



دانشگاه کاشان

University of Kashan

# لیست پیوندی

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

# 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

## مقدمه

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

# بازنمایی پیوندی

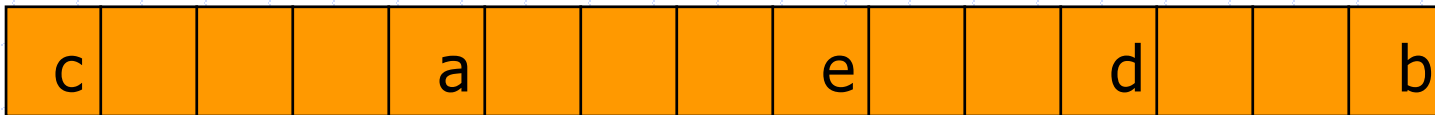
- راه حل مناسب : استفاده از بازنمایی پیوندی
- عناصر می توانند در هر جای حافظه قرار گیرند.
- در بازنمایی ترتیبی، ترتیب اعضای لیست با ترتیب نگهداری اعضا در حافظه یکسان است ولی در بازنمایی پیوندی لازم نیست ترتیب اعضای لیست با ترتیب نگهداری اعضا یکسان باشد.
- برای دستیابی صحیح به عناصر یک لیست، بایستی به 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.

# شمای حافظه

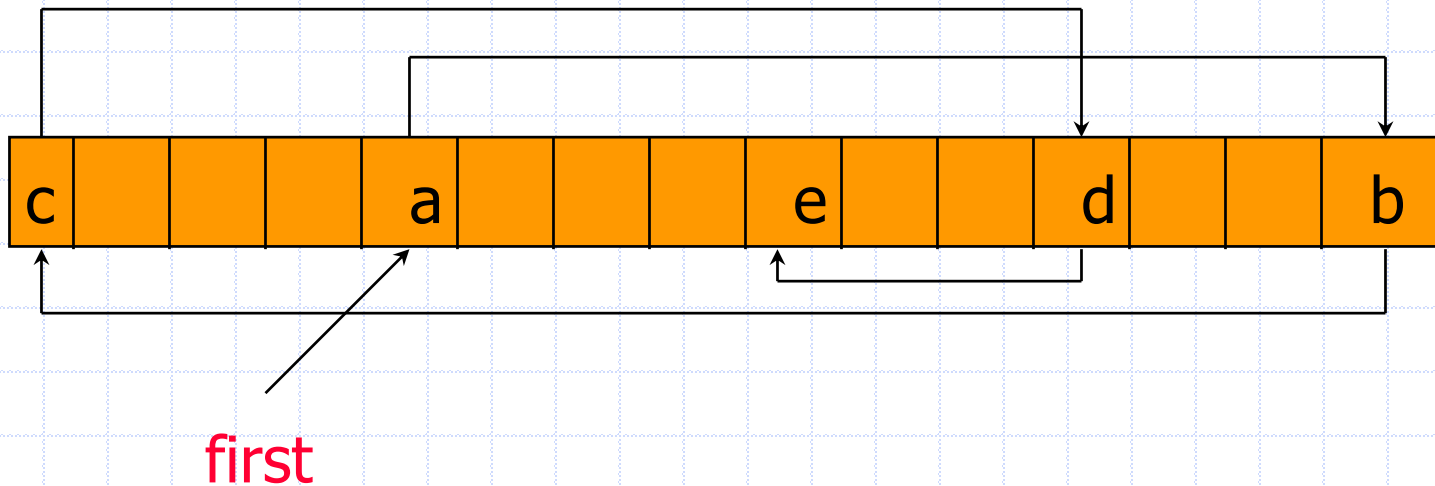
- شمای حافظه برای لیست  $L = (a,b,c,d,e)$
- بازنمایی ترتیبی (آرایه)



