 10x + C
  7. **Definite integral from 0 to 1**:
    - Expected Result: 419/30
#### Test Case 43
- Description: Polynomial addition, subtraction, multiplication, and
evaluation at x=2.
- Polynomials:
  - poly1(x) = 14x^7 - x^6 + 14x^5 - 9x^4 - 12x^3 - 11x^2 + 15x - 11
  - poly2(x) = -11x^7 + 13x^6 + 2x^5 + 13x^4 + 6x^3 - 11x^2 + 9x - 13
- Operations:
  1. **Addition**:
    - Expected Result: 3x^7 + 12x^6 + 16x^5 + 4x^4 - 6x^3 - 22x^2 + 24x - 24
  2. **Subtraction**:
    - Expected Result: 25x^7 - 14x^6 + 12x^5 - 22x^4 - 18x^3 + 0x^2 + 6x + 2
  3. **Multiplication**:
    - Expected Result: 143x^14 - 294x^13 + 399x^12 - 174x^11 + 77x^10 + ...
  4. **Evaluation at x = 2**:
    - Expected Result: 1911
#### Test Case 44
- Description: Polynomial addition, subtraction, multiplication, and
evaluation at x=2.
- Same polynomials as **Test Case 43**.
- Operations:
  1. **Addition**:
    - Expected Result: -24x^7 + 12x^6 + 9x^5 + 4x^4 - 13x^3 - 13x^2 + 14x -
4

```

```

2. **Subtraction**:
    - Expected Result:  $8x^7 - 17x^6 - 16x^5 + x^4 + 12x^3 + 9x^2 - 3x - 16$ 
3. **Multiplication**:
    - Expected Result: Same as **Test Case 43**
4. **Evaluation at  $x = 2$ **:
    - Expected Result: 1911
#### Test Case 45
- Description: Evaluate polynomial and test product of two polynomials.
- Polynomials:
    -  $p_1(x) = -6x^8 + 4x^7 + 8x^6 - 3x^5 + 14x^4 - 11x^3 + 15x^2 - x + 10$ 
    -  $p_2(x) = -10x^8 + 4x^7 + 5x^6 - 14x^5 + 9x^4 - 8x^3 - 7x^2 + 12$ 
- Operations:
    1. **Evaluation of  $p_1$  at  $x = 2$ **:
        - Expected Result: -404
    2. **Product of  $p_1$  and  $p_2$ **:
        - Expected Result: Polynomial with coefficients: [-100, -70, -92, ...]
#### Test Case 46
- Description: Integral of polynomials and finding real roots.
- Polynomials:
    -  $p_1(x) = -6x^8 + 4x^7 + 8x^6 - 3x^5 + 14x^4 - 11x^3 + 15x^2 - x + 10$ 
- Operations:
    1. **Indefinite integral**:
        - Expected Result:  $(1/9)x^9 + (4/8)x^8 + (1/7)x^7 + (8/6)x^6 + (14/5)x^5 + (14/5)x^4 + \dots$ 
    2. **Real roots of  $p_1$ **:
        - Expected Result: [-1.583, 1.719]
#### Test Case 47
- Description: Polynomial subtraction and definite integral.
- Polynomial:  $p_1(x) = 10x^7 + 7x^6 - 12x^5 + 3x^4 - 6x^3 + 9x^2 - 14x + 13$ 
- Operations:
    1. **Subtraction**:
        - Expected Result: Coefficients: [-11, 22, -20, ...]
    2. **Definite integral from 0 to 1**:
        - Expected Result: -4433/420
#### Test Case 48
- Description: Derivative and evaluation at  $x = 2$ .
- Polynomial:  $p_1(x) = 10x^7 + 7x^6 - 12x^5 + 3x^4 - 6x^3 + 9x^2 - 14x + 13$ 
- Operations:

```



```

1. **Derivative**:
    - Expected Result: Coefficients: [60, 42, -60, ...]
2. **Evaluation at x = 2**:
    - Expected Result: 676
#### Test Case 49
- Description: Polynomial addition and integral.
- Polynomials:
    -  $p_1(x) = 13x^9 + 2x^8 - 13x^7 + 8x^6 - 7x^5 + 3x^4 + 10$ 
    -  $p_2(x) = -10x^9 + 12x^8 - 7x^7 + 2x^6 + 13x^5$ 
- Operations:
    1. **Addition**:
        - Expected Result: Coefficients: [11, 1, 2, -21, ...]
    2. **Definite integral of p1 from 0 to 1**:
        - Expected Result: 34241/2520
#### Test Case 50
- Description: Integral and real roots.
- Polynomial:  $p_1(x) = 13x^9 + 2x^8 - 13x^7 + 8x^6 - 7x^5 + 3x^4 + 10$ 
- Operations:
    1. **Indefinite integral**:
        - Expected Result: Coefficients: [0, 13, (1/4)x^9, ...]
    2. **Definite integral of p1 from 0 to 1**:
        - Expected Result: 34241/2520
    3. **Real roots of p1**:
        - Expected Result: [-1.379]
#### Test Case 51
- Description: Polynomial product and evaluation.
- Polynomials:
    -  $p_1(x) = 11x^5 - 10x^4 - 7x^3 - 11x^2 + 0x - 13$ 
    -  $p_2(x) = -14x^5 - 2x^4 - 2x^3 - 6x^2 + 4x$ 
- Operations:
    1. **Product of p1 and p2**:
        - Expected Result:  $-52x^{10} + 78x^9 + 34x^8 + 64x^7 + 94x^6 + 350x^5 + 30x^4 + 122x^3 + 96x^2 + 118x - 154$ 
    2. **Evaluation of p1 at x = 2**:
        - Expected Result: 79
#### Test Case 52
- Description: Polynomial integral and finding real roots.
- Polynomial:

```

```

- p1(x) = 11x^5 - 10x^4 - 7x^3 - 11x^2 + 0x - 13
- Operations:
  1. **Indefinite integral**:
    - Expected Result: (11/6)x^6 - 2x^5 - (7/4)x^4 - (11/3)x^3 - 13x + C
  2. **Definite integral from 0 to 1**:
    - Expected Result: -223/12
  3. **Real root of p1**:
    - Expected Result: 1.73684576071411
#### Test Case 53
- Description: Polynomial derivative and evaluation.
- Polynomial:
  - p1(x) = -10x^11 + 9x^10 - 7x^9 + 7x^8 - 6x^7 + 15x^6 - 8x^5 - 6x^4 - 1x - 5
- Operations:
  1. **Derivative of p1**:
    - Expected Result: -110x^10 + 90x^9 - 63x^8 + 56x^7 - 42x^6 + 90x^5 - 40x^4 - 24x^3 - 1
  2. **Evaluation of p1 at x = 2**:
    - Expected Result: -31383
#### Test Case 54
- Description: Polynomial integral and finding real roots.
- Polynomial:
  - p1(x) = -10x^11 + 9x^10 - 7x^9 + 7x^8 - 6x^7 + 15x^6 - 8x^5 - 6x^4 - 1x - 5
- Operations:
  1. **Indefinite integral**:
    - Expected Result: -x^12/12 + (9/11)x^11 - (7/10)x^10 + (7/9)x^9 - (6/8)x^8 + (15/7)x^7 - (8/6)x^6 - (6/5)x^5 - x^2/2 - 5x + C
  2. **Evaluation of p1 at x = 2**:
    - Expected Result: -31383
  3. **Definite integral from 0 to 1**:
    - Expected Result: -235469/27720
  4. **Real root of p1**:
    - Expected Result: -2.1317879159301
#### Test Case 55
- Description: Polynomial operations (integral, subtraction, product, derivative, evaluation).
- Polynomials:
  - p1(x) = 11x^2 + 8x - 4
  - p2(x) = 15x^2 + 4x - 4

```

```

- Operations:
1. **Indefinite integral of p1**:
  - Expected Result: (11/3)x^3 + 4x^2 - 4x + C
2. **Subtraction of p2 from p1**:
  - Expected Result: -4x^2 + 4x
3. **Product of p1 and p2**:
  - Expected Result: 165x^4 + 164x^3 - 72x^2 - 48x + 16
4. **Derivative of p1**:
  - Expected Result: 22x + 8
5. **Evaluation of p1 at x = 2**:
  - Expected Result: 56
6. **Definite integral of p1 from 0 to 1**:
  - Expected Result: 11/3
7. **Real roots of p1**:
  - Expected Result: -1.06781515385589, 0.340542426583167

#### Test Case 56
- Description: Finding roots of a high-degree polynomial using Newton's
method.
- Polynomial:
  -  $p_1(x) = 13x^9 + 5x^8 - 4x^7 + 13x^6 + 10x^5 - 3x^4 + 2x^3 - x^2 - 10x - 1$ 
- Operations:
1. **Root near x = -0.1**:
  - Expected Result: -0.0915290543036897
2. **Root near x = 0.7**:
  - Expected Result: 0.727055676529518

#### Test Case 57
- Description: Finding roots of a 6th-degree polynomial.
- Polynomial:
  -  $p_1(x) = 6x^6 + 10x^5 - 2x^4 - 8x^3 + 6x^2 - 4x - 1$ 
- Operations:
1. **Root near x = -1.0**:
  - Expected Result: -1.03819717707786
2. **Root near x = -0.2**:
  - Expected Result: -0.186199186330593
3. **Root near x = 0.7**:
  - Expected Result: 0.749600188448983

#### Test Case 58

```

- Description: Finding roots of a 6th-degree polynomial.
- Same polynomial and operations as ****Test Case 57****.

Test Case 59

- Description: Finding roots of an 8th-degree polynomial.
- Polynomial:
 - $p_1(x) = 10x^8 + 4x^7 + 8x^6 - 3x^5 + 14x^4 - 11x^3 + 15x^2 - x - 6$
- Operations:
 1. ****Root near $x = -1.5$ ****:
 - Expected Result: -1.58327556220727
 2. ****Root near $x = 1.7$ ****:
 - Expected Result: 1.71953761729611

Test Case 60

- Description: Finding roots of a quadratic polynomial.
- Polynomial:
 - $p_1(x) = 11x^2 + 8x - 4$
- Operations:
 1. ****Root near $x = -1.0$ ****:
 - Expected Result: -1.06781515385589
 2. ****Root near $x = 0.5$ ****:
 - Expected Result: 0.340542426583167

Test Case 61

- Description: Test for polynomial with no real roots.
- Polynomial: $p_1(x) = x^4 + 2x^3 + 5x^2 + 6x + 10$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: No real roots, so root = Na

Test Case 62

- Description: Test for a 6th-degree polynomial with no real roots.
- Polynomial: $p_1(x) = 2x^6 + 3x^5 + 4x^4 + 5x^3 + 6x^2 + 7x + 8$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: No real roots, so root = NaN

Test Case 63

- Description: Test for a 4th-degree polynomial with no real roots.
- Polynomial: $p_1(x) = x^4 + 4x^3 + 6x^2 + 4x + 5$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: No real roots, so root = NaN

Test Case 64

- Description: Test for a polynomial with some real roots.
- Polynomial: $p_1(x) = 3x^5 + 2x^3 + x^2 + 8$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: Root ≈ -0.86382264

Test Case 65

- Description: Test for a cubic polynomial with a real root.
- Polynomial: $p_1(x) = x^3 + 2x^2 + 5x + 10$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: Root ≈ -0.5

Test Case 66

- Description: Test for a polynomial with irrational coefficients.
- Polynomial: $p_1(x) = x^4 + \sqrt{2}x^3 + \sqrt{3}x^2 + \sqrt{5}x + \sqrt{6}$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: No real roots, so root = NaN
 2. ****Check degree****:
 - Expected Result: Degree = 4
 3. ****Evaluate at x = 1****:
 - Expected Result: $p_1(1) = \sqrt{6} + \sqrt{5} + \sqrt{3} + \sqrt{2} + 1$
 4. ****Check derivative****:
 - Expected Result: Degree of derivative = 3

Test Case 67

- Description: Test for a 12th-degree polynomial with irrational coefficients.
- Polynomial: $p_1(x) = x^{12} + \sqrt{2}x^{11} + \sqrt{3}x^{10} + \sqrt{4}x^9 + \sqrt{5}x^8 + \sqrt{6}x^7 + \sqrt{7}x^6 + \sqrt{8}x^5 + \sqrt{9}x^4 + \sqrt{10}x^3 + \sqrt{11}x^2 + \sqrt{12}x + 1$
- Operations:
 1. ****Get root using Newton's method****:
 - Expected Result: Root ≈ -2.50562387 or -0.94087713
 2. ****Check degree****:
 - Expected Result: Degree = 12
 3. ****Evaluate at x = 1****:
 - Expected Result: $p_1(1) = 1 + \sqrt{12} + \sqrt{11} + \sqrt{10} + \sqrt{9} + \sqrt{8} + \sqrt{7} + \sqrt{6} + \sqrt{5} + \sqrt{4} + \sqrt{3} + \sqrt{2} + 1$
 4. ****Check derivative****:

```

- Expected Result: Degree of derivative = 11
#### Test Case 68
- Description: Test for a 20th-degree polynomial with real roots.
- Polynomial:  $p_1(x) = -71x^{20} + 67x^{19} + 61x^{18} + 59x^{17} - 53x^{16} + 47x^{15} - 43x^{14} + 41x^{13} - 37x^{12} + 31x^{11} - 29x^{10} + 23x^9 - 19x^8 + 17x^7 - 13x^6 + 11x^5 - 7x^4 + 5x^3 - 3x^2 + 2x + 1$ 
- Operations:
  1. **Get roots using Newton's method**:
    - Expected Result: Root1  $\approx 1.65737085$ , Root2  $\approx -0.28384746$ 
#### Test Case 69
- Description: Test for a 25th-degree polynomial.
- Polynomial:  $p_1(x) = x^{25} - 2x^{24} + 3x^{23} - 4x^{22} + 5x^{21} - 6x^{20} + 7x^{19} - 8x^{18} + 9x^{17} - 10x^{16} + 11x^{15} - 12x^{14} + 13x^{13} - 14x^{12} + 15x^{11} - 16x^{10} + 17x^9 - 18x^8 + 19x^7 - 20x^6 + 21x^5 - 22x^4 + 23x^3 - 24x^2 + 25x - 26$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: Root  $\approx 0.86072381$ 
#### Test Case 70
- Description: Test for a 30th-degree polynomial.
- Polynomial:  $p_1(x) = -31x^{30} + 30x^{29} - 29x^{28} + 28x^{27} - 27x^{26} + 26x^{25} - 25x^{24} + 24x^{23} - 23x^{22} + 22x^{21} - 21x^{20} + 20x^{19} - 19x^{18} + 18x^{17} - 17x^{16} + 16x^{15} - 15x^{14} + 14x^{13} - 13x^{12} + 12x^{11} - 11x^{10} + 10x^9 - 9x^8 + 8x^7 - 7x^6 + 6x^5 - 5x^4 + 4x^3 - 3x^2 + 2x - 1$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: Root  $\approx 0.0$ 
#### Test Case 71
- Description: Test for a high-degree polynomial with 51 terms.
- Polynomial:  $p_1(x) = 0 - x + 2x^2 - 3x^3 + 4x^4 - 5x^5 + \dots - 51x^{50}$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: Root  $\approx 0.0$ 
#### Test Case 72
- Description: Test for a polynomial with irrational coefficients.
- Polynomial:  $p_1(x) = 3.14 - 1.59x + 2.65x^2 - 3.58x^3 + 9.79x^4 - 2.65x^5 + 3.58x^6 - 9.79x^7 + \dots - 5.03x^{99}$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: No real roots (Expected NaN)
#### Test Case 73
- Description: Test for a polynomial with square roots and rational coefficients.

```

```

- Polynomial:  $p_1(x) = x^4 + \sqrt{2}x^3 + \sqrt{3}/2x^2 + \sqrt{5}/4x + 4$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: No real roots (Expected NaN)
#### Test Case 74
- Description: Test for a polynomial with square roots and simple coefficients.
- Polynomial:  $p_1(x) = x^4 + 3x^3 + \sqrt{3}/2x^2 + \sqrt{5}/4x + 4$ 
- Operations:
  1. **Get root using Newton's method**:
    - Expected Result: Roots  $\approx -2.47909554$  and  $-1.51010908$ 
#### Test Case 75
- Description: Test for a high-degree polynomial with equal coefficients.
- Polynomial:  $p_1(x) = 1 + x + x^2 + \dots + x^{1000}$ 
- Operations:
  1. **Integrate the polynomial**:
    - Expected Result: Integral  $= x + x^2/2 + x^3/3 + \dots + x^{1001}/1001$ 
#### Test Case 76
- Description: Test for a polynomial with rational and irrational coefficients.
- Polynomial:  $p_1(x) = 4x^3 + x^2 + 0x + \sqrt{2}$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -0.39196193$ 
  2. **Get second root using Newton's method**:
    - Initial Guess: -4.0
    - Expected Result: Root  $\approx -3.60803807$ 
#### Test Case 77
- Description: Test for a quadratic polynomial with irrational coefficients.
- Polynomial:  $p_1(x) = \sqrt{2}x + 1$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -1.41421356$ 
#### Test Case 78
- Description: Test for a quartic polynomial with an irrational coefficient.
- Polynomial:  $p_1(x) = \sqrt{2} + x^3 + 0x^2 + 0x$ 
- Operations:

```

