ساختمان داده‌ها

فصل چهارم

لیست های پیوندی

E-mail: Hadi.khademi@gmail.com

- 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

لیستهای پیوندی

E-mail: Hadi.khademi@gmail.com

- **لیست مرتب زیر را در نظر بگیرید**
- **(bat, cat, sat, vat)**

برای اضافه کردن کلمه mat

باید کلمات sat و vat یک مکان به راست شیفت داده شوند.

برای حذف کردن کلمه cat

باید کلمات sat و vat یک مکان به چپ شیفت داده شوند.

- **مشکلات بازنمایی ترتیبی**

۱- حذف و درج عناصر در آرایه ها بسیار وقت گیر است

۲- باذخیره کردن هر لیست در آرایه ای با حداکثر اندازه ، حافظه هدر می رود

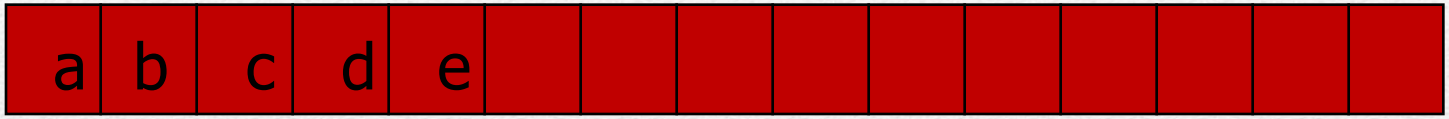
مقدمه

- **راه حل مناسب : استفاده از بازنمایی پیوندی**
  - عناصر می توانند در هر جای حافظه قرار گیرند.
  - در بازنمایی ترتیبی، ترتیب اعضای لیست با ترتیب نگهداری اعضا در حافظه یکسان است ولی در بازنمایی پیوندی لازم نیست ترتیب اعضای لیست با ترتیب نگهداری اعضا یکسان باشد.
  - برای دستیابی صحیح به عناصر یک لیست ، بایستی به همراه هر عنصر، آدرس یا موقعیت عنصر بعدی نیز ذخیره شود.
  - بنابراین برای هر عنصر لیست، یک نود وجود دارد که حاوی فیلدهای داده ای و اشاره گری به عنصر بعدی در لیست می باشد.

بازنمایی پیوندی

E-mail: Hadi.khademi@gmail.com

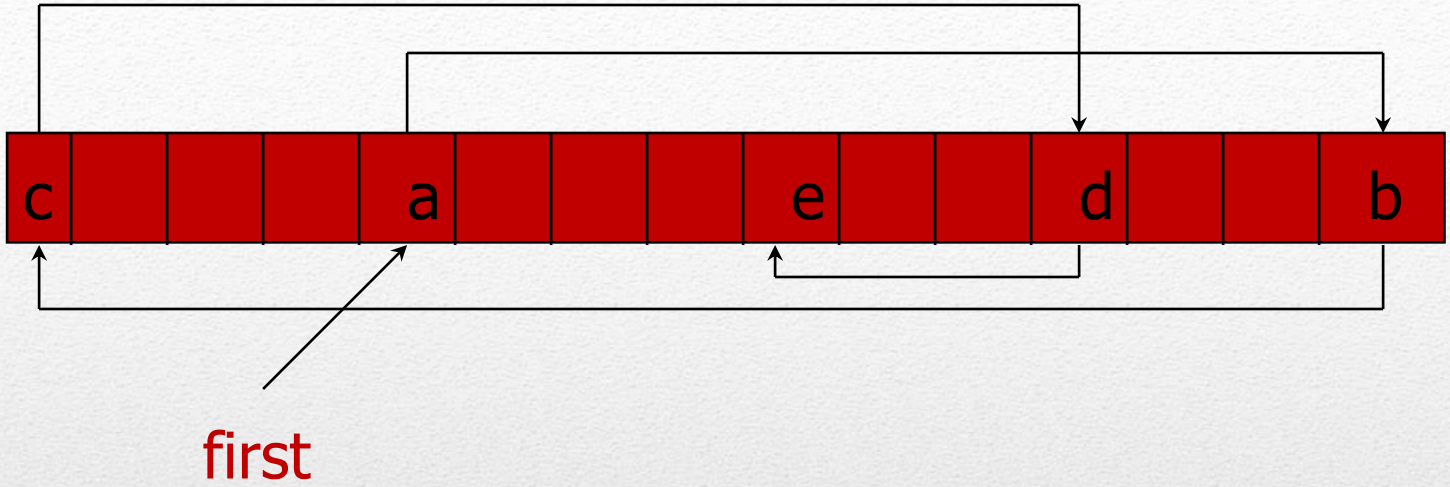
- **شمای حافظه برای لیست L = (a,b,c,d,e)**
  - بازنمایی ترتیبی (آرایه)



  - بازنمایی پیوندی

شمای حافظه

E-mail: Hadi.khademi@gmail.com



- اشاره گر در e برابر NULL است.
- از متغیر first برای دسترسی به عنصر اول استفاده می شود.

بازنمایی پیوندی

E-mail: Hadi.khademi@gmail.com

اشاره گرها

- **C به صورت مناسبی از اشاره گرها حمایت می کند.**
- دو عملگری که با اشاره گرها به کار می روند:

& **the address operator**

***** **the dereferencing (or indirection) operator**

- **مثال**
- اگر تعریف زیر را داشته باشیم

```
int i, *pi;
```

- آنگاه iیک متغیر صحیح و pi یک اشاره گر به یک متغیر صحیح است.

```
pi = &i;
```

- آدرس i را برگردانده و به عنوان مقدار pi نسبت می دهد. برای مقدار دهی به i

```
i = 10; or  *pi = 10;
```

E-mail: Hadi.khademi@gmail.com

اشاره گرها

- هر زمان که نیاز به حافظه جدیدی باشد می توان تابعی به نام malloc را فراخوانی و مقدار فضای لازم را درخواست کرد. اگر حافظه لازم وجود داشته باشد ، اشاره گری به ابتدای ناحیه حافظه مورد نیاز برگردانده می شود.

- زمانی که دیگر نیازی به آن حافظه نباشد ، می توان آن را با فراخوانی تابعی به نام free ، آزاد نمود.

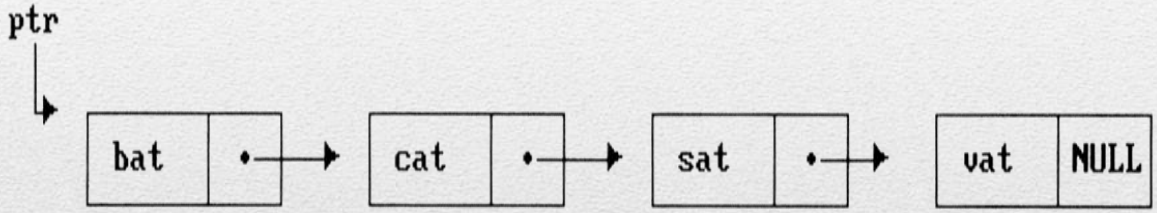
```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *) malloc(sizeof(float));
*pi = 1024;
*pf = 3.14;
printf("an integer = %d, a float = %f\n", *pi, *pf);
free(pi);
free(pf);
```

Request memory

Free memory

E-mail: Hadi.khademi@gmail.com

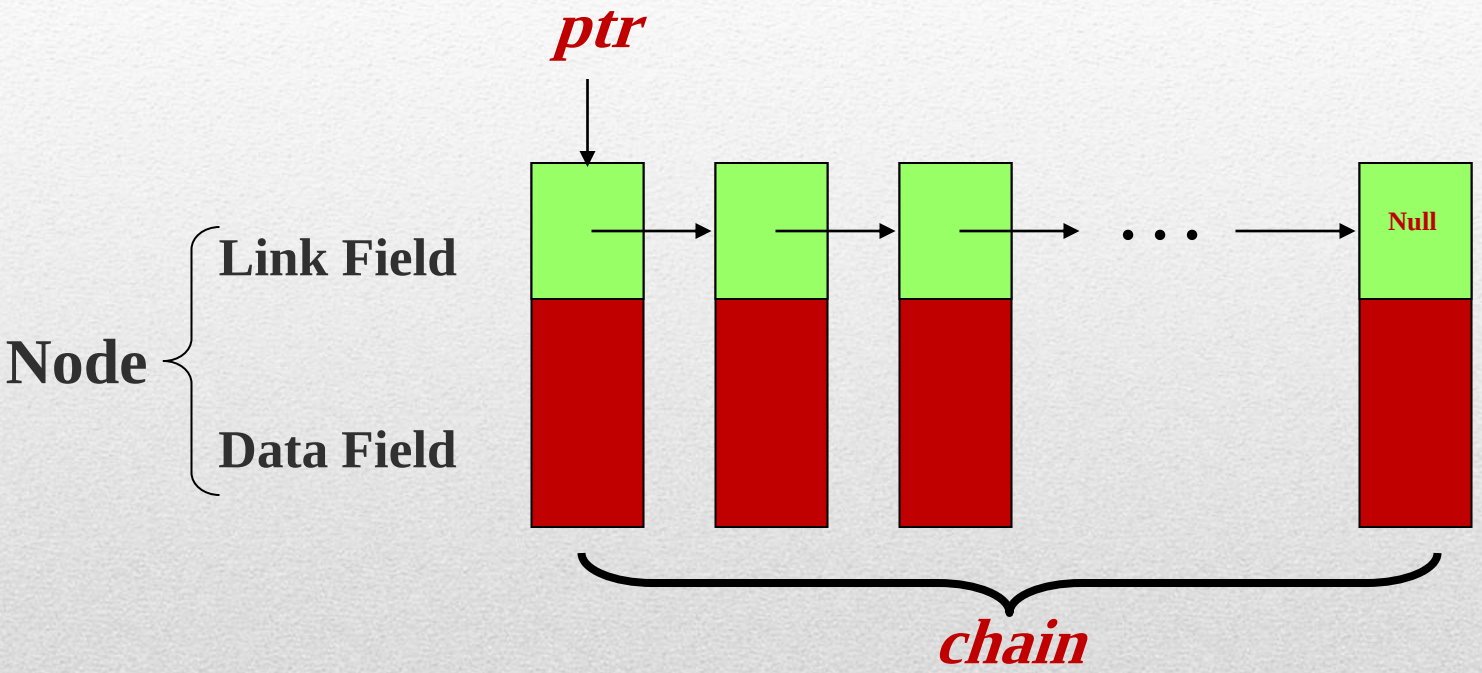
- لیست های پیوندی معمولا به وسیله گره هایی متوالی با اتصالاتی که به صورت فلش هایی نشان داده شده اند ارایه می گردند.



لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- گره ها واقعا در مکانهای پشت سر هم حافظه قرار نمی گیرند
- موقعیت گره ها در اجراهای مختلف می تواند تغییر کند.



چرا با استفاده از بازنمایی پیوندی حذف و اضافه ساده تر است؟

لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

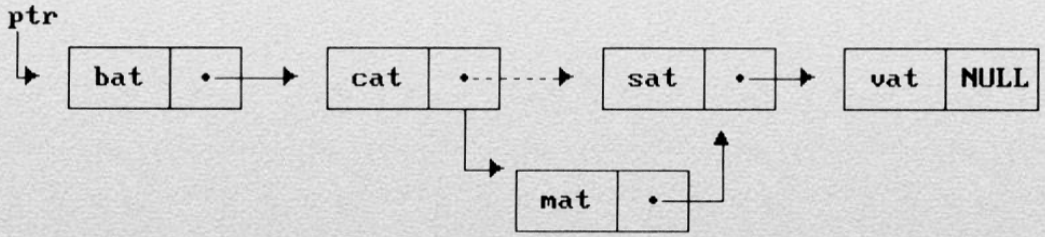
- **برای اضافه کردن کلمه mat بین cat و sat :**

۱- گره ی استفاده نشده ای را در نظر گرفته ، فرض کنید که آدرس آن paddr باشد.