- بازنمایی پیوندی



# بازنمایی پیوندی



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

# 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 *i* یک متغیر صحیح و *pi* یک اشاره گر به یک متغیر صحیح است.

و

```
pi = &i;
```

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

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

# 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);
```

---

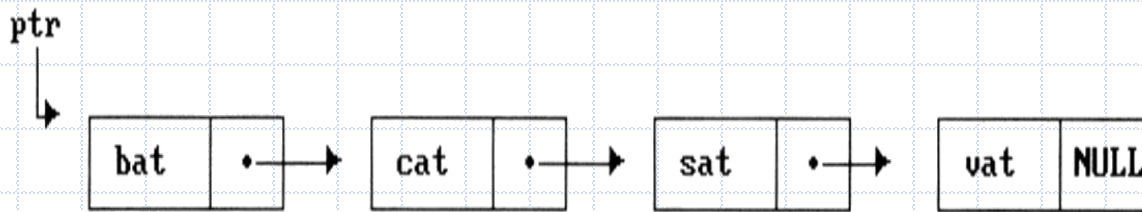
Request  
memory

Free memory



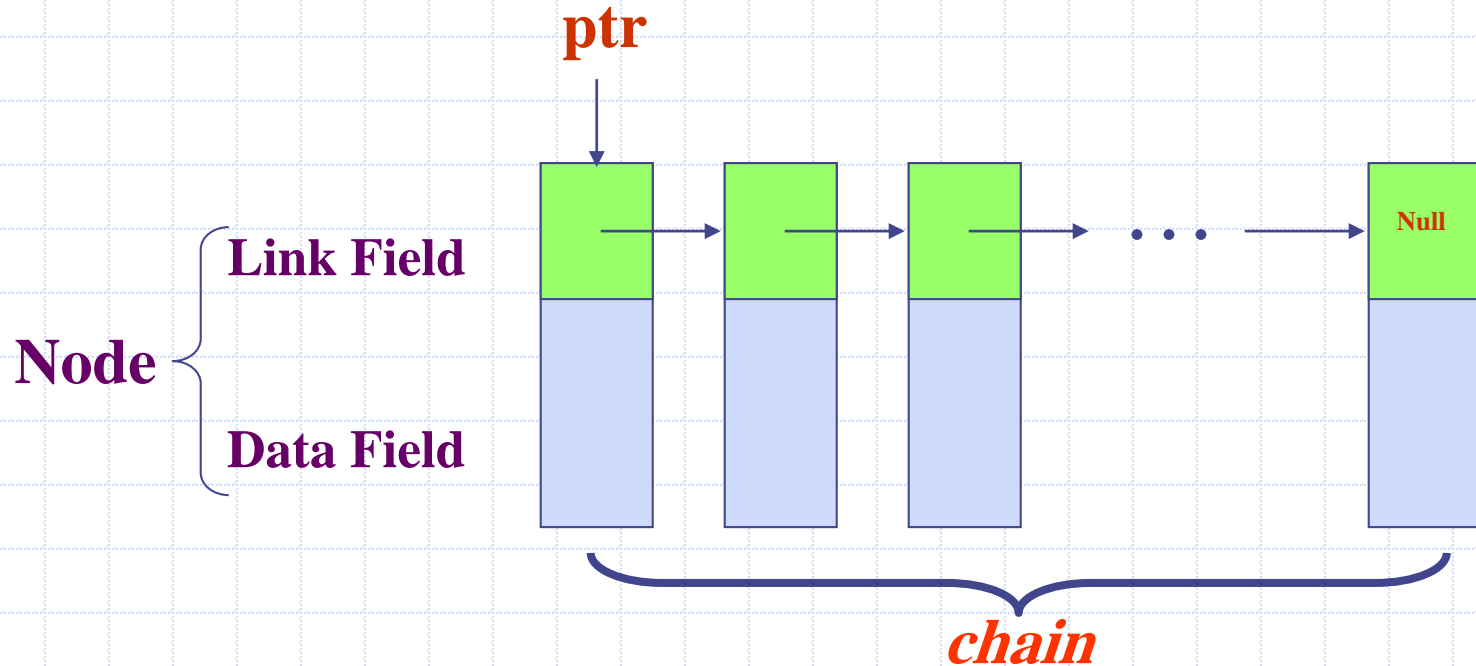
# لیست های تک پیوندی

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

- ✓ 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

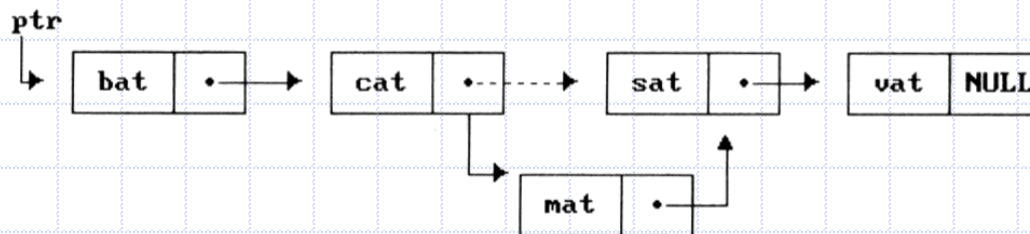
• برای اضافه کردن کلمه **mat** بین **cat** و **sat**:

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

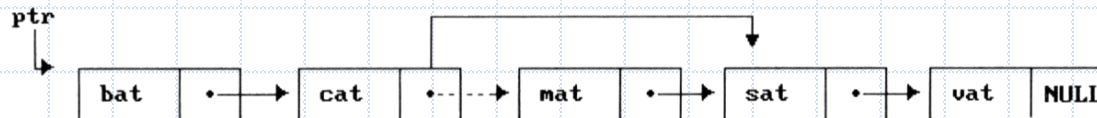
۴- فیلد اتصال گره حاوی **cat** را طوری تنظیم کنید که به **paddr** اشاره کند.



# لیست های تک پیوندی

## حذف mat از لیست

- ✓ 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 آن اشاره دارد
- ✓ ما هیچ داده ای را جابجا نکرده ایم و با وجود آنکه فیلد اتصال **mat** هنوز به **sat** اشاره می کند **mat** دیگر عضو لیست نیست.



# لیست های تک پیوندی

- 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;
```



