3.C Multiply by Term

Input: coefficient, which is the coefficient of the multiplied monomial
power, which is the degree of the multiplied term

Output: a new formed polynomial based on the map new_term, which is the product of the polynomial and the monomial

Method called: get_terms

1. Create a new instance of the Polynomial formed from the term obtained from the original polynomial by calling the method named get_term.
   Use the variable new_polynomial to refer to this new instance

2. Get the term of the polynomial by calling the method named get_term, which is a map that maps the powers and corresponding coefficients of each part of the polynomial.
   Use the variable terms to refer to this term

3. Get the key and corresponding value of terms as a sequence whose elements are tuples that consist of two element: One is the key representing power; Another one is the value representing corresponding coefficient
   Use the variable item to refer to this sequence

4. Iterate each number from 0 to the length of the item and assign the value to the variable index:
   - Transcribe each element whose index is index from a tuple to a sequence
   - Add the value of the first element of the element whose index is index inside the sequence item with the value of the given power
   - Mulitply the value of the second element of the element whose index is index inside the sequence item with the value of the given coefficient under z256

5. Transcribe the item from a sequence to a map and use the variable new_term to refer to this map

6. Return a new Polynomial instance formed from new_term

4.A Polynomial Addition

Input: other_polynomial, which is the added polynomial

Output: a new polynomial formed from a map obtained from reference, which is the sum of the original polynomial and other_polynomial

Method called: get_terms, add_term

1. Create a new instance of the Polynomial formed from the term obtained from the original polynomial by calling the method named get_terms.
   Use the variable self_polynomial to refer to this new instance

2. Get the map from the given other_polynomial by calling the method named get_terms.
   Use the variable other_terms to refer to this map

3. Get the key and corresponding value of other_term as a sequence whose elements are tuples that consist of two element: One is the key representing power; Another one is the value representing corresponding coefficient
   Use the variable other_terms to refer to this sequence

4. Use the variable reference to refer to self_polynomial

5. Iterate each number from 0 to the length of the other_terms and assign the value to the variable index:
   Get the value of the second element of the element whose index is index inside the sequence other_terms and assign the value to the variable coefficient
   Get the value of the first element of the element whose index is index inside the sequence other_terms and assign the value to the variable degree
   Add the term whose coefficient is the value of coefficient and whose power is the value of degree to reference

6. Return a new Polynomial instance formed from the term obtained from reference by calling the method named get_terms

4.C Polynomial Multiplication

Input: other_polynomial, which is the multiplied polynomial

Output: a new instance of the Polynomial formed from the term obtained from the variable result, which represents the sum of multiplying each term in this polynomial itself by the given other_polynomial

Methods called: get_terms, multiply_by_term, add_polynomial

1. Create a new instance of the Polynomial formed from the term obtained from the original polynomial by calling the method named get_terms.
   Use the variable self_polynomial to refer to this new instance

2. Get the map from the given other_polynomial by calling the method named get_terms.
   Use the variable other_terms to refer to this map

3. Get the key and corresponding value of other_term as a sequence whose elements are tuples that consist of two element: One is the key representing power; Another one is the value representing corresponding coefficient
   Use the variable other_terms to refer to this sequence

4. Get the map from the given self_polynomial by calling the method named get_terms.
   Use the variable self_terms to refer to this map

5. Get the key and corresponding value of self_terms as a sequence whose elements are tuples that consist of two element: One is the key representing power; Another one is the value representing corresponding coefficient
   Use the variable self_terms to refer to this sequence

6. Create an empty sequence and use the variable temp_product to refer to this sequence

7. Create a new polynomial instance and use the variable result to refer to this instance

8. Check if self_terms and other_polynomial are empty sequences:
   If the output is true:
      return result
   If the output is false:
      Iterate each number from 0 to the length of the other_terms and assign the value to the variable index:
         Get the value of the second element of the element whose index is index inside the sequence other_terms and assign the value to the variable coefficient
         Get the value of the first element of the element whose index is index inside the sequence other_terms and assign the value to the variable degree
         Multiply self_polynomial with the term whose coefficient is the value of coefficient and whose power is the value of degree. Use the variable reference to refer to the product.
         Get the terms of reference as a map. Use the variable reference_terms to refer to this map
         Add reference_terms into the sequence temp_product
9. Iterate each number from 0 to the length of the temp_product and assign the value to the variable index2:
      Create a new Polynomial instance based on the element whose index is index2 inside the sequence temp_product. Use the variable single_temp to refer to this instance
      Add result to single_temp. Use the variable result to refer to the sum
10. Return a new Polynomial instance formed from the term obtained from result by calling the method named get_terms

4.D Polynomial Remainder

Input: denominator, which is the given polynomial used as the denominator

Output: self_term a new instance of the Polynomial which represents the reaminder of dividing the original polynomial by the given denominator

Methods called: get_terms, get_degree, subtract_polynomial, multiply_by_term

Function used**:** divide_terms, whose four arguments is the coefficient and corresponding power of numerator and the coefficient and corresponding power of denominator

1. Create a new instance of the Polynomial formed from the term obtained from the original polynomial by calling the method named get_terms.
   Use the variable self_term to refer to this instance

1. Get the highest degree of the terms in polynomial and assign its value to the variable check_degree

2. Check if both the value of check_degree and if the value of the highest degree of denominator equal to zero:
   If the output is true:
   Return a new Polynomial instance formed from the map whose key and value both equal to zero

3. While the value of check_degree is bigger than or equals to the value of the highest degree of denominator and at the same time check_degree not equals to zero:
   Get the highest degree of self_term and assign its value to the variable max_degree
   Get the corresponding coefficient for the term in self_term whose power is max_degree assign its value to max_coefficient
   Get the highest degree of denominator and assign its value to the variable max_degree2
   Get the corresponding coefficient for the term in denominator whose power is max_degree assign its value to max_coefficient2
   Divide the numerator whose coefficient is max_coefficient and corresponding power is max_degree by the denominator whose coefficient is max_coefficient2 and corresponding power is max_degree2. Assign the value of the division to the variable quotient.
   Multiply quotient with denominator. Assign the value of the product to the variable product
   Subtract product from self_term. Assign the value of the difference to the variable self_term
   Get the degree of self_term and assign the value of the degree to check_degree

4. Return self_term

Discussion

1. The value of remainder calculated by using message dividing generator indicates my error correction bytes.

No, it is not possible.

However, it doesn't matter because when we encode the message, we need to combine the error correction bytes with the message block. Since the coefficient is zero, this represents that

the term with corresponding coefficient doesn't exist. Therefore it will not cause influence to the encoded data.

2. We can change the content of the module z256. We can still keep the name of the module as z256, but we change the content of the module as a module that calculate the addition, difference, product, power and division of polynomials under the normal regular arithmetic instead of z256 arithmetic.