۲- فیلد داده این گره را برابر با mat قرار دهید

۳- فیلد اتصال paddr را طوری تنظیم کنید که به ادرسی که در فیلد اتصال گره حاوی cat می باشد، اشاره کند

۴- فیلد اتصال گره حاوی cat را طوری تنظیم کنید که به paddr اشاره کند.



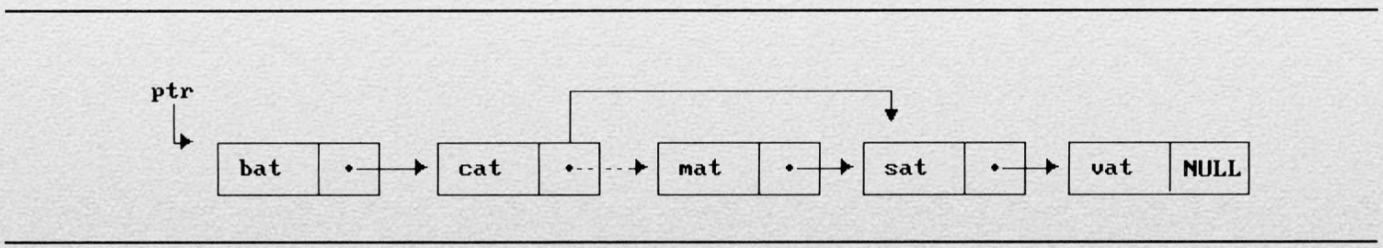
لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- **حذف mat از لیست**

۱- برای انجام این کار فقط لازم است که عنصر قبل از mat یعنی cat را پیدا و فیلد اتصال آنرا طوری تنظیم کنیم که به گره ای اشاره کند که در حال حاضر اتصال گره mat به ان اشاره دارد

۲- ما هیچ داده ای را جابجا نکرده ایم و با وجود آنکه فیلد اتصال mat هنوز به sat اشاره می کند matدیگر عضو لیست نیست.



لیست های تک پیوندی

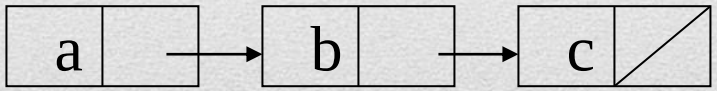
E-mail: Hadi.khademi@gmail.com

- یک یا چند فیلد ساختار اشاره گر به همین ساختار هستند

```
typedef struct list {
        char data;
        list *link;
        }
```

Construct a list with three nodes
item1.link=&item2;
item2.link=&item3;
malloc: obtain a node (memory)
free: release memory

```
list item1, item2, item3;
item1.data='a';
item2.data='b';
item3.data='c';
item1.link=item2.link=item3.link=NULL;
```

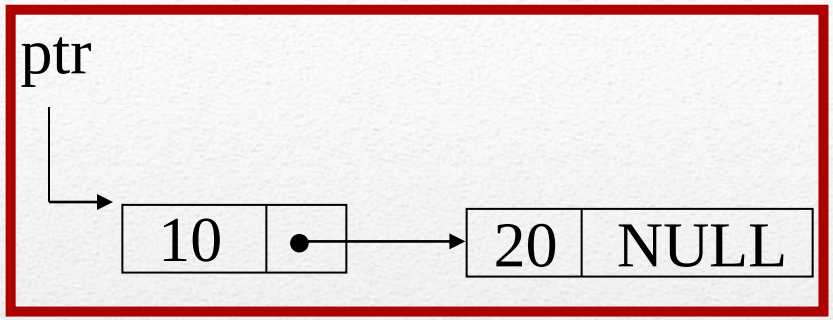


لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

Example [*Two-node linked list*]:

```
typedef struct list_node *list_pointer;
typedef struct list_node {
        int data;
        list_pointer link;
};
list_pointer ptr =NULL;
```



Program : Create a two-node list

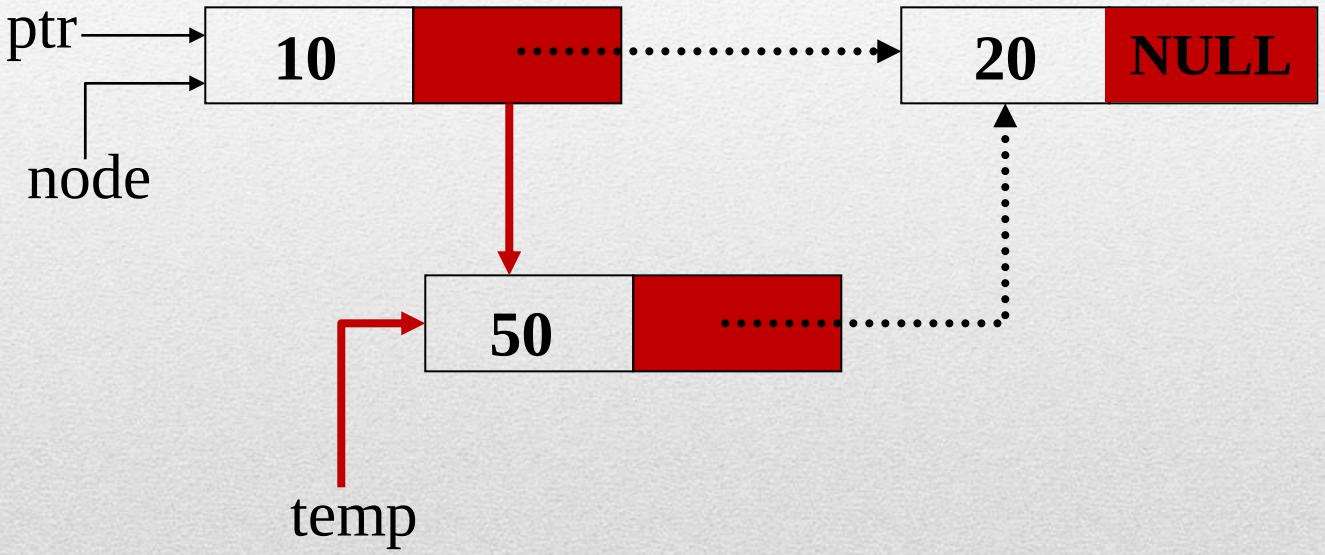
```
list_pointer create2( )
{
        /* create a linked list with two nodes */
        list_pointer first, second;
        first = (list_pointer) malloc(sizeof(list_node));
        second = (list_pointer) malloc(sizeof(list_node));
        second -> link = NULL;
        second -> data = 20;
        first -> data = 10;
        first ->link = second;
        return first;
}
```

لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- **اضافه کردن**

یک نود با داده ۵۰ به لیست ptr و بعد از node اضافه کنید.

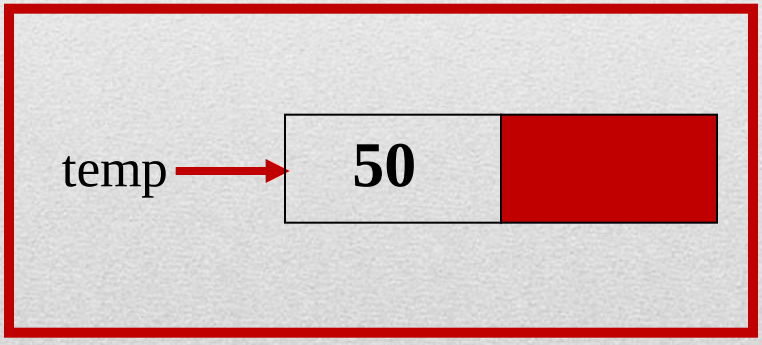


لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

Implement Insertion:

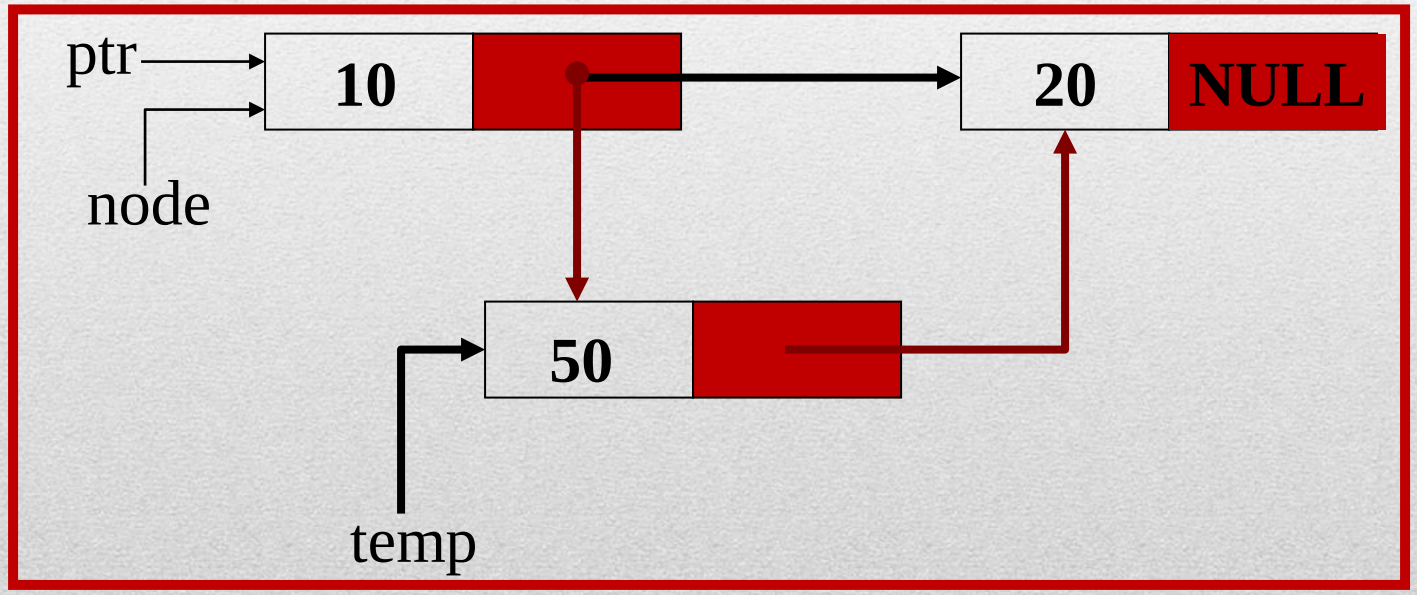
```
void insert(list_pointer *ptr, List_pointer node)
{
/* insert a new node with data = 50 into the list ptr after node */
    list_pointer temp;
    temp=(list_pointer)malloc(sizeof(list_node));
    if(IS_FULL(temp)){
            fprintf(stderr, "The memory is full\n");
            exit(1);
    }
    temp->data=50;
```



لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

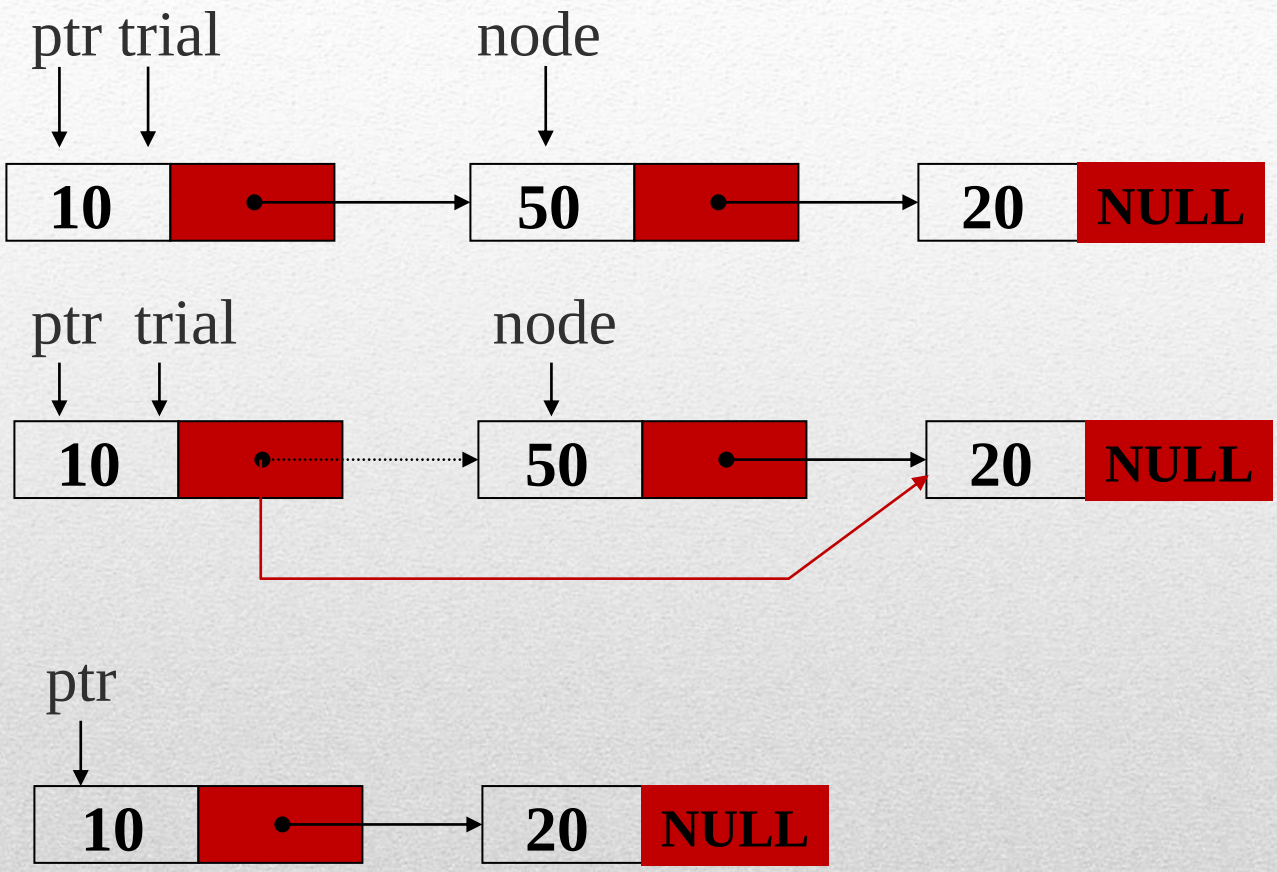
```
if(*ptr){//nonempty list
        temp->link = node->link;
        node->link = temp;
}
else{     //empty list
        temp->link = NULL;
        *ptr = temp;
}
```



لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- **حذف کردن** عضوی که node به آن اشاره می کند را حذف کنید

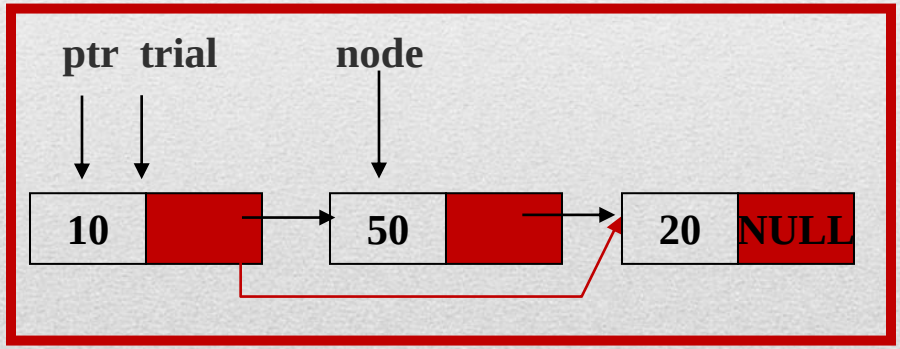


لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

Implement Deletion:

```
void delete(list_pointer *ptr, list_pointer trail, list_pointer node)
{
/* delete node from the list, trail is the preceding node ptr is the head of the
list */
        if(trail)
                trail->link = node->link;
        else
                *ptr = (*ptr)->link;
        free(node);
}
```



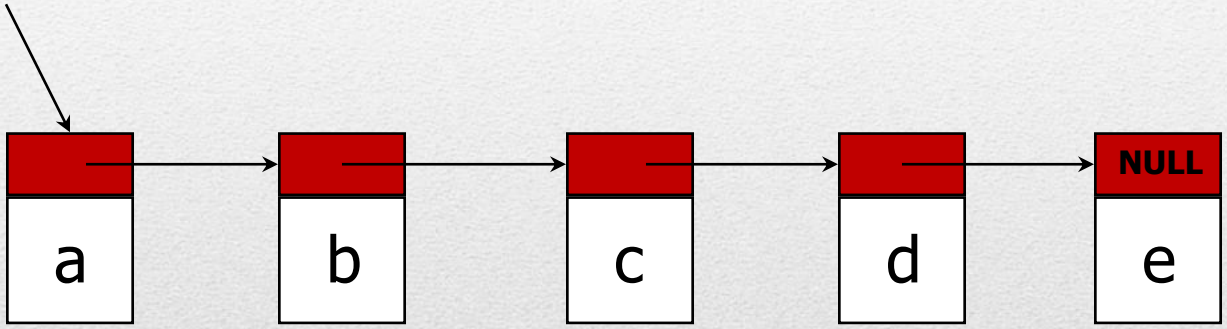
لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- پیمایش (چاپ) یک لیست پیوندی

Get(0)

first



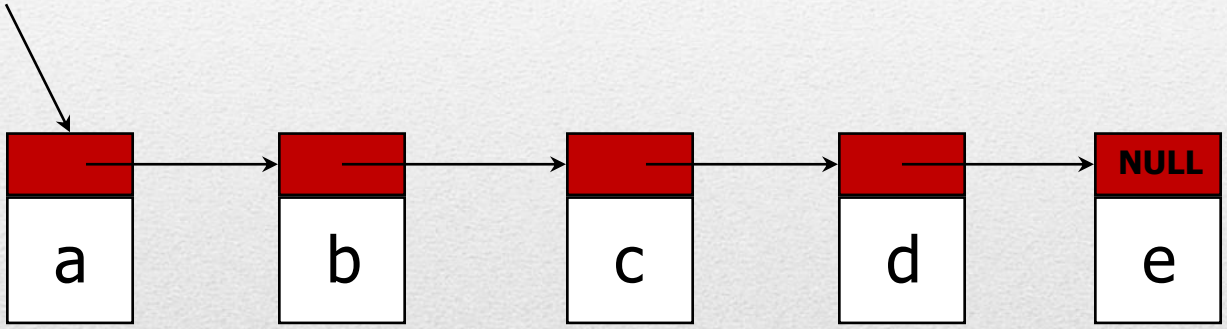

```
desiredNode = first; // gets you to first node
return desiredNode–>data;
```

لیست های تک پیوندی

- پیمایش (چاپ) یک لیست پیوندی

Get(1)

first



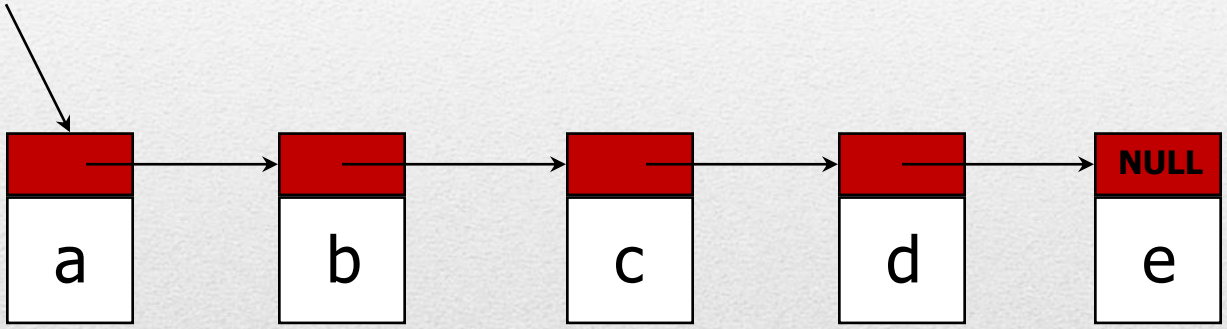

```
desiredNode = first–>link; // gets you to second node
return desiredNode–>data;
```

لیست های تک پیوندی

- پیمایش (چاپ) یک لیست پیوندی

Get(2)

first

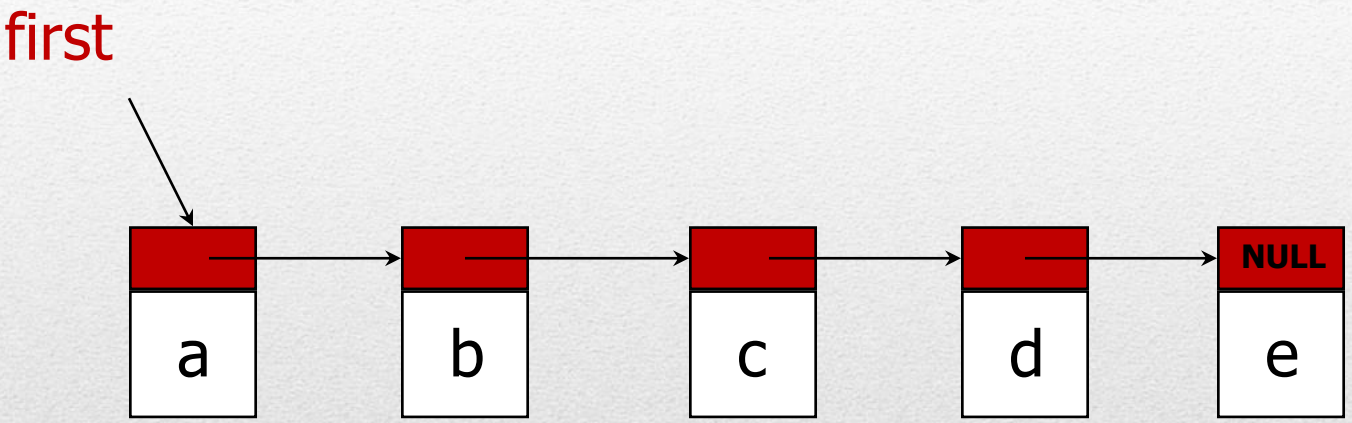



```
desiredNode = first–>link–>link; // gets you to third node
return desiredNode–>data;
```

لیست های تک پیوندی

- پیمایش (چاپ) یک لیست پیوندی

Get(5)




```
desiredNode = first–>link–>link–>link–>link–>link;
                                    // desiredNode = NULL
return desiredNode–>data;   // NULL.element
```

لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

- **پیمایش (چاپ) یک لیست پیوندی**

Program : Printing a list

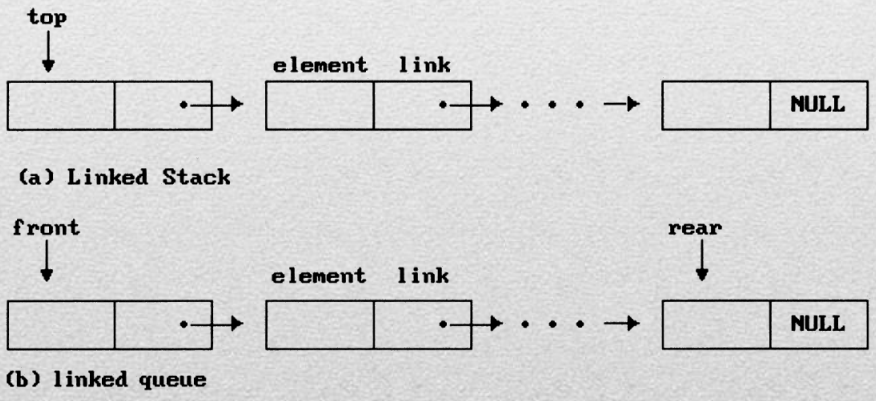
```
void print_list(list_pointer first)
{
    printf(“The list contains: “);
    for ( ; first; first= first->link)
    printf(“%4d”, first->data);
    printf(“\n”);
}
```

