لیست پیوندی-۳

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

دانشگاه کاشان- دانشکده مهندسی برق و کامپیوتر

لیستهای پیوندی

- اشاره گرها
- لیست ها
- لیست های دایره ای
- پشته ها و صفهای پیوندی
- چند جمله ای ها
- روابط هم ارزی
- لیستهای دو پیوندی و لیست های تعمیم یافته

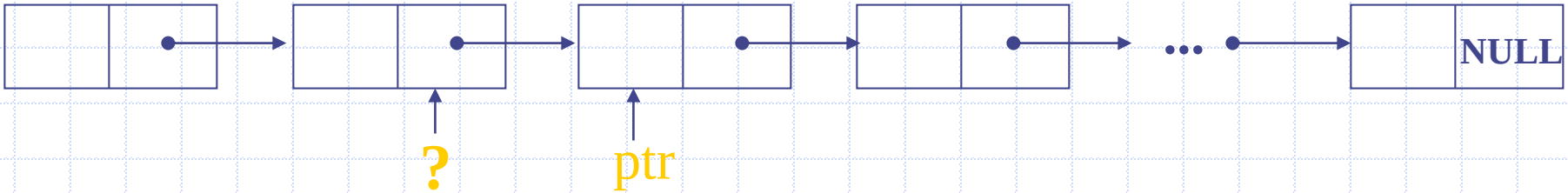
جمع چند جمله ای ها

- فرض کنیم که a و b اشاره گرهایی به ابتدای چند جمله ای ها باشند.
  - اگر توان دو چند جمله ای با هم برابر باشد ، ضرایب با هم جمع می شوند و گره جدیدی تشکیل می شود ، همچنین اشاره گرها را به گره های بعدی در a و b حرکت می دهیم.
  - اگر توان چند جمله ای a کمتر از توان متناظر در چند جمله ای b باشد، آنگاه یک جمله مشابه این جمله ایجاد و آنرا به نتیجه ، یعنی d ، اضافه می کنیم و اشاره گر را به جمله بعدی در b منتقل می کنیم.
  - اگر a-> expon >b- > expon باشد عملی مشابه را بر روی a انجام می دهیم .

لیست های دو پیوندی

- لیست های تک پیوندی بعضی از مشکلات را ایجاد می کند زیرا فقط می توانیم در جهت پیوندها حرکت کنیم.

مثال تنها راه یافتن گره ماقبل ptr پیمایش از ابتدای لیست می باشد. برای حذف یک گره دلخواه باید آدرس گره قبل را بدانیم.



- اگر نیازمند پیمایش لیست از هر دو جهت باشیم بهتر است از لیست دو پیوندی استفاده کنیم.

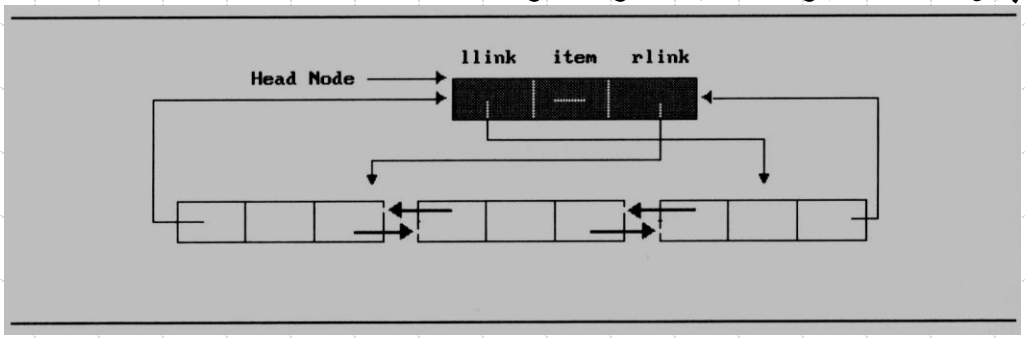
لیست دو پیوندی

- یک گره در یک لیست دو پیوندی حداقل سه فیلد داده ای دارد:
  - فیلد data
  - فیلد llink (اشاره گر به چپ)
  - فیلد rlink (اشاره گر به راست)
- لیست دوپیوندی می تواند دایره ای باشد یا دایره ای نباشد. می تواند دارای گره س باشد یا نباشد.

