ECOR 1606 Lab #10

Note: To prove your attendance at this lab, submit file "*lab10.cpp*" containing your program by the end of the lab period.

Introduction:
Arrays can be used to represent polynomials. The simplest approach is to store the coefficient for x^0 at array position 0, the coefficient for x^1 at array position 1, and so on. This approach is illustrated below.

polynomial:

$6x^3 - 4x^2 + x - 5$ (can also be written as $6x^3 - 4x^2 + 1x^1 - 5x^0$)

array representation:

[0]	[1]	[2]	[3]
-5	1	-4	6

The derivative of a polynomial is also a polynomial (and can be stored in the same way).

polynomial: $6x^3 - 4x^2 + x - 5$
derivative: $18x^2 - 8x + 1$

If a polynomial is of order "N", its derivative will be of order "N – 1".

Part 1:
Modify the starting point given (*lab10start.cpp*) by writing the two missing functions. You do **not** need to change the main program in part 1. The prototypes for these functions should be:

```
int readPolynomial (int maxOrder, double polynomial[]);
void displayPolynomial (int order, double polynomial[]);
```

Function *readPolynomial* should return the order of the polynomial read. It is similar to the *readArray* function in the notes. You will find details of what this function should do in the comments for the function, and also run the sample executable for part 1.

Note that the program is capable of handling up to 8th degree polynomials. That means that our array is nine elements long. If the user enters an unacceptable polynomial order, your program should reject it and have the user try again. Similarly, zero is not permitted for the first coefficient (as that would change the order of the polynomial).

Your program should behave in the same way as the supplied *sample1.exe*.

Function *displayPolynomia*l outputs the polynomial. This function is similar to the *writeArray* function in the notes. (See hint on next page.)

Hint: The easiest way of dealing with the plus and minus signs is to do something like the following:

```
if (polynomial[i] > 0) {
   cout << " + " << polynomial[i];
} else {
  cout << " - " << fabs(polynomial[i]);
}
```

Part 2:
Extend your program by having it read in values for *x* until 0 is entered. For each value entered your program should call new function *evaluatePolynomial* to calculate and return the corresponding value of the polynomial, which is output by the main program. Your program should behave like *sample2.exe*.

The function prototype is:

```
double evaluatePolynomial (int order, double polynomial[], double x);
```

The function should return the value of the polynomial at *x*. (Reminder: the loop is in the main program.)

Part 3:
Write a function that, given a polynomial, calculates the derivative of the polynomial (which is, of course, just another polynomial). The prototype for this function should be:

```
int differentiate (int order, double polynomial[], double derivative[]);
```

Your function should store the coefficients of the derivative in array *derivative* and return the order of the derivative.

Extend your program so that it calculates the derivative of the originally entered polynomial (using function *differentiate*), displays it (using function *displayPolynomial*), and then reads values for *x* until 0 is entered. For each value of *x* entered your program should output the corresponding derivative value (calculated using function *evaluatePolynomial*).

Compare your program's output to that produced by *sample3.exe*.

Optional Part 4:
If you have time left, add an integrate function -- see *sample4.exe*.