```

1. **Get root using Newton's method**:
  - Initial Guess: 1.0
  - Expected Result: Root ≈ -1.41421356
#### Test Case 79
- Description: Test for a polynomial with identical irrational coefficients.
- Polynomial:  $p_1(x) = \sqrt{2} + \sqrt{2}x + \sqrt{2}x^2 + \sqrt{2}x^3$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ -1.00000000
#### Test Case 80
- Description: Test for a polynomial where the result should be NaN.
- Polynomial:  $p_1(x) = \sqrt{4} + 3x + \sqrt{2}x^2 + x + 1$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root = NaN
#### Test Case 81
- Description: Test for a polynomial of degree 10 with various coefficients.
- Polynomial:  $p_1(x) = 5x^{10} - 7x^9 + 13x^8 + 10x^7 + 2x^6 + 15x^5 - 8x^4 - 3x^3 - 14x^2 - 4x + 10$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ -0.79058165
  2. **Get second root using a different guess**:
    - Initial Guess: -2.0
    - Expected Result: Root ≈ -1.04797069
#### Test Case 82
- Description: Test for a cubic polynomial with mixed coefficients.
- Polynomial:  $p_1(x) = -7x^3 + 13x^2 - 8x + 8$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ 1.59038197
#### Test Case 83
- Description: Test for a complex polynomial with varied coefficients.
- Polynomial:  $p_1(x) = -13x^9 - 8x^8 + 15x^7 + 6x^6 + 7x^5 - 11x^4 - 4x^3 - 6x^2 - 14x + 1$ 

```



```

- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -0.85992705$ 
  2. **Get second root using a different guess**:
    - Initial Guess: 0.5
    - Expected Result: Root  $\approx 0.06926055$ 
#### Test Case 84
- Description: Test for a quintic polynomial with varied coefficients.
- Polynomial:  $p_1(x) = x^5 + 6x^4 - 6x^3 - 9x^2 - 7x - 6$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: -7.0
    - Expected Result: Root  $\approx -6.71416292$ 
  2. **Get second root using a different guess**:
    - Initial Guess: 2.0
    - Expected Result: Root  $\approx 1.80907437$ 
  3. **Get third root using another guess**:
    - Initial Guess: 0.0
    - Expected Result: Root  $\approx -0.86048584$ 
#### Test Case 85
- Description: Test for a polynomial with integer coefficients.
- Polynomial:  $p_1(x) = -5x^8 - 4x^7 - 11x^6 - 5x^5 + 13x^4 + 6x^3 - 8x + 8$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -1.25405831$ 
  2. **Get second root using a different guess**:
    - Initial Guess: 0.5
    - Expected Result: Root  $\approx 0.91737553$ 
#### Test Case 86
- Description: Test for a polynomial of degree 11 with mixed coefficients.
- Polynomial:  $p_1(x) = 6x^{11} - 12x^{10} + 9x^9 + 8x^8 - 7x^7 + 11x^6 + 15x^5 - 13x^4 + 7x^3 - 14x^2 + 11x + 2$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -0.15023882$ 

```

Test Case 87

- Description: Test for a polynomial of degree 11 with mixed coefficients.
- Polynomial: $p_1(x) = 6x^{11} - 12x^{10} + 9x^9 + 8x^8 - 7x^7 + 11x^6 + 15x^5 - 13x^4 + 7x^3 - 14x^2 + 11x + 2$
- Operations:
 1. ****Get first root using Newton's method****:
 - Initial Guess: 1.0
 - Expected Result: Root ≈ 0.27271081
 2. ****Get second root using a different guess****:
 - Initial Guess: -1.0
 - Expected Result: Root ≈ -0.77265566

Test Case 88

- Description: Test for a polynomial with higher degree and varied coefficients.
- Polynomial: $p_1(x) = 5x^{10} + 8x^9 + 2x^8 + 10x^6 + 10x^5 + 12x^4 + 10x^3 + 11x^2 + 12x - 11$
- Operations:
 1. ****Get first root using Newton's method****:
 - Initial Guess: 1.0
 - Expected Result: Root ≈ -1.18719435
 2. ****Get second root using a different guess****:
 - Initial Guess: 0.5
 - Expected Result: Root ≈ 0.49405972

Test Case 89

- Description: Test for a polynomial with mixed coefficients.
- Polynomial: $p_1(x) = -12x^9 + 4x^8 + 5x^7 + 8x^6 - 14x^5 + 13x^4 - 12x^3 + 12x^2 + 10x + 8$
- Operations:
 1. ****Get root using Newton's method****:
 - Initial Guess: 1.0
 - Expected Result: Root ≈ 1.22634169

Test Case 90

- Description: Test for a complex polynomial of degree 9.
- Polynomial: $p_1(x) = x^9 + 15x^8 + 6x^7 + 15x^6 - 13x^5 + 5x^4 - 2x^3 + 12x^2 - 2x + 4$
- Operations:
 1. ****Get first root using Newton's method****:
 - Initial Guess: -15.0
 - Expected Result: Root ≈ -15.02190185

```

2. **Get second root using a different guess**:
  - Initial Guess: -1.0
  - Expected Result: Root ≈ -1.26757023
#### Test Case 91
- Description: Test for a polynomial with zero coefficients.
- Polynomial:  $p_1(x) = -1x^9 - 8x^8 - 3x^7 - 9x^6 + 8x^5 - 6x^4 + 15$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ -7.76317497
  2. **Get second root using a different guess**:
    - Initial Guess: -1.0
    - Expected Result: Root ≈ -1.0
  3. **Get third root using another guess**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ 0.96550363
#### Test Case 92
- Description: Test for a polynomial with various positive and negative coefficients.
- Polynomial:  $p_1(x) = 8x^{10} + 2x^9 - 6x^8 + x^7 - 10x^6 + 9x^5 + 3x^4 + 13x^3 - 5x^2 - 2x + 6$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ -1.49960077
  2. **Get second root using a different guess**:
    - Initial Guess: 0.0
    - Expected Result: Root ≈ -0.65517255
#### Test Case 93
- Description: Test for a polynomial with mixed positive and negative coefficients.
- Polynomial:  $p_1(x) = -6x^{11} + 11x^{10} - 2x^9 + 7x^8 + 12x^7 + 2x^6 + 3x^5 - 12x^4 - 11x^3 + 9x^2 + 4x + 6$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root ≈ 2.1388204
#### Test Case 94

```

```

- Description: Test for a polynomial with predominantly negative
coefficients.
- Polynomial:  $p_1(x) = -3x^{11} - 4x^{10} - 5x^9 - 11x^8 - 4x^7 - 7x^6 - 3x^5 + 5x^4 - 8x^3 - 6x^2 + 6x - 2$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: -1.0
    - Expected Result: Root  $\approx -1.56738018$ 
#### Test Case 95
- Description: Test for a polynomial with a small degree.
- Polynomial:  $p_1(x) = 11x^3 + 8x^2 - 5x - 10$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx 0.90002768$ 
#### Test Case 96
- Description: Test for a complex polynomial of degree 11.
- Polynomial:  $p_1(x) = 9x^{11} - 6x^{10} + 13x^9 - 2x^8 + x^7 + 9x^6 - 8x^5 + 13x^4 + 14x^3 + 4x^2 + 7x + 1$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -0.15000588$ 
#### Test Case 97
- Description: Test for a polynomial with a variety of coefficients.
- Polynomial:  $p_1(x) = -7x^9 + 15x^8 - 7x^7 - 3x^6 - 5x^5 + 13x^4 + 10x^3 - 14x^2 - 14x - 5$ 
- Operations:
  1. **Get root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx -0.70764823$ 
#### Test Case 98
- Description: Test for a polynomial with multiple significant roots.
- Polynomial:  $p_1(x) = 5x^9 - 14x^8 - 14x^7 - 6x^6 - 6x^5 - 14x^4 + 12x^3 - 4x^2 - 8x + 12$ 
- Operations:
  1. **Get first root using Newton's method**:
    - Initial Guess: 1.0
    - Expected Result: Root  $\approx 3.68442743$ 
  2. **Get second root using a different guess**:

```

```

- Initial Guess: -1.0
- Expected Result: Root ≈ -0.86557888
3. **Get third root using another guess**:
- Initial Guess: 0.5
- Expected Result: Root ≈ 0.73331757
#### Test Case 99
- Description: Test for a cubic polynomial with distinct roots.
- Polynomial:  $p_1(x) = 3x^2 + x - 8$ 
- Operations:
1. **Get first root using Newton's method**:
- Initial Guess: -1.0
- Expected Result: Root ≈ -1.80814297
2. **Get second root using a different guess**:
- Initial Guess: 2.0
- Expected Result: Root ≈ 1.47480963
#### Test Case 100
- Description: Test for a polynomial with negative coefficients.
- Polynomial:  $p_1(x) = -3x^7 + 2x^6 - 11x^5 + 5x^4 - 13x^3 - 14x^2 - 11x + 5$ 
- Operations:
1. **Get root using Newton's method**:
- Initial Guess: 0.5
- Expected Result: Root ≈ 0.30451905

```

Test Results (Failed)

ChatGPT

```
[ RUN      ] PolynomialTest.Test35
G:\FCAI\A1-T2-TestCases\test.cpp(390): error: The difference between root1 and -
2.41489316821433 is 1.7518819883336627, which exceeds tolerance, where
root1 evaluates to -0.66301117988066749,
-2.41489316821433 evaluates to -2.4148931682143302, and
tolerance evaluates to 1.0000000000000001e-05.
[ RUN      ] PolynomialTest.Test61
G:\FCAI\A1-T2-TestCases\test.cpp(752): error: Value of: std::isnan(root)
  Actual: false
Expected: true
[ FAILED   ] PolynomialTest.Test61 (1 ms)
[ RUN      ] PolynomialTest.Test63
G:\FCAI\A1-T2-TestCases\test.cpp(774): error: Value of: std::isnan(root)
  Actual: false
Expected: true
[ RUN      ] PolynomialTest.Test66
G:\FCAI\A1-T2-TestCases\test.cpp(811): error: Value of: isnan(root)
  Actual: false
Expected: true

[ RUN      ] PolynomialTest.Test67
G:\FCAI\A1-T2-TestCases\test.cpp(837): error: The difference between root and -2.50562387 is
2.1065216694538975, which exceeds tolerance, where
root evaluates to -0.39910220054610235,
-2.50562387 evaluates to -2.50562387, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(838): error: The difference between root and -0.94087713 is
0.5417749294538976, which exceeds tolerance, where
root evaluates to -0.39910220054610235,
-0.94087713 evaluates to -0.94087712999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test67 (3 ms)

[ RUN      ] PolynomialTest.Test68
G:\FCAI\A1-T2-TestCases\test.cpp(867): error: The difference between root1 and 1.65737085 is
1.0540055869584655, which exceeds tolerance, where
root1 evaluates to 0.60336526304153448,
1.65737085 evaluates to 1.65737085, and
tolerance evaluates to 9.999999999999995e-08.
```

```

G:\FCAI\A1-T2-TestCases\test.cpp(868): error: The difference between root2 and -0.28384746 is
0.88721272298115728, which exceeds tolerance, where
root2 evaluates to 0.60336526298115734,
-0.28384746 evaluates to -0.28384746, and
tolerance evaluates to 9.999999999999995e-08.
[ FAILED ] PolynomialTest.Test68 (3 ms)
[ RUN      ] PolynomialTest.Test69
G:\FCAI\A1-T2-TestCases\test.cpp(883): error: The difference between root1 and 0.86072381 is
0.30108905386498896, which exceeds tolerance, where
root1 evaluates to 1.1618128638649889,
0.86072381 evaluates to 0.86072380999999998, and
tolerance evaluates to 9.999999999999995e-08.
[ FAILED ] PolynomialTest.Test69 (2 ms)
[ RUN      ] PolynomialTest.Test70
G:\FCAI\A1-T2-TestCases\test.cpp(898): error: The difference between root1 and 0.0 is
3.3347631320993796, which exceeds tolerance, where
root1 evaluates to -3.3347631320993796,
0.0 evaluates to 0, and
tolerance evaluates to 9.999999999999995e-08.

[ RUN      ] PolynomialTest.Test73
G:\FCAI\A1-T2-TestCases\test.cpp(1046): error: Value of: isnan(root)
  Actual: false
Expected: true
[ FAILED ] PolynomialTest.Test73 (1 ms)
[ RUN      ] PolynomialTest.Test74
G:\FCAI\A1-T2-TestCases\test.cpp(1056): error: The difference between root and -2.47909554 is
2.4790956620218916, which exceeds tolerance, where
root evaluates to 1.2202189173755114e-07,
-2.47909554 evaluates to -2.4790955399999999, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1057): error: The difference between root and -1.51010908 is
1.5101092020218918, which exceeds tolerance, where
root evaluates to 1.2202189173755114e-07,
-1.51010908 evaluates to -1.5101090800000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test74 (3 ms)
[ RUN      ] PolynomialTest.Test76
G:\FCAI\A1-T2-TestCases\test.cpp(1073): error: The difference between root1 and -0.39196193 is
0.39196193006055913, which exceeds tolerance, where
root1 evaluates to 6.0559102527387256e-11,
-0.39196193 evaluates to -0.39196193000000001, and
tolerance evaluates to 9.999999999999995e-07.

```

```

G:\FCAI\A1-T2-TestCases\test.cpp(1078): error: The difference between root2 and -3.60803807 is
1.0567698832515036, which exceeds tolerance, where
root2 evaluates to -2.5512681867484965,
-3.60803807 evaluates to -3.6080380700000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test76 (3 ms)
[ RUN      ] PolynomialTest.Test77
G:\FCAI\A1-T2-TestCases\test.cpp(1089): error: The difference between root and -1.41421356 is
1.4142135861302172, which exceeds tolerance, where
root evaluates to 2.6130217099936629e-08,
-1.41421356 evaluates to -1.4142135600000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test77 (1 ms)
[ RUN      ] PolynomialTest.Test78
G:\FCAI\A1-T2-TestCases\test.cpp(1100): error: The difference between root and -1.41421356 is
1.4218840324096451, which exceeds tolerance, where
root evaluates to 0.0076704724096450568,
-1.41421356 evaluates to -1.4142135600000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test78 (1 ms)
[ RUN      ] PolynomialTest.Test80
G:\FCAI\A1-T2-TestCases\test.cpp(1122): error: Value of: isnan(root)
  Actual: false
Expected: true
[ FAILED ] PolynomialTest.Test80 (1 ms)

[ RUN      ] PolynomialTest.Test81
G:\FCAI\A1-T2-TestCases\test.cpp(1134): error: The difference between root1 and -0.79058165 is
0.47430983704805052, which exceeds tolerance, where
root1 evaluates to -1.2648914870480505,
-0.79058165 evaluates to -0.79058165000000002, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1139): error: The difference between root2 and -1.04797069 is
0.21692079654085683, which exceeds tolerance, where
root2 evaluates to -1.2648914865408569,
-1.04797069 evaluates to -1.0479706900000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test81 (3 ms)
[ RUN      ] PolynomialTest.Test82
G:\FCAI\A1-T2-TestCases\test.cpp(1150): error: The difference between root and 1.59038197 is
0.96160220735785729, which exceeds tolerance, where
root evaluates to 0.62877976264214264,
1.59038197 evaluates to 1.5903819699999999, and

```



```

tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test82 (1 ms)
[ RUN      ] PolynomialTest.Test83
G:\FCAI\A1-T2-TestCases\test.cpp(1163): error: The difference between root1 and -0.85992705 is
0.30296229245128981, which exceeds tolerance, where
root1 evaluates to -1.1628893424512898,
-0.85992705 evaluates to -0.85992705000000003, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1168): error: The difference between root2 and 0.06926055 is
0.75158835383172196, which exceeds tolerance, where
root2 evaluates to -0.68232780383172198,
0.06926055 evaluates to 0.069260550000000004, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test83 (2 ms)
[ RUN      ] PolynomialTest.Test84
G:\FCAI\A1-T2-TestCases\test.cpp(1180): error: The difference between root1 and -6.71416292 is
5.5520287511255741, which exceeds tolerance, where
root1 evaluates to -1.1621341688744258,
-6.71416292 evaluates to -6.7141629199999997, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1185): error: The difference between root2 and 1.80907437 is
1.2563054969642768, which exceeds tolerance, where
root2 evaluates to 0.5527688730357232,
1.80907437 evaluates to 1.80907437, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1190): error: The difference between root3 and -0.86048584 is
0.71154694177051803, which exceeds tolerance, where
root3 evaluates to -0.14893889822948189,
-0.86048584 evaluates to -0.86048583999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test84 (3 ms)
[ RUN      ] PolynomialTest.Test85
G:\FCAI\A1-T2-TestCases\test.cpp(1202): error: The difference between root1 and -1.25405831 is
2.3441244427612782, which exceeds tolerance, where
root1 evaluates to 1.0900661327612782,
-1.25405831 evaluates to -1.25405831, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1207): error: The difference between root2 and 0.91737553 is
1.714786618746023, which exceeds tolerance, where
root2 evaluates to -0.79741108874602296,
0.91737553 evaluates to 0.91737553000000005, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test85 (3 ms)
[ RUN      ] PolynomialTest.Test86

```