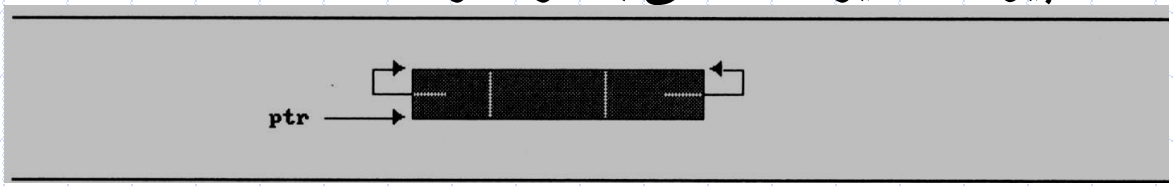
```
typedef struct node *node_pointer;
typedef struct node{
   node_pointer llink;
   element item;
   node_pointer rlink;
};
```

لیست دو پیوندی

- مثال
  - لیست دوپیوندی دایره ای با گره سر



  - لیست دوپیوندی دایره ای خالی با گره سر

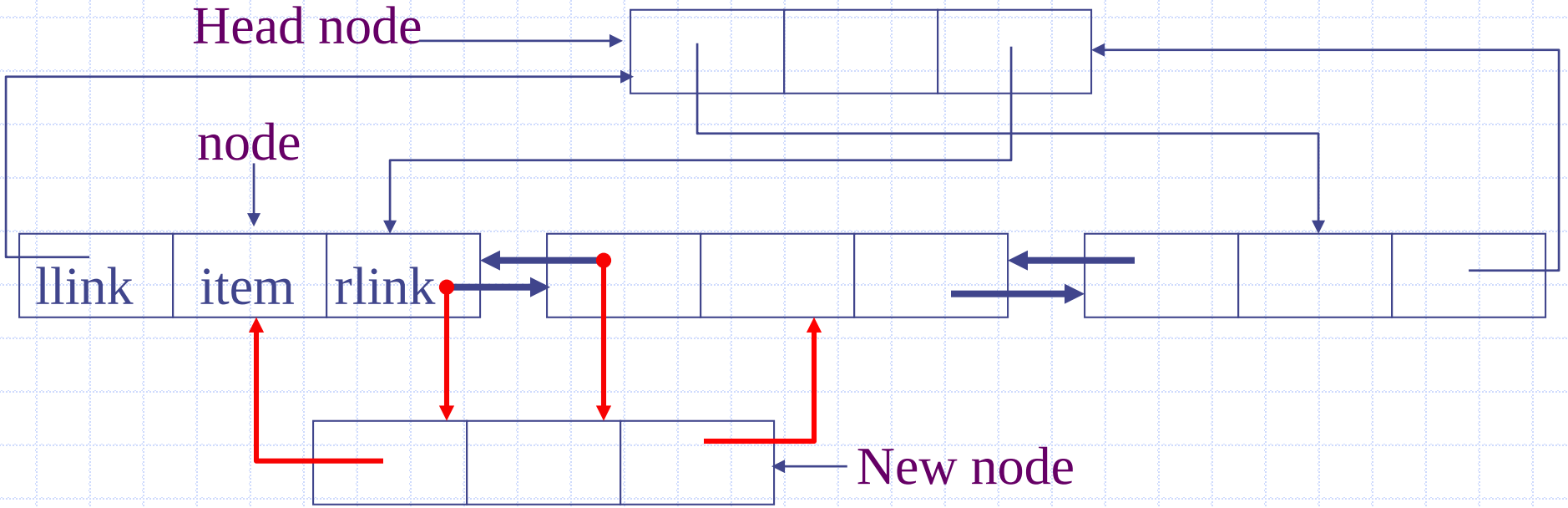


  - فرض کنید که ptr به گره ای دلخواه در لیست دوپیوندی اشاره کند آنگاه
    - ptr = ptr -> llink -> rlink = ptr -> rlink -> llink

