

● 다항식 - Representing Polynomials as a SLL

$$A(x) = a_m x^{e_m} + \dots + a_1 x^{e_1},$$

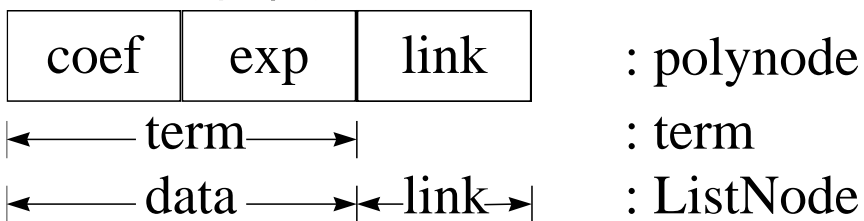
$$e_m > e_{m-1} > \dots > e_2 > e_1 \geq 0,$$

$$a_i \neq 0 (m \leq i \leq 1)$$

a_i : 0 이 아닌 계수,

e_i : 음수가 아닌 정수 지수 ($e_m > e_{m-1} > \dots > e_2 > e_1 \geq 0$)

. 노드의 구성



ex) $a = 3x^{14} + 2x^8 + 1$ $b = 8x^{14} - 3x^{10} + 10x^6$



(a) $3x^{14} + 2x^8 + 1$



(b) $8x^{14} - 3x^{10} + 10x^6$

```

struct Term {          //Term 의 모든 멤버는 묵시적으로 public
    int coef;  //계수
    int exp;   //지수
    Term Set(int c, int e){coef=c; exp=e; return *this;};
};

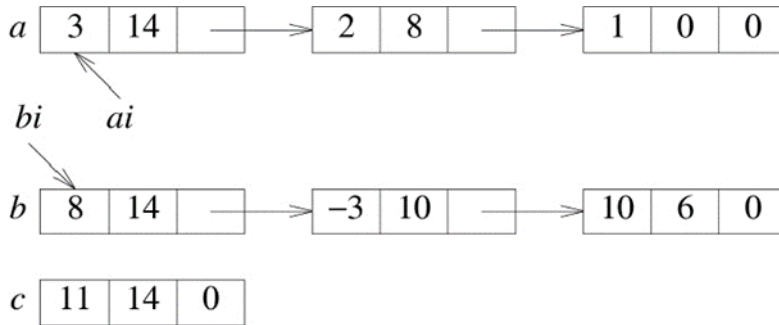
class Polynomial{
public:
    //정의된 공용 함수들
private:
    Chain<Term> poly;
};
    
```

● 덧셈 알고리즘 (Adding Polynomials)

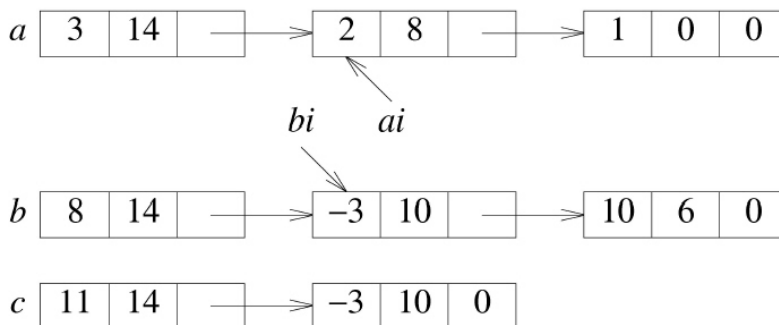
Ex) $C=a+b$ 의 처음 세 항을 생성

$$(a = 3x^{14} + 2x^8 + 1 \quad b = 8x^{14} - 3x^{10} + 10x^6)$$

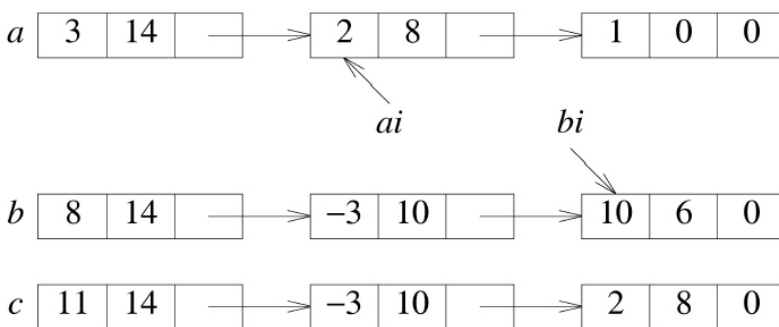
- 두항의 **지수(exp)**가 같으면, **계수(coef)**의 합을 구하고, 다른 지수가 큰항의 다항식부터 결과 다항식 c 에 첨가한다.



