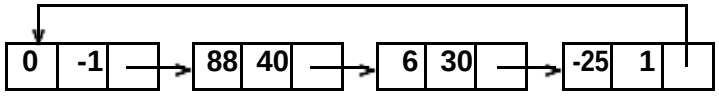
Univariate Polynomial Manipulation

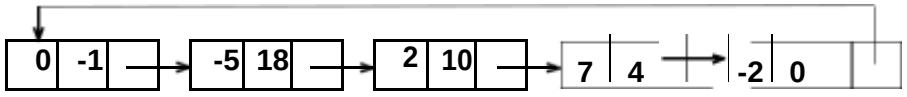
Task description. Design, implement and test an ADT, `Polynomial`, that provides some of the basic operations for univariate polynomials. For example, P (x) = $3x^4 - 7x + 18$ is such a polynomial.

Your ADT `Polynomial` should have a data member belonging to the class `CircularList` which keeps the terms (term consists of coefficient and exponent) in the polynomial in a circular linked list. The circular list representation of a polynomial has one node for each term that has non-zero coefficient. The terms are in decreasing order of exponent and the head node has its coefficient and exponent field equal to 0 and -1 respectively. The following figure gives some examples.

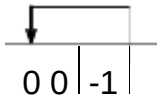
(a) P**1**(x) = $88x^{40} + 6x^{30} - 25x$



(b) P**2**(x) = $-5x^{18} + 2x^{10} + 7x^4 - 2$



(c) P**3**(x) =



The ADT `Polynomial` should support the following operations. Note that some operators should be overloaded to make your code more readable. You may add any other public/private member functions that you think are necessary.

(a) `Polynomial()`

- Create the zero polynomial, that is P (x) = 0. `Polynomial()` is the class default constructor.

(b) `friend istream& operator>>(istream&, const Polynomial&);`

- Read in a polynomial from `cin`. Each polynomial has the following form:

$$c_1\ e_1\ c_2\ e_2\ c_m\ e_m\ 0 - 1$$

where c_i and e_i are integers denoting the coefficient and exponent of the ith term, respectively. The last pair 0 − 1 denotes the end of polynomial.
You can assume that the exponents are in decreasing order; that is $e_1 > e_2 > \ldots > e_m \geq 0$, and there is no zero coefficient in the input; that is $c_i = 0$ for all i.

(c) `friend ostream& operator<<(ostream&, const Polynomial&);`

- Output the polynomial to `cout`. The output format should be the same as the input format. That is, the exponents should be in decreasing order and all coefficients are non-zero. Also it should end with the pair 0 − 1.

(d) `friend Polynomial& operator+(const Polynomial& p1, const Polynomial& p2);`

- Add the two polynomials `p1` and `p2` and return the result.

(e) `friend Polynomial& operator-(const Polynomial& p1, const Polynomial& p2);`

- Subtract the first polynomial `p1` from the second polynomial `p2` and return the result.

(f) `friend Polynomial& operator*(const Polynomial& p1, const Polynomial& p2);` - Multiply the two polynomials `p1` and `p2` and return the result.