لیست دو پیوندی

- اضافه کردن گره در یک لیست دایره ای دوپیوندی

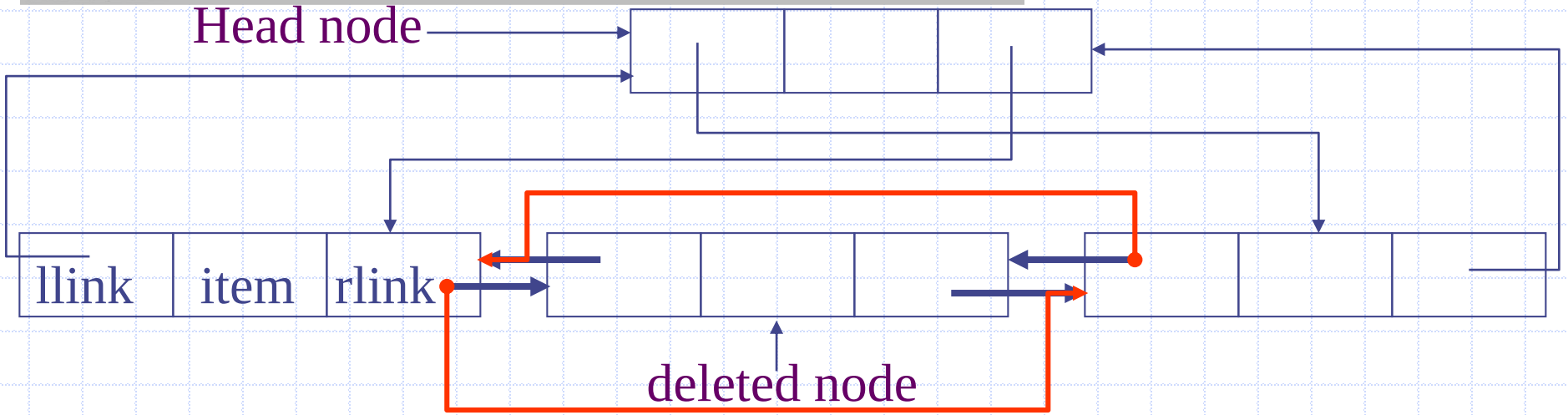
```
void dinsert(node_pointer node, node_pointer newnode)
{
/* insert newnode to the right of node */
   newnode->llink = node;
   newnode->rlink = node->rlink;
   node->rlink->llink = newnode;
   node->rlink = newnode;
}
```



لیست دو پیوندی

- حذف کردن گره در یک لیست دایره ای دوپیوندی

```
void ddelete(node_pointer node, node_pointer deleted)
{
/* delete from the doubly linked list */
   if (node == deleted)
      printf("Deletion of head node not permitted.\n");
   else {
      deleted->llink->rlink = deleted->rlink;
      deleted->rlink->llink = deleted->llink;
      free(deleted);
   }
}
```



لیست های تعمیم یافته

- تعریف: لیست تعمیم یافته دنباله ای متناهی از n عضو $a_0,\ldots,a_{n-1}$ است در آن a_iها یک عضو ساده یا لیست است. به عضو هایی که ساده نیستند زیر لیست گویند.
- نام تمام لیست ها با حروف بزرگ نمایش داده می شود و از حروف کوچک برای نمایش زیر لیست ها استفاده می شود.
- اگر n>=1 باشد آنگاه a_0 ابتدا یا اول لیست (head) و $a_1,\ldots,a_{n-1}$ انتها یا آخر لیست (tail) نام دارد.