```

G:\FCAI\A1-T2-TestCases\test.cpp(1218): error: The difference between root and -0.15023882 is
6.505830675967359, which exceeds tolerance, where
root evaluates to -6.6560694959673592,
-0.15023882 evaluates to -0.15023882, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test86 (1 ms)
[ RUN      ] PolynomialTest.Test87
G:\FCAI\A1-T2-TestCases\test.cpp(1231): error: The difference between root1 and expectedRoot1
is 6.9287803059673596, which exceeds tolerance, where
root1 evaluates to -6.6560694959673592,
expectedRoot1 evaluates to 0.27271081000000003, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1237): error: The difference between root2 and expectedRoot2
is 5.8834138359673593, which exceeds tolerance, where
root2 evaluates to -6.6560694959673592,
expectedRoot2 evaluates to -0.77265565999999997, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test87 (3 ms)
[ RUN      ] PolynomialTest.Test88
G:\FCAI\A1-T2-TestCases\test.cpp(1250): error: The difference between root1 and expectedRoot1
is 0.34487228489763433, which exceeds tolerance, where
root1 evaluates to -0.84232206510236562,
expectedRoot1 evaluates to -1.18719435, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1256): error: The difference between root2 and expectedRoot2
is 1.3363817851803055, which exceeds tolerance, where
root2 evaluates to -0.8423220651803055,
expectedRoot2 evaluates to 0.49405971999999998, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test88 (2 ms)
[ RUN      ] PolynomialTest.Test89
G:\FCAI\A1-T2-TestCases\test.cpp(1267): error: The difference between root and 1.22634169 is
0.41090827008780562, which exceeds tolerance, where
root evaluates to 0.81543341991219431,
1.22634169 evaluates to 1.2263416899999999, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test89 (1 ms)
[ RUN      ] PolynomialTest.Test90
G:\FCAI\A1-T2-TestCases\test.cpp(1279): error: The difference between root1 and expectedRoot1
is 14.23299092608471, which exceeds tolerance, where
root1 evaluates to -0.78891092391529039,
expectedRoot1 evaluates to -15.021901850000001, and
tolerance evaluates to 9.999999999999995e-07.

```

```

G:\FCAI\A1-T2-TestCases\test.cpp(1285): error: The difference between root2 and expectedRoot2
is 0.47865930257911304, which exceeds tolerance, where
root2 evaluates to -0.78891092742088698,
expectedRoot2 evaluates to -1.26757023, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test90 (2 ms)
[ RUN      ] PolynomialTest.Test91
G:\FCAI\A1-T2-TestCases\test.cpp(1298): error: The difference between root1 and expectedRoot1
is 8.798903855126424, which exceeds tolerance, where
root1 evaluates to 1.0357288851264246,
expectedRoot1 evaluates to -7.7631749699999997, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1310): error: The difference between root3 and expectedRoot3
is 0.070225255126424635, which exceeds tolerance, where
root3 evaluates to 1.0357288851264246,
expectedRoot3 evaluates to 0.96550362999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test91 (2 ms)
[ RUN      ] PolynomialTest.Test92
G:\FCAI\A1-T2-TestCases\test.cpp(1323): error: The difference between root1 and expectedRoot1
is 0.026714704394017907, which exceeds tolerance, where
root1 evaluates to -1.5263154743940179,
expectedRoot1 evaluates to -1.49960077, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1329): error: The difference between root2 and expectedRoot2
is 0.87114292494304457, which exceeds tolerance, where
root2 evaluates to -1.5263154749430445,
expectedRoot2 evaluates to -0.65517254999999996, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test92 (2 ms)
[ RUN      ] PolynomialTest.Test93
G:\FCAI\A1-T2-TestCases\test.cpp(1340): error: The difference between root and 2.1388204 is
1.6712729087597311, which exceeds tolerance, where
root evaluates to 0.46754749124026912,
2.1388204 evaluates to 2.1388204000000002, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test93 (2 ms)
[ RUN      ] PolynomialTest.Test94
G:\FCAI\A1-T2-TestCases\test.cpp(1350): error: The difference between root and -1.56738018 is
0.92937282659589715, which exceeds tolerance, where
root evaluates to -0.63800735340410286,
-1.56738018 evaluates to -1.56738018, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test94 (1 ms)

```

```

[ RUN      ] PolynomialTest.Test95
G:\FCAI\A1-T2-TestCases\test.cpp(1360): error: The difference between root and 0.90002768 is
0.21104926298809734, which exceeds tolerance, where
root evaluates to 1.1110769429880973,
0.90002768 evaluates to 0.90002768, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test95 (1 ms)
[ RUN      ] PolynomialTest.Test96
G:\FCAI\A1-T2-TestCases\test.cpp(1370): error: The difference between root and -0.15000588 is
6.5163993549596997, which exceeds tolerance, where
root evaluates to -6.6664052349596998,
-0.15000588 evaluates to -0.15000588000000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test96 (1 ms)
[ RUN      ] PolynomialTest.Test97
G:\FCAI\A1-T2-TestCases\test.cpp(1380): error: The difference between root and -0.70764823 is
0.70548326372333903, which exceeds tolerance, where
root evaluates to -1.413131493723339,
-0.70764823 evaluates to -0.70764822999999999, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test97 (1 ms)
[ RUN      ] PolynomialTest.Test98
G:\FCAI\A1-T2-TestCases\test.cpp(1392): error: The difference between root1 and expectedRoot1
is 3.4130148373956977, which exceeds tolerance, where
root1 evaluates to 0.27141259260430223,
expectedRoot1 evaluates to 3.6844274299999999, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1398): error: The difference between root2 and expectedRoot2
is 0.28971733551314405, which exceeds tolerance, where
root2 evaluates to -1.1552962155131441,
expectedRoot2 evaluates to -0.86557888000000005, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1404): error: The difference between root3 and expectedRoot3
is 0.46190497737364489, which exceeds tolerance, where
root3 evaluates to 0.27141259262635509,
expectedRoot3 evaluates to 0.73331756999999997, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test98 (4 ms)
[ RUN      ] PolynomialTest.Test99
G:\FCAI\A1-T2-TestCases\test.cpp(1417): error: The difference between root1 and expectedRoot1
is 1.2550893565979937, which exceeds tolerance, where
root1 evaluates to -0.55305361340200621,
expectedRoot1 evaluates to -1.80814297, and
tolerance evaluates to 9.999999999999995e-07.

```

```
G:\FCAI\A1-T2-TestCases\test.cpp(1423): error: The difference between root2 and expectedRoot2
is 0.79675601651029637, which exceeds tolerance, where
root2 evaluates to 0.67805361348970361,
expectedRoot2 evaluates to 1.47480963, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test99 (2 ms)
[ RUN      ] PolynomialTest.Test100
G:\FCAI\A1-T2-TestCases\test.cpp(1434): error: The difference between root and 0.30451905 is
2.979347733994409, which exceeds tolerance, where
root evaluates to 3.2838667839944091,
0.30451905 evaluates to 0.30451905000000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test100 (1 ms)
```

Claude

```
[ RUN      ] PolynomialTest.Test35
Roots found:
-0.663011
-2.41489
G:\FCAI\A1-T2-TestCases\test.cpp(390): error: The difference between root1 and -
2.41489316821433 is 1.7518819883336627, which exceeds tolerance, where
root1 evaluates to -0.66301117988066749,
-2.41489316821433 evaluates to -2.4148931682143302, and
tolerance evaluates to 1.0000000000000001e-05.
Roots found:
-0.663011
-2.41489
[ FAILED   ] PolynomialTest.Test35 (9 ms)
[ RUN      ] PolynomialTest.Test67
Roots found:
-0.399102
-1.06284
G:\FCAI\A1-T2-TestCases\test.cpp(837): error: The difference between root and -2.50562387 is
2.1065216694538975, which exceeds tolerance, where
root evaluates to -0.39910220054610235,
-2.50562387 evaluates to -2.50562387, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(838): error: The difference between root and -0.94087713 is
0.5417749294538976, which exceeds tolerance, where
root evaluates to -0.39910220054610235,
-0.94087713 evaluates to -0.94087712999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test67 (6 ms)
[ RUN      ] PolynomialTest.Test68
Roots found:
0.603365
Roots found:
0.603365
G:\FCAI\A1-T2-TestCases\test.cpp(867): error: The difference between root1 and 1.65737085 is
1.0540055869584655, which exceeds tolerance, where
root1 evaluates to 0.60336526304153448,
1.65737085 evaluates to 1.65737085, and
tolerance evaluates to 9.999999999999995e-08.
G:\FCAI\A1-T2-TestCases\test.cpp(868): error: The difference between root2 and -0.28384746 is
0.88721272298115728, which exceeds tolerance, where
root2 evaluates to 0.60336526298115734,
```

```

-0.28384746 evaluates to -0.28384746, and
tolerance evaluates to 9.999999999999995e-08.
[ FAILED ] PolynomialTest.Test68 (7 ms)
[ RUN      ] PolynomialTest.Test69
Roots found:
1.16181
G:\FCAI\A1-T2-TestCases\test.cpp(883): error: The difference between root1 and 0.86072381 is
0.30108905386498896, which exceeds tolerance, where
root1 evaluates to 1.1618128638649889,
0.86072381 evaluates to 0.86072380999999998, and
tolerance evaluates to 9.999999999999995e-08.
[ FAILED ] PolynomialTest.Test69 (3 ms)
[ RUN      ] PolynomialTest.Test70
No roots found.
G:\FCAI\A1-T2-TestCases\test.cpp(898): error: The difference between root1 and 0.0 is nan,
which exceeds tolerance, where
root1 evaluates to nan,
0.0 evaluates to 0, and
tolerance evaluates to 9.999999999999995e-08.
[ FAILED ] PolynomialTest.Test70 (7 ms)
[ RUN      ] PolynomialTest.Test73
Roots found:
1.19747e-09
G:\FCAI\A1-T2-TestCases\test.cpp(1046): error: Value of: isnan(root)
    Actual: false
    Expected: true
[ FAILED ] PolynomialTest.Test73 (3 ms)
[ RUN      ] PolynomialTest.Test74
Roots found:
1.22022e-07
-0.662204
G:\FCAI\A1-T2-TestCases\test.cpp(1056): error: The difference between root and -2.47909554 is
2.4790956620218916, which exceeds tolerance, where
root evaluates to 1.2202189173755114e-07,
-2.47909554 evaluates to -2.4790955399999999, and
tolerance evaluates to 9.999999999999995e-07.
G:\FCAI\A1-T2-TestCases\test.cpp(1057): error: The difference between root and -1.51010908 is
1.5101092020218918, which exceeds tolerance, where
root evaluates to 1.2202189173755114e-07,
-1.51010908 evaluates to -1.5101090800000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test74 (4 ms)

```

```

[ RUN      ] PolynomialTest.Test76
Roots found:
6.05591e-11
-2.55127
-0.277159
G:\FCAI\A1-T2-TestCases\test.cpp(1073): error: The difference between root1 and -0.39196193 is
0.39196193006055913, which exceeds tolerance, where
root1 evaluates to 6.0559102527387256e-11,
-0.39196193 evaluates to -0.39196193000000001, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-2.55127
-0.277159
0
G:\FCAI\A1-T2-TestCases\test.cpp(1078): error: The difference between root2 and -3.60803807 is
1.0567698832515036, which exceeds tolerance, where
root2 evaluates to -2.5512681867484965,
-3.60803807 evaluates to -3.6080380700000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test76 (4 ms)
[ RUN      ] PolynomialTest.Test77
Roots found:
2.61302e-08
-0.707107
G:\FCAI\A1-T2-TestCases\test.cpp(1089): error: The difference between root and -1.41421356 is
1.4142135861302172, which exceeds tolerance, where
root evaluates to 2.6130217099936629e-08,
-1.41421356 evaluates to -1.4142135600000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test77 (2 ms)
[ RUN      ] PolynomialTest.Test78
Roots found:
0.00767047
-0.707107
0
0.00971551
G:\FCAI\A1-T2-TestCases\test.cpp(1100): error: The difference between root and -1.41421356 is
1.4218840324096451, which exceeds tolerance, where
root evaluates to 0.0076704724096450568,
-1.41421356 evaluates to -1.4142135600000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test78 (2 ms)

```



```

[ RUN      ] PolynomialTest.Test81
Roots found:
-1.26489
-0.954225
G:\FCAI\A1-T2-TestCases\test.cpp(1134): error: The difference between root1 and -0.79058165 is
0.47430983704805052, which exceeds tolerance, where
root1 evaluates to -1.2648914870480505,
-0.79058165 evaluates to -0.79058165000000002, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-1.26489
-0.954225
G:\FCAI\A1-T2-TestCases\test.cpp(1139): error: The difference between root2 and -1.04797069 is
0.21692079654085683, which exceeds tolerance, where
root2 evaluates to -1.2648914865408569,
-1.04797069 evaluates to -1.04797069000000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test81 (5 ms)
[ RUN      ] PolynomialTest.Test82
Roots found:
0.62878
G:\FCAI\A1-T2-TestCases\test.cpp(1150): error: The difference between root and 1.59038197 is
0.96160220735785729, which exceeds tolerance, where
root evaluates to 0.62877976264214264,
1.59038197 evaluates to 1.5903819699999999, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED   ] PolynomialTest.Test82 (2 ms)
[ RUN      ] PolynomialTest.Test83
Roots found:
-1.16289
G:\FCAI\A1-T2-TestCases\test.cpp(1163): error: The difference between root1 and -0.85992705 is
0.30296229245128981, which exceeds tolerance, where
root1 evaluates to -1.1628893424512898,
-0.85992705 evaluates to -0.85992705000000003, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-0.682328
-1.16289
G:\FCAI\A1-T2-TestCases\test.cpp(1168): error: The difference between root2 and 0.06926055 is
0.75158835383172196, which exceeds tolerance, where
root2 evaluates to -0.68232780383172198,
0.06926055 evaluates to 0.069260550000000004, and
tolerance evaluates to 9.999999999999995e-07.

```

```

[ FAILED ] PolynomialTest.Test83 (5 ms)
[ RUN      ] PolynomialTest.Test84
Roots found:
-1.16213
-0.148939
0.552769
G:\FCAI\A1-T2-TestCases\test.cpp(1180): error: The difference between root1 and -6.71416292 is
5.5520287511255741, which exceeds tolerance, where
root1 evaluates to -1.1621341688744258,
-6.71416292 evaluates to -6.7141629199999997, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
0.552769
-1.16213
-0.148939
G:\FCAI\A1-T2-TestCases\test.cpp(1185): error: The difference between root2 and 1.80907437 is
1.2563054969642768, which exceeds tolerance, where
root2 evaluates to 0.5527688730357232,
1.80907437 evaluates to 1.80907437, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-0.148939
-1.16213
0.552769
G:\FCAI\A1-T2-TestCases\test.cpp(1190): error: The difference between root3 and -0.86048584 is
0.71154694177051803, which exceeds tolerance, where
root3 evaluates to -0.14893889822948189,
-0.86048584 evaluates to -0.86048583999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test84 (8 ms)
[ RUN      ] PolynomialTest.Test85
Roots found:
1.09007
-0.797411
G:\FCAI\A1-T2-TestCases\test.cpp(1202): error: The difference between root1 and -1.25405831 is
2.3441244427612782, which exceeds tolerance, where
root1 evaluates to 1.0900661327612782,
-1.25405831 evaluates to -1.25405831, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-0.797411
1.09007

```

```

G:\FCAI\A1-T2-TestCases\test.cpp(1207): error: The difference between root2 and 0.91737553 is
1.714786618746023, which exceeds tolerance, where
root2 evaluates to -0.79741108874602296,
0.91737553 evaluates to 0.91737553000000005, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test85 (4 ms)
[ RUN      ] PolynomialTest.Test86
Roots found:
-6.65607
G:\FCAI\A1-T2-TestCases\test.cpp(1218): error: The difference between root and -0.15023882 is
6.505830675967359, which exceeds tolerance, where
root evaluates to -6.6560694959673592,
-0.15023882 evaluates to -0.15023882, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test86 (2 ms)
[ RUN      ] PolynomialTest.Test87
Roots found:
-6.65607
G:\FCAI\A1-T2-TestCases\test.cpp(1231): error: The difference between root1 and expectedRoot1
is 6.9287803059673596, which exceeds tolerance, where
root1 evaluates to -6.6560694959673592,
expectedRoot1 evaluates to 0.27271081000000003, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-6.65607
G:\FCAI\A1-T2-TestCases\test.cpp(1237): error: The difference between root2 and expectedRoot2
is 5.8834138359673593, which exceeds tolerance, where
root2 evaluates to -6.6560694959673592,
expectedRoot2 evaluates to -0.77265565999999997, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test87 (4 ms)
[ RUN      ] PolynomialTest.Test88
Roots found:
-0.842322
2.02405
G:\FCAI\A1-T2-TestCases\test.cpp(1250): error: The difference between root1 and expectedRoot1
is 0.34487228489763433, which exceeds tolerance, where
root1 evaluates to -0.84232206510236562,
expectedRoot1 evaluates to -1.18719435, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-0.842322
2.02405

```

