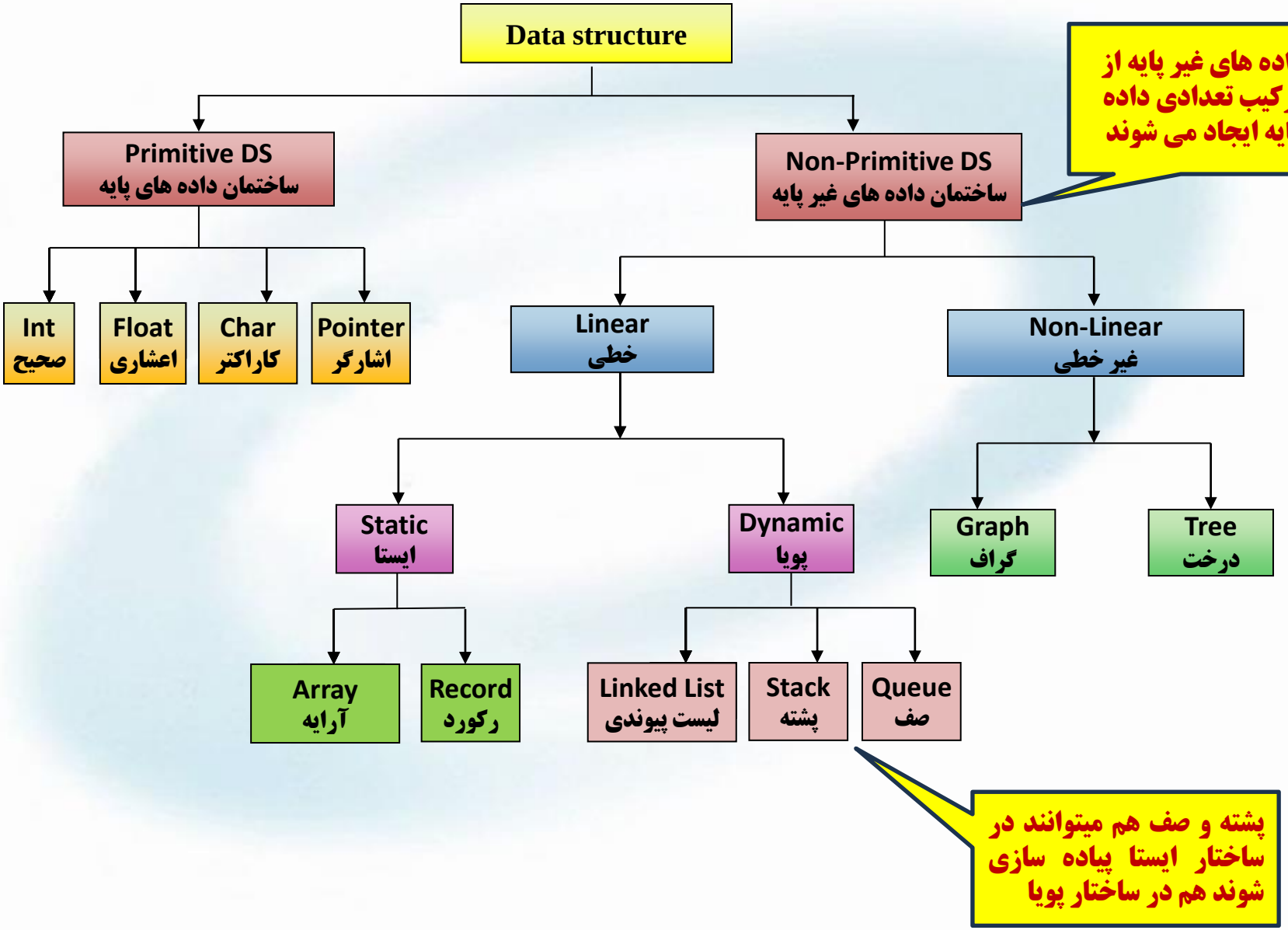
ساختمان داده ها
Data Structure

فصل چهارم
لیست پیوندی Linked List

تهیه و تنظیم : محمد نعیمی

عضو هیات علمی دانشگاه آزاد اسلامی

دسته بندی ساختمان داده ها



تعریف لیست پیوندی

آرایه به عنوان یک ساختمان داده جهت ذخیره مجموعه داده ها استفاده میشود. ویژگی آرایه پیوستگی خانه های آن می باشد.
مزیت: آدرس هر خانه با یک فرمول ریاضی محاسبه و دسترسی به تمام خانه ها سریع می باشد. O(1)
عیب: لازم است ابتدا تعداد خانه ها مشخص باشد و امکان افزودن به خانه ها وجود ندارد چون ممکن است در حافظه، خانه بعد از آرایه خالی نباشد و پیوستگی رعایت نشود. پس باید بیش از نیاز معمول حافظه ابتدا معرفی کرد یا در صورت نیاز خانه خالی نخواهیم داشت.

لیست پیوندی: ساختار لیست پیوندی مانند زنجیر می باشد.

- هر **گره** لیست پیوندی دو قسمت دارد یکی **قسمت داده ای** که مقادیر در آن ذخیره می شود و دیگری **قسمت اشاره گر** که در آن آدرس گره بعد ذخیره می گردد.
- لیست پیوندی هم فقط توسط یک اشاره گر معرفی می شود.

بر خلاف آرایه که عناصر در فواصل ثابتی از هم قرار می گرفتند، در لیست پیوندی عناصر می توانند در هر جای حافظه قرار گیرند و با استفاده از قسمت آدرس محل عنصر بعد را مشخص نمایند.
مزیت: به اندازه نیاز میتوان حافظه گرفته و در صورت عدم نیاز برگرداند.
عیب: برای دسترسی به عناصرداخلی لیست پیچیدگی o(n) خواهیم داشت و فقط دسترسی به عناصر ابتدای لیست O(1) میباشد.
همچنین برای ذخیره هر مقدار باید یک حافظه اضافی برای ذخیره آدرس عنصر بعد در نظر بگیریم.

```
struct node {
        type item;
        node *next;
    }
```

تعریف رکورد گره های لیست