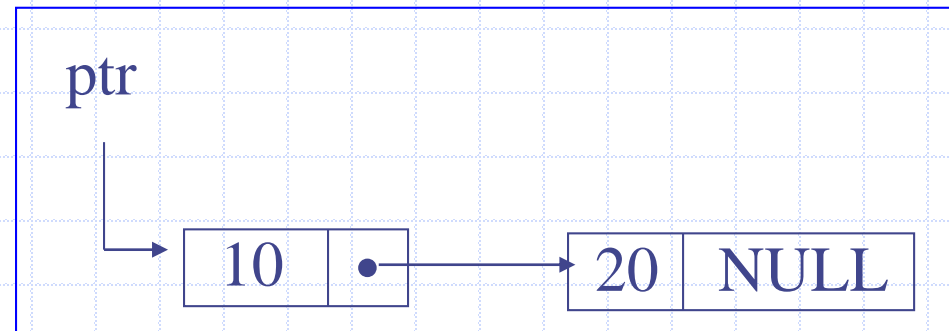
# لیست های تک پیوندی

- **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

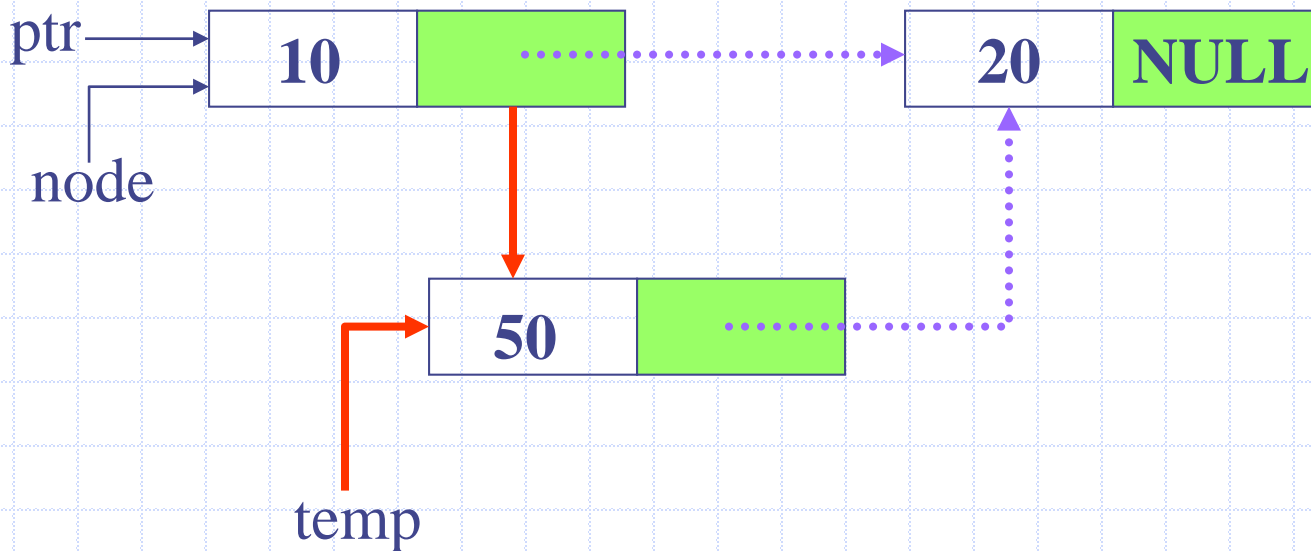
```
list_pointer create2( )  
{  
    /* create a linked list with two nodes */  
    list_pointer first, second;  
    first = (list_pointer) malloc(sizeof(list_node));  
    second = (list_pointer) malloc(sizeof(list_node));  
    second -> link = NULL;  
    second -> data = 20;  
    first -> data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



# لیست های تک پیوندی

- اضافه کردن

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



## لیست های تک پیوندی

- 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));
```

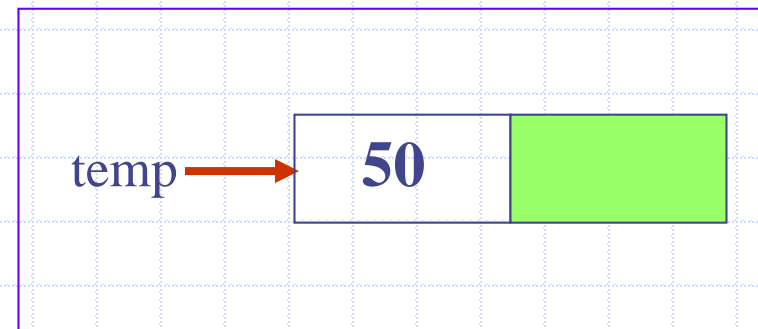
```
if(IS_FULL(temp)){
```

```
    fprintf(stderr, "The memory is full\n");
```

```
    exit(1);
```

```
}
```