```

G:\FCAI\A1-T2-TestCases\test.cpp(1256): error: The difference between root2 and expectedRoot2
is 1.3363817851803055, which exceeds tolerance, where
root2 evaluates to -0.8423220651803055,
expectedRoot2 evaluates to 0.49405971999999998, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test88 (6 ms)
[ RUN      ] PolynomialTest.Test89
Roots found:
0.815433
G:\FCAI\A1-T2-TestCases\test.cpp(1267): error: The difference between root and 1.22634169 is
0.41090827008780562, which exceeds tolerance, where
root evaluates to 0.81543341991219431,
1.22634169 evaluates to 1.2263416899999999, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test89 (2 ms)
[ RUN      ] PolynomialTest.Test90
Roots found:
-0.788911
-0.0665695
G:\FCAI\A1-T2-TestCases\test.cpp(1279): error: The difference between root1 and expectedRoot1
is 14.23299092608471, which exceeds tolerance, where
root1 evaluates to -0.78891092391529039,
expectedRoot1 evaluates to -15.021901850000001, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-0.788911
-0.0665695
G:\FCAI\A1-T2-TestCases\test.cpp(1285): error: The difference between root2 and expectedRoot2
is 0.47865930257911304, which exceeds tolerance, where
root2 evaluates to -0.78891092742088698,
expectedRoot2 evaluates to -1.26757023, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test90 (8 ms)
[ RUN      ] PolynomialTest.Test91
Roots found:
1.03573
-1
-0.128813
G:\FCAI\A1-T2-TestCases\test.cpp(1298): error: The difference between root1 and expectedRoot1
is 8.798903855126424, which exceeds tolerance, where
root1 evaluates to 1.0357288851264246,
expectedRoot1 evaluates to -7.7631749699999997, and
tolerance evaluates to 9.999999999999995e-07.

```

```

Roots found:
-1
-0.128813
1.03573
Roots found:
1.03573
-1
-0.128813
G:\FCAI\A1-T2-TestCases\test.cpp(1310): error: The difference between root3 and expectedRoot3
is 0.070225255126424635, which exceeds tolerance, where
root3 evaluates to 1.0357288851264246,
expectedRoot3 evaluates to 0.96550362999999995, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test91 (9 ms)
[ RUN      ] PolynomialTest.Test92
Roots found:
-1.52632
-0.666844
G:\FCAI\A1-T2-TestCases\test.cpp(1323): error: The difference between root1 and expectedRoot1
is 0.026714704394017907, which exceeds tolerance, where
root1 evaluates to -1.5263154743940179,
expectedRoot1 evaluates to -1.49960077, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-1.52632
-0.666844
G:\FCAI\A1-T2-TestCases\test.cpp(1329): error: The difference between root2 and expectedRoot2
is 0.87114292494304457, which exceeds tolerance, where
root2 evaluates to -1.5263154749430445,
expectedRoot2 evaluates to -0.65517254999999996, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test92 (5 ms)
[ RUN      ] PolynomialTest.Test93
Roots found:
0.467547
G:\FCAI\A1-T2-TestCases\test.cpp(1340): error: The difference between root and 2.1388204 is
1.6712729087597311, which exceeds tolerance, where
root evaluates to 0.46754749124026912,
2.1388204 evaluates to 2.1388204000000002, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test93 (2 ms)
[ RUN      ] PolynomialTest.Test94

```

```

Roots found:
-0.638007
G:\FCAI\A1-T2-TestCases\test.cpp(1350): error: The difference between root and -1.56738018 is
0.92937282659589715, which exceeds tolerance, where
root evaluates to -0.63800735340410286,
-1.56738018 evaluates to -1.56738018, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test94 (2 ms)
[ RUN      ] PolynomialTest.Test95
Roots found:
1.11108
G:\FCAI\A1-T2-TestCases\test.cpp(1360): error: The difference between root and 0.90002768 is
0.21104926298809734, which exceeds tolerance, where
root evaluates to 1.1110769429880973,
0.90002768 evaluates to 0.90002768, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test95 (2 ms)
[ RUN      ] PolynomialTest.Test96
Roots found:
-6.66641
G:\FCAI\A1-T2-TestCases\test.cpp(1370): error: The difference between root and -0.15000588 is
6.5163993549596997, which exceeds tolerance, where
root evaluates to -6.6664052349596998,
-0.15000588 evaluates to -0.15000588000000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test96 (3 ms)
[ RUN      ] PolynomialTest.Test97
Roots found:
-1.41313
G:\FCAI\A1-T2-TestCases\test.cpp(1380): error: The difference between root and -0.70764823 is
0.70548326372333903, which exceeds tolerance, where
root evaluates to -1.413131493723339,
-0.70764823 evaluates to -0.7076482299999999, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test97 (3 ms)
[ RUN      ] PolynomialTest.Test98
Roots found:
0.271413
-1.1553
1.36367
G:\FCAI\A1-T2-TestCases\test.cpp(1392): error: The difference between root1 and expectedRoot1
is 3.4130148373956977, which exceeds tolerance, where
root1 evaluates to 0.27141259260430223,

```

```

expectedRoot1 evaluates to 3.6844274299999999, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
-1.1553
0.271413
1.36367
G:\FCAI\A1-T2-TestCases\test.cpp(1398): error: The difference between root2 and expectedRoot2
is 0.28971733551314405, which exceeds tolerance, where
root2 evaluates to -1.1552962155131441,
expectedRoot2 evaluates to -0.86557888000000005, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
0.271413
-1.1553
1.36367
G:\FCAI\A1-T2-TestCases\test.cpp(1404): error: The difference between root3 and expectedRoot3
is 0.46190497737364489, which exceeds tolerance, where
root3 evaluates to 0.27141259262635509,
expectedRoot3 evaluates to 0.73331756999999997, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test98 (10 ms)
[ RUN      ] PolynomialTest.Test99
Roots found:
-0.553054
0.678054
G:\FCAI\A1-T2-TestCases\test.cpp(1417): error: The difference between root1 and expectedRoot1
is 1.2550893565979937, which exceeds tolerance, where
root1 evaluates to -0.55305361340200621,
expectedRoot1 evaluates to -1.80814297, and
tolerance evaluates to 9.999999999999995e-07.
Roots found:
0.678054
-0.553054
G:\FCAI\A1-T2-TestCases\test.cpp(1423): error: The difference between root2 and expectedRoot2
is 0.79675601651029637, which exceeds tolerance, where
root2 evaluates to 0.67805361348970361,
expectedRoot2 evaluates to 1.47480963, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test99 (6 ms)
[ RUN      ] PolynomialTest.Test100
Roots found:
3.28387

```

```
G:\FCAI\A1-T2-TestCases\test.cpp(1434): error: The difference between root and 0.30451905 is
2.979347733994409, which exceeds tolerance, where
root evaluates to 3.2838667839944091,
0.30451905 evaluates to 0.30451905000000001, and
tolerance evaluates to 9.999999999999995e-07.
[ FAILED ] PolynomialTest.Test100 (1 ms)
```


Gtest Code

```
#include "pch.h"
#include "Polynomial.h"
#include <cmath>
#include <stdexcept>
#include <sstream>
using namespace std;

class PolynomialTest : public ::testing::Test {
protected:
    Polynomial p1;
    Polynomial p2;

    void SetUp() override {
        p1 = Polynomial({ 1, -2, 3 }); // Represents  $1 - 2x + 3x^2$ 
        p2 = Polynomial({ -1, 4 });    // Represents  $-1 + 4x$ 
    }
};

// Test Case 1
TEST_F(PolynomialTest, TestCase1) {
    Polynomial p1({ -1, 0, 1 });      //  $x^2 - 1$ 
    Polynomial p2({ -1, 0, 0, 1 });   //  $x^3 - 1$ 
    Polynomial result = p2.compose(p1);
    EXPECT_EQ(result, Polynomial({ -2, 0, 3, 0, -3, 0, 1 })); //  $x^6 - 3x^4 + 3x^2 - 2$ 
}

// Test Case 2
TEST_F(PolynomialTest, TestCase2) {
    Polynomial p1({ 0, 1 });          //  $x$ 
    Polynomial p2({ -1, 0, 0, 1 });   //  $x^3 - 1$ 
    Polynomial sum = p1 + p2;
    EXPECT_EQ(sum, Polynomial({ -1, 1, 0, 1 })); //  $x^3 + x - 1$ 
}

// Test Case 3
TEST_F(PolynomialTest, TestCase3) {
    Polynomial p1({ 0, 1, 1 });       //  $x^2 + x$ 
    Polynomial p2({ -2, 0, 0, 1 });   //  $x^3 - 2$ 
```

```

    Polynomial difference = p1 - p2;
    EXPECT_EQ(difference, Polynomial({ 2, 1, 1, -1 })); //  $-x^3 + x^2 + x + 2$ 
}

// Test Case 4
TEST_F(PolynomialTest, TestCase4) {
    Polynomial p1({ 1, 0, 0, 1 }); //  $x^3 + 1$ 
    Polynomial p2({ -1, 0, 0, 0, 0, 1 }); //  $x^5 - 1$ 
    Polynomial product = p1 * p2;
    EXPECT_EQ(product, Polynomial({ -1, 0, 0, -1, 0, 1, 0, 0, 1 })); //  $x^8 - x^3 - 1$ 
}

// Test Case 5
TEST_F(PolynomialTest, TestCase5) {
    Polynomial p({ -1, 2, 1 }); //  $x^2 + 2x - 1$ 
    double result = p.evaluate(3);
    EXPECT_DOUBLE_EQ(result, 14.0); //  $P(3) = 3^2 + 2*3 - 1 = 14$ 
}

// Test Case 6
TEST_F(PolynomialTest, TestCase6) {
    Polynomial p({ -1, 2, 1 }); //  $x^2 + 2x - 1$ 
    Polynomial integral = p.integral();
    EXPECT_EQ(integral, Polynomial({ 0, -1, 1, 1.0 / 3.0 })); //  $(1/3)x^3 + x^2 - x + C$ 
}

// Test Case 7
TEST_F(PolynomialTest, TestCase7) {
    Polynomial p({ -1, 2, 1 }); //  $x^2 + 2x - 1$ 
    double integral = p.integral(0, 9); // Assuming integral method calculates the definite integral
    EXPECT_DOUBLE_EQ(integral, 315); // Integral from 0 to 9
}

// Test Case 8
TEST_F(PolynomialTest, TestCase8) {
    Polynomial p({ 1 }); //  $x$ 
    Polynomial derivative = p.derivative();
    EXPECT_EQ(derivative, Polynomial({ 0 })); // 0
}

```

```

// Test Case 9
TEST_F(PolynomialTest, TestCase9) {
    Polynomial p({ 1 }); // x
    Polynomial integral = p.integral();
    EXPECT_EQ(integral, Polynomial({ 0, 1 })); // x
}

// Test Case 10
TEST_F(PolynomialTest, TestCase10) {
    Polynomial p({ 0, -1, 0, 1, 0, 1 }); //  $x^5 + x^3 - x$ 
    double result = p.evaluate(2);
    EXPECT_DOUBLE_EQ(result, 38.0); //  $P(2) = 32 + 8 - 2 = 38$ 
}

// Test Case 11
TEST_F(PolynomialTest, TestCase11) {
    Polynomial p({ 0, -1, 0, 1, 0, 1 }); //  $x^5 + x^3 - x$ 
    Polynomial derivative = p.derivative();
    EXPECT_EQ(derivative, Polynomial({ -1, 0, 3, 0, 5 })); //  $5x^4 + 3x^2 - 1$ 
}

// Test Case 12
TEST_F(PolynomialTest, TestCase12) {
    Polynomial p({ 0, -1, 0, 1, 0, 1 }); //  $x^5 + x^3 - x$ 
    Polynomial integral = p.integral();
    EXPECT_EQ(integral, Polynomial({ 0, 0, -0.5, 0, 0.25, 0, (1.0 / 6) })); //  $(1/6)x^6 + (1/4)x^4 - (1/2)x^2 + C$ 
}

// Test Case 13
TEST_F(PolynomialTest, TestCase13) {
    Polynomial p({ 0, 0, -1, 0, 1, 0, -1, 0, 1 }); //  $x^8 - x^6 + x^4 - x^2$ 
    double result = p.evaluate(3);
    EXPECT_DOUBLE_EQ(result, 5904); //  $P(3) = 5904$ 
}

// Test Case 14
TEST_F(PolynomialTest, TestCase14) {
    Polynomial p({ 0, 0, -1, 0, 1, 0, -1, 0, 1 }); //  $x^8 - x^6 + x^4 - x^2$ 
    Polynomial derivative = p.derivative();
    EXPECT_EQ(derivative, Polynomial({ 0, -2, 0, 4, 0, -6, 0, 8 })); //  $8x^7 - 6x^5 + 4x^3 - 2x$ 
}

```

```

}

// Test Case 15
TEST_F(PolynomialTest, TestCase15) {
    Polynomial p({ 0, 0, -1, 0, 1, 0, -1, 0, 1 }); //  $x^8 - x^6 + x^4 - x^2$ 
    Polynomial second_derivative = p.derivative().derivative();
    EXPECT_EQ(second_derivative, Polynomial({ -2, 0, 12, 0, -30, 0, 56 })); //  $56x^6 - 30x^4 + 12x^2 - 2$ 
}

// Test Case 16
TEST_F(PolynomialTest, TestCase16) {
    Polynomial p({ 0, 0, -1, 0, 1, 0, -1, 0, 1 }); //  $x^8 - x^6 + x^4 - x^2$ 
    double result = p.evaluate(10);
    EXPECT_DOUBLE_EQ(result, 99009900); //  $P(10) = 99009900$ 
}

// Test Case 17
TEST_F(PolynomialTest, TestCase17) {
    Polynomial p({ -1, 1, -3, 5 }); //  $5x^4 - 3x^2 + x - 1$ 
    double result = p.evaluate(0);
    EXPECT_DOUBLE_EQ(result, -1); //  $P(0) = -1$ 
}

// Test Case 18
TEST_F(PolynomialTest, TestCase18) {
    Polynomial p({ -1, 1, -3, 0, 5 }); //  $5x^4 - 3x^2 + x - 1$ 
    Polynomial derivative = p.derivative();
    EXPECT_EQ(derivative, Polynomial({ 1, -6, 0, 20 })); //  $20x^3 - 6x + 1$ 
}

// Test Case 19
TEST_F(PolynomialTest, TestCase19) {
    Polynomial p({ -1, 1, -3, 0, 5 }); //  $5x^4 - 3x^2 + x - 1$ 
    Polynomial integral = p.integral();
    EXPECT_EQ(integral, Polynomial({ 0, -1, 0.5, -1.0, 0, 1.0 })); //  $x^5 - x^3 + (1/2)x^2 - x + C$ 
}

// Test Case 20
TEST_F(PolynomialTest, TestCase20) {

```

```

    Polynomial p({ -1, 1, -3, 0, 5 }); //  $5x^4 - 3x^2 + x - 1$ 
    double integral = p.integral(0, 2); // Definite integral from 0 to 2
    EXPECT_DOUBLE_EQ(integral, 24); // Calculate this value based on the polynomial
}

// Test Case 21
TEST_F(PolynomialTest, TestCase21) {
    // 1. Evaluate P at x = -1
    Polynomial P({ 5, 4, 3 }); //  $3x^2 + 4x + 5$ 
    EXPECT_EQ(P.evaluate(-1), 4);

    // 2. Find the derivative of P
    Polynomial derivative = P.derivative();
    EXPECT_EQ(derivative, Polynomial({ 4, 6 })); //  $6x + 4$ 

    // 3. Compute the definite integral from 0 to 2
    double integralValue = P.integral(0, 2);
    EXPECT_DOUBLE_EQ(integralValue, 26); // Adjusted expected value
}

// Test Case 22
TEST_F(PolynomialTest, TestCase22) {
    Polynomial P({ 1, 0, -2, 0, 3, 0, 1 }); // Represents  $x^6 + 3x^4 - 2x^2 + 1$ 
    // 1. Evaluate P at x = 30
    EXPECT_EQ(P.evaluate(30), 731428201);

    // 2. Find the derivative of P
    Polynomial derivative = P.derivative();
    EXPECT_EQ(derivative, Polynomial({ 0, -4, 0, 12, 0, 6 })); //  $6x^5 + 12x^3 - 4x$ 
}

// Test Case 23
TEST_F(PolynomialTest, TestCase23) {
    Polynomial P({ (sqrt(10) / 3), 0, 0, (sqrt(6) / 3), (sqrt(3) / 2) });
    // 1. Evaluate P at x = sqrt(3)
    EXPECT_DOUBLE_EQ(P.evaluate(sqrt(2)), ((sqrt(10) + 10 * sqrt(3)) / 3)); // Use
    EXPECT_DOUBLE_EQ for floating-point comparison

    // 2. Find the derivative of P
    Polynomial derivative = P.derivative();

```

```

    EXPECT_EQ(derivative, Polynomial({ 0, 0, sqrt(6), 2 * sqrt(3) })); // Ensure this is
correct
    derivative = derivative.derivative(); // 2nd Derivative
    EXPECT_EQ(derivative, Polynomial({ 0, 2 * sqrt(6), 6 * sqrt(3) })); // Ensure this is
correct
}

// Test Case 24
TEST_F(PolynomialTest, TestCase24) {
    Polynomial P({ (sqrt(10) / 3), 0, 0, (sqrt(6) / 3), (sqrt(3) / 2) });

    // Compute the indefinite integral
    Polynomial Integral = P.integral();
    EXPECT_EQ(Integral, Polynomial({ 0, (sqrt(10) / 3), 0, 0, (sqrt(6) / 12), (sqrt(3) / 10)
})); // Adjust expected values if needed

    // Compute the definite integral from sqrt(3) to sqrt(10)
    double definiteIntegral = P.integral(sqrt(3), sqrt(10));

