**Introduction**

Artificial intelligence (AI) has become an integral part of modern society, influencing nearly every sector and aspect of our lives. From revolutionizing healthcare and automating industries to enhancing customer experiences and driving scientific advancements, AI offers immense potential to solve complex problems efficiently. Its ability to process vast amounts of data, identify patterns, and make intelligent decisions is transforming the way we live and work.

One area where AI is making a particularly significant impact is programming. Traditionally, coding required manual effort to design, develop, and debug applications, but AI tools have introduced new levels of automation and intelligence into the programming process. AI-powered programming assistants, such as prompt-based models, are changing how developers approach their work.

Prompt AI, a recent innovation in the field of artificial intelligence, allows programmers to interact with intelligent systems by providing natural language prompts to generate code, solve problems, and enhance productivity. This approach makes coding more accessible, reducing the need for deep expertise in complex algorithms or syntax. AI tools like GitHub Copilot, ChatGPT, and others assist developers in writing clean code, debugging, and even explaining code structures—all based on prompts provided by the user.

By leveraging prompt AI, developers can streamline workflows, automate mundane coding tasks, and explore new levels of creativity. Whether it’s generating boilerplate code, suggesting optimizations, or automating testing, prompt-based AI brings a higher level of efficiency and support, allowing programmers to focus on problem-solving and innovation rather than manual coding efforts.

In this document we used Chat GPT and Compilot for generating functions for provided header and we will talk about this experience in details.

**Chat GPT**

ChatGPT is an advanced AI language model developed by OpenAI, designed to understand and generate human-like text in a wide range of contexts. It can engage in meaningful conversations, providing detailed responses and helpful insights based on the input it receives. Whether it's answering questions, explaining complex topics, assisting with coding, or generating creative content, ChatGPT offers a versatile platform for users across various fields. Its ability to respond to prompts makes it especially useful for tasks like brainstorming ideas, solving technical problems, or simplifying complex concepts. With its intuitive interface, ChatGPT allows users to interact naturally, using plain language, which makes it accessible even to those without deep technical knowledge. This blend of intelligence and ease of use makes ChatGPT a powerful tool for enhancing productivity, creativity, and learning in everyday tasks and professional settings alike.

In this assignment we used Chat GPT for Prompt AI, and it helps very well. At first, we put the header file, which is provided in assignment in Chat GPT and we told him to generate functions for these prototypes in header and he did it but some of them works properly and some when we enter some input it output wrong answer so we asked chat GPT to edit more times and it takes much time from us.

One of these functions was (retRoot) function. at first the ai forgot to call the function in main then we asked him that you have an error and you forgot to add this function in main, after that, the function doesn’t work at all so we fall in a long dialog with Chat GPT asking him that the function doesn’t work at all, and after an hour from trying we could get a function that works well for some cases ,so, I told him that ,but he damaged the code and we couldn’t get a better version than which works for some cases so we use it now.

Another function was (Degree) function. At first, ai didn’t understand what we need as we want when we enter the coefficient of the two polynomials to tell us the degree of each one, but he misunderstood us as he asked us to enter a coefficient and he till us - if this coefficient exist - the power of the variable(x) which is multiplied with him, and if this coefficient doesn’t exist it output out of range. So, we tried with the ai and told his the problem and we get another function which works properly for most of test cases.

**Chat GPT Code**

The code is an implementation of the class Polynomial supports a variety of operations, such as addition, subtraction, multiplication, evaluation, differentiation, integration, and root-finding using Newton's method.

Here is an explanation for the code from Chat GPT:

1. **Constructors and Destructor**
   * Default Constructor: Polynomial::Polynomial() : coeffs (1, 0) {}
   * This constructor initializes the polynomial as a zero polynomial with one coefficient, 0. The coeffs vector is initialized with one element set to 0, which means the polynomial is effectively P(x) = 0.
   * Constructor with Coefficients: Polynomial::Polynomial (const vector<double>& coefficients): coeffs(coefficients) {}
   * Initializes the polynomial with the provided coefficients from the input vector.
   * Copy Constructor: Polynomial::Polynomial (const Polynomial& other) : coeffs (other. coeffs) {}
   * Creates a new polynomial as a copy of an existing polynomial.
   * Destructor: Polynomial::~Polynomial() {}
   * Destructor for the polynomial class; it cleans up resources when a polynomial object is destroyed.
2. **Assignment Operator**
   * Assignment Operator: Polynomial& Polynomial::operator=(const Polynomial& other) {}
   * Overloads the assignment operator to copy coefficients from one polynomial to another, ensuring self-assignment is handled properly.
3. **Polynomial Operations**
   * Addition of Two Polynomials: Polynomial Polynomial::operator+(const Polynomial& other) const {}
   * Adds two polynomials by summing their corresponding coefficients.
   * Subtraction of Two Polynomials: Polynomial Polynomial::operator-(const Polynomial& other) const {}
   * Subtracts one polynomial from another by subtracting their corresponding coefficients.
   * Multiplication of Two Polynomials: Polynomial Polynomial::operator\*(const Polynomial& other) const {}
   * Multiplies two polynomials using the distributive property to compute the coefficients of the resultant polynomial.
   * Equality Operator: bool Polynomial::operator==(const Polynomial& other) const {}
   * Checks for equality between two polynomials by comparing their coefficient vectors.
4. **Output Operator**
   * Output Operator: ostream& operator<<(ostream& out, const Polynomial& poly) {}
   * Outputs the polynomial in a human-readable format, including proper handling of positive and negative coefficients.
5. **Utility Functions**
   * Degree of the Polynomial: int Polynomial::degree() const {}
   * Returns the highest degree of the polynomial by iterating from the end of the coefficient vector.
   * Evaluate the Polynomial: double Polynomial::evaluate(double x) const {}
   * Evaluates the polynomial at a given value of xxx using Horner's method for efficient computation.
   * Derivative of the Polynomial: Polynomial Polynomial::derivative() const {}
   * Computes the derivative of the polynomial, returning a new polynomial representing the derivative.
6. **Polynomial Composition**
   * Composition of Polynomials: Polynomial Polynomial::compose(const Polynomial& q) const {}
   * Composes this polynomial with another polynomial q, effectively substituting q for each xxx in the polynomial.
7. **Integration**
   * Indefinite Integral: Polynomial Polynomial::integral() const {}
   * Computes the indefinite integral of the polynomial, returning a new polynomial representing the integral.
   * Definite Integral: double Polynomial::integral(double x1, double x2) const{}
   * Calculates the definite integral of the polynomial from x1x1x1 to x2x2x2 by evaluating the indefinite integral at both limits.
8. **Root Finding with Newton's Method**
   * Finding a Root: double Polynomial::getRoot(double guess, double tolerance, int maxIter) {}
   * Implements Newton’s method to find a root of the polynomial. The algorithm iteratively improves the guess by using the polynomial's evaluation and its derivative, stopping when the improvement is less than the specified tolerance or when the maximum iterations are reached.
9. **Setters and Getters**
   * Set Coefficients: void Polynomial::setCoefficients(const vector<double>& coefficients) {}
   * Sets the polynomial's coefficients directly by copying the input vector.
   * Get Coefficient: double Polynomial::getCoefficient(int degree) const{}
   * Returns the coefficient for the specified degree of the polynomial. If the degree is larger than the highest degree of the polynomial, it returns 0.

**Compilot**

GitHub Copilot is an AI-powered coding assistant developed by GitHub in collaboration with OpenAI. It acts as a co-pilot for developers, helping them write code more efficiently by suggesting lines or blocks of code based on natural language prompts or partially written code. Copilot can understand the context of what the developer is working on, offering relevant suggestions for functions, methods, and even entire algorithms. Its ability to provide real-time code completions and recommendations makes it a valuable tool for speeding up development, reducing repetitive tasks, and improving code quality. By learning from a vast range of open-source codebases, Copilot can assist with a wide variety of programming languages and frameworks. This seamless integration into the coding environment helps developers focus more on solving problems and building features, rather than worrying about syntax or boilerplate code. With Copilot, programming becomes more intuitive, collaborative, and productive, allowing developers of all skill levels to work more efficiently.

In this assignment we used Compilot for Prompt AI, and it helps very well. At first, we put the header file, which is provided in assignment in Compilot and we told him to generate functions for these prototypes in header and he did it but some of them works properly and some when we enter some input it output wrong answer so we asked Compilot edit more times and it takes much time from us.

One of these functions was (retRoot) function. at first the ai forgot to call the function in main then we asked him that you have an error and you forgot to add this function in main, after that, the function doesn’t work at all so we fall in a long dialog with Compilot asking him that the function doesn’t work at all, and after an hour from trying we could get a function that works well for some cases ,so, I told him that ,but he damaged the code and we couldn’t get a better version than which works for some cases so we use it now.

