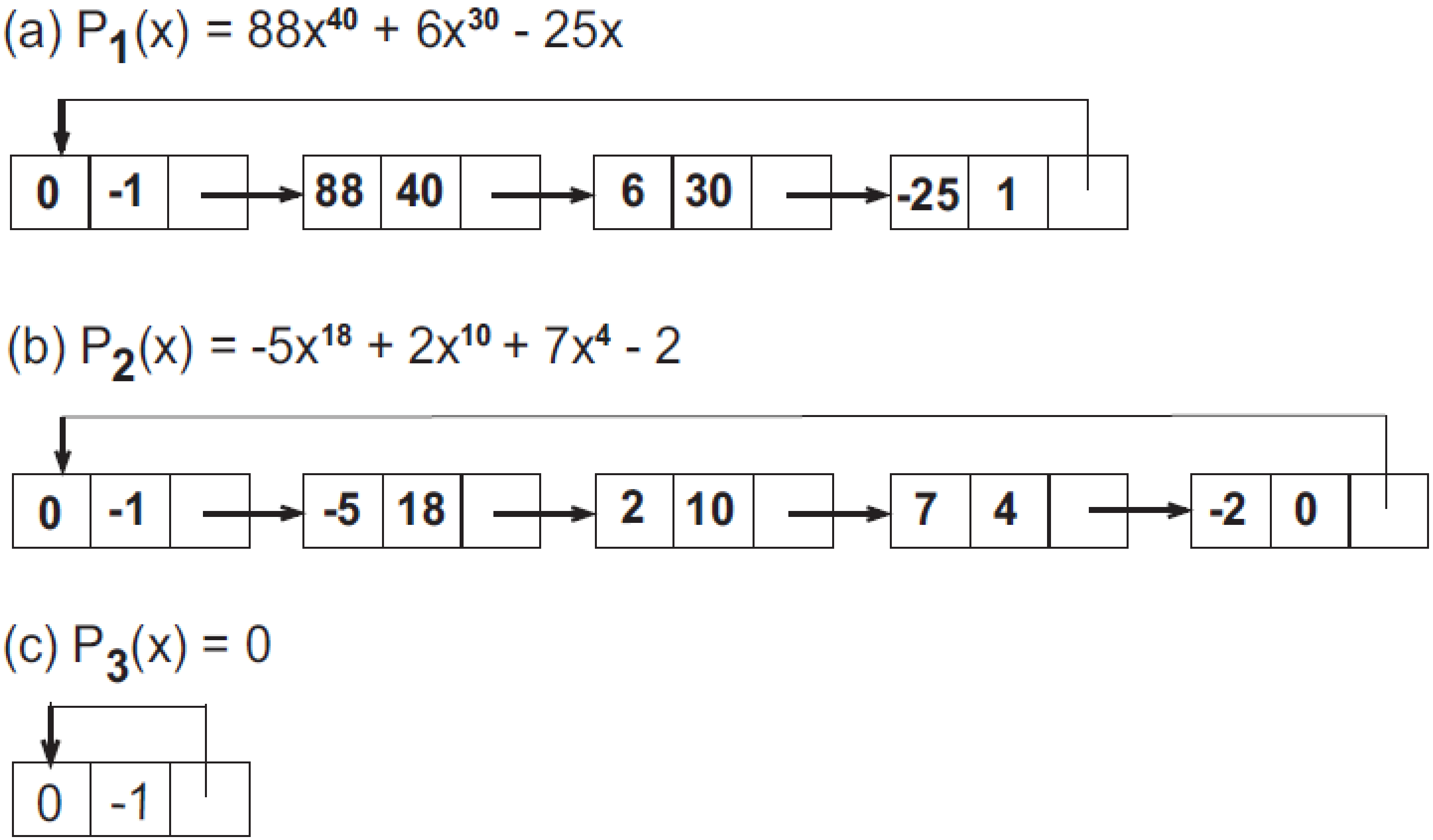
HW #5 (Polynomial Arithmetic)

- Develop in C++ a class **Polynomial**.
- Polynomial should have a private data member belonging to the class **CircularList** which keeps the terms (term consists of coefficient and exponent, both are integers) 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.
- Node class must be hidden from your Polynomial class.

HW #5 (2)

- Note that it is a good programming style to destroy nodes when they are no longer needed.
- 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) $P_3(x) = 0$

HW #5 (3)

- Develop a full class containing proper constructor, copy constructor, and destructor functions as well as *set* and *get* functions.
- Besides, provide a derivative function, which computes the derivative of a polynomial; for example, *p.derivative().derivative()* evaluates the second derivative of a Polynomial *p*.

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

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

$$c_1\ e_1\ c_2\ e_2\ \ldots\ c_m\ e_m\ 0\ \text{-}1$$

HW #5 (4)

where c_i and e_i are integers denoting the coefficient and exponent of the i-th 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 $c_i \neq 0$ for all i.

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.

HW #5 (5)

- The class should also provide the following overloaded operator capabilities:

  1. Overload the addition operator (+) to add two *Polynomials*.
  2. Overload the subtraction operator (-) to subtract two *Polynomials*.
  3. Overload the assignment operator (=) to assign one *Polynomial* to another.
  4. Overload the multiplication operator (*) to multiply two *Polynomials*, or to multiple an integer with a *Polynomial*.

HW #5 (6)

5. Overload the addition assignment operator (+=),
   the subtraction assignment operator (-=),
   and the multiplication assignment operator (*=).
6. Overload the equality operator (==) to
   determine the equality of two *Polynomials.*
7. Overload the inequality operator (!=).

- You may add any other **private** data members (for example, **degree** of the polynomial) or member functions that you think are necessary.
- Write a test program which implements all of the above methods.