    // Use EXPECT_DOUBLE_EQ for floating-point comparison
    EXPECT_NEAR(definiteIntegral, 72.15514444, 1e-5); // Allow for some floating-point
tolerance
}

// Test Case 25
TEST_F(PolynomialTest, TestCase25) {
    Polynomial P({ 1, 0, (0.5), (5.0 / 4.0), (2.0 / 3.0) }); // Represents  $(2/3)x^4 + (5/4)x^3 + (1/2)x^2 + 1$ 
    // Compute the indefinite integral
    Polynomial Integral = P.integral();
    EXPECT_EQ(Integral, Polynomial({ 0, 1, 0, (0.5 / 3.0), (5.0 / 16.0), (2.0 / 15.0) })); //
Expected values adjusted for integration

    // Compute the definite integral from sqrt(3)/2 to sqrt(13)
    double definiteIntegral = Integral.integral(0.5, 2.5);

    // Use EXPECT_DOUBLE_EQ for exact floating-point comparison
    EXPECT_DOUBLE_EQ(definiteIntegral, 10337.0/640); // Expected value for the definite
integral
}

// Test Case 26
TEST_F(PolynomialTest, TestCase26) {

```

```

    Polynomial p1({ 0, 3, 0, 0, (sqrt(10) / 5), 0, 0, 0, (sqrt(3) / 3) }); // Represents
    (sqrt(3)/3)x^8 + (sqrt(10)/5)x^6 + 3x

    Polynomial p2({ 0, -(sqrt(3) / 2), 0, 0, (sqrt(6) / 4) }); // Represents (sqrt(6)/4)x^4 -
    (sqrt(3)/2)x

    // Compute p3 as the product of p1 and p2.
    Polynomial p3 = p1 * p2;
    // Compute the indefinite integral of p3 (the antiderivative).
    Polynomial integral = p3.integral();
    double expected_value = 117614.75952999208;
    double definiteIntegral = integral.integral((sqrt(3) / 2), sqrt(13));

    // Replace with the actual expected definite integral value.
    EXPECT_DOUBLE_EQ(definiteIntegral, expected_value);
}

// Test Case 27
TEST_F(PolynomialTest, TestCase27) {
    // Define the polynomial P(x) = x^2 - 1
    Polynomial P({ -1, 0, 1 });
    // Use getRoot to find the root
    double root = P.getRoot(2, 1e-6, 100);
    // Verify that the root is close to the expected value
    EXPECT_NEAR(P.evaluate(root), 0.0, 1e-6);
}

// Test Case 28
// Test case for the polynomial x^2 + 1, which has complex roots (i and -i)
TEST_F(PolynomialTest, TestCase28) {
    // Polynomial represents x^2 + 1, which has no real roots.
    Polynomial p({ 1, 0, 1 });
    double initialGuess = 1.0; // Starting guess
    double tolerance = 1e-6;   // Small tolerance
    int maxIter = 100;         // Maximum iterations

    // Get the root (or indication that no root exists)
    double root = p.getRoot(initialGuess, tolerance, maxIter);

    // Check if the root is NaN (or any special value you choose to indicate no real root)
    EXPECT_TRUE(std::isnan(root)); // If using NaN to indicate no real root
    // Or if you use a specific value to indicate no real root, you can check against that
    value

```

```

    // EXPECT_EQ(root, -1.0); // Assuming -1.0 indicates no real root (use the value you
    decide)
}

// Test Case 29
TEST_F(PolynomialTest, TestCase29) {
    Polynomial poly({ 1, -2, 5 }); // Represents the polynomial  $x^2 - 2x + 5$ , which has no
    real roots.

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-5;    // Tolerance for convergence
    int maxIter = 10;           // Maximum iterations

    // Get the root (or indication that no root exists)
    double root = poly.getRoot(initialGuess, tolerance, maxIter);

    // Check if the root is NaN (or any special value you choose to indicate no real root)
    EXPECT_TRUE(isnan(root)); // Assuming you return NaN to indicate no real root
    // Or check against a specific value indicating no real root
    // EXPECT_EQ(root, -1.0); // Use this if you choose a specific value to indicate no real
    root
}

// Test 30
TEST_F(PolynomialTest, Test30) {
    //Test -->27
    Polynomial p1({ -9 });
    Polynomial p2({ 12 });
    EXPECT_EQ(p1.integral(), Polynomial({ 0, -9 }));
    EXPECT_EQ(p1 - p2, Polynomial({ -21 }));
    EXPECT_EQ(p1 * p2, Polynomial({ -108 }));
    EXPECT_EQ(p1.derivative(), Polynomial({ 0 }));
    EXPECT_EQ(p1.evaluate(2), -9);
    EXPECT_DOUBLE_EQ(p1.integral(0, 1), -9);
}

// Test 31
TEST_F(PolynomialTest, Test31) {
    Polynomial p1({ 5 }); // Polynomial 1: 5

```



```

Polynomial p2({ 1 }); // Polynomial 2: 1

// Sum
EXPECT_EQ((p1 + p2), Polynomial({ 6 })); // Sum: 6

// Subtraction
EXPECT_EQ((p1 - p2), Polynomial({ 4 })); // Subtraction: 4

// Product
EXPECT_EQ((p1 * p2), Polynomial({ 5 })); // Product: 5

// Degree
EXPECT_EQ(p1.degree(), 0); // Degree of Polynomial: 0

// Evaluation
EXPECT_DOUBLE_EQ(p1.evaluate(2), 5); // Evaluation at x=2: 5

// Derivative
EXPECT_EQ(p1.derivative(), Polynomial({ 0 })); // Derivative: 0

// Integral
EXPECT_EQ(p1.integral(), Polynomial({ 0 , 5 })); // Integral: 5*x

// Definite integral
EXPECT_EQ(p1.integral(0, 1), 5); // Integral: 5*x

double initialGuess = 0.0; // Starting point for Newton's method
double tolerance = 1e-5;    // Tolerance for convergence
int maxIter = 10;           // Maximum iterations

// Expect the result to be NaN or another indication of no real roots
double root = p1.getRoot(initialGuess, tolerance, maxIter);

// Check if the root is NaN
EXPECT_TRUE(std::isnan(root)); // If you want to check for NaN
// Alternatively, if your implementation defines a specific behavior for no real root
}

// Test 32
TEST_F(PolynomialTest, Test32) {

```

```

//Test -->28
Polynomial p1({ -1,-10,10,-1,2,-3,13,5,-4,13,13});
Polynomial p2({ 4,12,7,12,3,3,-8,-1,7,9,12 });
EXPECT_EQ(p1 .derivative(), Polynomial({-10,20,-3,8,-15,78,35,-32,117,130}));
EXPECT_EQ(p1.evaluate(2), 20363);
}

// Test 33
TEST_F(PolynomialTest, Test33) {
    //Test -->28
    Polynomial p1({ -1,-10,10,-1,2,-3,13,5,-4,13,13 });
    Polynomial p2({ 4,12,7,12,3,3,-8,-1,7,9,12 });
    EXPECT_EQ(p1.integral(), Polynomial({ 0,-1,-5,10.0/3 , -1.0/4 , 2.0/5 , -0.5 , 13.0/7 , 5.0
/8 ,-4.0/9 ,13.0 /10 , 13.0/11 }));
    EXPECT_NEAR(p1.integral(0, 1), 41659.0 / 27720, 1e-5);
}

// Test 34
TEST_F(PolynomialTest, Test34) {
    // Polynomial 1:  $12x^8 - 10x^7 + x^6 + x^5 + 10x^4 - 12x^3 - 6x^2 - 9x - 6$ 

    Polynomial p1({ -6, -9, -6, -12, 10, 1, 1, -10, 12 });
    //Polynomial 2:  $-2x^8 + 3x^7 + 2x^6 - x^5 - 6x^4 + 8x^3 + 12x^2 - x - 6$   $14x + 11$ 

    Polynomial p2({ -6, -1, 12, 8, -6, -1, 2, 3, -2 });
    EXPECT_EQ((p1 - p2), Polynomial({ 0, -8, -18, -20, 16, 2, -1, -13, 14 })); // Product: 5
    EXPECT_EQ(p1.integral(0, 1), -367.0 / 28);
}

// Test 35
TEST_F(PolynomialTest, Test35) {
    // Polynomial 1:  $-4x^{10} - 4x^9 + 9x^8 - 6x^7 + 7x^6 - 9x^5 + 6x^4 - 11x^3 + 8x^2 - x - 11$ 
    Polynomial p1({ -11, -1, 8, -11, 6, -9, 7, -6, 9, -4, -4 });

    // Real roots of Polynomial 1: [-2.41489316821433, -0.663011179796566]
    double initialGuess1 = -2.0; // Starting guess close to -2.41489
    double initialGuess2 = -0.5; // Starting guess close to -0.66301
    double tolerance = 1e-5; // Smaller tolerance
    int maxIter = 200; // Increased maximum iterations

    // Test for the first root

```

```

    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, -2.41489316821433, tolerance); // Check close to the expected root

    // Test for the second root
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, -0.663011179796566, tolerance); // Check close to the expected root
}

TEST_F(PolynomialTest, Test36) {
    //Test -->26
    Polynomial p1({ -1,-4,6,-8,-2,10,6,9 });
    Polynomial p2({ -14,-7,4,7,-10,8,0,15});
    EXPECT_EQ(p1*p2, Polynomial({ 14,63,-60,47,90,-84,-310,-29,-73,52,-37,-72,222,90,135 }));
    EXPECT_EQ(p1.evaluate(2), 1775);
}

TEST_F(PolynomialTest, Test37) {
    //Test -->26
    Polynomial p1({ -1,-4,6,-8,-2,10,6,9 });
    Polynomial p2({ -14,-7,4,7,-10,8,0,15 });
    EXPECT_EQ(p1 - p2, Polynomial({ 13,3,2,-15,8,2,6,-6 }));
    EXPECT_DOUBLE_EQ(p1.integral(0, 1), 209.0/ 840);
}

TEST_F(PolynomialTest, Test38) {
    //Test -->26
    Polynomial p1({ -1,-4,6,-8,-2,10,6,9 });
    Polynomial p2({ -14,-7,4,7,-10,8,0,15 });
    EXPECT_EQ(p1.derivative(), Polynomial({ -4,12,-24,-8,50,36 ,63}));
    EXPECT_DOUBLE_EQ(p1.degree(), 7);
}

TEST_F(PolynomialTest, Test39) {
    //Test -->46
    Polynomial p1({ -13,12,15,5,12,-2,13,-10,2,12,15 });
    Polynomial p2({ -12,-14,-10,-2,1,-3,14,1,-8,-10,-11 });

```

```

    EXPECT_EQ(p1+p2, Polynomial({ -25,-2,5,3,13,-5,27,-9,-6,2,4 }));
    EXPECT_DOUBLE_EQ(p1.degree(), 10);
}

TEST_F(PolynomialTest, Test40) {
    //Test -->46
    Polynomial p1({ -13,12,15,5,12,-2,13,-10,2,12,15 });
    Polynomial p2({ -12,-14,-10,-2,1,-3,14,1,-8,-10,-11 });
    EXPECT_EQ(p1 * p2, Polynomial({156,38,-218,-364,-401,-173,-461,49,313,-17,-253,-721,-273,-
310,-219,97,163,9,-262,-282,-165 }));
    EXPECT_DOUBLE_EQ(p1.integral(0,1), 16319.0 / 3465);
    EXPECT_DOUBLE_EQ(p1.evaluate(2), 21807);
}

TEST_F(PolynomialTest, Test41) {
    Polynomial p1({ -13,12,15,5,12,-2,13,-10,2,12,15 });
    Polynomial p2({ -12,-14,-10,-2,1,-3,14,1,-8,-10,-11 });
    EXPECT_EQ(p1.integral(), Polynomial({0,-13,6,5,5.0 / 4,12.0 / 5,-1.0 / 3,13.0 / 7,-5.0 /
4,2.0 / 9,6.0 / 5,15.0 / 11}));
    EXPECT_EQ(p1.derivative(), Polynomial({12,30,15,48,-10,78,-70,16,108,150}));
}

TEST_F(PolynomialTest, Test42) {
    // Create two polynomials
    Polynomial poly1({ 10, 4, 0, 10, 4, -8 }); // Represents  $-8x^5 + 4x^4 + 10x^3 + 4x + 10$ 
    Polynomial poly2({ 12, 15, 8, 1, 11, -14 }); // Represents  $-14x^5 + 11x^4 + x^3 + 8x^2 + 15x + 12$ 

    // Test addition
    Polynomial sum = poly1 + poly2;
    EXPECT_EQ(sum, Polynomial({ 22, 19, 8, 11, 15, -22 })); //  $-22x^5 + 15x^4 + 11x^3 + 8x^2 + 19x + 22$ 

    // Test subtraction
    Polynomial difference = poly1 - poly2;
    EXPECT_EQ(difference, Polynomial({ -2, -11, -8, 9, -7, 6 })); //  $6x^5 - 7x^4 + 9x^3 - 8x^2 - 11x - 2$ 

    // Test multiplication
    Polynomial product = poly1 * poly2;
    EXPECT_EQ(product, Polynomial({ 120, 198, 140, 162, 312, -52, -134, 50, -104, -144, 112
})); //  $112x^{10} - 144x^9 - 104x^8 + 50x^7 - 134x^6 - 52x^5 + 312x^4 + 162x^3 + 140x^2 + 198x + 120$ 

```

```

// Test degree
EXPECT_EQ(poly1.degree(), 5);
EXPECT_EQ(poly2.degree(), 5);

// Test evaluation at x=2
EXPECT_DOUBLE_EQ(poly1.evaluate(2), -94); // Evaluation of poly1 at x=2

// Test derivative
Polynomial derivative = poly1.derivative();
EXPECT_EQ(derivative, Polynomial({ 4, 0, 30, 16, -40 })); //  $4 + 0x + 30x^2 + 16x^3 - 40x^4$ 

// Test integral
Polynomial integral = poly1.integral();
EXPECT_EQ(integral, Polynomial({ 0, 10, 2, 0, 5.0 / 2, 4.0 / 5, -4.0 / 3 })); //  $0 + 10x + 2x^2 + 0x^3 + 5/2x^4 + 4/5x^5 - 4/3x^6$ 
// Test definite integral from 0 to 1
EXPECT_DOUBLE_EQ(poly1.integral(0, 1), 419.0 / 30); // Definite integral from x=0 to x=1
}

TEST_F(PolynomialTest, Test43) {
    // Create two polynomials
    Polynomial poly1({ -11, 15, -11, -12, -9, 14, -1, 14 }); //  $14x^7 - x^6 + 14x^5 - 9x^4 - 12x^3 - 11x^2 + 15x - 11$ 
    Polynomial poly2({ -13, 9, -11, 6, 13, 2, 13, -11 }); //  $-11x^7 + 13x^6 + 2x^5 + 13x^4 + 6x^3 - 11x^2 + 9x - 13$ 

    // Test addition
    Polynomial sum = poly1 + poly2;
    EXPECT_EQ(sum, Polynomial({ -24, 24, -22, -6, 4, 16, 12, 3 })); //  $14x^{11} + 9x^{10} + 4x^9 - 2x^8 + 7x^7 - 7x^6 - 14x^5 + x^4 - 13x^3 - 13x^2 + 14x - 4$ 

    // Test subtraction
    Polynomial difference = poly1 - poly2;
    EXPECT_EQ(difference, Polynomial({ 2, 6, 0, -18, -22, 12, -14, 25 })); //  $8x^{11} - 17x^{10} - 16x^9 - 14x^8 + x^7 + 9x^6 + 12x^5 - 3x^4 - 15x^3 - 15x^2 + 4x + 16$ 

    // Test multiplication
    Polynomial product = poly1 * poly2;
    EXPECT_EQ(product, Polynomial({ 143, -294, 399, -174, 77, -24, -90, -261, -228, -31, 114, 461, -139, 193, -154 })); // Product coefficients

```

```

// Test degree
EXPECT_EQ(poly1.degree(), 7);
EXPECT_EQ(poly2.degree(), 7);

// Test evaluation at x=2
EXPECT_DOUBLE_EQ(poly1.evaluate(2), 1911); // Evaluation of poly1 at x=2

// Test derivative
Polynomial derivative = poly1.derivative();
EXPECT_EQ(derivative, Polynomial({ 15, -22, -36, -36, 70, -6, 98 })); // Derivative
coefficients

// Test definite integral from 0 to 1
EXPECT_DOUBLE_EQ(poly1.integral(0, 1), -3371.0 / 420); // Definite integral
}

TEST_F(PolynomialTest, Test44) {
    // Create two polynomials
    Polynomial poly1({ -11, 15, -11, -12, -9, 14, -1, 14 }); //  $14x^7 - x^6 + 14x^5 - 9x^4 - 12x^3 - 11x^2 + 15x - 11$ 
    Polynomial poly2({ -13, 9, -11, 6, 13, 2, 13, -11 }); //  $-11x^7 + 13x^6 + 2x^5 + 13x^4 + 6x^3 - 11x^2 + 9x - 13$ 

    // Test addition
    Polynomial sum = poly1 + poly2;
    EXPECT_EQ(sum, Polynomial({ -24, 24, -22, -6, 4, 16, 12, 3 })); //  $14x^{11} + 9x^{10} + 4x^9 - 2x^8 + 7x^7 - 7x^6 - 14x^5 + x^4 - 13x^3 - 13x^2 + 14x - 4$ 