**Compilot** **Code**

The code is an implementation of the class Polynomial supports a variety of operations, such as addition, subtraction, multiplication, evaluation, differentiation, integration, and root-finding using Newton's method.

Here is an explanation for the code from Copilot:

**1. Constructors and Destructor**

* + Default Constructor: Polynomial::Polynomial() : coeffs({0})
  + Initializes a polynomial with a constant term of zero. The coeffs vector stores the coefficients of the polynomial, where each element corresponds to the coefficient of a particular degree.
  + Parameterized Constructor: Polynomial::Polynomial(const vector<double>& coefficients)
  + Initializes the polynomial with a specific set of coefficients provided as a vector. Each index of the vector represents the coefficient of the corresponding degree.
  + Copy Constructor: Polynomial::Polynomial(const Polynomial& other)
  + Creates a new polynomial that is a copy of another polynomial (other). It copies the coefficients vector.
  + Destructor: Polynomial::~Polynomial()
  + Since no dynamic memory allocation is used, this destructor is just a placeholder.

**2. Operator Overloading**

* + Assignment Operator: Polynomial& Polynomial::operator=(const Polynomial& other)
  + This function allows assignment of one polynomial to another. It ensures that the current object is not assigned to itself and then copies the coeffs vector from the other polynomial.
  + Addition: Polynomial Polynomial::operator+(const Polynomial& other) const
  + Adds two polynomials by adding their corresponding coefficients. The result is stored in a new vector with size equal to the larger of the two polynomials, handling polynomials of different degrees.
  + Subtraction: Polynomial Polynomial::operator-(const Polynomial& other) const
  + Similar to the addition operator, but it subtracts the coefficients of the second polynomial from the first.
  + Multiplication: Polynomial Polynomial::operator\*(const Polynomial& other) const
  + Multiplies two polynomials. The result is a polynomial whose degree is the sum of the degrees of the two input polynomials. The multiplication is done by multiplying each term of one polynomial with each term of the other.
  + Equality: bool Polynomial::operator==(const Polynomial& other) const
  + Checks if two polynomials are equal by comparing their coefficient vectors.
  + Output Operator: ostream& operator<<(ostream& out, const Polynomial& poly)
  + This is a friend function that allows the polynomial to be printed in human-readable form. It formats the output, skipping zero coefficients, and adds the appropriate "x^n" for each term.

**3. Polynomial Operations**

* + Degree: int Polynomial::degree() const
  + Returns the degree of the polynomial, which is the highest exponent with a non-zero coefficient.
  + Evaluate: double Polynomial::evaluate(double x) const
  + Evaluates the polynomial at a given value of x using Horner's method, which is an efficient algorithm for polynomial evaluation.
  + Compose: Polynomial Polynomial::compose(const Polynomial& q) const
  + Composes two polynomials. It replaces every x in the current polynomial with the polynomial q, creating a new polynomial as a result.

**4. Calculus-related Operations**

* + Derivative: Polynomial Polynomial::derivative() const
  + Returns the derivative of the polynomial. It computes the derivative by multiplying each coefficient by the degree of its corresponding term and then reduces the degree by one.
  + Indefinite Integral: Polynomial Polynomial::integral() const
  + Returns the indefinite integral (anti-derivative) of the polynomial by dividing each coefficient by its degree plus one and increasing the degree of each term.
  + Definite Integral: double Polynomial::integral(double x1, double x2) const
  + Computes the definite integral of the polynomial between x1 and x2. It first computes the indefinite integral and then evaluates it at the two bounds (x2 and x1), subtracting the results to get the area under the curve between the two points.

**5. Root Finding with Newton's Method**

* + getRoot: double Polynomial::getRoot(double guess, double tolerance, int maxIter)
  + Implements Newton’s method for finding a root (solution where the polynomial equals zero) starting from an initial guess. The algorithm iteratively improves the guess by applying the formula:
  + Where is the derivative of the polynomial at ​. The method stops when the improvement is smaller than the given tolerance or when the maximum number of iterations is reached. It includes a check to avoid division by zero if the derivative is too small.

**6. Utility Functions**

* + setCoefficients: void Polynomial::setCoefficients(const vector<double>& coefficients)
  + Sets the polynomial's coefficients directly by copying the input vector.
  + getCoefficient: double Polynomial::getCoefficient(int degree) const
  + Returns the coefficient for the specified degree of the polynomial. If the degree is larger than the highest degree of the polynomial, it returns 0.