لیست های تعمیم یافته

D = ( )	the null or empty list, its length is zero.
A = (a, (b,c))	a list of length two; its first element is the atom 'a' and its second element is the linear list (b,c).
B = (A,A,( ))	a list of length three whose first two elements are the lists A, the third element the null list.
C = (a, C)	a recursive list of length two. C corresponds to the infinite list C = (a,(a,(a,

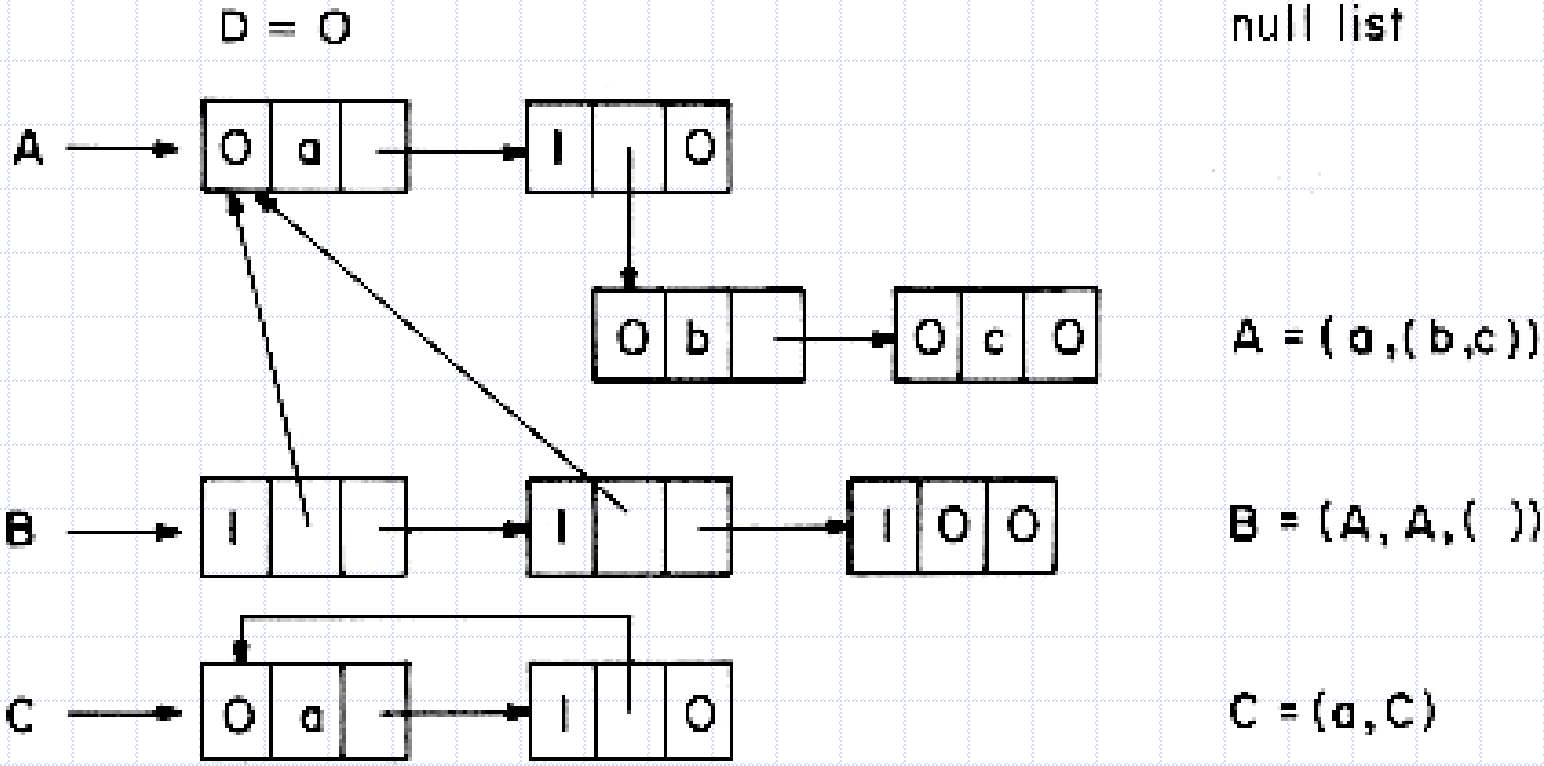
head (A) = 'a', tail (A) = ((b,c)).
head (B)) = A, tail (B)) = (A, ( ))
head (tail(B)) = A, tail (tail(B)) = (( ))