    // Test subtraction
    Polynomial difference = poly1 - poly2;
    EXPECT_EQ(difference, Polynomial({ 2, 6, 0, -18, -22, 12, -14, 25 })); //  $8x^{11} - 17x^{10} - 16x^9 - 14x^8 + x^7 + 9x^6 + 12x^5 - 3x^4 - 15x^3 - 15x^2 + 4x + 16$ 

    // Test multiplication
    Polynomial product = poly1 * poly2;
    EXPECT_EQ(product, Polynomial({ 143, -294, 399, -174, 77, -24, -90, -261, -228, -31, 114, 461, -139, 193, -154 })); // Product coefficients

    // Test degree
    EXPECT_EQ(poly1.degree(), 7);
    EXPECT_EQ(poly2.degree(), 7);

    // Test evaluation at x=2
    EXPECT_DOUBLE_EQ(poly1.evaluate(2), 1911); // Evaluation of poly1 at x=2

```

```

    // Test derivative
    Polynomial derivative = poly1.derivative();
    EXPECT_EQ(derivative, Polynomial({ 15, -22, -36, -36, 70, -6, 98 })); // Derivative
coefficients
    // Test definite integral from 0 to 1
    EXPECT_DOUBLE_EQ(poly1.integral(0, 1), -3371.0 / 420); // Definite integral
}

TEST_F(PolynomialTest, Test45) {
    //Test -->85
    Polynomial p1({ 10,-1,15,-11,14,-3,8,4,-6 });
    Polynomial p2({-10,-8,5,-14,-10,9,4,-10,-10});
    EXPECT_EQ(p1.evaluate(2), -404);
    EXPECT_EQ(p1*p2, Polynomial({ -100,-70,-92,-155,-63,-247,49,-174,-159,-72,-177,74,18,-88,-
144,20,60 }));
}

TEST_F(PolynomialTest, Test46) {
    //Test -->85
    // Remmeber to do Real roots of Polynomial 1: [-1.58327556220727, 1.71953761729611]

    Polynomial p1({ 10,-1,15,-11,14,-3,8,4,-6 });
    Polynomial p2({ -10,-8,5,-14,-10,9,4,-10,-10 });
    EXPECT_EQ(p1.integral(), Polynomial({0,10,-0.5,5 ,(-11.0/4) ,(14.0/5) , -0.5,8.0/7 ,0.5 ,
-2.0/3}));
}

TEST_F(PolynomialTest, Test47) {
    //Test -->86
    Polynomial p1({ -14,9,-6,3,-12,7,10});
    Polynomial p2({-3,-13,14,-2,-14,-1,12 });
    EXPECT_EQ(p1-p2, Polynomial({-11,22,-20,5,2,8,-2 }));
    EXPECT_DOUBLE_EQ(p1.integral(0,1), -4433.0 / 420);
}

TEST_F(PolynomialTest, Test48) {
    //Test -->86
    // Remmeber to do Real roots of Polynomial 1: [-1.76254517978563, 1.04918639904969]

    Polynomial p1({ -14,9,-6,3,-12,7,10 });
    Polynomial p2({ -3,-13,14,-2,-14,-1,12 });
    EXPECT_EQ(p1.derivative(), Polynomial({9,-12,9,-48,35,60}));
    EXPECT_EQ(p1.evaluate(2), 676);
}

TEST_F(PolynomialTest, Test49) {

```

```

//Test -->89

Polynomial p1({ 13,8,0,-10,3,-7,-1,-13,2,12});
Polynomial p2({ -2 , -7 , 2 , -11 , -9 , -5 , 14 , -6 , 12 });
EXPECT_EQ(p1 + p2 , Polynomial({11,1,2,-21,-6,-12,13,-19,14,12 }));
EXPECT_EQ(p1.integral(0,1), 34241.0 / 2520);
}

TEST_F(PolynomialTest, Test50) {
    //Test -->89
    // Remmeber to do Real roots of Polynomial 1: [-1.37922798199990]
    Polynomial p1({ 13,8,0,-10,3,-7,-1,-13,2,12 });
    Polynomial p2({ -2 , -7 , 2 , -11 , -9 , -5 , 14 , -6 , 12 });
    EXPECT_EQ(p1 .integral (), Polynomial({0,13,4,0,-5.0/2 , 3.0/5 , -7.0/6 , -1.0/7 , -13.0/8
,2.0/9,6.0/5}));
    EXPECT_EQ(p1.integral(0, 1), 34241.0 / 2520);
}

TEST_F(PolynomialTest, Test51) {
    //Test -->90
    Polynomial p1({ -13,0,-11,-7,-10,11 });
    Polynomial p2({ 4,-6,-6,-2,-2,-14 });
    EXPECT_EQ(p1*p2, Polynomial({ -52,78,34,64,94,350,30,122,96,118,-154 }));
    EXPECT_EQ(p1.evaluate(2),79);
}

TEST_F(PolynomialTest, Test52) {
    //Test -->90
    // Remmeber to do Real roots of Polynomial 1: [1.73684576071411]

    Polynomial p1({ -13,0,-11,-7,-10,11 });
    Polynomial p2({ 4,-6,-6,-2,-2,-14 });
    EXPECT_EQ(p1.integral(), Polynomial({ 0,-13,0,-11.0 / 3,-7.0 / 4,-2,11.0 / 6 }));
    EXPECT_EQ(p1.integral(0,1), -223.0/12);
}

TEST_F(PolynomialTest, Test53) {
    //Test -->93
    Polynomial p1({ -5,-1,0,-6,-8,15,-9,7,-7,9,-14,-10 });
    Polynomial p2({10,-8,-12,-1,15,2,-4,-7,-8,6,10,12});
    EXPECT_EQ(p1.derivative(), Polynomial({-1,0,-18,-32,75 , -54 , 49 , -56 , 81,-140,-110}));
    EXPECT_EQ(p1.evaluate(2), -31383);
}

TEST_F(PolynomialTest, Test54) {
    //Test -->93

```



```

// Remmeber to Real roots of Polynomial 1: [-2.13178791593010]

Polynomial p1({ -5,-1,0,-6,-8,15,-9,7,-7,9,-14,-10 });
Polynomial p2({ 10,-8,-12,-1,15,2,-4,-7,-8,6,10,12 });
EXPECT_EQ(p1.integral(), Polynomial({0,-5,-0.5,0,-3.0/2,-8.0/5,5.0 /2 ,-9.0/7, 7.0/8,-
7.0/9,9.0/10,-14.0/11,-5.0/6 }));

EXPECT_EQ(p1.evaluate(2), -31383);
EXPECT_EQ(p1.integral(0, 1), -235469.0 / 27720);
}

TEST_F(PolynomialTest, Test55) {
    //Test -->97
    Polynomial p1({ -4,8,11 });
    Polynomial p2({ -4,4,15 });
    EXPECT_EQ(p1.integral(), Polynomial({ 0,-4,4,11.0/3 }));
    EXPECT_EQ(p1-p2, Polynomial({ 0,4,-4 }));
    EXPECT_EQ(p1 * p2, Polynomial({ 16,-48,-72,164,165}));
    EXPECT_EQ(p1.derivative(), Polynomial({ 8,22 }));
    EXPECT_EQ(p1.evaluate(2), 56);
    EXPECT_DOUBLE_EQ(p1.integral(0, 1), 11.0/3);
    //Real roots of Polynomial 1: [-1.06781515385589, 0.340542426583167]
}

TEST_F(PolynomialTest, Test56) {
    // Polynomial:  $P(x) = -1x^9 - 10x^8 + 10x^7 - 1x^6 + 2x^5 - 3x^4 + 13x^3 + 5x^2 - 4x + 13$ 
    Polynomial p1({ -1, -10, 10, -1, 2, -3, 13, 5, -4, 13, 13 });

    // Define initial guesses for the roots
    double guess1 = -0.1; // Close to the first root
    double guess2 = 0.7; // Close to the second root
    double tolerance = 1e-5; // Tolerance for convergence
    int maxIter = 100; // Maximum iterations

    // Get the roots using the getRoot method
    double root1 = p1.getRoot(guess1, tolerance, maxIter);
    double root2 = p1.getRoot(guess2, tolerance, maxIter);

    // Check if the roots are close to the expected values
    EXPECT_NEAR(root1, -0.0915290543036897, tolerance);
    EXPECT_NEAR(root2, 0.727055676529518, tolerance);
}

```

```

TEST_F(PolynomialTest, Test57) {
    // Polynomial coefficients for:  $-1 * x^6 - 4 * x^5 + 6 * x^4 - 8 * x^3 - 2 * x^2 + 10 * x + 6$ 
    Polynomial p1({ -1, -4, 6, -8, -2, 10, 6, 9 });

    double tolerance = 1e-6; // Set the tolerance level for root finding
    int maxIter = 100; // Set maximum iterations for Newton's method

    // Initial guesses based on expected roots
    double guess1 = -1.0; // Near the first expected root
    double guess2 = -0.2; // Near the second expected root
    double guess3 = 0.7; // Near the third expected root

    // Get roots using the getRoot function
    double root1 = p1.getRoot(guess1, tolerance, maxIter);
    double root2 = p1.getRoot(guess2, tolerance, maxIter);
    double root3 = p1.getRoot(guess3, tolerance, maxIter);

    // Check that the found roots are close to the expected values
    EXPECT_NEAR(root1, -1.03819717707786, tolerance);
    EXPECT_NEAR(root2, -0.186199186330593, tolerance);
    EXPECT_NEAR(root3, 0.749600188448983, tolerance);
}

TEST_F(PolynomialTest, Test58) {
    // Polynomial coefficients for:  $-1 * x^6 - 4 * x^5 + 6 * x^4 - 8 * x^3 - 2 * x^2 + 10 * x + 6$ 
    Polynomial p1({ -1, -4, 6, -8, -2, 10, 6, 9 });

    double tolerance = 1e-6; // Set the tolerance level for root finding
    int maxIter = 100; // Set maximum iterations for Newton's method

    // Initial guesses based on expected roots
    double guess1 = -1.0; // Near the first expected root
    double guess2 = -0.2; // Near the second expected root
    double guess3 = 0.7; // Near the third expected root

    // Get roots using the getRoot function
    double root1 = p1.getRoot(guess1, tolerance, maxIter);
    double root2 = p1.getRoot(guess2, tolerance, maxIter);
    double root3 = p1.getRoot(guess3, tolerance, maxIter);

```

```

    // Check that the found roots are close to the expected values
    EXPECT_NEAR(root1, -1.03819717707786, tolerance);
    EXPECT_NEAR(root2, -0.186199186330593, tolerance);
    EXPECT_NEAR(root3, 0.749600188448983, tolerance);
}

TEST_F(PolynomialTest, Test59) {
    // Polynomial coefficients for:  $10 * x^8 - 1 * x^7 + 15 * x^6 - 11 * x^5 + 14 * x^4 - 3 * x^3 + 8 * x^2 + 4 * x - 6$ 
    Polynomial p1({ 10, -1, 15, -11, 14, -3, 8, 4, -6 });

    double tolerance = 1e-6; // Set the tolerance level for root finding
    int maxIter = 100; // Set maximum iterations for Newton's method

    // Initial guesses based on expected roots
    double guess1 = -1.5; // Near the first expected root
    double guess2 = 1.7; // Near the second expected root

    // Get roots using the getRoot function
    double root1 = p1.getRoot(guess1, tolerance, maxIter);
    double root2 = p1.getRoot(guess2, tolerance, maxIter);

    // Check that the found roots are close to the expected values
    EXPECT_NEAR(root1, -1.58327556220727, tolerance);
    EXPECT_NEAR(root2, 1.71953761729611, tolerance);
}

TEST_F(PolynomialTest, Test60) {
    // Polynomial coefficients for:  $-4 * x^2 + 8 * x + 11$ 
    Polynomial p1({ -4, 8, 11 });

    double tolerance = 1e-6; // Set the tolerance level for root finding
    int maxIter = 100; // Set maximum iterations for Newton's method

    // Initial guesses based on expected roots
    double guess1 = -1.0; // Near the first expected root
    double guess2 = 0.5; // Near the second expected root

    // Get roots using the getRoot function
    double root1 = p1.getRoot(guess1, tolerance, maxIter);
    double root2 = p1.getRoot(guess2, tolerance, maxIter);

```

```

    // Check that the found roots are close to the expected values
    EXPECT_NEAR(root1, -1.06781515385589, tolerance);
    EXPECT_NEAR(root2, 0.340542426583167, tolerance);
}

TEST_F(PolynomialTest, Test61) {
    // Polynomial coefficients for:  $x^4 + 2x^3 + 5x^2 + 6x + 10$ 
    Polynomial p1({ 10, 6, 5, 2, 1 });

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // Expect that getRoot does not return a real root (you can also check if it returns NaN)
    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(std::isnan(root)); // or check the return value against a known condition for
no real roots
}

TEST_F(PolynomialTest, Test62) {
    // Polynomial coefficients for:  $2x^6 + 3x^5 + 4x^4 + 5x^3 + 6x^2 + 7x + 8$ 
    Polynomial p1({ 8, 7, 6, 5, 4, 3, 2 });

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // Expect that getRoot does not return a real root (you can also check if it returns NaN)
    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(std::isnan(root)); // or check the return value against a known condition for
no real roots
}

TEST_F(PolynomialTest, Test63) {
    Polynomial p1({ 5, 4, 6, 4, 1 });

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(std::isnan(root)); // Expect no real roots
}

```

```

TEST_F(PolynomialTest, Test64) {
    Polynomial p1({ 8, 0, 1, 2, 0, 3 });

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(root , -0.86382264 ); // Expect no real roots
}

TEST_F(PolynomialTest, Test65) {
    Polynomial p1({ 10, 5, 2, 1 });

    double initialGuess = 0.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(root ,-0.5); // Expect no real roots
}

/*converging properly to the expected roots. Here are a few considerations to determine
whether the issue lies within the function or the parameters being used:
*/

TEST_F(PolynomialTest, Test66) {
    // Polynomial:  $x^4 + \sqrt{2}x^3 + \sqrt{3}x^2 + \sqrt{5}x + \sqrt{6}$ 
    // This polynomial does not have any real roots.
    Polynomial poly1({ sqrt(6), sqrt(5), sqrt(3), sqrt(2), 1 }); // Represents  $x^4 + \sqrt{2}x^3 + \sqrt{3}x^2 + \sqrt{5}x + \sqrt{6}$ 

    // Check that there are no real roots
    double initialGuess = 0.0; // Starting guess
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // Since the polynomial is expected to have no real roots,
    // we can test that calling getRoot returns a value that is not a real root
    double root = poly1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(isnan(root));

    // Additional checks for properties of the polynomial
    EXPECT_EQ(poly1.degree(), 4); // Check degree

```

```

    EXPECT_DOUBLE_EQ(poly1.evaluate(1), sqrt(6) + sqrt(5) + sqrt(3) + sqrt(2) + 1); //
    Evaluate at x=1

    // Test derivative
    Polynomial derivative = poly1.derivative();
    EXPECT_EQ(derivative.degree(), 3); // Check that derivative is degree 3
}

TEST_F(PolynomialTest, Test67) {
    // Polynomial: x^12 + sqrt(2)x^11 + sqrt(3)x^10 + sqrt(4)x^9 + sqrt(5)x^8 +
    // sqrt(6)x^7 + sqrt(7)x^6 + sqrt(8)x^5 + sqrt(9)x^4 + sqrt(10)x^3 +
    // sqrt(11)x^2 + sqrt(12)x + 1
    // This polynomial does not have any real roots.
    Polynomial poly1({ 1, sqrt(12), sqrt(11), sqrt(10), sqrt(9),
                      sqrt(8), sqrt(7), sqrt(6), sqrt(5),
                      sqrt(4), sqrt(3), sqrt(2), 1 }); // Represents x^12 + sqrt(2)x^11 +
... + 1
    double initialGuess = 1.0; // Starting guess
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // Check that there are no real roots
    // Since the polynomial is expected to have no real roots,
    double root = poly1.getRoot(initialGuess, tolerance, maxIter);
    // we can test that calling getRoot returns a value that is not equal to the known real
    roots
    EXPECT_NEAR(root, -2.50562387, tolerance); // Check if it is not approximately equal to -
2.50562387
    EXPECT_NEAR(root, -0.94087713, tolerance); // Check if it is not approximately equal to -
0.94087713

    // Additional checks for properties of the polynomial
    EXPECT_EQ(poly1.degree(), 12); // Check degree
    EXPECT_DOUBLE_EQ(poly1.evaluate(1), 1 + sqrt(12) + sqrt(11) + sqrt(10) +
                      sqrt(9) + sqrt(8) + sqrt(7) +
                      sqrt(6) + sqrt(5) + sqrt(4) +
                      sqrt(3) + sqrt(2) + 1); // Evaluate at x=1

    // Test derivative
    Polynomial derivative = poly1.derivative();
    EXPECT_EQ(derivative.degree(), 11); // Check that derivative is degree 11
}