لیست های تک پیوندی

E-mail: Hadi.khademi@gmail.com

پشته ها و صف های پیوندی

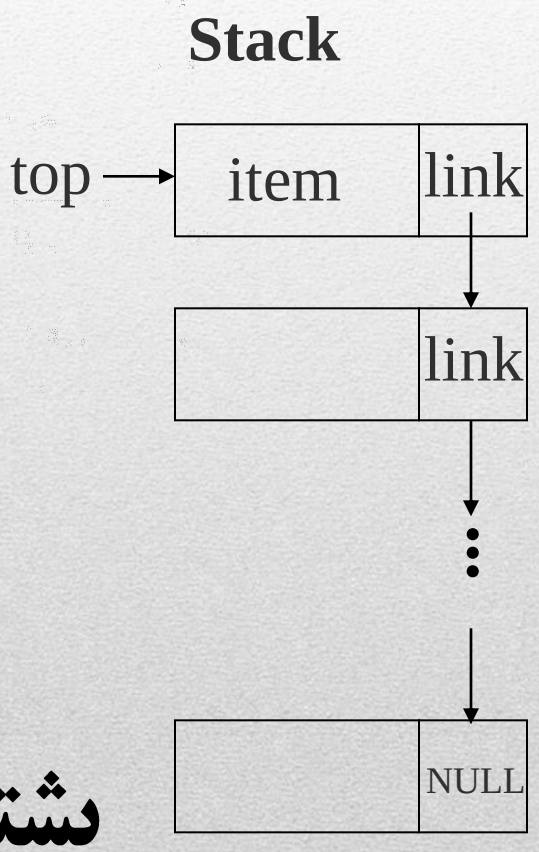
- هنگامی که چندین صف و پشته وجود داشته باشد روش ترتیبی کارایی برای بازنمایی آنها وجود ندارد.
- در بازنمایی پیوندی جهت اشاره گر برای پشته و صف به صورتی است که عملیات حذف کردن و اضافه کردن گره ها در انها به اسانی انجام شود.
- به اسانی می توانید یک گره را به بالای پشته اضافه و یا از آن حذف کنید.
- به اسانی می توانید یک گره به آخر صف اضافه کنید یا عمل اضافه کردن و حذف کردن را در اول صف انجام دهید (هر چند اضافه کردن گره در اول صف معمولا انجام نمی شود)



E-mail: Hadi.khademi@gmail.com

- بازنمایی n پشته

```
#define MAX_STACKS 10 /*maximum number of stacks*/
typedef struct {
        int key;
        /* other fields */
        } element;
typedef struct stack *stack_pointer;
typedef struct stack {
        element item;
        stack_pointer link;
        };
stack_pointer top[MAX_STACKS];
```

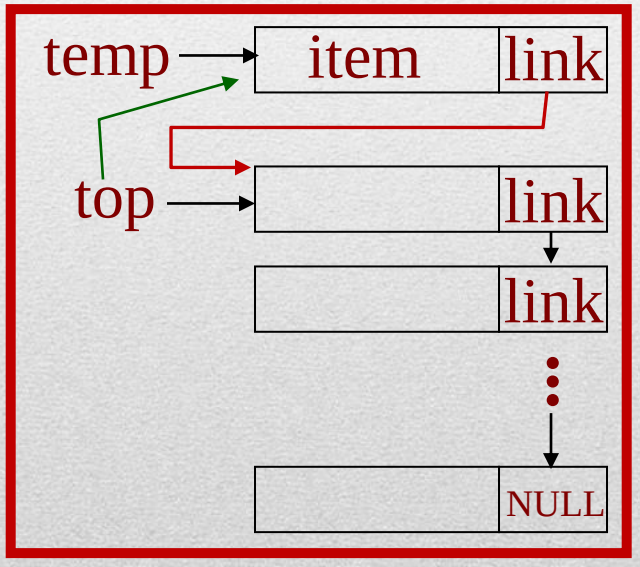


پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

Push in the linked stack

```
void add(stack_pointer *top, element item){
        /* add an element to the top of the stack */ Push
        stack_pointer temp = (stack_pointer) malloc (sizeof (stack));
        if (IS_FULL(temp)) {
                fprintf(stderr, " The memory is full\n");
                exit(1);
        }
        temp->item = item;
        temp->link = *top;
        *top= temp;
}
```

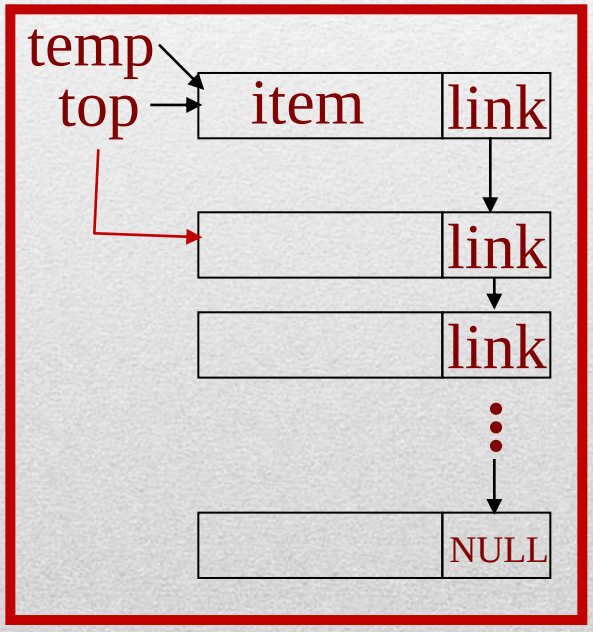


پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

Pop from the linked stack

```
element delete(stack_pointer *top) {
/* delete an element from the stack */    Pop
    stack_pointer temp = *top;
    element item;
    if (IS_EMPTY(temp))  {
            fprintf(stderr,  "The stack is empty\n");
            exit(1);
    }
    item = temp->item;
    *top = temp->link;
    free(temp);
    return item;
}
```

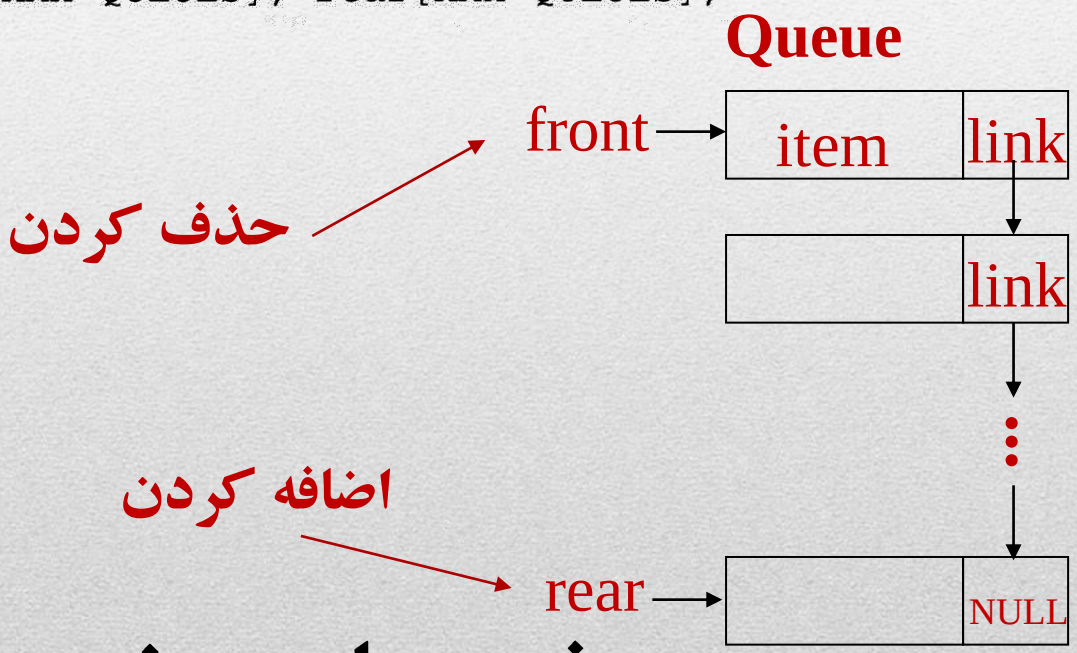


پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

- بازنمایی n صف

```
#define MAX_QUEUES 10 /* maximum number of queues */
typedef struct queue *queue_pointer;
typedef struct queue {
        element item;
        queue_pointer link;
        };
queue_pointer front[MAX_QUEUES], rear[MAX_QUEUES];
```

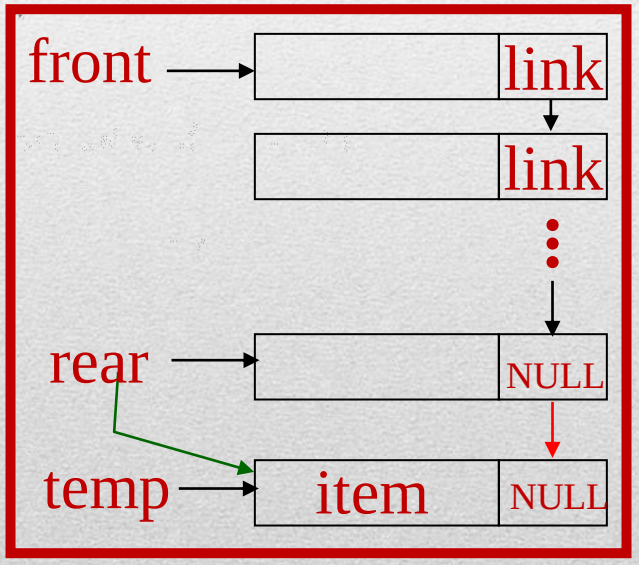


پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

اضافه کردن به صف پیوندی

```
void addq(queue_pointer *front, queue_pointer *rear,
                                element item)
{
/* add an element to the rear of the queue */
   queue_pointer temp =
                    (queue_pointer) malloc(sizeof(queue));
   if (IS_FULL(temp)) {
      fprintf(stderr, "The memory is full\n");
      exit(1);
   }
   temp->item = item;
   temp->link = NULL;
   if (*front) (*rear)->link = temp;
   else *front = temp;
   *rear = temp;
}
```

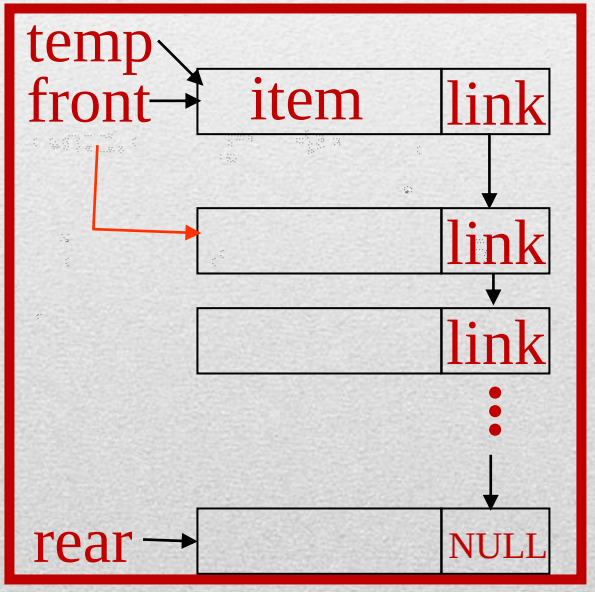


پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

حذف کردن از صف پیوندی

```
element deleteq(queue_pointer *front)
{
/* delete an element from the queue */
   queue_pointer temp = *front;
   element item;
   if (IS_EMPTY(*front)) {
      fprintf(stderr, "The queue is empty\n");
      exit(1);
   }
   item = temp->item;
   *front= temp->link;
   free(temp);
   return item;
}
```



پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

- راهکار ارائه شده برای مسائل n-stack و m-queue هم از نظر محاسباتی و هم از نظر مفهومی ساده هستند.
  - لازم نیست برای ایجاد فضای خالی پشته ها و یا صف ها شیفت داده شوند.
  - تا زمانی که حافظه وجود داشته باشد می توان از ان استفاده کرد.