ساختار گره یک لیست تعمیم یافته

یک لیست تعمیم یافته با یک اشاره گر
GenListNode * first

```
enum  Boolean {false, true};
Struct GenListNode {
    GenListNode *link;
    Boolean tag;
    union {
    char data;
    GenListNode *dlink;
    };
};
```

نمایش لیست های عمومی



نمونه کاربرد لیست های عمومی

- فرض کنید می خواهیم چند جمله ایهای چند متغیره مانند مثال زیر را بازنمایی کنیم

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

COEF	EXPX	EXPY
EXPZ	LINK	

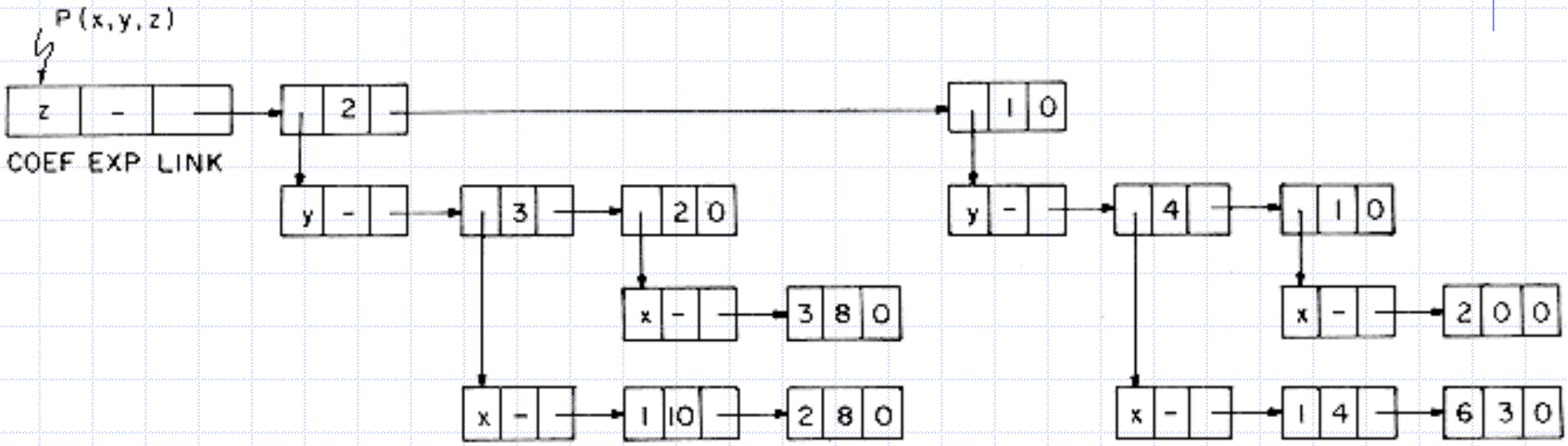
- راهکار: استفاده از ساختاری مانند
- عیب : چند جمله ایهای با چند متغیر مختلف نیازمند تعداد مختلفی فیلد است که باعث بروز مشکلات متعددی در رابطه با نمایش ترتیبی چن جمله ایها می شود

نمونه کاربرد لیست هاي عمومي

$P(x,y,z)=$

$=x^{10}\,y^3\,z^2 + 2x^8\,y^3\,z^2 + 3x^8\,y^2\,z^2 + x^4\,y^4\,z + 6x^3\,y^4\,z + 2yz$