لیست پیوندی توسط یک متغیر (first) از نوع اشاره گر سازماندهی می شود زیرا باید به گره اول لیست اشاره کند و آدرس آن را نشان دهد.
تعریف اشاره گر لیست و مقدار دهی اولیه آن که ابتدا خالی (NULL) است.

```
node * first;
first =NULL;
```

تعریف لیست پیوندی – مثال

ساختاری از یک لیست پیوندی

در این ساختار first یک اشاره گر از نوع لیست است که به اولین گره اشاره میکند(آدرس اولین گره را دارد).
فلش ها بیانگر آدرس گره بعدی هستند. گره آخر چون بعدی ندارد مقدار NULL دارد به معنی هیچ

نمایی از حافظه لیست بالا

first

				6584	47					33					9845	NULL								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
							1856		64												3654		54	
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
			4256		17								2561		5									
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75

- در شکل بالا نمایی از حافظه برای لیست بالا نمایش داده شده است. هر گره لیست به صورت قسمت داده ای (آبی) و اشارگر (زرد) نمایش داده شده است. چون قسمت داده ای int است دو خانه برای آن در نظر گرفته شده است و یک خانه برای آدرس.
- اشارگر لیست پیوندی (first) در آدرس 12 قرار دارد.
- در خانه first عدد 33 نوشته شده یعنی اولین عنصر در آدرس 33 میباشد.
- در خانه 33 عدد 1856 نوشته شده که اولین عدد لیست میباشد و در قسمت زرد رنگ آن عدد 64 نوشته شده یعنی عنصر بعد در آدرس 64 می باشد.
- با همین ترتیب میتوان گفت مقادیر لیست ما به ترتیب در خانه های 33,64,5,47,53,17 قرار دارد.
- در قسمت آدرس خانه 17 بجای آدرس عبارت NULL قرار دارد یعنی عنصر بعد وجود ندارد و این آخرین خانه لیست می باشد.