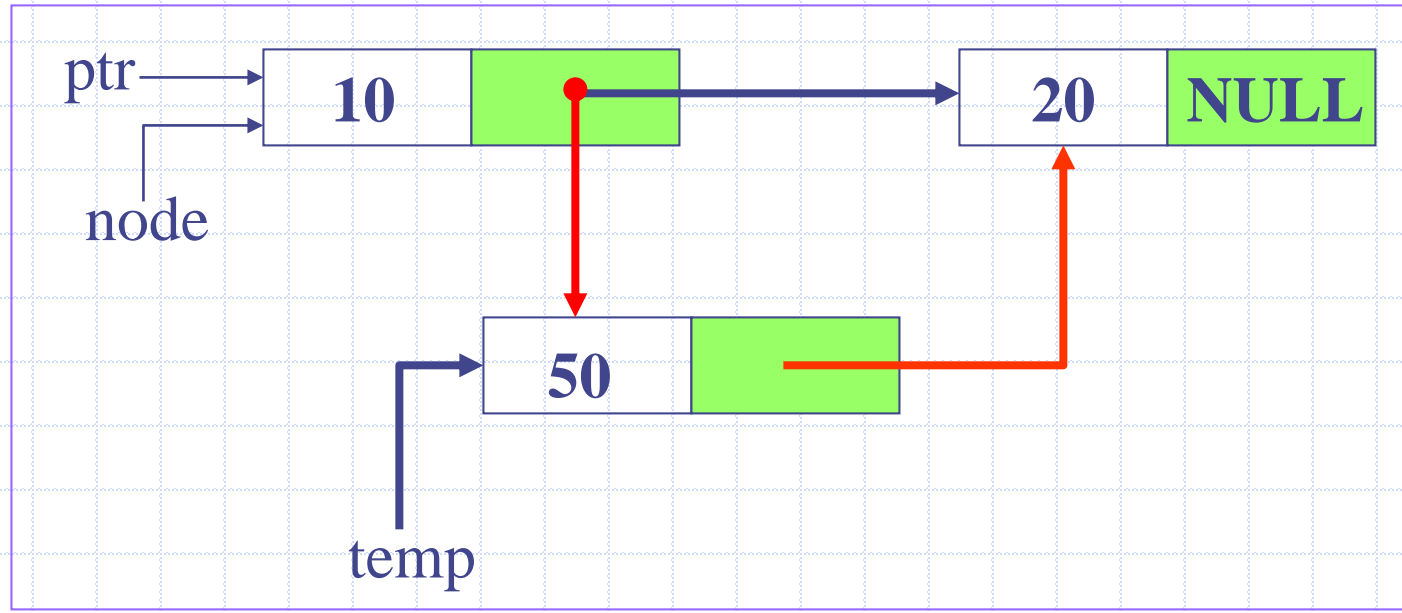
```
temp->data=50;
```





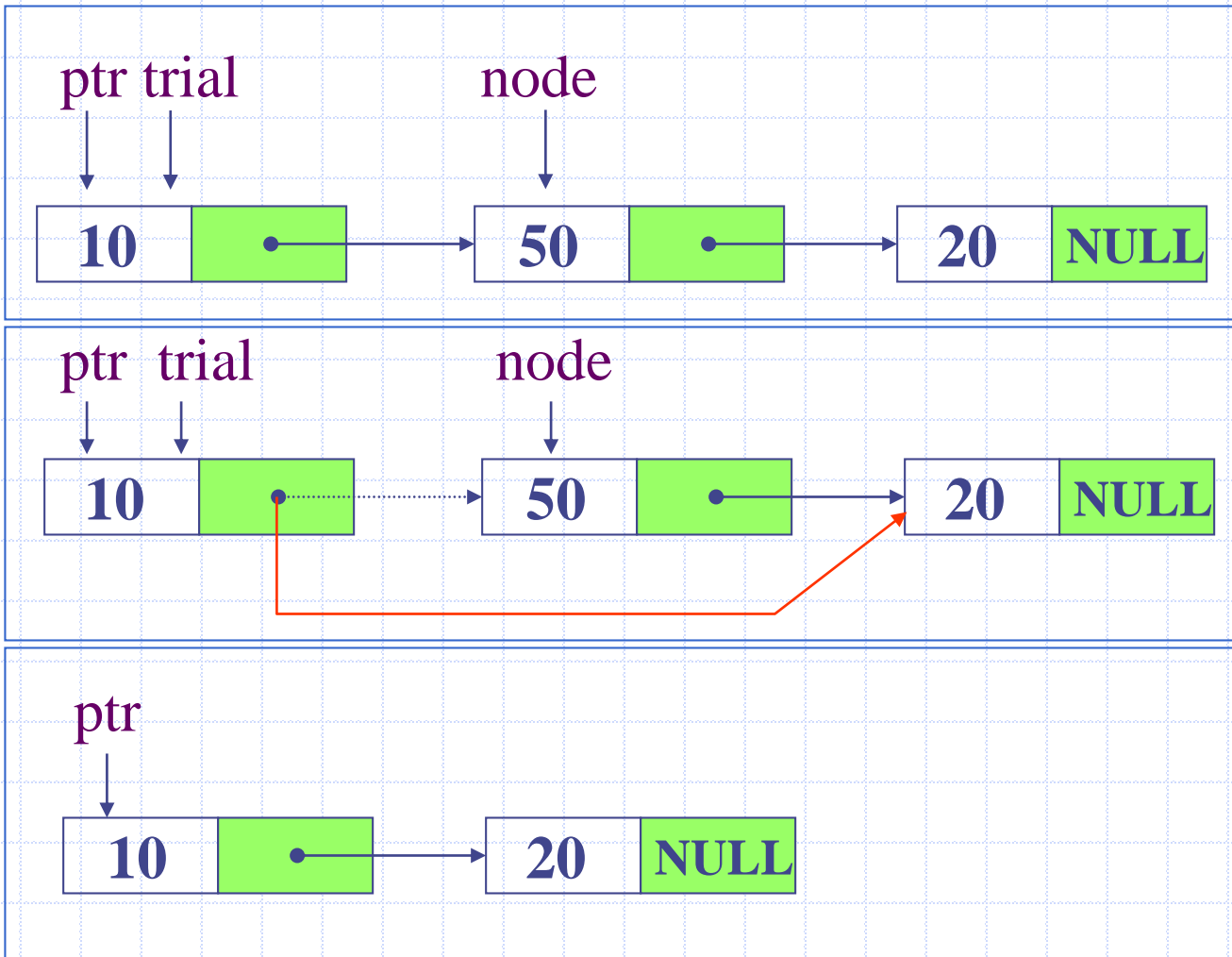
# لیست های تک پیوندی

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



# لیست های تک پیوندی

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

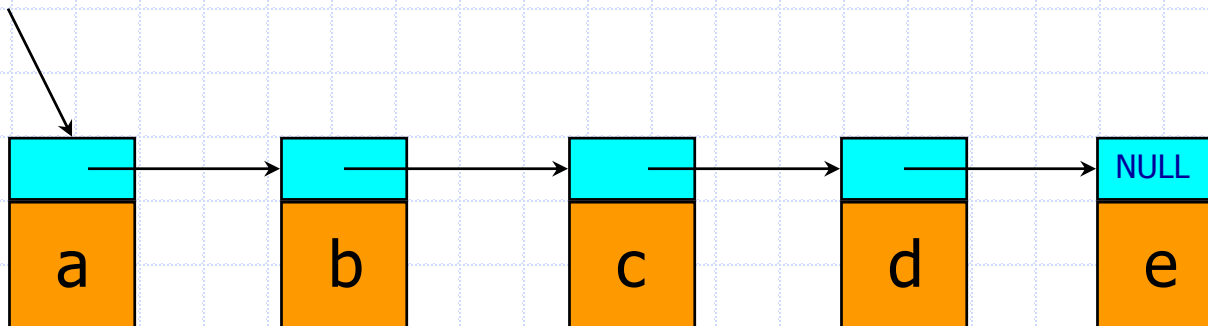


# لیست های تک پیوندی

پیمایش (چاپ) یک لیست پیوندی