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



نمونه کاربرد لیست های عمومی

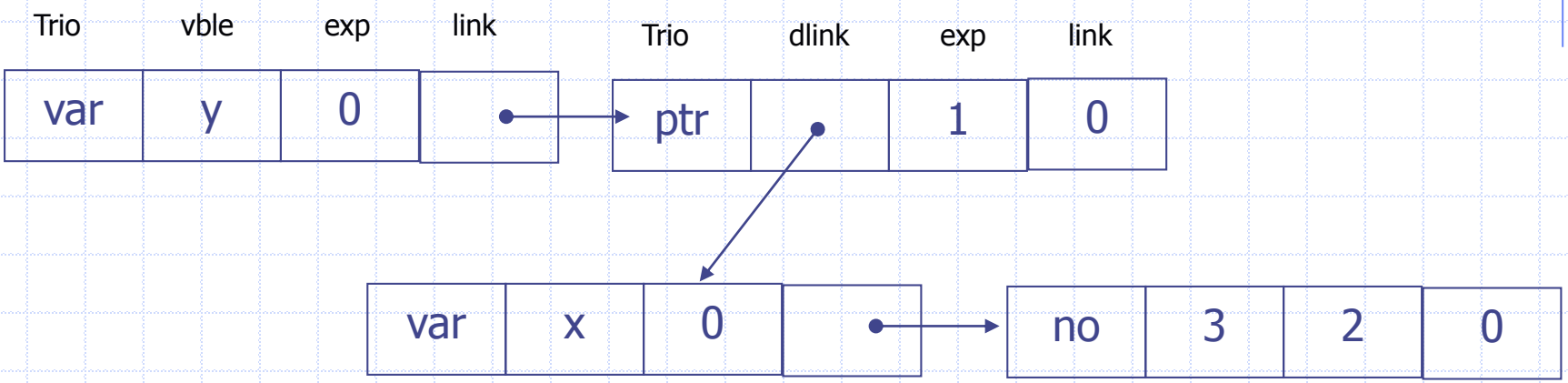
- ساختار نود برای چند جمله ای

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

نمونه کاربرد لیست های عمومی

مثال

$P = 3x^2y$



الگوریتم بازگشتی برای لیست ها

کپی یک لیست غیر بازگشتی که در آن هیچ زیر لیست مشترکی وجود ندارد.

```
GenListNode *copy (GenListNode *p)
{
/* copy the nonrecursive list with no shared sublists pointed at by p */
GenListNode *q=0;
If (p) {
        q=new GenListNode;
        q->tag=p->tag;
        If (!p->tag ) q->data=p->data;
        else q->dlink= copy( p->dlink);
        q->link=copy(p->link);
        }
return q;
}
```

زمان اجرای الگوریتم o(m) یا خطی است .

یک کران بالا برای حداقل عمق بازگشتی یا معادل آن تعداد مکان های مورد نیاز پشته ی بازگشتی m یعنی تعداد کل گره ها است که برای A=((((((a)))))) قابل دستیابی است

الگوریتم بازگشتی برای لیست ها

برابری دو لیست: لیست ها باید ساختار یکسان داشته و عضوهای داده ای متناظرشان نیز باید داده های یکسانی داشته باشند

```
int  equal (GenListNode *s, GenListNode *t)
{
int  x;
if ( !s )  && ( !t )  return 1;
if ( s  && t  && ( s->tag==t->tag) )
{
        if ( !s->tag)
                if (s->data == t->data ) x=1; else x=0;
        else  x=equal (s->dlink, t->dlink);
        if (x) return equal (s->link , t->link);
}
return 0;
}
```

زمان اجرای الگوریتم خطی است .

الگوریتم بازگشتی برای لیست ها

عمق یک لیست:

عمق لیست خالی برابر صفر و در حالت کلی

$$depth(S) = \begin{cases} 0, \text{ if } S \text{ is an atom} \\ 1 + \max\{depth(x_1), \ldots, depth(x_n)\}, \text{ if } S \text{ is the list} \\ (x_1, \ldots, x_n),\ n \geq 1. \end{cases}$$

