PS5

- Don't wait till the last moment.

`Q1`

Using dynamic arrays, implement a polynomial class with infix operators `+, -, *.` Discussion: In a polynomial, a variable is placeholder for coefficient. If term is missing, the coefficient is 0. For example, the cubic polynomial `2x3 -3x + 4` or, written out as C++ code, `2*x*x*x -3*x + 4` has the coefficients for terms as listed:

- degree 3 term has coefficient 2
- degree 2 term has coefficient 0
- degree 1 term has coefficient -3
- degree 0 term has coefficient 4

Note that the term with degree 2 is missing, but we will use a coefficient of 0 to indicate that. Observe that the size of the coefficient array is 4, one more than the degree.

Use of sparse matrix techniques not recommended. We will assume that there are few missing terms.

Provide these member functions:

- default constructor
- copy constructor
- operator=
- destructor
- parameterized constructor to create an arbitrary polynomial
- operator+
- operator-
- operator*
- assign and inspect function (or functions) for
- coefficients, indexed by exponent
- function to evaluate polynomial as a value of type double

The student is to decide on whether these are to be member, friend, or neither (standalone).

NOTES: The default constructor creates an empty polynomial. A zero polynomial has degree 0, since it has only the zero degree coefficient.

In the coefficient array, the index is the value of the exponent of term having this coefficient. For example, the index 0 entry is the constant coefficient, the index 1 entry is coefficient of the linear term, the index 2 entry is the coefficient of the quadratic term (term in x2), etc.

The size of the coefficient array include a degree 0 entry, so the size is the degree of the polynomial + 1.

Odd error messages associated with overloading some of the operators occur if you try to pass a Polynomial object to one of the functions by const reference and do not implement both these operator[] implementations,

```
//This version of operator[] is used when an indexed
//expression is used as an l-value.
double& operator[](int degree);
//This version of operator[] is used when an indexed
//expression is used as an r-value.
const double& operator[](int degree) const;
```

Instead of difficult-to-understand compiler error messages, you may get linker errors, which is an even harder situation, since linker errors are not associated with any particular line in your source.

The alternative is to pass by value. In this example, that is not a problem but in general, passing a class object has the potential for a large amount of copying of data, so const reference is the desirable way to do it.

Neither of the operator functions for multiply, add or subtract check for zero lead coefficients. For polynomials that have many zero terms, these functions should account for the zero entries. Two polynomials of high degree with few nonzero

terms will waste considerable time multiplying zero entries using this technique. The fix is the use of a linked list (Chapter 17), or the STL list (Chapter 19.)

Use the following main function (Change test data. Create objects with different data):

```
int main()
{
   Polynomial empty;
   double one[] = {1};
   Polynomial One(one, 1);
   double quad[] = {3, 2, 1};
   double cubic[] = {1, 2, 0, 3};
   Polynomial q(quad, 3); // q is 3 + 2*x + x*x
   Polynomial c(cubic, 4);// c is 1 + 2*x + 0*x*x + 3*x*x*x
   Polynomial p = q;  // test copy constructor
   Polynomial r;
   r = q;              //test operator=
   r = c;

   cout << "Polynomial q " << endl;
   {for(int i = 0; i < 3; i++)
     cout << "term with degree " << i
          << " has coefficient " <<  q[i] << endl;
   }
   cout << "Polynomial c " << endl;
   {for(int i = 0; i < 4; i++)
     cout << "term with degree " << i
          << " has coefficient " <<  c[i] << endl;
   }
   cout << "value of q(2) is " << evaluate(q, 2) << endl;
   cout << "value of p(2) is " << evaluate(p, 2) << endl;
   cout << "value of r(2) is " << evaluate(r, 2) << endl;
   cout << "value of c(2) is " << evaluate(c, 2) << endl;

   r = q + c;
   cout << "value of (q + c)(2) is " << evaluate(r, 2) << endl;

   r = q - c;
   cout << "value of (q - c)(2) is " << evaluate(r, 2) << endl;

   r = q * c;
   cout << "size of q*c is " << r.mySize() << endl;
   cout << "Polynomial r (= q*c) " << endl;

   for(int i = 0; i < r.mySize(); i++)
     cout << "term with degree " << i
          << " has coefficient " <<  r[i] << endl;


   cout << "value of (q * c)(2) is " << evaluate(r, 2) << endl;

   return 0;
}
```

Q2

Create a class named **Student** that has three member variables:

- name – A string that stores the name of the student

- numClasses – An integer that tracks how many courses the student is currently enrolled in
- classList – A dynamic array of strings used to store the names of the classes that the student is enrolled in

Write appropriate constructor(s), mutator, and accessor methods for the class along with the following: - A method that inputs all values from the user, including the list of class names. This method will have to support input for an arbitrary number of classes.
- A method that outputs the name and list of all courses. - A method that resets the number of classes to 0 and the classList to an empty list. - An overloaded assignment operator that correctly makes a new copy of the list of courses. - A destructor that releases all memory that has been allocated.

Write a main method that tests all of your functions.

CodeMate hint: Recall that cin » variable leaves a newline in the buffer. This can be a problem if you are mixing cin » variable and getline. Use cin.ignore to discard the newline.

Use the following main function (Change the object data):

```
int main()
{

  Student s1, s2;

  s1.InputData();       // Input data for student 1
  cout << "Student 1's data:" << endl;
  s1.OutputData();      // Output data for student 1

  cout << endl;

  s2 = s1;
  cout << "Student 2's data after assignment from student 1:" << endl;
  s2.OutputData();      // Should output same data as for student 1

  s1.ResetClasses();
  cout << "Student 1's data after reset:" << endl;
  s1.OutputData();      // Should have no classes

  cout << "Student 2's data, should still have original classes:" << endl;
  s2.OutputData();      // Should still have original classes

  cout << endl;
  return 0;
}
```

Turn In

- Make and submit a zip file(`<your_full_name>_PS5.zip`) which includes the following:
  - Source code of `Q1`: `q1.cpp`
  - Source code of `Q2`: `q2.cpp`
  - Run `Q1` and `Q2` and attach screenshots(in `jpg` format, not exceeding `300kb` each) which show that your programs are running.
  - At least `1` screenshot for each question.