پشته ها و صف های پیوندی

E-mail: Hadi.khademi@gmail.com

- اضافه کردن یک گره به انتهای لیست پیوندی

```
void attach(list_pointer first, list_pointer last, List_pointer newnode)
{
        if(first==0) first=last=newnode;
        else
        {
                last->link=newnode;
            last=newnode;
        }
}
```

- فرض می کنیم عضو داده ای last وجود دارد که به گره آخر لیست پیوندی اشاره می کند

عملیات روی لیست های پیوندی

E-mail: Hadi.khademi@gmail.com

- معکوس کردن لیست پیوندی

```
void Invert(list_pointer first)
{
   list_pointer p=first, q=0, r=0;     //q trails p
        while( p) {
                        r=q; q=p;                 //r trails q
              p=p->link;                  //p moves to next node
              q->link=r;                  //link q to preceding node
        }
        first =q;
}
```

برای لیست به طول m زمان اجرا o(m)

عملیات روی لیست های پیوندی

E-mail: Hadi.khademi@gmail.com

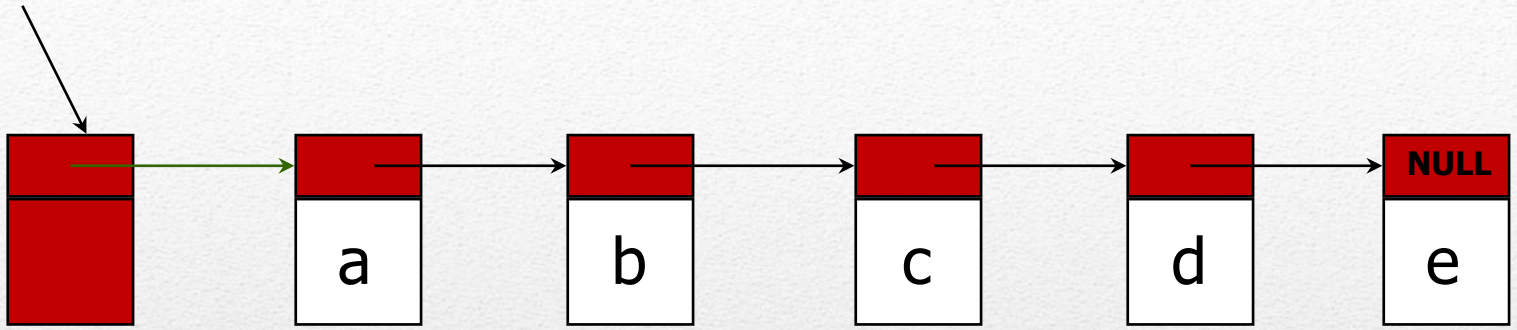
- **اتصال دو زنجیر**

```
void Concatenate(list_pointer first_a, list_pointer first_b)
{
        if (!first_a) { first_a=first_b; return;}
  if (first_b) {
      for (list_pointer  p=first; p->link; p=p->link); no body
                  p->link=first_b;
        }
}
```

زمان اجرا بر حسب طول زنجیر اول خطی است

عملیات روی لیست های پیوندی

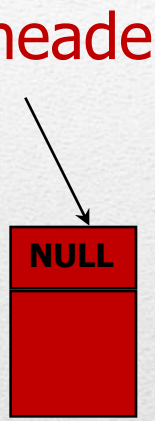
E-mail: Hadi.khademi@gmail.com



- حال اضافه/حذف از سمت چپ (اندیس صفر) متفاوت با حذف و اضافه کردن بقیه نودها نیست و کد حذف/ اضافه ساده تر می شود.

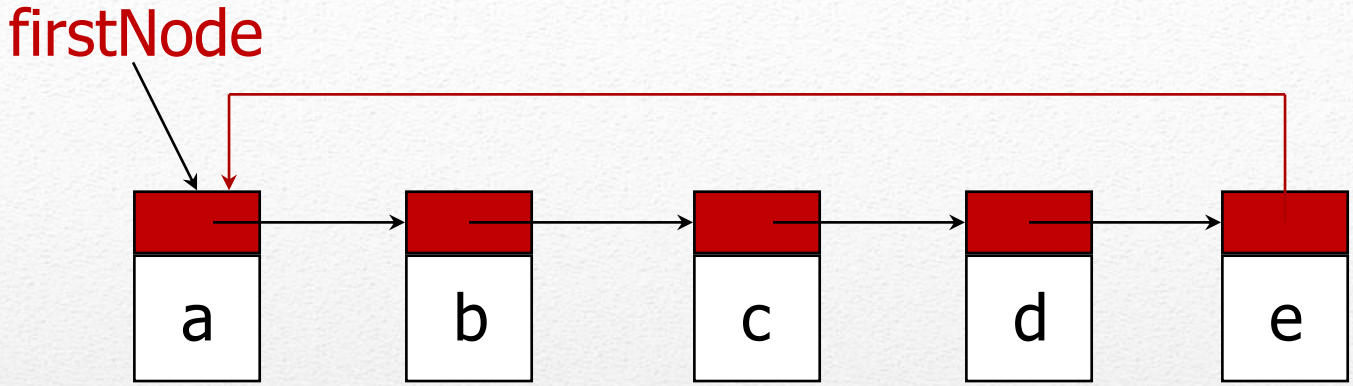
زنجیر با گره سر

E-mail: Hadi.khademi@gmail.com



زنجیر خالی با گره سر

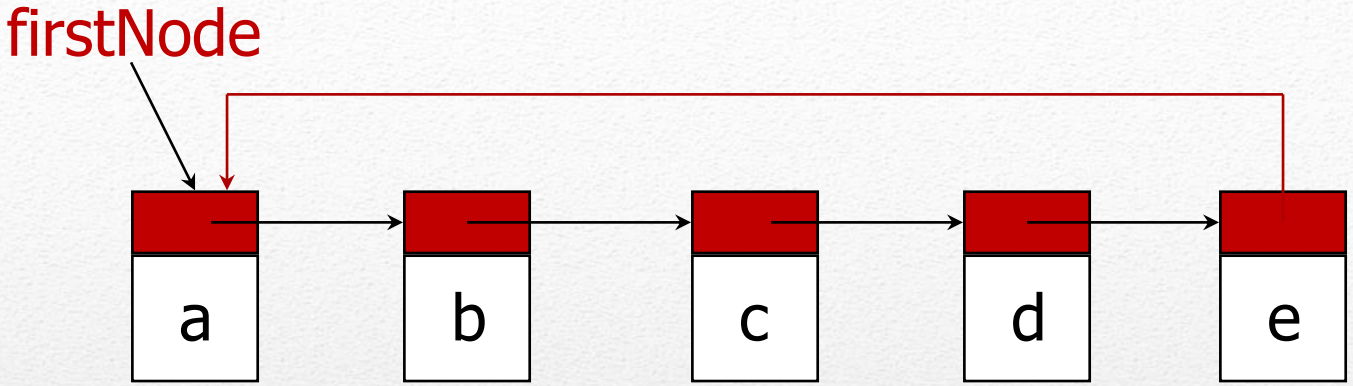
E-mail: Hadi.khademi@gmail.com



- برای بررسی اینکه ایا اشاره گر current به گره آخر لیست دایره ای اشاره می کند به جای مقایسه (current->link==0) مقایسه ی (current->link==first) انجام می شود.
- الگوریتم های حذف/ اضافه کردن از/ به لیست دایره ای باید تضمین کند که فیلد اشاره گر گره آخر به گره اول لیست اشاره کند.

لیست دایره ای

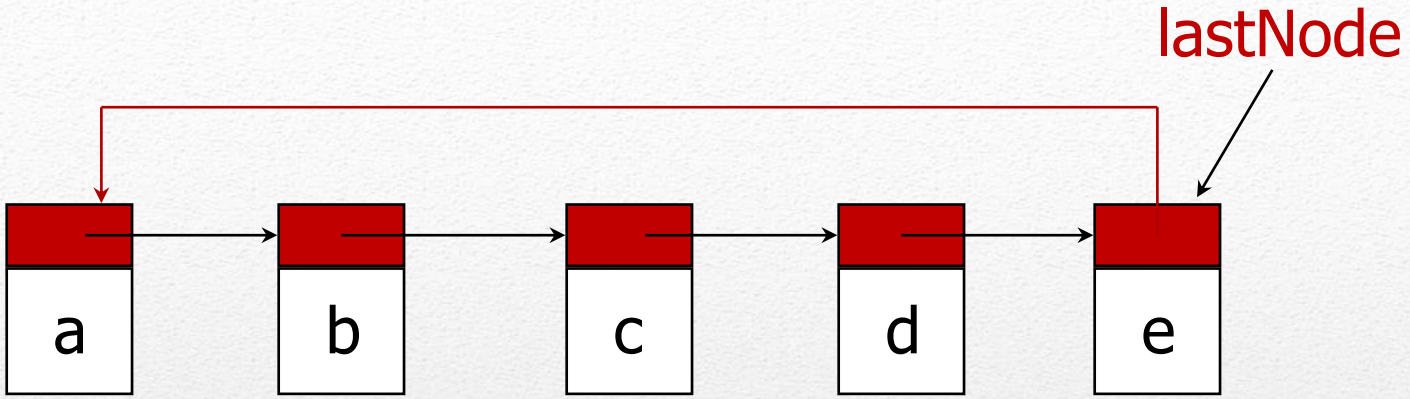
E-mail: Hadi.khademi@gmail.com



- می خواهیم گره جدیدی به اول لیست بالا اضافه کنیم
- باید اشاره گر گره آخر (گره ای که حاوی e است) را تغییر دهیم.
- باید تا پیدا نشدن گره آخر در طول لیست حرکت کنیم.

لیست دایره ای

E-mail: Hadi.khademi@gmail.com



- هنگامی که از گره سر استفاده نمی کنیم بهتر است اشاره گر دسترسی به لیست دایره ای به جای گره اول به گره آخر لیست اشاره کند.
- در اینصورت اضافه کردن یک گره در اول و یا در آخر لیست دایره ای در مدت زمان ثابتی انجام می شود.

لیست دایره ای

E-mail: Hadi.khademi@gmail.com

لیست دایره ای

- اضافه کردن گره ای که x به آن اشاره می کند در اول لیست

```
void InsertFront(list_pointer last, list_pointer x)
// insert the node pointed at by x at the front of the circular list
// last points to the last node in the list
{
        if (!last) {  // empty list
                last=x; x->link=x;
   }
   else {
      x->link=last->link; last->link=x;
         }
}
```

زمان اجرا O(1) است

E-mail: Hadi.khademi@gmail.com

- اضافه کردن گره ای که x به آن اشاره می کند در انتهای لیست

```
void InsertRear(list_pointer last, list_pointer x)
// insert the node pointed at by x at the rear of the circular list
// last points to the last node in the list
{
        if (!last) {  // empty list
                last=x; x->link=x;
    } else {
        x->link=last->link; last->link=x;
                last=x;
        }
}
```

تفاوت با کد قبلی

زمان اجرا O(1) است

لیست دایره ای

E-mail: Hadi.khademi@gmail.com

- بازنمایی چند جمله ایها با لیست های پیوندی

$$A(x) = a_{m-1}x^{e_{m-1}} + \cdots + a_0 x^{e_0}$$

- a_i ها ضرایب غیر صفر و e_i ها توانهای صحیح غیر منفی هستند به قسمی که

$$e_{m-1} > e_{m-2} > \ldots > e_1 > e_0 \geqq 0$$

- هر عبارت با یک گره که شامل سه فیلد ضریب، توان و اشاره گر به گره بعدی است نشان داده می شود.