Get(0)

first



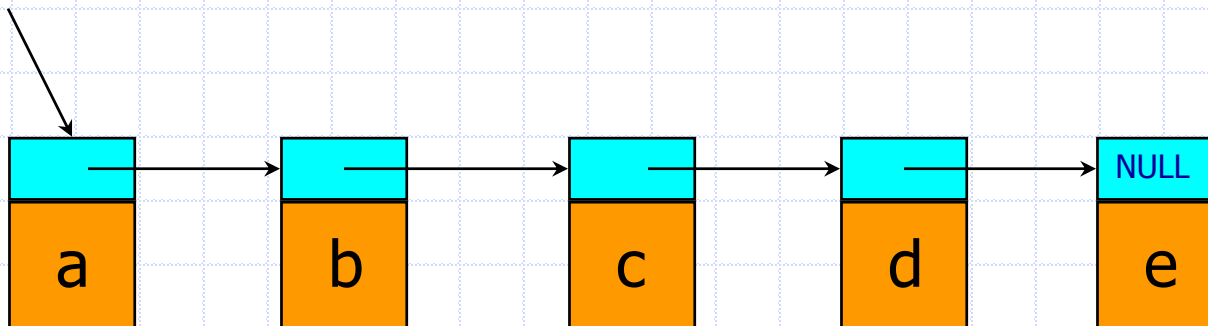
```
desiredNode = first; // gets you to first node  
return desiredNode->data;
```

# لیست های تک پیوندی

• پیمایش (چاپ) یک لیست پیوندی

Get(1)

first



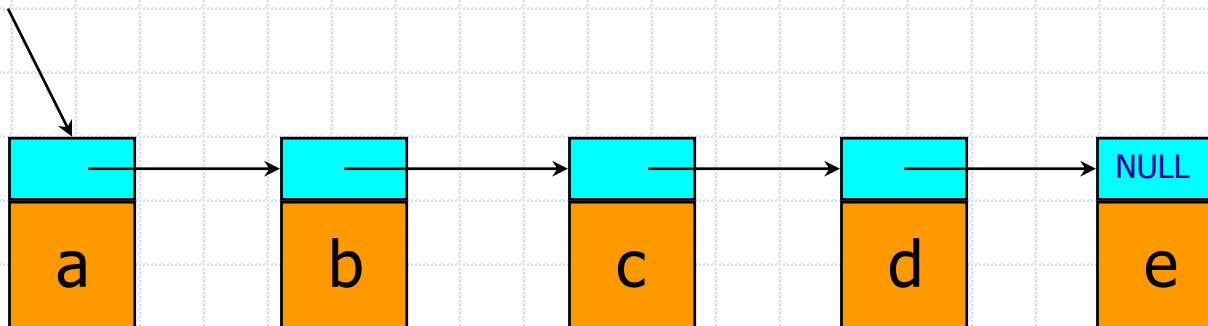
```
desiredNode = first->link; // gets you to second node  
return desiredNode->data;
```

