DESIGN DOCUMENT: Project 0

NAME: Naveela Sookoo
STUDENT ID: nnsookoo

The assigned task was to create a C++ program that performs operations on a single variable polynomial by using a linked list to store the coefficients of the polynomial. A polynomial class was designed in order to accommodate the services stated in the project description.

The member variables of the polynomial class include a struct object, called Term, which its members are coefficient (double coefficient), exponent (int exponent) and a pointer to the next term, called next (Term* next). The member variables also include, a pointer to the 'Term's' head (Term* head), which denotes the beginning of the polynomial linked list, and a size (int size) variable which was used to define the number of coefficients in the polynomial.

The member functions included:

1. **Constructors**:
    a. To create an empty polynomial: **Polynomial();**
    This was designed to make an empty polynomial by defining the head of the polynomial linked list to be NULL and the size to be 0.
    b. To create a defined polynomial: **Polynomial(const int size, const double coeff[]);**
    This was designed to take a sorted array of coefficients as well as the size of the polynomial to create a linked list. Each "node" or "term" of the polynomial would be assigned a coefficient.
    c. To create a copy of an already defined polynomial: **Polynomial(const Polynomial& polynomial_to_copy);**
    This constructor was designed to take an already defined polynomial and assign its contents to the "this->" polynomial. First, the constructor checked if the head of the "this->" polynomial is NULL. If it isn't NULL, the list will be destroyed using the destroy() function. Then, the function proceeds to copy the contents of the defined polynomial to the "this->" polynomial using a 'for loop'. Within the 'for loop', the function InsertTerm is used to copy each term at a time.
2. Destructor: **~Polynomial();**
    The destructor calls the destroy() function which is discussed later on.
3. **destroy()**
    This function is used to clear the contents of the passed polynomial. It is designed as a 'void' function and hence does not return anything. It also does not have any parameters assigned it.
4. **copy()**
    This function is used to copy the contents of one polynomial to the passed polynomial. It is designed as a 'void' function and hence also does not return anything. Its single parameter is a polynomial, passed by reference, whose contents will be copied to another polynomial.
5. **Assignment operator (=)**
    This function is used to override the = operator. It utilises the copy() function. Its single parameter is a polynomial, passed by reference, whose contents is then passed to another polynomial. The contents of the polynomial, for example: p1, is copied to another polynomial, for example: p2, by implemented "p2 = p1".
6. **Equivalence operators**:
    a. Function to test equivalence(==)
    This function is used to override the == operator. It is designed as a 'bool' function in which it will return true if two polynomials are equal and false if two polynomials are not

equal. Its parameter is a polynomial, passed by reference, which is compared to another polynomial.

b. Function to test non-equivalence(!=)
This function is used to override the != operator. It has the same design as the equivalence operator.

7. **size_of()**
This function is used to get the current size of the passed polynomial. It is designed as an 'int' function in which it will return the size of the polynomial. It has no parameters but the size of a polynomial, for example: p1, is returned by implementing "p1.size_of();".
8. **InsertTerm()**
This is used to insert a term into the end of a polynomial linked list. It is designed as a 'void' function. Its parameters are the exponent of the term (int exp) and the coefficient of the term (double coeff).
9. **init()**
This function is used to define the size of the polynomial. It is designed as a 'bool' function. Its parameter is the size, m. The function always returns true to indicate a successful initialization of the size of the polynomial.
10. **To define the coefficients of a polynomial**:
The following functions returns a polynomial. Its parameter is an array containing coefficients of type double.
    a. To define current polynomial: **coeffp1();**
    b. To define second polynomial: **coeffp2();**
    c. To define expected polynomial: **expected();**
11. **get()**
This function compares the two polynomials. It is a 'bool' function and returns true if the two polynomials have the same size and have the same coefficients. Its parameter is a polynomial, passed by reference, which is utilised in the comparison.
12. **evaluate()**
This function evaluates a polynomial in which its parameters are a number to evaluate the polynomial to and the expected value of the evaluation. It is a 'bool' function and return true if the expected value is equal to the evaluated value.
13. The following functions were designed to add two polynomials.
    a. add()
    b. Polynomial operator+(const Polynomial& right);

    Function (a)'s parameter is a polynomial, passed by reference, which is added to another polynomial. Function (a) returns the sum polynomial. Function (b) is used to override the + operator.

14. The following functions were designed to multiply two polynomials.
    a. mult()
    b. Polynomial operator*(const Polynomial& right);

    Function (a)'s parameter is a polynomial, passed by reference, which is added to another polynomial. Function (a) returns the product polynomial. Function (b) is used to override the * operator.