چند جمله ای ها

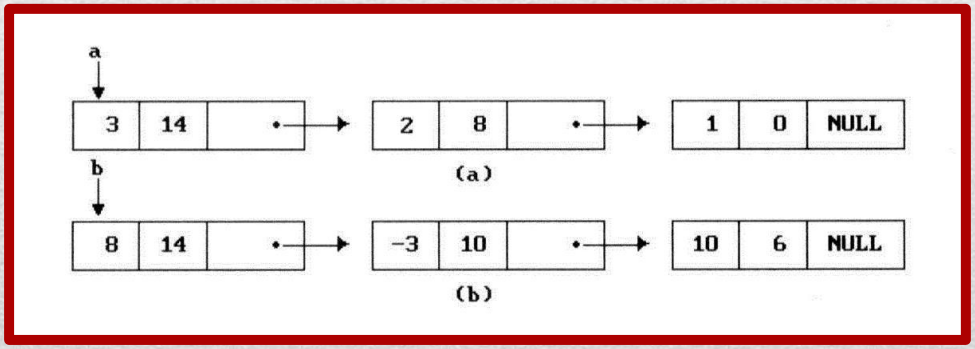
E-mail: Hadi.khademi@gmail.com

```
typedef struct poly_node *poly_pointer;
typedef struct poly_node {
      int coef;
      int expon;
     poly_pointer link;
};
poly_pointer a,b,d;
```

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

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

coef	expon	link



چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

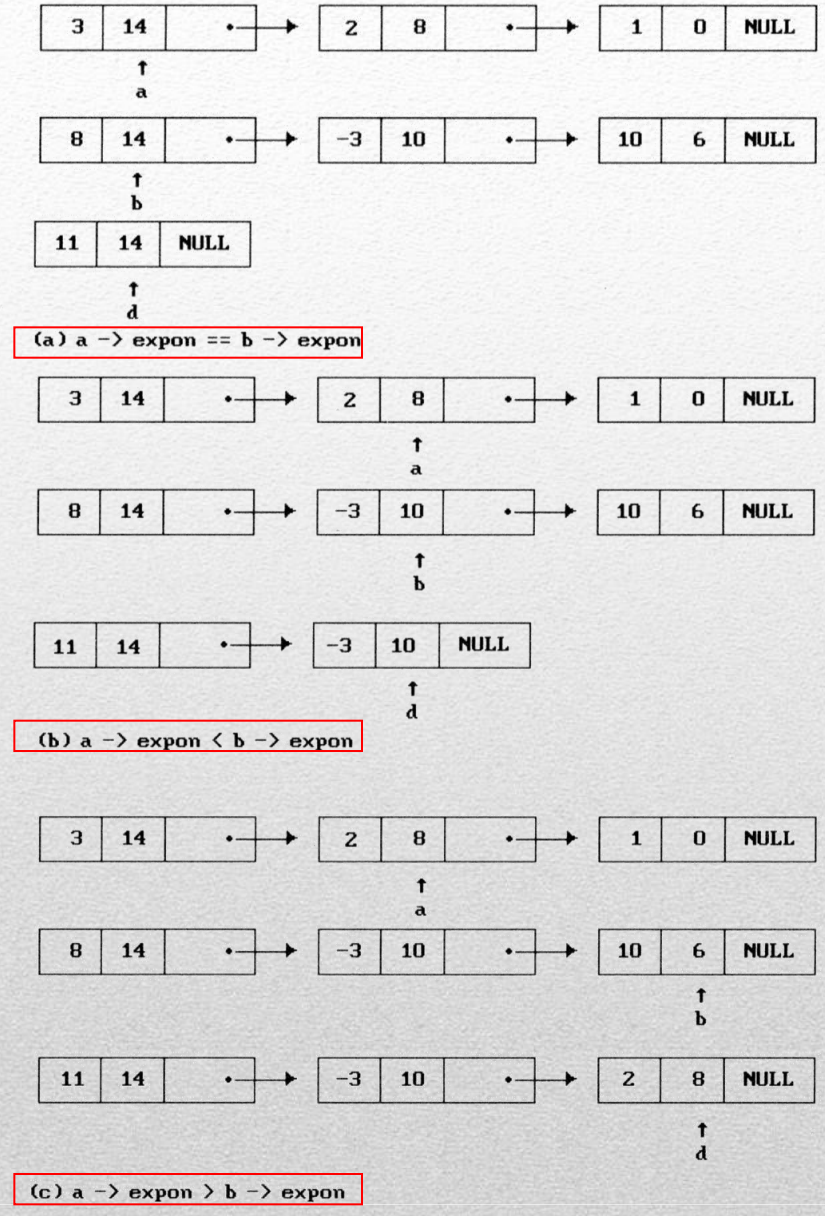
- فرض کنیم که a و b اشاره گرهایی به ابتدای چند جمله ای ها باشند.
- اگر توان دو چند جمله ای با هم برابر باشد ، ضرایب با هم جمع می شوند و گره جدیدی تشکیل می شود ، همچنین اشاره گرها را به گره های بعدی در a و b حرکت می دهیم.
- اگر توان چند جمله ای a کمتر از توان متناظر در چند جمله ای b باشد، آنگاه یک جمله 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 .

جمع چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

جمع چند جمله ای ها

$d = a+b$



E-mail: Hadi.khademi@gmail.com

```
poly_pointer padd(poly_pointer a, poly_pointer b)
{
/* return a polynomial which is the sum of a and b */
   poly_pointer front, rear, temp;
   int sum;
   rear = (poly_pointer)malloc(sizeof(poly_node));
   if (IS_FULL(rear)) {
      fprintf(stderr, "The memory is full\n");
      exit(1);
   }
   front = rear;
   while (a && b)
      switch (COMPARE(a->expon,b->expon)) {
         case -1: /* a->expon < b->expon */
               attach(b->coef,b->expon,&rear);
               b = b->link;
               break;
         case 0: /* a->expon = b->expon */
               sum = a->coef + b->coef;
               if (sum) attach(sum,a->expon,&rear);
               a = a->link;  b = b->link; break;
         case 1: /* a->expon > b->expon */
               attach(a->coef,a->expon,&rear);
               a = a->link;
      }
   /* copy rest of list a and then list b */
   for (; a; a = a->link) attach(a->coef,a->expon,&rear);
   for (; b; b = b->link) attach(b->coef,b->expon,&rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = front; front = front->link;  free(temp);
   return front;
}
```

جمع چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

```
void attach(float coefficient, int exponent, poly_pointer *ptr){
/* create a new node with coef = coefficient and expon = exponent,
   attach it to the node pointed to by ptr.  Ptr is updated to point to this
   new node */
   poly_pointer temp;
   temp = (poly_pointer) malloc(sizeof(poly_node));
   /* create new node */
   if (IS_FULL(temp)) {
         fprintf(stderr, "The memory is full\n");
         exit(1);
   }
   temp->coef = coefficient;          /* copy item to the new node */
   temp->expon = exponent;
   (*ptr)->link = temp;      /* attach */
   *ptr = temp;                       /* move ptr to the end of the list */
}
```

جمع چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

جمع چند جمله ای ها

- تحلیل جمع چند جمله ای ها

$$A(x)(=a_{m-1}x^{e_{m-1}}+\cdots+a_0x^{e_0})+B(x)(=b_{n-1}x^{f_{n-1}}+\cdots+b_0x^{f_0})$$

- جمع ضرایب

$0 \le \text{additions} \le \min(m, n)$
where m (n) denotes the number of terms in A (B).

- مقایسه توان ها

extreme case:
$e_{m-1} > f_{m-1} > e_{m-2} > f_{m-2} > \ldots > e_1 > f_1 > e_0 > f_0$
m+n-1 comparisons

- ایجاد گره جدید برای d

extreme case: maximum number of terms in d is m+n

m + n new nodes
summary: O(m+n)

الگوریتم جمع چند جمله ای با فاکتور ثابت بهینه است.

E-mail: Hadi.khademi@gmail.com

$e(x) = a(x) * b(x) + d(x)$

```
poly_pointer a, b, d, e;
...
a = read_poly();
b = read_poly();
d = read_poly();
temp = pmult(a, b);
e = padd(temp, d);
print_poly(e);
```

read_podly()

print_poly()

padd()

psub()

pmult()

temp برای ذخیره ی مقدار میانی استفاده شده است با بازگرداندن گره های این چند جمله ای می توان از گره های آن برای نگهداری چندجمله ایهای دیگر استفاده کرد.

چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

- حذف یک چند جمله ای

```
void erase (poly_pointer *ptr){
/* erase the polynomial pointed to by ptr */
        poly_pointer temp;
        while ( *ptr){
                temp = *ptr;
                *ptr = (*ptr) -> link;
                free(temp);
        }
}
```

چند جمله ای ها

E-mail: Hadi.khademi@gmail.com

- با استفاده از لیست دایره ای به جای زنجیر می توانیم تمام گره های لیست را به صورت کارایی آزاد کنیم.

- مثال

نمایش ptr = 3x14+2x8+1 به صورت لیست دایره ای

نمایش چند جمله ایها با لیست دایره ای

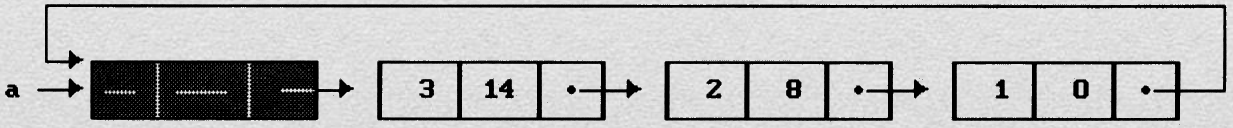
E-mail: Hadi.khademi@gmail.com

- باید چند جمله ای صفر را به صورت یک حالت خاص در نظر بگیریم. برای اجتناب از چنین حالت خاصی می توانیم در هر چند جمله ای یک گره سر تعریف کنیم.

  هر چند جمله ای، چه چند جمله ای صفر یا غیر صفر، یک گره اضافی دارد

  عضوهای داده expon و coef برای این گره معنی ندارند.

(a) Zero polynomial



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

نمایش چند جمله ایها با لیست دایره ای

E-mail: Hadi.khademi@gmail.com

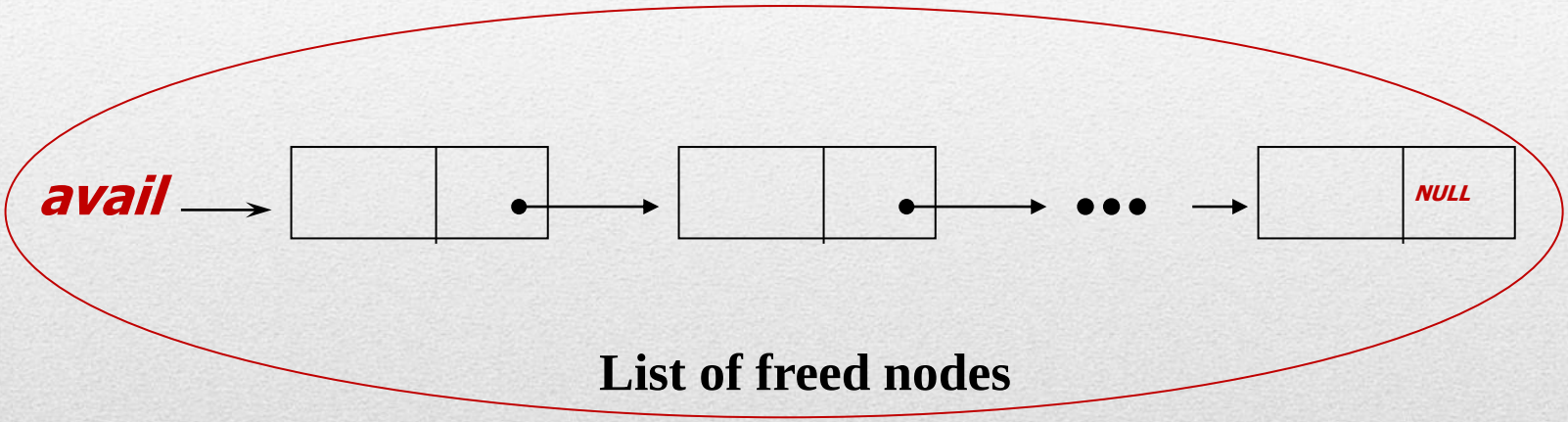
```
poly_pointer cpadd(poly_pointer a, poly_pointer b)
{
/* polynomials a and b are singly linked circular lists
with a head node. Return a polynomial which is the sum
of a and b */
   poly_pointer starta, d, lastd;
   int sum, done = FALSE;
   starta = a;              /* record start of a */
   a = a->link;             /* skip head node for a and b*/
   b = b->link;
   d = get_node();          /* get a head node for sum */
   d->expon = -1; lastd = d;     /* head node */
   do {
      switch (COMPARE(a->expon, b->expon)) {
         case -1: /* a->expon < b->expon */
                  attach(b->coef,b->expon,&lastd);
                  b = b->link;
                  break;
         case 0:  /* a->expon = b->expon */
                  if (starta == a)  done = TRUE;
                  else { /*a->expon=-1, so b->expont = -1 */
                     sum = a->coef + b->coef;
                     if (sum) attach(sum,a->expon,&lastd);
                     a = a->link; b = b->link;
                  }
                  break;
         case 1:  /* a->expon > b->expon */
                  attach(a->coef,a->expon,&lastd);
                  a = a->link;
      }
   } while (!done);
   lastd->link = d;       /* link to the first node */
   return d;
}
```

