Here the constant term, 7, is first, then the linear (x) term, 1, then the quadratic (x2) term, 0, and finally the highest degree term with a coefficient of 4. In addition, the degree of the polynomial, 3, is stored in the member variable degree.

Storing the polynomials in this order, rather than in the order in which we normally write them, will make it easier to perform arithmetic operations. It will make it slightly more work to print, but I think you will appreciate the trade-off..

The class *invariant* ,i.e., the thing that must be true about the class whenever a member function is *finished* making changes, is:

1. The coefficients are stored in a linked list with the constant term at the front of the list and the highest order term at the end of the list.
2. The zero polynomial is represented by a list with a single item, zero. The degree of the zero polynomial is 0.
3. The list for a non-zero polynomial does not end with a zero coefficient. In other words, the highest degree coefficient is only zero if this is the zero polynomial.
4. The value of the degree member variable is one less than the length of the list.

What you have to handle:

* Constructors:
  + A default constructor could be handy. Obviously, the result would be a Polynomial with just a the constant term, which would be zero.
  + A constructor that takes a vector of its coefficients in order from the highest degree term to lowest
* +=
* +
* ==
* !=
* <<
  + Use the caret ^ for exponentiation. So: 5x^3 represents five times the term with an exponent of three.
  + Do not display terms with zero coefficients. Unless, of course, it is the zero polynomial. (If I didn't say that, someone would ask.)
  + Do not display coefficients whose value is 1. So you would not display 4x^3 + 1x^2, but rather just 4x^3 + x^2. (Unless it is the low order coefficient, i.e. the constant term.)
* A method eval that takes a single argument and evaluates the polynomial for that value of "x". So if the variable popy represents the polynomial 4x3 + x + 7, then poly.eval(2) would return 41.
* copy control, i.e. destructor, copy constructor and assignment operator

**Notes**

* Lets keep things simple and restrict coefficients to integers.
* **No vectors!** Aside from the constructor that takes in a vector of the coefficients, there *should not be any use* of vectors in the entire program!

**Test program:** attached is a test program and corresponding output.  Do feel free to expand on it.  You may not be familiar with all of the syntax used in this program.

* boolalpha.  This *manipulator* tells the output stream that when it is printing a bool value, true or false, to print it that way, i.e. as the strings "true" and "false".  Without this manipulator, C++ will print true as 1 and false as 0.
* {2, 4, 8}.  The use of braces around a comma-delimited sequence of values can be used to initialize a vector to hold that sequence of values.  Please note that this use of braces requires C++11.

**What to hand in:**

Header and implementation files for polynomial (polynomial.h and polynomial.cpp).