# لیست های تک پیوندی

• پیمایش (چاپ) یک لیست پیوندی

Get(2)

first



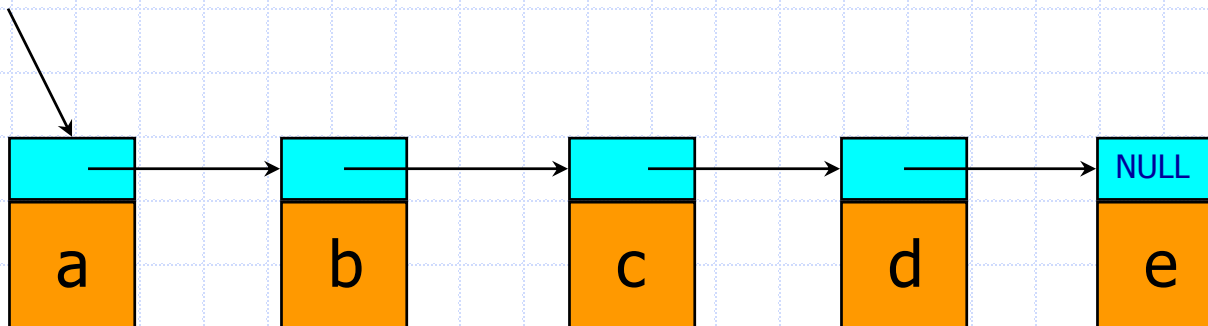
```
desiredNode = first->link->link; // gets you to third node  
return desiredNode->data;
```

# لیست های تک پیوندی

پیمایش (چاپ) یک لیست پیوندی

Get(5)

first



```
desiredNode = first->link->link->link->link->link;
```

```
// desiredNode = NULL
```

```
return desiredNode->data; // NULL.element
```

# لیست های تک پیوندی

• 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");
}
```

# پشته ها و صف های پیوندی

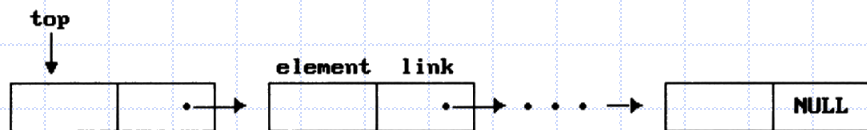
هنگامی که چندین صف و پشته وجود داشته باشد روش ترتیبی کارایی برای بازنمایی آنها وجود ندارد.

در بازنمایی پیوندی جهت اشاره گر برای پشته و صف به صورتی است که عملیات حذف کردن و اضافه کردن گره ها در آنها به اسانی انجام شود.

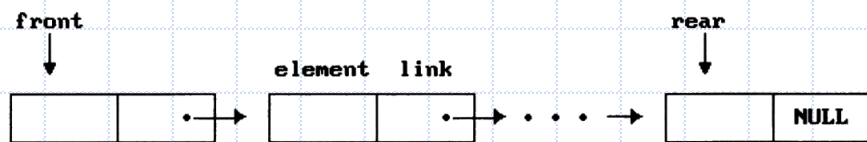
- به اسانی می توانید یک گره را به بالای پشته اضافه و یا از آن حذف کنید.

- 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)



(a) Linked Stack



(b) linked queue

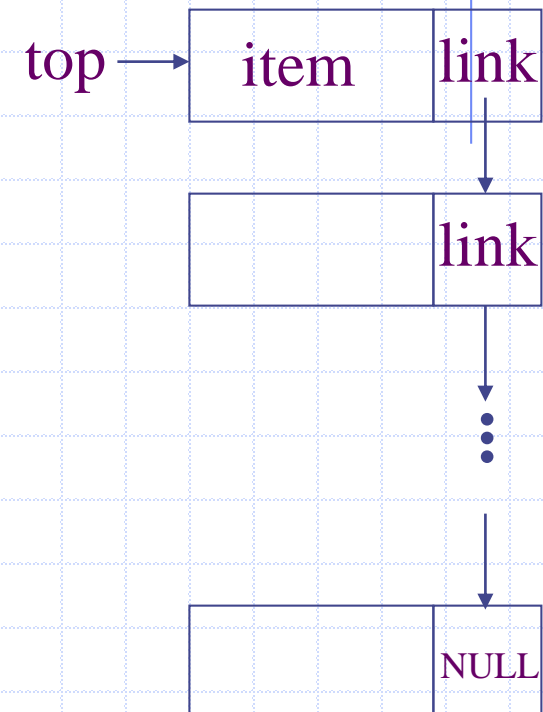


# پشته ها و صف های پیوندی

• 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];
```

Stack



# پشته ها و صف های پیوندی

- 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);
```

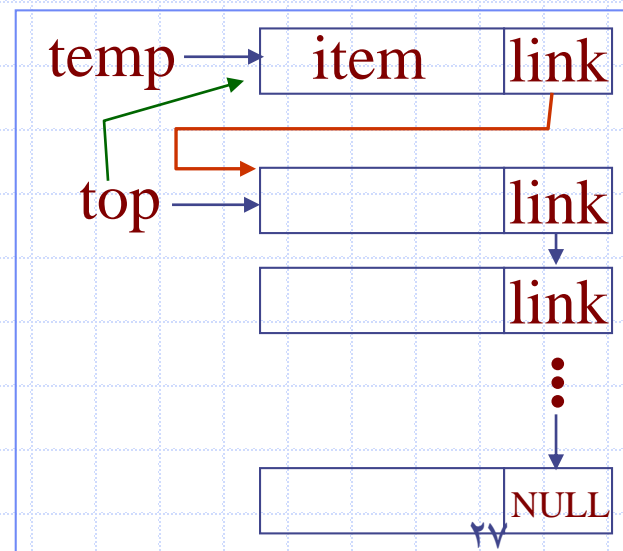
```
}
```

```
temp->item = item;
```

```
temp->link = *top;
```

```
*top= temp;
```

```
}
```



# پشته ها و صف های پیوندی

## • Pop from the linked stack

```
element delete(stack_pointer *top) {  
/* delete an element from the stack */
```