(i) $ai \rightarrow exp == bi \rightarrow exp$

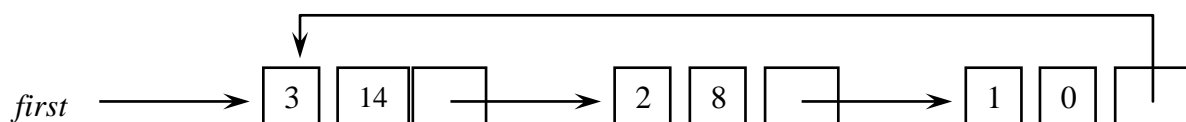


(ii) $ai \rightarrow exp < bi \rightarrow exp$



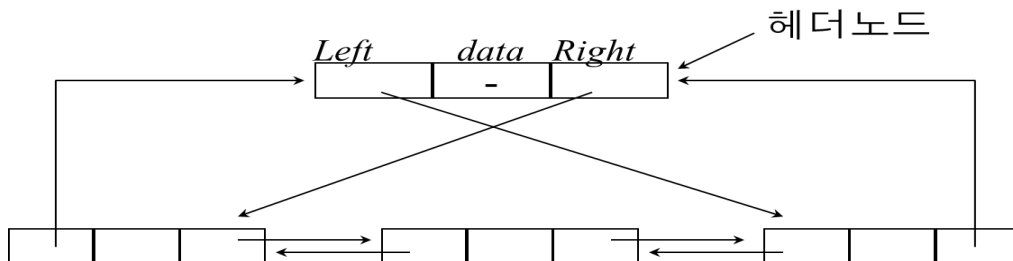
(iii) $ai \rightarrow exp > bi \rightarrow exp$

● 다항식의 원형리스트 표현

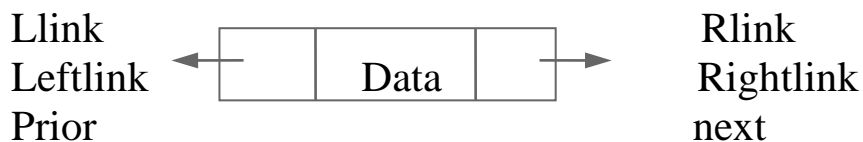


3. Doubly Linked List

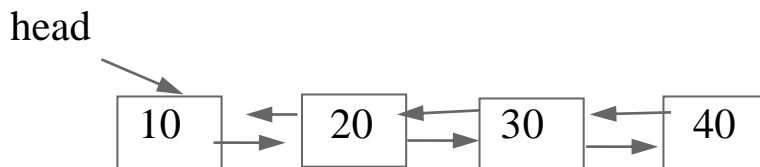
- SLL 의 단점: 특정노드 P 의 이전노드를 찾기 위해서는, 처음부터 전체 list 검색해야한다. => O(n) time
 ⇒ DLL 은 이 문제를 2개의 link 로 해결



■ DLL 의 정의



- 특성: $\text{ptr} = \text{ptr} \rightarrow \text{llink} \rightarrow \text{rlink} = \text{ptr} \rightarrow \text{rlink} \rightarrow \text{llink}$



■ 노드선언(Declaration)

```
class Node {
    private:
        int data;
        char name[10];
        Node *next;
        Node *prev;
        Node (int val, char str[])
            { data = val; strcpy(name, str); next = 0; prev = 0;}
    friend class List;
};
```

```

class List {
    private:
        Node *head;
    public:
        List();
        ~List();
        void insertList(int, char[]);
        void deleteList(int);
        void forwardList();
        void backwardList();
        void searchList(int);
        int  isEmpty();
};

```

변수 및 함수선언부	설명
void insertList(int,char[]); void deleteList(int); void forwardList(); void backwardList(); void searchList(int) int isEmpty();	연결 리스트에 노드를 삽입하는 함수 연결 리스트에 노드를 삭제하는 함수 노드들의 내용을 head 부터 출력 노드들의 내용을 끝 노드부터 출력 연결 리스트에서 데이터를 찾는 함수 연결 리스트가 비었는지의 여부를 검사