TEST_F(PolynomialTest, Test68) {

```

```

// Polynomial:  $P(x) = -71x^{20} + 67x^{19} + 61x^{18} + 59x^{17} - 53x^{16} + 47x^{15} - 43x^{14} + 41x^{13} - 37x^{12} + 31x^{11} - 29x^{10} + 23x^9 - 19x^8 + 17x^7 - 13x^6 + 11x^5 - 7x^4 + 5x^3 - 3x^2 + 2x + 1$ 
Polynomial p1({ -71, 67, 61, 59, -53, 47, -43, 41, -37, 31, -29, 23, -19, 17, -13, 11, -7, 5, -3, 2, 1 });

// Define initial guesses for the roots
double guess1 = 1.5; // Adjusted to be closer to the expected real root
double guess2 = -0.2; // Adjusted to be closer to the second expected real root
double tolerance = 1e-7; // Tighter tolerance for convergence
int maxIter = 100; // Maximum iterations

// Get the roots using the getRoot method
double root1 = p1.getRoot(guess1, tolerance, maxIter);
double root2 = p1.getRoot(guess2, tolerance, maxIter);

// Check if the roots are close to the expected values
EXPECT_NEAR(root1, 1.65737085, tolerance);
EXPECT_NEAR(root2, -0.28384746, tolerance);
}

TEST_F(PolynomialTest, Test69) {
// Polynomial:  $P(x) = x^{25} - 2x^{24} + 3x^{23} - 4x^{22} + 5x^{21} - 6x^{20} + 7x^{19} - 8x^{18} + 9x^{17} - 10x^{16} + 11x^{15} - 12x^{14} + 13x^{13} - 14x^{12} + 15x^{11} - 16x^{10} + 17x^9 - 18x^8 + 19x^7 - 20x^6 + 21x^5 - 22x^4 + 23x^3 - 24x^2 + 25x - 26$ 
Polynomial p1({ -26, 25, -24, 23, -22, 21, -20, 19, -18, 17, -16, 15, -14, 13, -12, 11, -10, 9, -8, 7, -6, 5, -4, 3, -2, 1 });

// Define initial guess for the real root
double guess1 = 0.9; // Adjusted to be closer to the expected real root
double tolerance = 1e-7; // Tighter tolerance for convergence
int maxIter = 100; // Maximum iterations

// Get the root using the getRoot method
double root1 = p1.getRoot(guess1, tolerance, maxIter);

// Check if the root is close to the expected value
EXPECT_NEAR(root1, 0.86072381, tolerance);
}

TEST_F(PolynomialTest, Test70) {
// Polynomial:  $P(x) = -31x^{30} + 30x^{29} - 29x^{28} + 28x^{27} - 27x^{26} + 26x^{25} - 25x^{24} + 24x^{23} - 23x^{22} + 22x^{21} - 21x^{20} + 20x^{19} - 19x^{18} + 18x^{17} - 17x^{16} + 16x^{15} - 15x^{14} + 14x^{13} - 13x^{12} + 12x^{11} - 11x^{10} + 10x^9 - 9x^8 + 8x^7 - 7x^6 + 6x^5 - 5x^4 + 4x^3 - 3x^2 + 2x - 1$ 
Polynomial p1({ -31, 30, -29, 28, -27, 26, -25, 24, -23, 22, -21, 20, -19, 18, -17, 16, -15, 14, -13, 12, -11, 10, -9, 8, -7, 6, -5, 4, -3, 2, -1, 0 });

```

```

    // Define initial guess for the root
    double guess1 = 0.0;    // Close to the expected real root
    double tolerance = 1e-7; // Tighter tolerance for convergence
    int maxIter = 100; // Maximum iterations

    // Get the root using the getRoot method
    double root1 = p1.getRoot(guess1, tolerance, maxIter);

    // Check if the root is close to the expected value
    EXPECT_NEAR(root1, 0.0, tolerance);
}

TEST_F(PolynomialTest, Test71) {
    Polynomial p1({ 0, -1, 2, -3, 4, -5, 6, -7, 8, -9, 10, -11, 12, -13, 14, -15, 16, -17, 18,
-19, 20, -21, 22, -23, 24, -25, 26, -27, 28, -29, 30, -31, 32, -33, 34, -35, 36, -37, 38, -39,
40, -41, 42, -43, 44, -45, 46, -47, 48, -49, 50, -51 }
    );

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(root,0); // Expect no real roots
}

TEST_F(PolynomialTest, Test72) {
    Polynomial p1({ 3.14, -1.59, 2.65, -3.58, 9.79, -2.65, 3.58, -9.79,
2.71, -8.28, 4.67, -1.41, 5.12, -7.22, 0.58, -2.71,
4.14, -1.61, 9.26, -5.35, 3.18, -2.73, 1.62, -4.56,
7.98, -0.37, 1.05, -2.19, 6.28, -7.15, 9.62, -1.44,
3.32, -8.23, 0.58, -9.21, 4.75, -5.81, 6.22, -3.83,
7.45, -8.06, 1.91, -2.71, 2.45, -3.67, 4.85, -9.02,
0.42, -6.39, 8.14, -5.67, 1.23, -4.35, 9.99, -2.56,
7.84, -3.43, 0.76, -5.45, 6.72, -1.92, 3.74, -8.12,
4.56, -9.11, 2.64, -7.23, 3.59, -1.85, 5.37, -4.29,
8.11, -6.19, 1.78, -2.54, 9.83, -0.77, 4.12, -8.41,
6.73, -3.65, 0.67, -5.83, 1.54, -2.33, 7.98, -4.12,
5.49, -3.66, 0.88, -6.74, 9.25, -4.82, 1.19, -2.76,
8.56, -7.35, 3.21, -5.90, 6.82, -0.49, 4.57, -2.99,
1.32, -8.31, 7.20, -3.87, 0.53, -5.91, 2.69, -4.74,

```


6.15, -1.23, 3.87, -9.77, 5.83, -2.44, 8.10, -6.22,
 4.56, -3.68, 0.77, -8.45, 9.01, -5.14, 2.72, -1.91,
 7.18, -0.43, 5.88, -3.29, 6.47, -2.91, 1.07, -4.90,
 0.44, -8.11, 9.29, -7.21, 2.78, -5.89, 3.58, -0.95,
 1.48, -4.13, 6.30, -3.64, 2.10, -7.53, 8.22, -1.60,
 5.49, -0.98, 3.92, -2.71, 9.80, -6.04, 1.74, -7.12,
 0.96, -4.86, 8.67, -3.50, 5.81, -1.19, 2.83, -9.28,
 4.99, -5.34, 3.73, -8.05, 7.14, -0.62, 6.77, -2.45,
 1.24, -5.86, 0.34, -4.27, 8.51, -6.85, 2.22, -3.48,
 9.99, -0.36, 5.77, -1.12, 3.10, -2.57, 6.22, -4.91,
 1.90, -7.99, 0.83, -8.40, 5.03, -9.49, 3.50, -6.78, 22, 3.14, -1.59, 2.65, -3.58, 9.79, -
 2.65, 3.58, -9.79,
 2.71, -8.28, 4.67, -1.41, 5.12, -7.22, 0.58, -2.71,
 4.14, -1.61, 9.26, -5.35, 3.18, -2.73, 1.62, -4.56,
 7.98, -0.37, 1.05, -2.19, 6.28, -7.15, 9.62, -1.44,
 3.32, -8.23, 0.58, -9.21, 4.75, -5.81, 6.22, -3.83,
 7.45, -8.06, 1.91, -2.71, 2.45, -3.67, 4.85, -9.02,
 0.42, -6.39, 8.14, -5.67, 1.23, -4.35, 9.99, -2.56,
 7.84, -3.43, 0.76, -5.45, 6.72, -1.92, 3.74, -8.12,
 4.56, -9.11, 2.64, -7.23, 3.59, -1.85, 5.37, -4.29,
 8.11, -6.19, 1.78, -2.54, 9.83, -0.77, 4.12, -8.41,
 6.73, -3.65, 0.67, -5.83, 1.54, -2.33, 7.98, -4.12,
 5.49, -3.66, 0.88, -6.74, 9.25, -4.82, 1.19, -2.76,
 8.56, -7.35, 3.21, -5.90, 6.82, -0.49, 4.57, -2.99,
 1.32, -8.31, 7.20, -3.87, 0.53, -5.91, 2.69, -4.74,
 6.15, -1.23, 3.87, -9.77, 5.83, -2.44, 8.10, -6.22,
 4.56, -3.68, 0.77, -8.45, 9.01, -5.14, 2.72, -1.91,
 7.18, -0.43, 5.88, -3.29, 6.47, -2.91, 1.07, -4.90,
 0.44, -8.11, 9.29, -7.21, 2.78, -5.89, 3.58, -0.95,
 1.48, -4.13, 6.30, -3.64, 2.10, -7.53, 8.22, -1.60,
 5.49, -0.98, 3.92, -2.71, 9.80, -6.04, 1.74, -7.12,
 0.96, -4.86, 8.67, -3.50, 5.81, -1.19, 2.83, -9.28,
 4.99, -5.34, 3.73, -8.05, 7.14, -0.62, 6.77, -2.45,
 1.24, -5.86, 0.34, -4.27, 8.51, -6.85, 2.22, -3.48,
 9.99, -0.36, 5.77, -1.12, 3.10, -2.57, 6.22, -4.91,
 1.90, -7.99, 0.83, -8.40, 5.03, -9.49, 3.50, -6.78, 22, 3.14, -1.59, 2.65, -3.58, 9.79, -
 2.65, 3.58, -9.79,
 2.71, -8.28, 4.67, -1.41, 5.12, -7.22, 0.58, -2.71,
 4.14, -1.61, 9.26, -5.35, 3.18, -2.73, 1.62, -4.56,
 7.98, -0.37, 1.05, -2.19, 6.28, -7.15, 9.62, -1.44,
 3.32, -8.23, 0.58, -9.21, 4.75, -5.81, 6.22, -3.83,

7.45, -8.06, 1.91, -2.71, 2.45, -3.67, 4.85, -9.02,
 0.42, -6.39, 8.14, -5.67, 1.23, -4.35, 9.99, -2.56,
 7.84, -3.43, 0.76, -5.45, 6.72, -1.92, 3.74, -8.12,
 4.56, -9.11, 2.64, -7.23, 3.59, -1.85, 5.37, -4.29,
 8.11, -6.19, 1.78, -2.54, 9.83, -0.77, 4.12, -8.41,
 6.73, -3.65, 0.67, -5.83, 1.54, -2.33, 7.98, -4.12,
 5.49, -3.66, 0.88, -6.74, 9.25, -4.82, 1.19, -2.76,
 8.56, -7.35, 3.21, -5.90, 6.82, -0.49, 4.57, -2.99,
 1.32, -8.31, 7.20, -3.87, 0.53, -5.91, 2.69, -4.74,
 6.15, -1.23, 3.87, -9.77, 5.83, -2.44, 8.10, -6.22,
 4.56, -3.68, 0.77, -8.45, 9.01, -5.14, 2.72, -1.91,
 7.18, -0.43, 5.88, -3.29, 6.47, -2.91, 1.07, -4.90,
 0.44, -8.11, 9.29, -7.21, 2.78, -5.89, 3.58, -0.95,
 1.48, -4.13, 6.30, -3.64, 2.10, -7.53, 8.22, -1.60,
 5.49, -0.98, 3.92, -2.71, 9.80, -6.04, 1.74, -7.12,
 0.96, -4.86, 8.67, -3.50, 5.81, -1.19, 2.83, -9.28,
 4.99, -5.34, 3.73, -8.05, 7.14, -0.62, 6.77, -2.45,
 1.24, -5.86, 0.34, -4.27, 8.51, -6.85, 2.22, -3.48,
 9.99, -0.36, 5.77, -1.12, 3.10, -2.57, 6.22, -4.91,
 1.90, -7.99, 0.83, -8.40, 5.03, -9.49, 3.50, -6.78, 22, 3.14, -1.59, 2.65, -3.58, 9.79, -
 2.65, 3.58, -9.79,
 2.71, -8.28, 4.67, -1.41, 5.12, -7.22, 0.58, -2.71,
 4.14, -1.61, 9.26, -5.35, 3.18, -2.73, 1.62, -4.56,
 7.98, -0.37, 1.05, -2.19, 6.28, -7.15, 9.62, -1.44,
 3.32, -8.23, 0.58, -9.21, 4.75, -5.81, 6.22, -3.83,
 7.45, -8.06, 1.91, -2.71, 2.45, -3.67, 4.85, -9.02,
 0.42, -6.39, 8.14, -5.67, 1.23, -4.35, 9.99, -2.56,
 7.84, -3.43, 0.76, -5.45, 6.72, -1.92, 3.74, -8.12,
 4.56, -9.11, 2.64, -7.23, 3.59, -1.85, 5.37, -4.29,
 8.11, -6.19, 1.78, -2.54, 9.83, -0.77, 4.12, -8.41,
 6.73, -3.65, 0.67, -5.83, 1.54, -2.33, 7.98, -4.12,
 5.49, -3.66, 0.88, -6.74, 9.25, -4.82, 1.19, -2.76,
 8.56, -7.35, 3.21, -5.90, 6.82, -0.49, 4.57, -2.99,
 1.32, -8.31, 7.20, -3.87, 0.53, -5.91, 2.69, -4.74,
 6.15, -1.23, 3.87, -9.77, 5.83, -2.44, 8.10, -6.22,
 4.56, -3.68, 0.77, -8.45, 9.01, -5.14, 2.72, -1.91,
 7.18, -0.43, 5.88, -3.29, 6.47, -2.91, 1.07, -4.90,
 0.44, -8.11, 9.29, -7.21, 2.78, -5.89, 3.58, -0.95,
 1.48, -4.13, 6.30, -3.64, 2.10, -7.53, 8.22, -1.60,
 5.49, -0.98, 3.92, -2.71, 9.80, -6.04, 1.74, -7.12,
 0.96, -4.86, 8.67, -3.50, 5.81, -1.19, 2.83, -9.28,

```

4.99, -5.34, 3.73, -8.05, 7.14, -0.62, 6.77, -2.45,
1.24, -5.86, 0.34, -4.27, 8.51, -6.85, 2.22, -3.48,
9.99, -0.36, 5.77, -1.12, 3.10, -2.57, 6.22, -4.91,
1.90, -7.99, 0.83, -8.40, 5.03, -9.49, 3.50, -6.78, 22, 3.14, -1.59, 2.65, -3.58, 9.79, -
2.65, 3.58, -9.79,
2.71, -8.28, 4.67, -1.41, 5.12, -7.22, 0.58, -2.71,
4.14, -1.61, 9.26, -5.35, 3.18, -2.73, 1.62, -4.56,
7.98, -0.37, 1.05, -2.19, 6.28, -7.15, 9.62, -1.44,
3.32, -8.23, 0.58, -9.21, 4.75, -5.81, 6.22, -3.83,
7.45, -8.06, 1.91, -2.71, 2.45, -3.67, 4.85, -9.02,
0.42, -6.39, 8.14, -5.67, 1.23, -4.35, 9.99, -2.56,
7.84, -3.43, 0.76, -5.45, 6.72, -1.92, 3.74, -8.12,
4.56, -9.11, 2.64, -7.23, 3.59, -1.85, 5.37, -4.29,
8.11, -6.19, 1.78, -2.54, 9.83, -0.77, 4.12, -8.41,
6.73, -3.65, 0.67, -5.83, 1.54, -2.33, 7.98, -4.12,
5.49, -3.66, 0.88, -6.74, 9.25, -4.82, 1.19, -2.76,
8.56, -7.35, 3.21, -5.90, 6.82, -0.49, 4.57, -2.99,
1.32, -8.31, 7.20, -3.87, 0.53, -5.91, 2.69, -4.74,
6.15, -1.23, 3.87, -9.77, 5.83, -2.44, 8.10, -6.22,
4.56, -3.68, 0.77, -8.45, 9.01, -5.14, 2.72, -1.91,
7.18, -0.43, 5.88, -3.29, 6.47, -2.91, 1.07, -4.90,
0.44, -8.11, 9.29, -7.21, 2.78, -5.89, 3.58, -0.95,
1.48, -4.13, 6.30, -3.64, 2.10, -7.53, 8.22, -1.60,
5.49, -0.98, 3.92, -2.71, 9.80, -6.04, 1.74, -7.12,
0.96, -4.86, 8.67, -3.50, 5.81, -1.19, 2.83, -9.28,
4.99, -5.34, 3.73, -8.05, 7.14, -0.62, 6.77, -2.45,
1.24, -5.86, 0.34, -4.27, 8.51, -6.85, 2.22, -3.48,
9.99, -0.36, 5.77, -1.12, 3.10, -2.57, 6.22, -4.91,
1.90, -7.99, 0.83, -8.40, 5.03 }
);

