The project comprises 4 files: main.cpp in which we can find the main function of my program,as well as all the operations we were asked to perform on the polynomias; we can also find 3 headers: Stack.h,CircularQueue.h and Term.h.

The Term.h file contains the class Term,used to define a term of the polynomial. For example 3x^2 wil be represented as a term with the coefficient value set to 3 and the exponent set to 2. This file also has a method that overloads the << operator so that we can display the content of the circularQueue using cout << term.

Stack.h has a constructor, destructor and push(),pop(), dispay() methods.

As we well know, using push() we add an element to the top level if the stack is not already full. Pop is used to delete the last element in the stack. If the stack is not empty, we copy the value of topLevel, decrement the topLevel and then return the copied value. In the display() method I used a for to go through all elements of the stack and display them starting with the last element.

CircularQueue.h has the method enqueue(x),used to add an element in the queue if the queue is not already full. If it isn’t, we add the element. Next function is dequeue(). I checked to see if the tail and head have the value -1. If this condition in fulfilled, then the queue is empty. Otherwise, we store the value of the head in a temporary variable , check if the tail= head. If true, we set the head to -1. We decrement the current size and display the values saved in a temporary variable. The display() function prints on the screen the elements of the queue.

CircularQueue also has a constructor and a destructor.

Both CircularQueue.h and Stack.h contain a copy constructor and copy assignment operator overload. When you pass an object as a parameter to a function in C++ the copy constructor is used to create a new object that is used in the context of the function. Because both the Stack and the Queue contain a pointer to an array respectively stackArray and queueArray when passing these objects as value parameters to functions we need a custom copy contructor that will copy not just the pointer to the memory but the memory itself. Otherwise both the object in the function and the object from the external calling context will point to the same memory area and the program will function unexpectedly.

Moreover we also need to define copy assignment operator overload for the same reasons stated above.

The class PolynomialOperations contains:

* Void readPolynomsFromCmd: first,we are asked to insert the value for which the polynomials are going to be computed. I created two polynomials: first we get the degree for the first polynomial,then using a for structure we get all the coefficients and each polyTerm having a coefficient and a exponent is added in the queue. I did the same for the second polynomial.
* Void displayPolynom:prints on the screen a polynom stored in a queue
* Void displayPolynomFromStack: prints on the screen a polynom stored in a stack
* Int polyValue: I created the stackPartialSum(1) and performed the push(0) operation so that the stack will have one element which in the beginning is going to be 0;we use pop() to get the sum from stackPartialSum,respectively push() to add back in the stack the new sum obtained after the computations are done
* invertPolynom: I created invertedPolySack and used the dequeue operation to get the last term from the polynomial. As long as there are elements in the queue,we push them in the invertedPolyStack. After the procedure is done the program will extract all the terms from the stack and add them to a new inverted polynom
* void substractPolynoms: I created stackResult to save the computation result,then I performed the dequeue operation on the two polynomials. While the exponents of the firstPoly and secondPoly are different from -1,I add the new coefficient in a new variable(substrCoef) and create substrTerm having the new coefficient and the exponent of one of the two polynomials. Then we move on to the following terms.
* Void multiplyPolynoms: I created stackResult to store the multiplication result.

Using dequeue, I get a term from firstPoly which I am going to multiply with every term from secondPoly and add them to stackReslut. This process is going to take place as long as he exponent of firstPolyTerm is different from -1; If the condition is true,I store the terms of the secondPoly in an auxiliar queue and perform the dequeue operation once. While the exponent of secondPolyTerm(obtained by dequeuing auxQueue) is different from -1,we multiply the coefficients and add the exponents and add them into new variables,variables that are going to define a new term which is going to be pushed in stackResult. Inside the first while we perform the dequeue operation of firstPoly again.

Next we have to sum the terms having the same exponent.In order to do that, we copy the elements from stackResult to another stack,an auxiliary one.We perform the pop() operation on stackResult and the term is going to be compared with all the terms in the auxiliary stack.Once we fine two terms having the same exponent,we sum their coefficients.The algorithm is pretty much similar to the one used above. The only difference is that we need the vector solvedExponents to store the terms having a particular exponent so that if there are two terms alike in the stack we don’t perform the procedure twice or more times.