● 함수

1) isempty 함수

```

int List::isEmpty()
{
    return (head == 0);
}

```

2) insert 함수

```
void List::insertList(int data, char name[]) //숫자의 경우(오름차순)
```

```
{
```

```
    Node *temp = new Node(data, name);
```

```
    Node *p, *q;
```

```
    if (head == 0)      // 첫노드일때
```

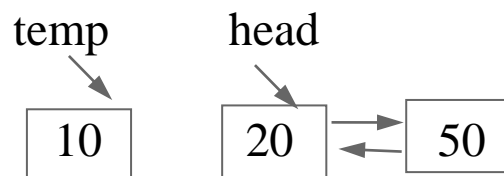
```
        head = temp;
```

```
    else if (temp->data < head->data) { //head node 앞에 삽입
```

```
        temp->next = head;
```

```
        head->prev = temp;
```

```
        head = temp; }
```



```
    else {                // 가운데 삽입
```

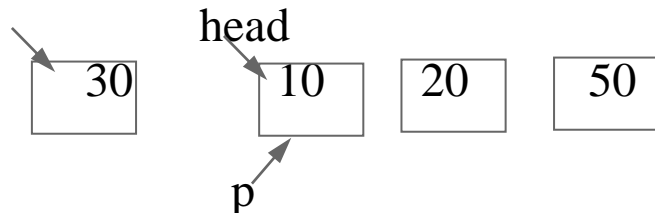
```
        p = head;    q = head;
```

```
        while ((p != 0) && (p->data < temp->data)) { //이동
```

```
            q = p;
```

```
            p = p->next;
```

```
        }
```



```
    if (p != 0) { // 중간에 삽입
```

```
        temp->next = p;
```

```
        temp->prev = q;
```

```
        q->next = temp;
```

```
        p->prev = temp;
```

```
    }
```

```
    else { // temp 가 큰 경우
```

```
        q->next = temp;
```

```
        temp->prev = q;
```

```
    }
```

```
}
```

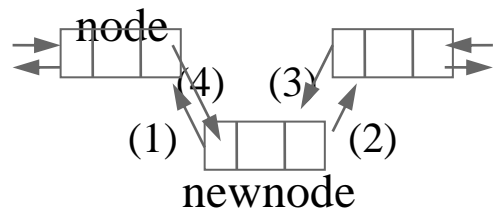
```
}
```

- Insert (after)

```

void Dinsert_after(ptr c_node, ptr head)
{
    get_newnode(p);
    p->data = newdata;
    if (head == NULL) {
        head = p;    p->Rlink=p;  p->Llink=p;  pos = 1}
    else
    {
        p ->Llink = c_node; (1)
        p->Rlink = c_node->Rlink; (2)
        c_node->Rlink->Llink = p; (3)
        c_node->Rlink = p;  (4)
    }
    c_node = p;
    size = size+1;
}

```

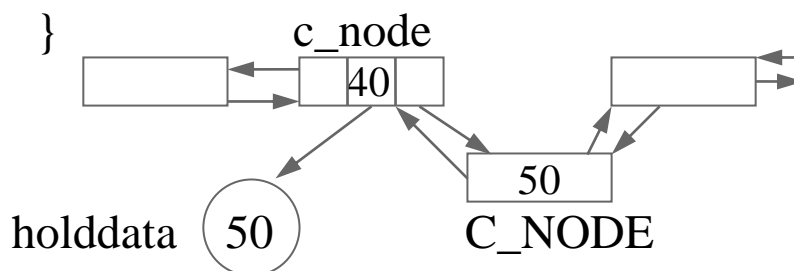


- Insert (before)

```

void Dinsert_before ( ..... )
{ if (head == NULL)  Dinsert_after(newdata);
  else
  {   holddata = c_node->data;
      c_node->data = newdata;
      Dinsert_after(holddata);
      c_node = c_node->Llink;
      pos = pos -1;
  }
}