```
stack_pointer temp = *top;
```

```
element item;
```

```
if (IS_EMPTY(temp)) {
```

```
    fprintf(stderr, "The stack is empty\n");
```

```
    exit(1);
```

```
}
```

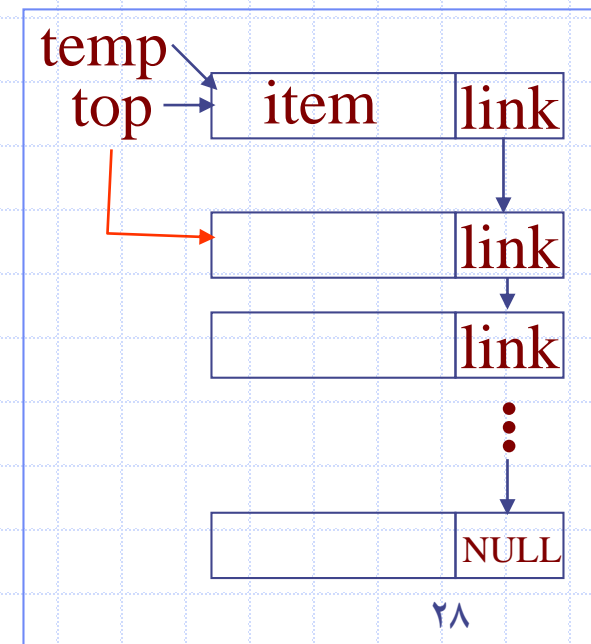
```
item = temp->item;
```

```
*top = temp->link;
```

```
free(temp);
```

```
return item;
```

```
}
```



# پشته ها و صف های پیوندی

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];
```

