This project is meant to be yet another practice for library creation.
You should create a class HunPolynomial. It should be submitted in two files: HunPolynomial.h and HunPolynomial.cpp. This class represents polynomials with <u>integer coefficients</u> and <u>a single variable x</u> and <u>positive integer exponents</u>. The examples of such polynomials are

    5*x^3 – 12*x – 1
    – 34*x^2 + x

Pay attention to how we print polynomials above:

- we separate + and – signs with spaces unless polynomial starts with the – sign. If polynomial starts with the – sign, that sign doesn't have leading space. Polynomial cannot start with the + sign.
- "power of" is denoted by ^ symbol.
- multiplication is denoted by * and is never omitted.
- multiplication symbol * doesn't have spaces to the left or to the right.
- variable name is always lower-case x.
- terms with the coefficient of 0 aren't printed
- terms with a coefficient of 1 or -1 are printed without the coefficient unless it's a constant term (the rightmost term with no variable).
- + and – never follow each other.

You must use this format when printing polynomials.
We will represent polynomials using vectors of int. Each element of the vector corresponds to a polynomial coefficient. The element with index 0 corresponds to the most significant term. Coefficient with the value 0 are stored in vector (even though they are not printed). For example,

    5*x^3 – 12*x – 1   => std::vector<int>{5, 0, -12, -1}
    – 34*x^2 + x   => std::vector<int>{-34, 0, 1, 0}

You must implement a decent program structure and decent programming style.

Class HunPolymonial has the following public methods:

- Default constructor that constructs an empty polynomial
- Parametrized constructor that takes a vector of integers and initializes polynomial according to the polynomial representation rules above.
- Method Set that takes a vector of integers that represents a polynomial and sets polynomial to that value.

- Operator +. It uses standard polynomial addition rules.
- Operator –. It uses standard polynomial subtraction rules.
- Operator *. It uses standard polynomial multiplication rules.
- Operator << that allows printing. Printing should happen according to the printing rules discussed above. Attention, operator<< won't be a class member. Writing such operator is discussed in our textbook in C++ Interlude 6, section 2 (page 441). In general, C++ Interlude 6 (page 435) discusses overloading operators for your class.

- Operator () that takes a <u>float</u> x0 and returns a float value of polynomial in point x0 . Ignore a possibility of overflows.

Pay attention, there is no need for custom destructor, copy-assignment, and copy-constructor, if you don't use dynamic memory allocation here. And there is <u>no need</u> for dynamic memory allocation in this assignment. Regular vector class is all that you require.
I guess by now the importance of matching names to the specification is clear for everyone.
Feel free to create additional private methods if needed.
And, please, <u>don't</u> put using namespace std in the header file.

TESTING.
Please find a test file attached. As discussed in class, the test file shouldn't be changed. Your program should work with it "as is". If you think the test file has an error in it (which is possible), write me and I'll fix it for everyone or explain you why it is correct.
As always, the test file isn't meant to be complete and doesn't guarantee that your program is bug-free.
On Linux lab computers, your library should be placed in the same folder with the test file and compiled with the command

```
g++  -std=c++17  TestPolynomial.cpp  HunPolynomial.cpp  -o prog
```