FINAL ASSIGNMENT

MAT 685

1. Download the `Rings` library described in class. Be sure you understand its structure.
2. Design a new `Polynomial` class, which should have the following features, *some of which I have already written for you!* Make note especially of certain protected methods I've written to help you implement other methods correctly, but *which are not listed below*.
   - It also exists within the `Rings` namespace.
   - It inherits from `Commutative_Ring_Element` and implements all remaining pure virtual functions in a reasonable way. (If you need help deciding what constitutes a "reasonable" answer, I can provide *a lot* of help.)
   - Its coefficients are parameterized or templated. That is, I should be able to instantiate `Polynomial<Modp<int, 5> >` if I wish, and I would have a polynomial whose coefficients have type `Modp<int, 5>`. In what remains, I will refer to the parametrized type as `R`.
   - A `typedef` defines `DEGREE_TYPE` as meaning `uint32_t`.
   - It has at least the following constructors:
     - Default constructor creates a zero polynomial.
     - Copy constructor.
     - Single-argument `const initializer_list<R> &` assigns the first entry to the constant term, the second entry to the linear term, etc., so that the last entry is the leading term. For instance, `Polynomial<int> p = { 1, 3, 0, -2 } ;` would initialize $-2x^3 + 3x + 1$.
     - Single-argument `const vector<R> &` assigns the first entry to the constant term, the second entry to the linear term, etc.
   - It accepts the following queries:
     - `DEGREE_TYPE degree() const` returns the polynomial's degree.
     - `const R & coeff(DEGREE_TYPE) const` returns the coefficient at the specificied degree.
     - `const R & operator() (const R &) const` returns the value of the polynomial at the specified point.
       (We haven't talked much about this operator, but `operator() (...)` allows you to treat the polynomial as if it were a function.)
   - It accepts the following modifications:
     - `void set_coeff(DEGREE_TYPE, const R &)` sets the coefficient of the specified degree to the specified value.
     - `const Polynomimal<R> & operator =(const Ring_Element &)` is the assignment operator and returns the polynomial itself.
   - It has methods that implement the following arithmetic and comparison operators:
     - `bool operator == (const Ring_Element & ) const;`

Date: Spring 2017.

  - `bool operator == (const R &) const;`
    (This last one should be `true` if and only if the polynomial is constant and its constant term equals the value passed in.)
  - `const Polynomial<R> & operator +(const Ring_Element &) const;`
  - `const Polynomial<R> & operator -(const Ring_Element &) const;`
  - `const Polynomial<R> & operator *(const Ring_Element &) const;`
  - `const Polynomial<R> & operator /(const Polynomial<R> &) const;`
  - `const Polynomial<R> & operator %(const Polynomial<R> &) const;`

  Remark. `operator /` and `operator %` should give the quotient and divisor from division (respectively) when dividing by a *monic* polynomial.[1] These will probably take the most work.

  Remark (Extra Credit). When you compute the quotient by hand, you get the remainder for free, and vice versa. That should happen in your program, too. Trouble is, the operators only ask for one or the other, so you lose valuable information. For extra credit, work out a way to "cache" these results so that they are remembered. That way, if a client asks for both the quotient and remainder in succession, you don't have to waste time computing them twice; you just have to make sure the client is asking for the quotient and remainder *with respect to the same divisor*.

- It can perform output using the following function that is *outside* the class but *inside* the namespace:
  - `ostream & operator << (ostream &, Polynomial<R>);`

3. I have included the program `test_polynomial.hpp` that you can use to test your implementation. *This program must compile and run with your code.* You can also find documentation in the directory, or online.

[1] Monic polynomials have leading coefficient equal to 1. If the leading coefficient is not 1, things are more complicated.