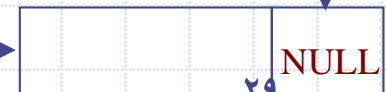
حذف کردن

front

Queue



⋮



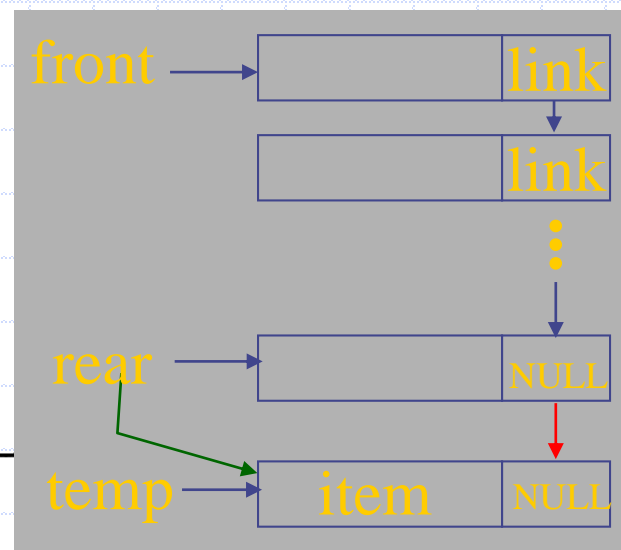
rear

اضافه کردن

# پشته ها و صف های پیوندی

## • اضافه کردن به صف پیوندی

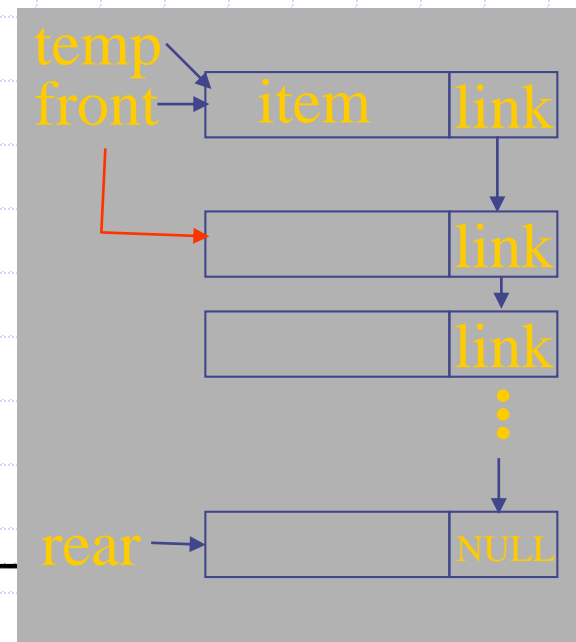
```
void addq(queue_pointer *front, queue_pointer *rear,
          element item)
{
    /* add an element to the rear of the queue */
    queue_pointer temp =
        (queue_pointer) malloc(sizeof(queue));
    if (IS_FULL(temp)) {
        fprintf(stderr, "The memory is full\n");
        exit(1);
    }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



# پشته ها و صف های پیوندی

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

- 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 آن استفاده کرد.

# عملیات روی لیست های پیوندی

- 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

- 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  $O(m)$

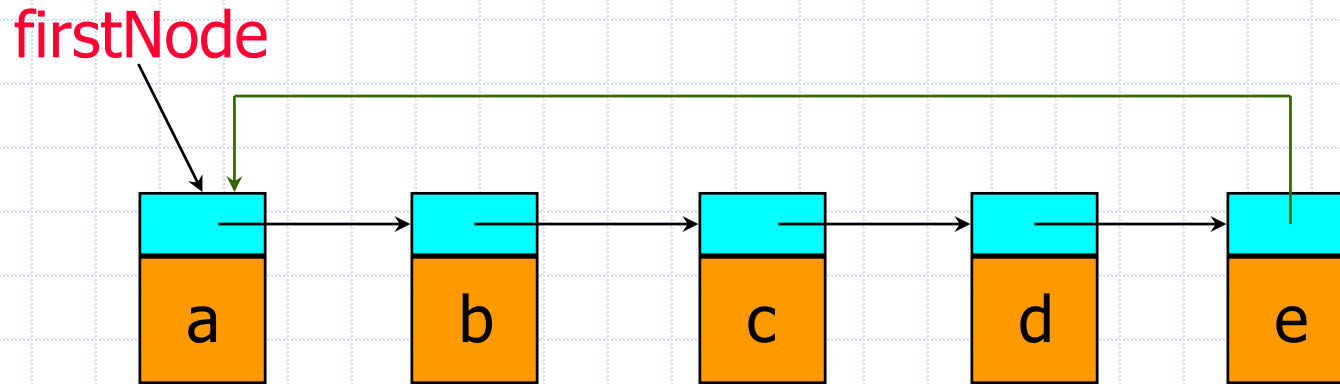
# عملیات روی لیست های پیوندی

- 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

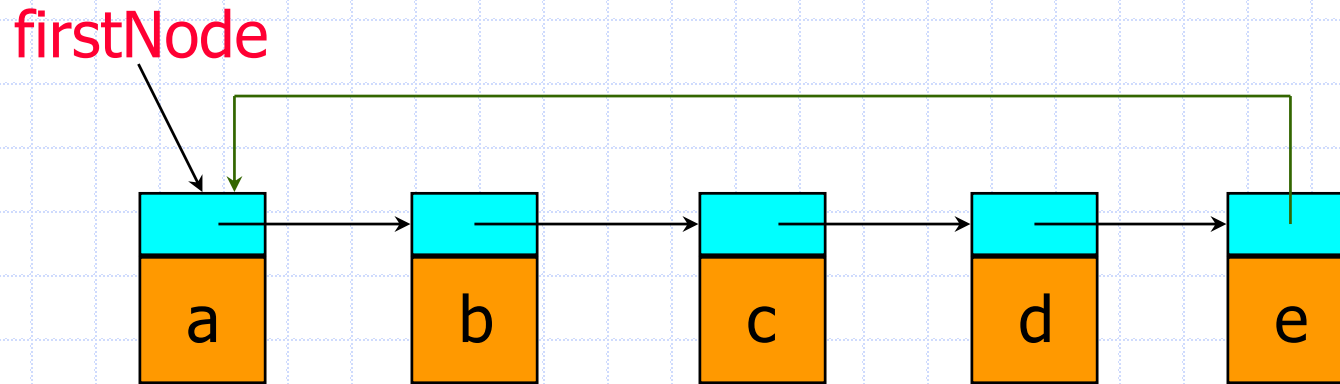
# لیست دایره ای



□ برای بررسی اینکه آیا اشاره گر `current` به گره آخر لیست دایره ای اشاره می کند به جای مقایسه  $(current \rightarrow link == 0)$  مقایسه ی  $(current \rightarrow link == first)$  انجام می شود.

□ الگوریتم های حذف / اضافه کردن از / به لیست دایره ای باید تضمین کند که فیلد اشاره گر آخر به گره اول لیست اشاره کند.

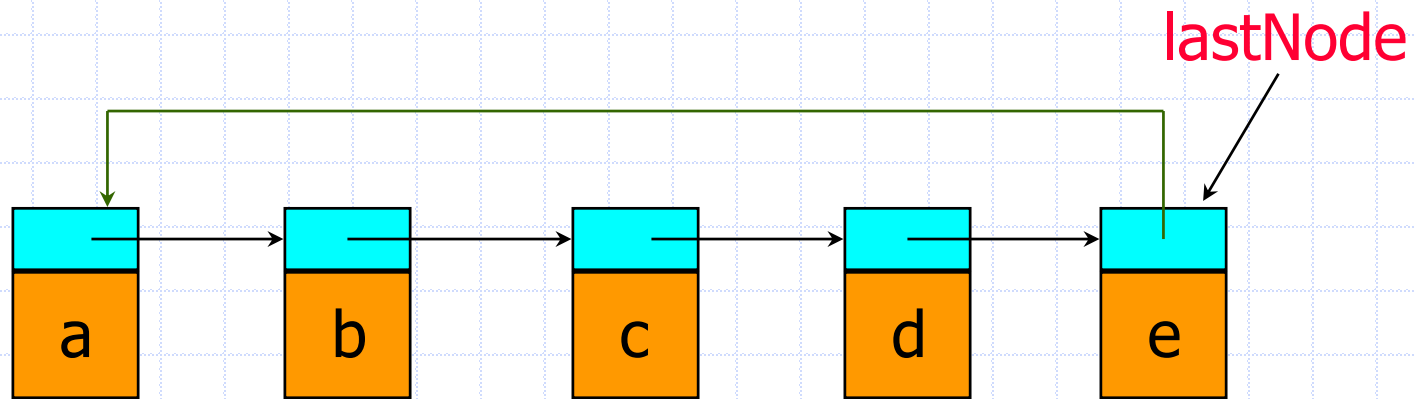
# لیست دایره ای



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



□ 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

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

# لیست دایره ای

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