Goals

The goal of this lab are

- Design and implement a polynomial library to support various operations such as addition, multiplication and comparison
- Create a fast polynomial multiplication algorithm (hint: threads may help).

The way that you approach the problem is up to you, but the design of your solution is a key part of the lab experience.

Deadlines

There are two deadlines for this project

Mon November 21st 9pm

All functions must be implemented as described, but timing doesn't matter for this submission. We strongly suggest getting the following to work before working on anything else

- addition
- multiplication
- constructor
- canonical form

note: We use canonical form to compare your solutions to ours. This means that if your canonical form doesn't work we have no way to verify if your solution is correct. So it's imperative that your canonical form is correct.

Thu December 1st, 9pm

All functions must be implemented as described. Timing matters for this submission, and we'll provide more timing information closer to this deadline.

Restrictions

- You must implement all functions/operators described in the header file
- You ARE allowed to add fields and functions to the header file, but you're not allowed to remove any fields/methods that we give you. This means you can define whatever helper functions/extra fields that you want to help with the project.
- You're not allowed to use any external libraries (except pthreads) in your solution. This is because we're only linking against the pthreads library, so we won't be able to use whatever libraries you link against.
- You must turn in a poly.cpp file and a poly.h file. Each function/operator defined in poly.h must have a definition in poly.cpp.

Polynomial files

We're not giving you any grading code this time, but we are going to give you a sample polynomial file that you can use if you want (you'll need to write a parser for it if you want to use it for testing, or you can do testing by changing main.cpp to include examples for testing.)

The format of the polynomial file is

```
<coefficient>x^<power>
<coefficient>x^<power>
....
<coefficient>x^<power>;
```

note: the semicolon is there to make parsing easier, since it marks the end of the polynomial. Do not assume the powers are in any particular order (ascending, descending etc.).

It's also important to notice that there are no addition signs between the polynomials, the addition is implied. So if you wanted to write

$$3x^{10} - 7x^2 + 1$$

This would be represented in the polynomial file as

```
3x^10
-7x^2
1x^0
;
```

Thread examples

This is a short example of how to use threads, and also has documentation for what functions threads have.

Here is a more in depth explanation of how to use threads.

Running your code

We've given you a main.cpp file that has some code in it that you can use to time your own code. That will help you figure out how long your polynomial multiplication takes to run.

We'll give more timing advice closer to the deadline. Additionally this lab won't work well on ecegrid when you time your program (ecegrid is slow). Instead you should run all your timing tests on `eceprog`. See here for instructions on how to do that.

note: the default version of gcc on eceprog is 11, so there's no need for module load gcc on this cluster. But other clusters (such as ececomp listed in the URL above) still may need it. Please make sure to use gcc –version to check you are using a version that is at least 8.3 or newer. Please also ensure that your code compiles with the -std=c++17 flag (this means you cannot use anything from C++20 onwards.) as this is a requirement for our auto-grader.

Hints and Advice

- Start immediately - you are on your own for testing, so budget plenty of time.
- First get a correct sequential implementation. This will ensure you at least get credit for this portion, and will give you a starting point for your optimizations for running time.
- What is a good data structure to use to represent a polynomial? You are free to use any STL containers of your choice, but you should carefully think through what a good choice is, and how to use it.
- How would you handle sparse polynomials? E.g., consider a polynomial with just two terms, $x^{100000} + 4$. What is a good way to represent it? How would the representation impact addition and multiplication?
- To do multiplication, you can potential speedup with multiple threads. What is a good design to obtain the parallelism?
- a * b and b * a should have similar speeds. Does your design do this? Only worry about this once you've gotten a parallel implementation working

How to use TA time

You should be using TA time to get feedback on your design and ideas, and questions related to threads. Don’t expect straight answers on what a good design is, but expect TAs to make you think of the pros and cons of a design you propose.