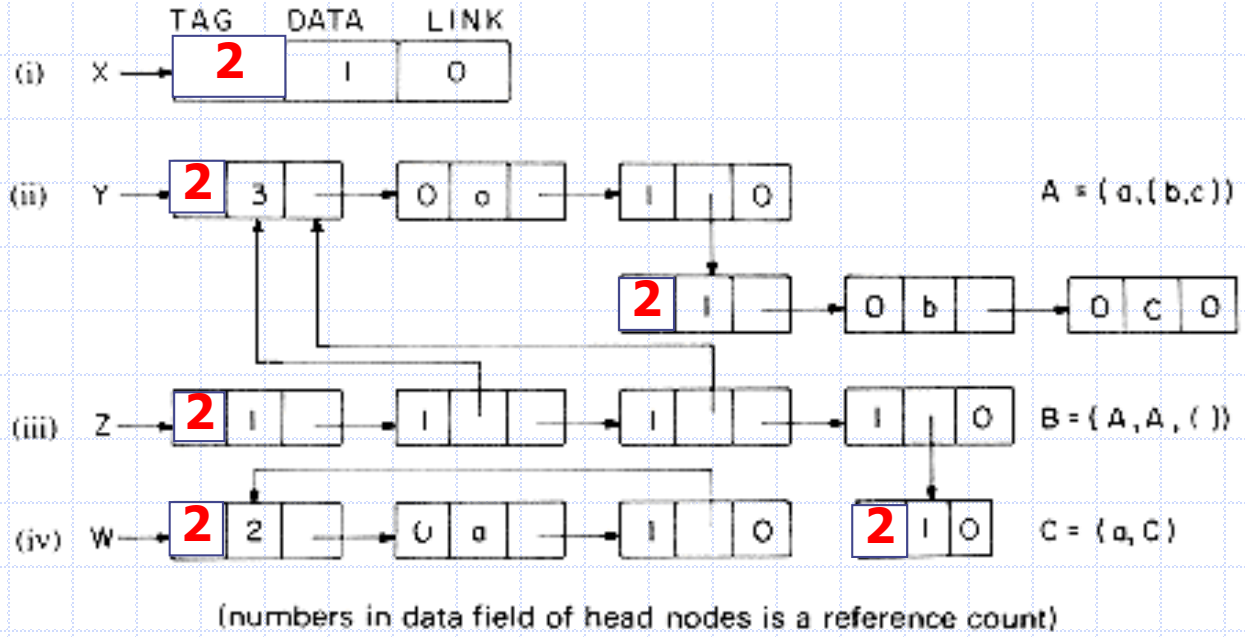
الگوریتم بازگشتی برای لیست ها

عمق لیست

```
int  depth(GenListNode *s)
{
if ( !s ) return 0;
GenListNode *p=s; int m=0;
while (p)  {
            if (p->tag) {
                        int n= depth( p->dlink);
                        if (m<n) m=n;
            }
            p=p->link;
}
return m+1;
}
```

تعداد دفعات ارجاع به لیست

- در لیست هایی که لیست مشترک دارند هرگاه بخواهیم گره ای را در اول لیست اضافه یا از آن حذف کنیم مشکلاتی را در کار ایجاد می کند
- علاوه بر ان می توان از فیلد data/dlink گره های سر یرای نگهداری تعداد دفعات ارجاع به لیست مربوطه استفاده کرد.



در این ساختار می توان فیلد tag را اصلاح کرد تا ۳ مقدار ممکن 0، 1 و 2 را اختیار کند.

0= data

1= dlink

2= ref

حذف یک لیست به صورت بازگشتی

- تنها یک واحد از تعداد دفعات ارجاع کم می شود. تنها اگر تعداد دفعات ارجاع برابر صفر باشد گره های لیست به صورت فیزیکی به حافظه بر گردانده می شود.

```
void delete(GenListNode *x)
{
X->ref - -;
if  (!x ->ref )
    {
    GenListNode *y =x;     //traverse top level of x;
    while (y-> link) { y=y-> link; if (y->tag ==1) delete (y-> dlink) ; }
    y-> link =av; //attach top- llevel nodes to av list
    av=x;
    }
}
```

حذف یک لیست به صورت بازگشتی

- در لیست های بازگشتی تعداد دفعات ارجاع هیچگاه صفر نمی شود.
- این مطلب در مورد بازگشتی غیر مستقیم هم صادق است

مثال حذف R و سپس S در شکل زیر