فیلد expon از گره سر را برابر 1- قرار می دهیم حال اگر تمام گره های a بررسی شوند a- خواهد شد و از انجا که a->expona=-1 b->expon>=-1 است بقیه جملات b در اجرای بعدی دستور case کپی خواهد شد همین مطلب در مورد گره های b قبل از a نیز صادق است

E-mail: Hadi.khademi@gmail.com

با ذخیره گره هایی که حذف شده اند به صورت زنجیر، یک الگوریتم حذف کارا برای لیست های دایره ای به دست می آوریم.

به جای استفاده از توابع malloc و free از تابع های get_node و ret_node استفاده می کنیم.



لیست دایره ای

E-mail: Hadi.khademi@gmail.com

لیست دایره ای

- هرگاه احتیاج به گره جدیدی داشته باشیم لیست آزاد را بررسی می کنیم. اگر این لیست خالی نباشد آنگاه از یکی از گره های موجود در آن استفاده می کنیم. فقط وقتی لیست خالی است از دستور malloc برای ایجاد گره جدید استفاده می کنیم.

```
poly_pointer get_node(void)
/* provide a node for use */
{
   poly_pointer node;
   if (avail) {
      node = avail;
      avail = avail->link;
   }
   else {
      node = (poly_pointer) malloc(sizeof(poly_node));
      if (IS_FULL(node)) {
         fprintf(stderr, "The memory is full\n");
         exit(1);
      }
   }
   return node;
}
```

E-mail: Hadi.khademi@gmail.com

لیست دایره ای

- برگرداندن یک گره

avail متغیری از نوع listNode * است که به زنجیری از گره هایی که حذف شده اند اشاره می کند.

این زنجیر یا لیست را لیست حافظه آزاد می نامیم.

در آغاز avail برابر NULL است.

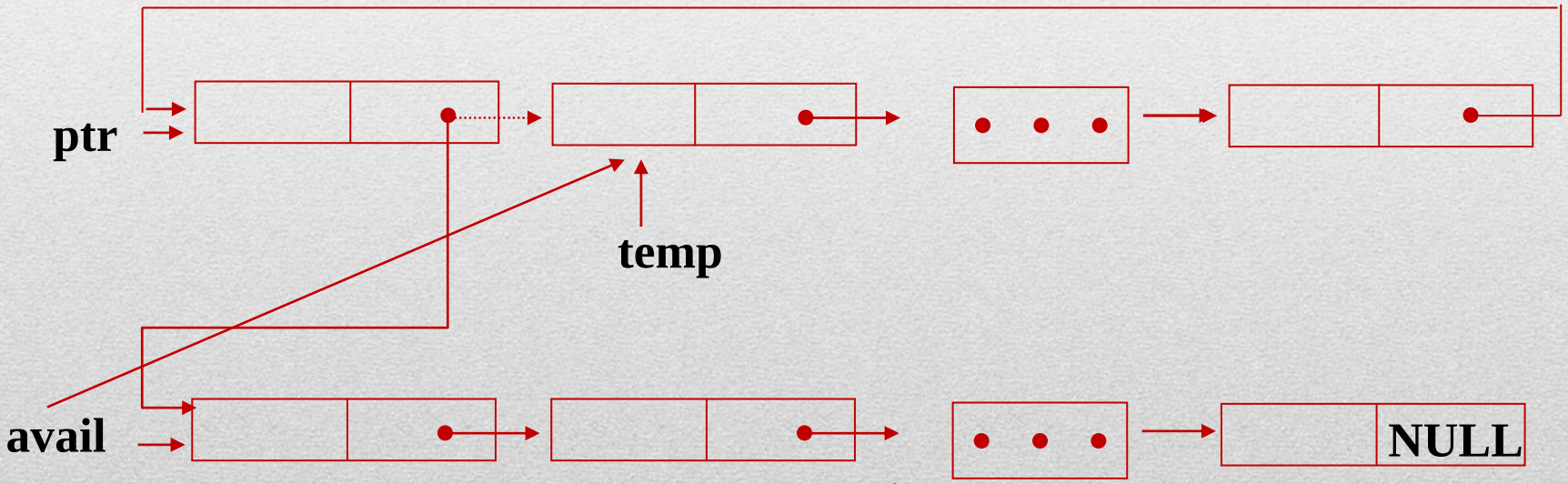
اضافه کردن ptr به ابتدای لیست

```
void ret_node(poly_pointer ptr)
{
/* return a node to the available list */
   ptr->link = avail;
   avail = ptr;
}
```

E-mail: Hadi.khademi@gmail.com

```
void cerase(poly_pointer *ptr)
{
/* erase the circular list ptr */
   poly_pointer temp;
   if (*ptr) {
      temp = (*ptr)->link;
      (*ptr)->link = avail;
      avail = temp;
      *ptr = NULL;
   }
}
```

لیست دایره ای را می توان در زمان ثابت (O(1)) مستقل از تعداد گره های لیست حذف کرد.



حذف یک لیست دایره ای

E-mail: Hadi.khademi@gmail.com

- تحلیل تابع هم ارزی
- فاز اول

مقدار دهی اولیه به seq و out O(n)

پردازش هر زوج ورودی O(m)

O(n+m)

- فاز دوم

هر گره حداکثر یک بار در پشته قرار می گیرد. از آنجا که 2m گره وجود دارد و حلقه for، n بار اجرا می شود زمان اجرای این مرحله O(n+m) است

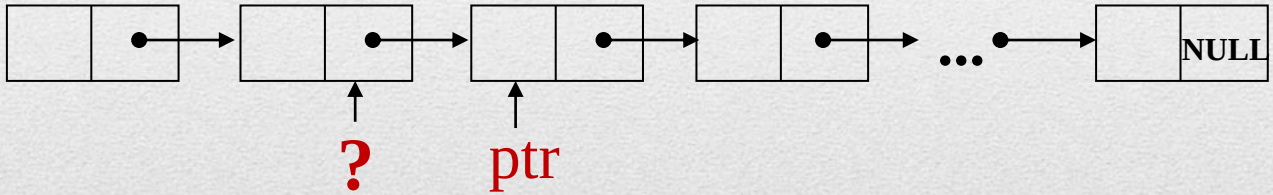
- تابع بیان شده با ضریب ثابت بهینه است.
- فضای لازم برای این الگوریتم O (m + n) است.

روابط هم ارزی

E-mail: Hadi.khademi@gmail.com

- 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 های دو پیوندی

E-mail: Hadi.khademi@gmail.com

- یک گره در یک لیست دو پیوندی حداقل سه فیلد داده ای دارد:

  فیلد data

  فیلد llink (اشاره گر به چپ)

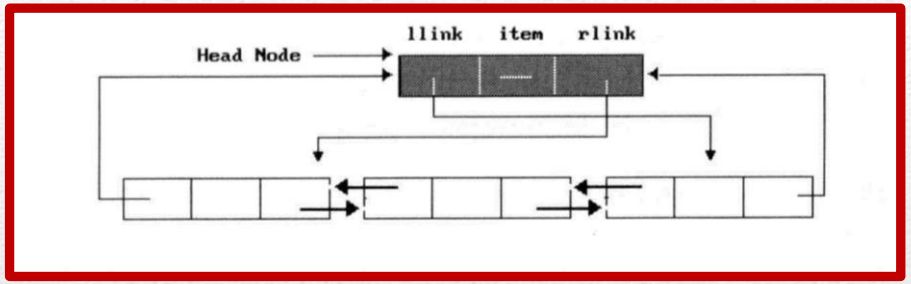
  فیلد rlink (اشاره گر به راست)
- لیست دوپیوندی می تواند دایره ای باشد یا دایره ای نباشد. می تواند دارای گره سر باشد یا نباشد.

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

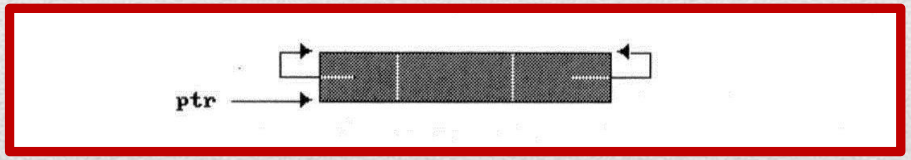
لیست دو پیوندی

E-mail: Hadi.khademi@gmail.com

- 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

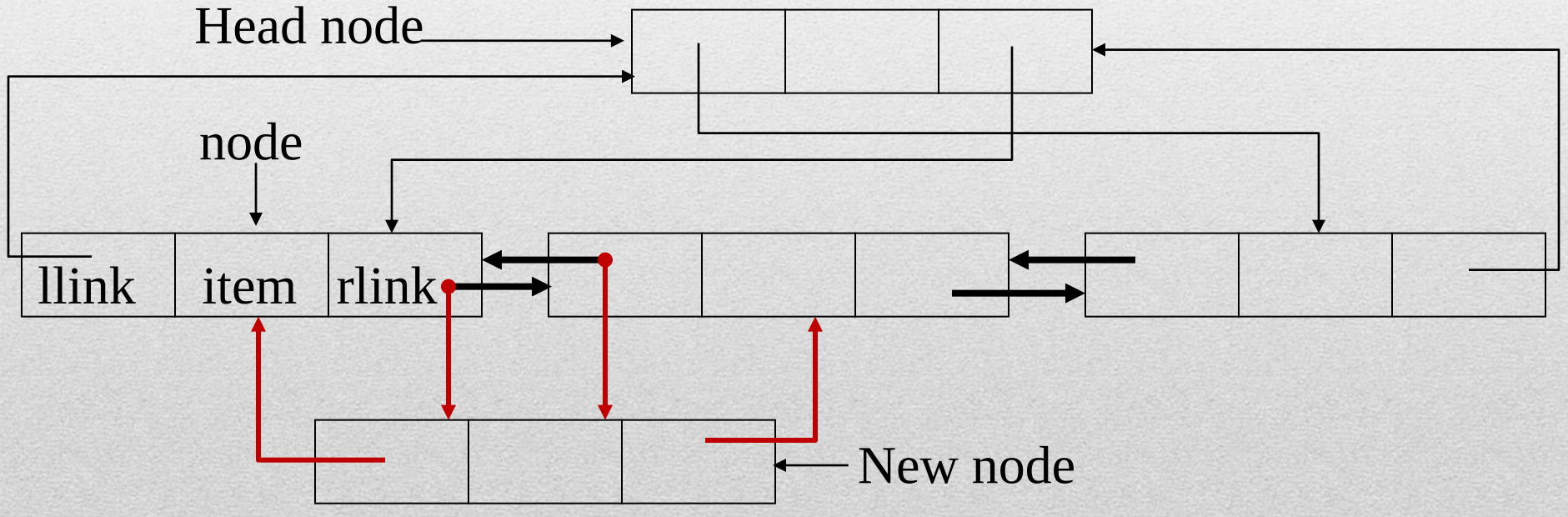
ptr = ptr -> llink -> rlink = ptr -> rlink -> llink