```



3) Delete 함수

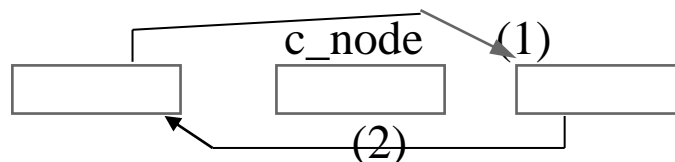
```
void List::deleteList(int key)
{   Node *p, *q;

    if (head->data == key) { // 삭제될 노드가 head 일 경우
        p = head;    head = head->next;    head->prev = 0;
        delete p;
    }
    else {                // 가운데 노드가 삭제될 경우
        q = head;    p = head;
        while (p != 0 && p->data != key) {
            q = p;    p = p->next;
        }
        if (p != 0) {
            q->next = p->next;
            if (p->next != 0)    p->next->prev = q;    //
            delete p;
        }
        else
            cout << key << " is not in the list\n";
    } }
```

- Ex

c_node->Llink->Rlink = c_node->Rlink; (1)

c_node->Rlink->Llink = c_node->Llink; (2)



```
p = c_node;
c_node = c_node->Rlink;
size = size-1;
delete p;
```

4) forward 함수

```
void List::forwardList()
{
    if (!isEmpty()) {
        Node *p = head;
        cout << "----- Forward List -----\\n";
        while (p!= 0) {
            cout << p->data << p->name << endl;
            p = p->next;    //move right
        }
    }
    else
        cout << "List is empty!\\n";
}
```

5) backward 함수

```
void List::backwardList()
{
    if (!isEmpty()) {
        Node *p = head;
        while (p->next != 0)    // find the node
            p = p->next;
        cout << "----- Backward List -----\\n";
        while (p!= 0) {
            cout << p->data << p->name << endl;
            p = p->prev;    // move left
        }
    }
    else
        cout << "List is empty!\\n";
}
```


6) search 함수

```
void List::searchList(int key)
{
    if (!isEmpty()) {
        Node *p = head;
        while (p != 0 && p->data != key)
            p = p->next;
        if (p != 0)
            cout << p->data << " is in the list\n";
        else
            cout << key << " is not in the list\n";
    }
    else
        cout << "List is empty!\n";
}
```

7) List::~~List()

```
List::~~List()
{
    Node *p;

    while (head != 0) {
        p = head;
        head = head->next;
        delete p;
    }
}
```

4. Generalized List : 일반리스트(범용리스트)

- . 선형 리스트 $A = (\alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_n), n \geq 0$,
- . 일반리스트는 α_i 가 (원자, 리스트) 일수 있기 때문에, 다차원의 구조를 가질 수 있다.

[정의: 일반리스트 A 는 원자 또는 list 원소들의 유한순차 $\alpha_i, \dots, \alpha_n (n \geq 0)$ 이다.

원소 $(\alpha_i (1 \leq i \leq n))$ 가 list 일때 이를 A 의 sublist 라 한다.)]

* 표현

- 리스트 A 자체는 $A = (\alpha_0, \dots, \alpha_{n-1})$ 라고 표기
- A 는 리스트 이름, n 은 리스트의 길이
- 모든 리스트의 이름은 대문자로 표기, 소문자는 원자를 표현
- $n \geq 1$ 일 때, α_0 는 A 의 head, $(\alpha_1, \dots, \alpha_{n-1})$ A 의 tail

ex) $D = ()$: NULL/empty list, $n = 0$

$A = (a, (b, c))$: $n = 2, \alpha_1 = a, \alpha_2 = (b, c), \text{Head}(A) = a, \text{Tail}(A) = (b, c)$

$B = (A, A, ())$: $n = 3, \alpha_1 = A, \alpha_2 = A, \alpha_3 = \text{NULL},$
 $\text{Head}(B) = A, \text{Tail}(B) = (A, ())$

$C = (a, C)$: $n = 2, C = (a, (a, (a, \dots)))$ 무한리스트

*범용리스트 노드 구조 1):

```
enum Boolean { FALSE, TRUE };
class GenList; // forward declaration
class GenListNode {
friend class GenList;
private:
    GenListNode *link;
    Boolean tag;
    union {
        char data;
        GenListNode *dlink;
    };
};
```

tag=false/true	data/down	next
----------------	-----------	------

```
class Genlist{
public:
    // 리스트 연산조작
private:
    GenlistNode *first;
};
```


Ex) $A=(4,6)$ A:


0	4	
---	---	--

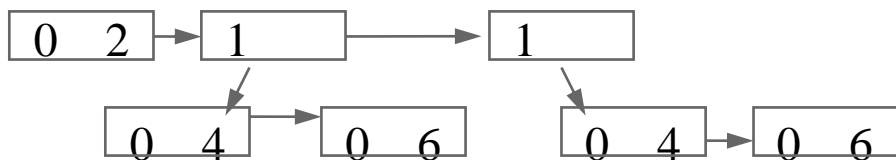
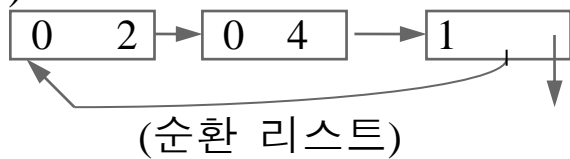
 \longrightarrow

0	6	
---	---	--

 \longrightarrow

$B = ((4, 6), 8)$ $B :$ 

$C = (((4)), 6)$ C: 

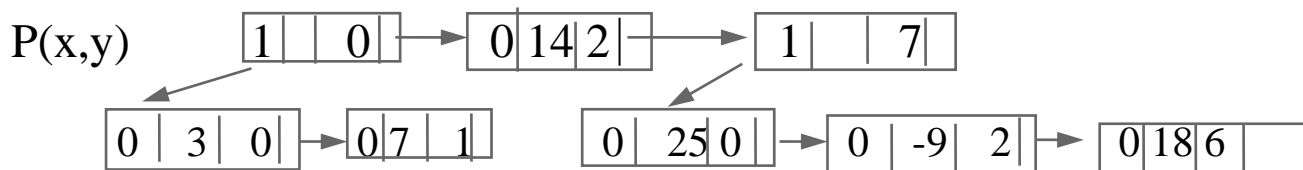
$$D = (2, A, A) \Rightarrow (2, (4,6), (4,6))$$

$$E = (2, 4, E)$$


● < 범용 리스트 이용한 다항식표현 >

tag=false/true	data/down	next
----------------	-----------	------

\Rightarrow	Tag(T/F)	Coef/dlink	exp	link
---------------	----------	------------	------------	------

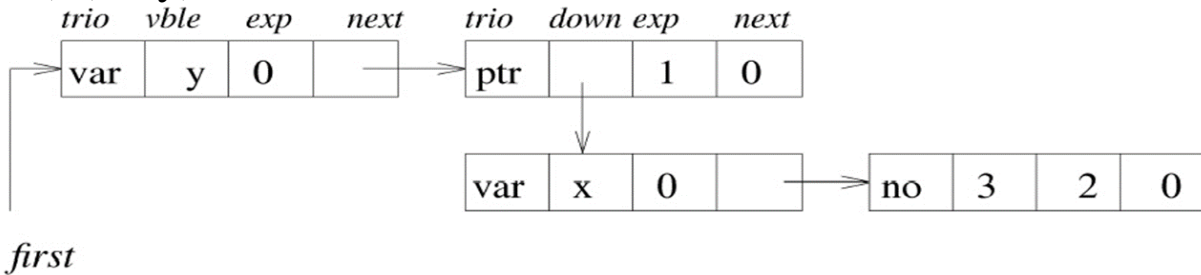
$$\begin{aligned} \text{ex) } P(x,y) &= 3+7x+14y^2 + 25y^7 - 9x^2y^7 + 18x^6y^7 \\ &= (3+7x) + 14y^2 + (25 - 9x^2 + 18x^6)y^7 \end{aligned}$$



◆PolyNode 타입의 노드 정의

```
enum Triple { var, ptr, no };
class PolyNode {
    PolyNode *link;
    int exp;
    Triple trio;
    union {
        char vble;
        PolyNode *dlink;
        int coef;
    };
};
```

Ex) $(3x^2y)$ 의 표현



ex) $P(x,y,z) = x^{10}y^3z^2 + 2x^8y^3z^2 + 3x^8y^2z^2 + x^3y^4z + 6x^3y^4z + 2yz$

⇒ 순차적으로 표현불가능, 범용리스트에서는

$$((x^{10}+2x^8)y^3 + 3x^8y^2)z^2 + ((x^4+6x^3)y^4 + 2y)z$$

$$* Cz^2 + Dz$$

$$* C(x,y) = Ey^3 + Fy^2 \quad D(x,y) = Gy^4 + 2y$$