EXPECT_EQ(p1.degree(), 1000);
}
TEST_F(PolynomialTest, Test73) {
    Polynomial p1({0,1,sqrt(2) , sqrt(3) /2 , sqrt(5)/4 , 4 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);

```


260,1.0	/	261,1.0	/	262,1.0	/	263,1.0	/	264,1.0	/	265,1.0	/	266,1.0	/	267,1.0	/	268,1.0	/
269,1.0	/	270,1.0	/	271,1.0	/	272,1.0	/	273,1.0	/	274,1.0	/	275,1.0	/	276,1.0	/	277,1.0	/
278,1.0	/	279,1.0	/	280,1.0	/	281,1.0	/	282,1.0	/	283,1.0	/	284,1.0	/	285,1.0	/	286,1.0	/
287,1.0	/	288,1.0	/	289,1.0	/	290,1.0	/	291,1.0	/	292,1.0	/	293,1.0	/	294,1.0	/	295,1.0	/
296,1.0	/	297,1.0	/	298,1.0	/	299,1.0	/	300,1.0	/	301,1.0	/	302,1.0	/	303,1.0	/	304,1.0	/
305,1.0	/	306,1.0	/	307,1.0	/	308,1.0	/	309,1.0	/	310,1.0	/	311,1.0	/	312,1.0	/	313,1.0	/
314,1.0	/	315,1.0	/	316,1.0	/	317,1.0	/	318,1.0	/	319,1.0	/	320,1.0	/	321,1.0	/	322,1.0	/
323,1.0	/	324,1.0	/	325,1.0	/	326,1.0	/	327,1.0	/	328,1.0	/	329,1.0	/	330,1.0	/	331,1.0	/
332,1.0	/	333,1.0	/	334,1.0	/	335,1.0	/	336,1.0	/	337,1.0	/	338,1.0	/	339,1.0	/	340,1.0	/
341,1.0	/	342,1.0	/	343,1.0	/	344,1.0	/	345,1.0	/	346,1.0	/	347,1.0	/	348,1.0	/	349,1.0	/
350,1.0	/	351,1.0	/	352,1.0	/	353,1.0	/	354,1.0	/	355,1.0	/	356,1.0	/	357,1.0	/	358,1.0	/
359,1.0	/	360,1.0	/	361,1.0	/	362,1.0	/	363,1.0	/	364,1.0	/	365,1.0	/	366,1.0	/	367,1.0	/
368,1.0	/	369,1.0	/	370,1.0	/	371,1.0	/	372,1.0	/	373,1.0	/	374,1.0	/	375,1.0	/	376,1.0	/
377,1.0	/	378,1.0	/	379,1.0	/	380,1.0	/	381,1.0	/	382,1.0	/	383,1.0	/	384,1.0	/	385,1.0	/
386,1.0	/	387,1.0	/	388,1.0	/	389,1.0	/	390,1.0	/	391,1.0	/	392,1.0	/	393,1.0	/	394,1.0	/
395,1.0	/	396,1.0	/	397,1.0	/	398,1.0	/	399,1.0	/	400,1.0	/	401,1.0	/	402,1.0	/	403,1.0	/
404,1.0	/	405,1.0	/	406,1.0	/	407,1.0	/	408,1.0	/	409,1.0	/	410,1.0	/	411,1.0	/	412,1.0	/
413,1.0	/	414,1.0	/	415,1.0	/	416,1.0	/	417,1.0	/	418,1.0	/	419,1.0	/	420,1.0	/	421,1.0	/
422,1.0	/	423,1.0	/	424,1.0	/	425,1.0	/	426,1.0	/	427,1.0	/	428,1.0	/	429,1.0	/	430,1.0	/
431,1.0	/	432,1.0	/	433,1.0	/	434,1.0	/	435,1.0	/	436,1.0	/	437,1.0	/	438,1.0	/	439,1.0	/
440,1.0	/	441,1.0	/	442,1.0	/	443,1.0	/	444,1.0	/	445,1.0	/	446,1.0	/	447,1.0	/	448,1.0	/
449,1.0	/	450,1.0	/	451,1.0	/	452,1.0	/	453,1.0	/	454,1.0	/	455,1.0	/	456,1.0	/	457,1.0	/
458,1.0	/	459,1.0	/	460,1.0	/	461,1.0	/	462,1.0	/	463,1.0	/	464,1.0	/	465,1.0	/	466,1.0	/
467,1.0	/	468,1.0	/	469,1.0	/	470,1.0	/	471,1.0	/	472,1.0	/	473,1.0	/	474,1.0	/	475,1.0	/
476,1.0	/	477,1.0	/	478,1.0	/	479,1.0	/	480,1.0	/	481,1.0	/	482,1.0	/	483,1.0	/	484,1.0	/
485,1.0	/	486,1.0	/	487,1.0	/	488,1.0	/	489,1.0	/	490,1.0	/	491,1.0	/	492,1.0	/	493,1.0	/
494,1.0</																	

```
980,1.0 / 981,1.0 / 982,1.0 / 983,1.0 / 984,1.0 / 985,1.0 / 986,1.0 / 987,1.0 / 988,1.0 /
989,1.0 / 990,1.0 / 991,1.0 / 992,1.0 / 993,1.0 / 994,1.0 / 995,1.0 / 996,1.0 / 997,1.0 /
998,1.0 / 999,1.0 / 1000,1.0 / 1001 }));
```

```
}
```

```
TEST_F(PolynomialTest, Test76) {
    Polynomial p1({ 0, 1 , 4, sqrt(2) });

    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // First root: starting point close to -0.39
    double initialGuess1 = 1.0;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, -0.39196193, tolerance); // Check for root near -0.39

    // Second root: starting point close to -3.60
    double initialGuess2 = -4.0;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, -3.60803807, tolerance); // Check for root near -3.60
}
```

```
TEST_F(PolynomialTest, Test77) {
    Polynomial p1({ 0, 1 , sqrt(2) });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, -1.41421356, tolerance);
}
```

```
TEST_F(PolynomialTest, Test78) {
    Polynomial p1({ 0, 0 ,0, 1,sqrt(2) });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
}
```

```

    EXPECT_NEAR(root, -1.41421356, tolerance);
}

TEST_F(PolynomialTest, Test79) {
    Polynomial p1({ sqrt(2) , sqrt(2) , sqrt(2) , sqrt(2) });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, -1.00000000e+00, tolerance);
}

TEST_F(PolynomialTest, Test80) {
    Polynomial p1({ 1,sqrt(2) , 3 , sqrt(4) , (5) });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_TRUE(isnan(root));
}

TEST_F(PolynomialTest, Test81) {
    Polynomial p1({ 5, -7, 13, 10, 2, 15, -8, -3, -14, -4, 10 });

    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // First root: Use an initial guess near -0.79
    double initialGuess1 = 1.0;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, -0.79058165, tolerance);

    // Second root: Use a different initial guess near -1.04
    double initialGuess2 = -2.0;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, -1.04797069, tolerance);
}

```

```

}

TEST_F(PolynomialTest, Test82) {
    Polynomial p1({ -7 ,13, -8, 8 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, 1.59038197, tolerance);
}

TEST_F(PolynomialTest, Test83) {
    Polynomial p1({ -13, -8, 15, 6, 7, -11, -4, -6, -14, 1 });

    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // First root: Use an initial guess near -0.86
    double initialGuess1 = 1.0;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, -0.85992705, tolerance); // Check first root

    // Second root: Use a different initial guess near 0.07
    double initialGuess2 = 0.5;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, 0.06926055, tolerance); // Check second root
}

TEST_F(PolynomialTest, Test84) {
    Polynomial p1({ 1, 6, -6, -9, -7, -6 });

    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    // First root: Use an initial guess near -6.71
    double initialGuess1 = -7.0;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);

```



```

    EXPECT_NEAR(root1, -6.71416292, tolerance); // Check first root

    // Second root: Use a different initial guess near 1.81
    double initialGuess2 = 2.0;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, 1.80907437, tolerance); // Check second root

    // Third root: Use a different initial guess near -0.86
    double initialGuess3 = 0.0;
    double root3 = p1.getRoot(initialGuess3, tolerance, maxIter);
    EXPECT_NEAR(root3, -0.86048584, tolerance); // Check third root
}

TEST_F(PolynomialTest, Test85) {
    Polynomial p1({ -5, -4, -11, -5, 13, 6, 0, -8, 8 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // First root: Use an initial guess near -1.25
    double initialGuess1 = 1.0;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, -1.25405831, tolerance); // Check first root

    // Second root: Use a different initial guess near 0.92
    double initialGuess2 = 0.5;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, 0.91737553, tolerance); // Check second root
}

TEST_F(PolynomialTest, Test86) {
    Polynomial p1({ 6, -12, 9, 8, -7, 11, 15, -13, 7, -14, 11, 2 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100; // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, -0.15023882, tolerance);
}

```

```

}
TEST_F(PolynomialTest, Test87) {
    Polynomial p1({ 6, -12, 9, 8, -7, 11, 15, -13, 7, -14, 11, 2 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = 1.0;
    double expectedRoot1 = 0.27271081;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root
    double initialGuess2 = -1.0;
    double expectedRoot2 = -0.77265566;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);
}

TEST_F(PolynomialTest, Test88) {
    Polynomial p1({ 5, 8, 2, 0, 10, 10, 12, 10, 11, 12, -11 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = 1.0;
    double expectedRoot1 = -1.18719435;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root
    double initialGuess2 = 0.5; // You might need to adjust this guess based on the expected
root
    double expectedRoot2 = 0.49405972;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);
}

TEST_F(PolynomialTest, Test89) {

```

```

Polynomial p1({ -12 , 4 , 5 , 8 , -14 , 13 , -12 , 12 , 10 , 8 });

double initialGuess = 1.0; // Starting point for Newton's method
double tolerance = 1e-6;    // Tolerance for convergence
int maxIter = 100;          // Maximum iterations

double root = p1.getRoot(initialGuess, tolerance, maxIter);
EXPECT_NEAR(root, 1.22634169, tolerance);
}

TEST_F(PolynomialTest, Test90) {
    Polynomial p1({ 1, 15, 0, 6, 15, -13, 5, -2, 12, -2, 4 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = -15.0;
    double expectedRoot1 = -15.02190185;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root
    double initialGuess2 = -1.0; // Adjust this guess based on where you expect the second
root
    double expectedRoot2 = -1.26757023;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);
}

TEST_F(PolynomialTest, Test91) {
    Polynomial p1({ -1, -8, -3, -9, 0, 0, 8, 0, -6, 15 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = 1.0;
    double expectedRoot1 = -7.76317497;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);
}

```

```

    // Test for the second root
    double initialGuess2 = -1.0; // Adjust this guess based on where you expect the second
root
    double expectedRoot2 = -1.0;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);

    // Test for the third root
    double initialGuess3 = 1.0; // Adjust this guess based on where you expect the third root
    double expectedRoot3 = 0.96550363;
    double root3 = p1.getRoot(initialGuess3, tolerance, maxIter);
    EXPECT_NEAR(root3, expectedRoot3, tolerance);
}

TEST_F(PolynomialTest, Test92) {
    Polynomial p1({ 8, 2, -6, 1, -10, 9, 3, 13, -5, -2, 6 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;      // Maximum iterations

    // Test for the first root
    double initialGuess1 = 1.0;
    double expectedRoot1 = -1.49960077;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root
    double initialGuess2 = 0.0; // Adjust this guess based on where you expect the second root
    double expectedRoot2 = -0.65517255;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);
}

TEST_F(PolynomialTest, Test93) {
    Polynomial p1({ -6 , 11 , -2 , 7 , 12 , 2 , 3 , -12 , -11 , 9, 4, 6 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);

```

```

    EXPECT_NEAR(root, 2.1388204, tolerance);
}
TEST_F(PolynomialTest, Test94) {
    Polynomial p1({ -3 , -4 , -5 , -11 , -4 , -7 , -3 , 5 , -8 , -6 , 6 , -2 });

    double initialGuess = -1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, -1.56738018, tolerance);
}
TEST_F(PolynomialTest, Test95) {
    Polynomial p1({ 11 , 8 , -5 , -10 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, 0.90002768, tolerance);
}
TEST_F(PolynomialTest, Test96) {
    Polynomial p1({ 9 , -6 , 13 , -2 , 1 , 9 , -8 , 13 , 14 , 4 , 7 , 1 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, -0.15000588, tolerance);
}
TEST_F(PolynomialTest, Test97) {
    Polynomial p1({ -7 , 15 , -7 , -3 , -5 , 13 , 10 , -14 , -14 , -5 });

    double initialGuess = 1.0; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);

```

```

    EXPECT_NEAR(root, -0.70764823, tolerance);
}

TEST_F(PolynomialTest, Test98) {
    Polynomial p1({ 5, -14, -14, -6, -6, -14, 12, -4, -8, 12 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = 1.0;
    double expectedRoot1 = 3.68442743;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root
    double initialGuess2 = -1.0; // Adjust this guess based on where you expect the second
root
    double expectedRoot2 = -0.86557888;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);

    // Test for the third root
    double initialGuess3 = 0.5; // Adjust this guess based on where you expect the third root
    double expectedRoot3 = 0.73331757;
    double root3 = p1.getRoot(initialGuess3, tolerance, maxIter);
    EXPECT_NEAR(root3, expectedRoot3, tolerance);
}

TEST_F(PolynomialTest, Test99) {
    Polynomial p1({ 3, 1, -8 });

    double tolerance = 1e-6; // Tolerance for convergence
    int maxIter = 100;       // Maximum iterations

    // Test for the first root
    double initialGuess1 = -1.0;
    double expectedRoot1 = -1.80814297;
    double root1 = p1.getRoot(initialGuess1, tolerance, maxIter);
    EXPECT_NEAR(root1, expectedRoot1, tolerance);

    // Test for the second root

```

```

    double initialGuess2 = 2.0; // Adjust this guess based on where you expect the second root
    double expectedRoot2 = 1.47480963;
    double root2 = p1.getRoot(initialGuess2, tolerance, maxIter);
    EXPECT_NEAR(root2, expectedRoot2, tolerance);
}

TEST_F(PolynomialTest, Test100) {
    Polynomial p1({ -3 , 2 , -11 , 5 , -13 , -14 , -11 , 5 });

    double initialGuess = 0.5; // Starting point for Newton's method
    double tolerance = 1e-6;    // Tolerance for convergence
    int maxIter = 100;          // Maximum iterations

    double root = p1.getRoot(initialGuess, tolerance, maxIter);
    EXPECT_NEAR(root, 0.30451905, tolerance);
}

```

Task 3

Technology Training opportunities

1. Information Technology Institute (ITI) - Egypt

Who offers it and where it is located:

The Information Technology Institute (ITI) is a prestigious training institute in Egypt with campuses in Cairo, Alexandria, Assiut, and Mansoura, offering cutting-edge technology training for fresh graduates.

What it offers:

ITI provides specialized software development and IT programs, including a 1-month summer camp and a 9-month professional diploma. These programs focus on areas like Artificial Intelligence, Data Science, and more.

When it opens and how long it is, and whether it's online or offline:

The 1-month program runs during the summer, while the 9-month diploma starts every fall. Both are primarily offline, with some programs offering blended learning options (a mix of online and offline sessions).

Conditions for application and acceptance, and fees (if any):

Applicants typically need a university degree in fields like engineering, computer science, or a related area. The programs are free for Egyptians, as they are funded by the Egyptian government.

Different learning tracks:

Tracks include Software Engineering, Artificial Intelligence, Data Science, UI/UX Design, and more.

Preferred Track & Why:

Selected by Abdullah Mohammed [Software Engineering].

Reason: I am passionate about building large-scale systems and solving complex challenges using programming languages. The ITI program offers hands-on experience with leading-edge technologies, which aligns with my career goals.

2. Digital Egypt Pioneers Initiative (Mubadara Ruwwad Misr al-Raqmiya) - Egypt

Who offers it and where it is located:

The Digital Egypt Pioneers Initiative is a national project launched by the Ministry of Communications and Information Technology (MCIT). It is designed to prepare Egypt's future tech leaders and operates nationwide, with opportunities in various governorates and through online platforms.

What it offers:

The initiative provides free training programs in advanced technology fields, such as Artificial Intelligence, Cloud Computing, and Cybersecurity. It also supports entrepreneurship and offers scholarships for master's degrees in technology-related fields.

When it opens and how long it is, and whether it's online or offline:

Programs are offered at different times throughout the year. The duration varies, with some courses lasting just a few weeks, while others, like scholarship programs, may last up to a year. Training is available both online and offline, depending on the specific program.

Conditions for application and acceptance, and fees (if any):

Most programs are free. Applicants typically need a background in IT or engineering, and some programs require passing an entrance exam or interview. The initiative is mainly aimed at fresh graduates and young professionals.

Different learning tracks:

Learning tracks include Software Development, Artificial Intelligence, Data Science, Cybersecurity, Digital Arts, Business Analysis, Embedded Systems, and Enterprise Resource Planning (ERP). The initiative also provides entrepreneurship training.

Preferred Track & Why:

Selected by Loai Hataba [Data Science].

Reason: I have a deep interest in Data Science as it plays a crucial role in critical business decisions. The Digital Egypt Pioneers training will help me gain the necessary foundation for a career in Data science.

3. FreeCodeCamp (Outside Egypt)

Who offers it and where it is located:

freeCodeCamp is a well-known international online learning platform offering free coding lessons to learners worldwide.

What it offers:

The platform provides a wide range of free online courses, covering areas such as Web Development, Machine Learning, Data Visualization, and more.

When it opens and how long it is, and whether it's online or offline:

Courses are self-paced, allowing participants to start at any time and complete them at their own speed. All courses are delivered online.

Conditions for application and acceptance, and fees (if any):

There are no application requirements, and the platform is entirely free, making it accessible to anyone, anywhere in the world.

Different learning tracks:

Tracks include Web Development, JavaScript Algorithms, Machine Learning with Python, Data Science, and others.

Preferred Track & Why:

Selected by Hossam Abdelaziz [Web Development].

Reason: I want to focus on developing and maintaining websites and web applications. freeCodeCamp provides a comprehensive curriculum with hands-on projects that will help me develop practical skills for a career in web development.