لیست دو پیوندی

E-mail: Hadi.khademi@gmail.com

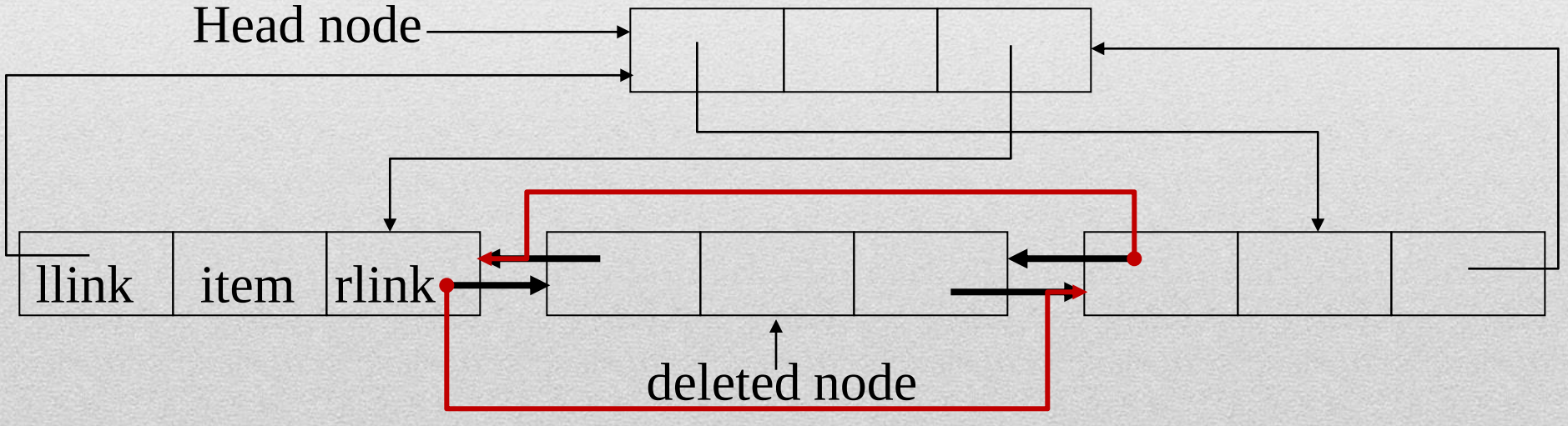
- اضافه کردن گره در یک لیست دایره ای دوپیوندی

```
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;
}
```



- 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_i ها 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.

E-mail: Hadi.khademi@gmail.com

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)) = (( ))

لیست های تعمیم یافته

E-mail: Hadi.khademi@gmail.com

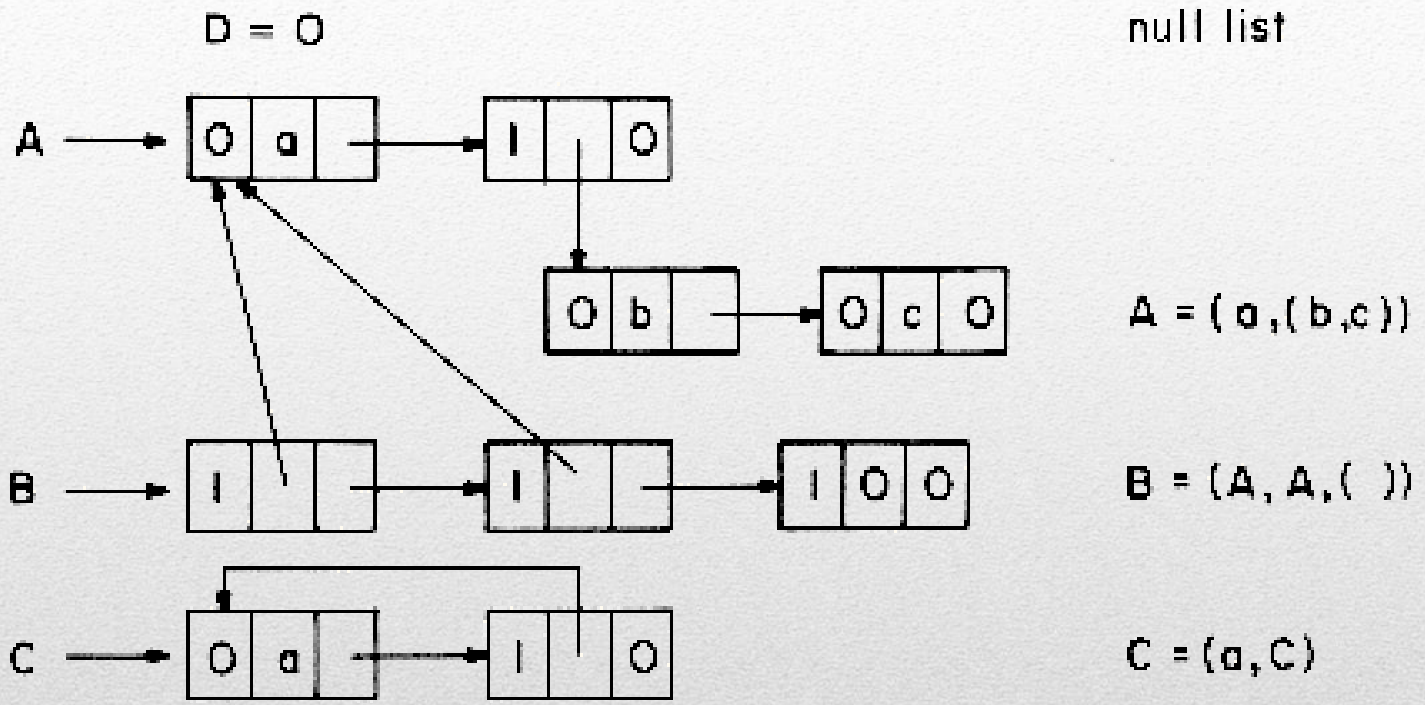

```
enum  Boolean {false, true};
Struct GenListNode {
    GenListNode *link;
    Boolean tag;
    union {
    char data;
    GenListNode *dlink;
    };
};
```

یک لیست تعمیم یافته با یک اشاره گر

GenListNode * first

ساختار گره یک لیست تعمیم یافته

E-mail: Hadi.khademi@gmail.com



نمایش لیست های عمومی

E-mail: Hadi.khademi@gmail.com

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

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

- راهکار: استفاده از ساختاری مانند

COEF	EXPX	EXPY
EXPZ	LINK	

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

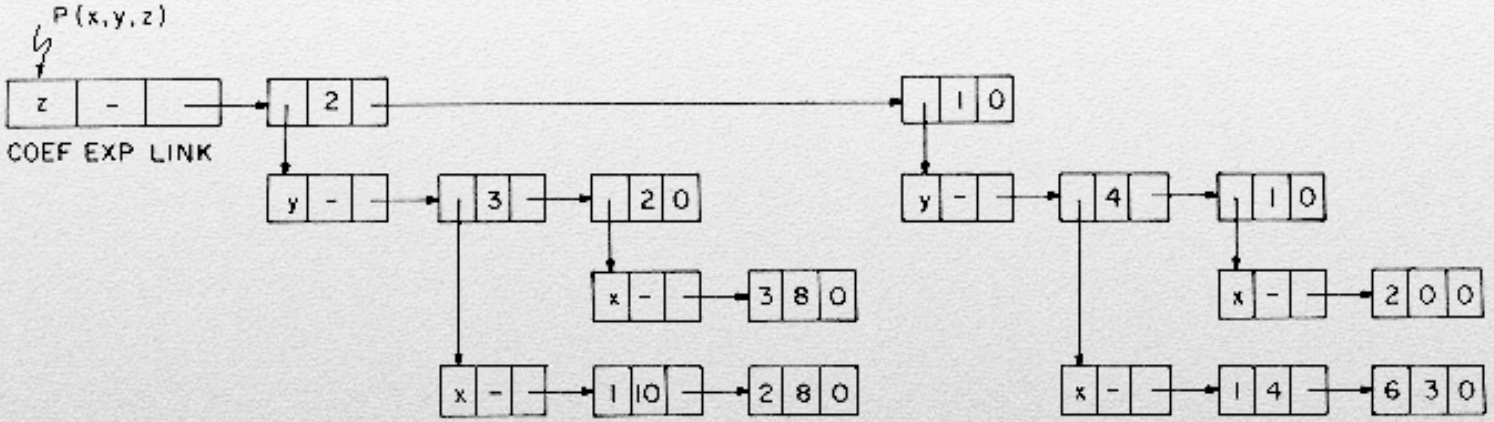
نمونه کاربرد لیست های عمومی

E-mail: Hadi.khademi@gmail.com

$P(x,y,z)=$

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

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



نمونه کاربرد لیست های عمومی

E-mail: Hadi.khademi@gmail.com

- ساختار نود برای چند جمله ای

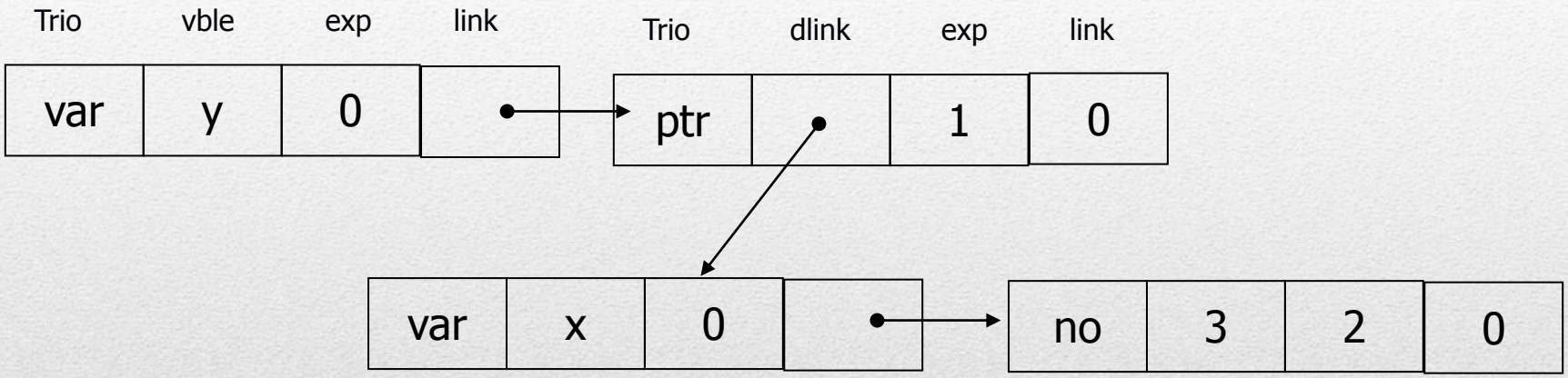
```
enum Triple {var, ptr, no}
Struct PolyNode {
        PolyNode *link;
        int exp;
        Triple trio;
        union {
                Char vble;
                PolyNode *dlink;
                int coef;
        };
};
```

نمونه کاربرد لیست های عمومی

E-mail: Hadi.khademi@gmail.com

- مثال

$P = 3x2y$



نمونه کاربرد لیست های عمومی

E-mail: Hadi.khademi@gmail.com

- 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

E-mail: Hadi.khademi@gmail.com

- 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

E-mail: Hadi.khademi@gmail.com