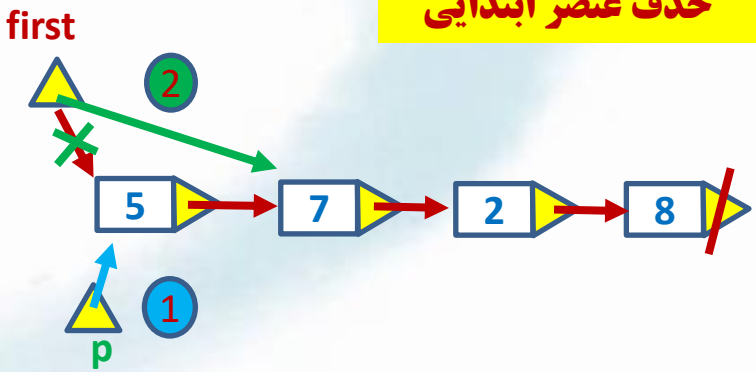
نکات کلیدی در خصوص دستورات لیست پیوندی

- همیشه با رسم شکل ابتدا ساختار عملیات را مشخص کنید. فلش های سبز و آبی کاری است که باید انجام شود.
- در ترتیب انجام فلش ها ابتدا باید لینک گره یا اشاره گر جدید به لیست متصل شود بعد لینک های گره های لیست اصلاح شود.
- همیشه در حذف ابتدا باید گره ای که قرار است حذف شود در اشاره گری ذخیره شود.

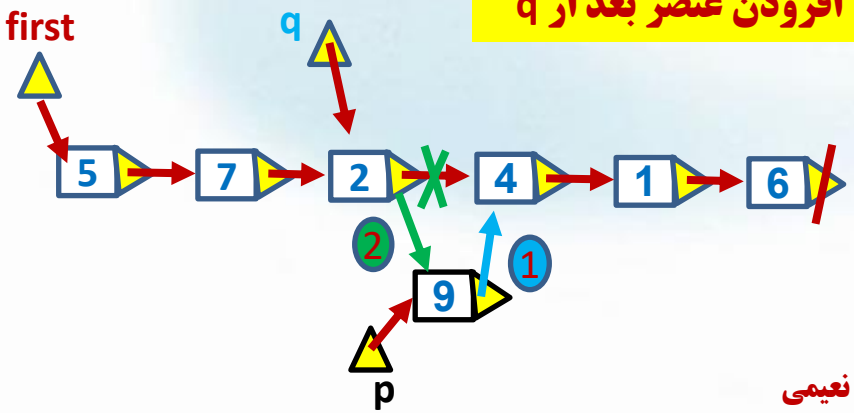
شکلهای زیر از نظر تصویری نحوه عملیات را نشان می دهند

افزودن عنصر به ابتدا

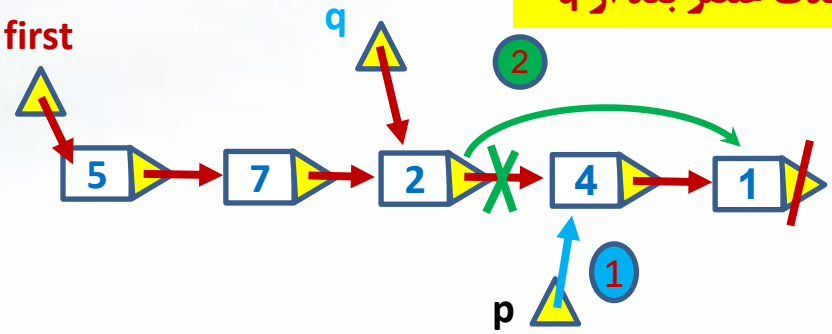
حذف عنصر ابتدایی



افزودن عنصر بعد از q



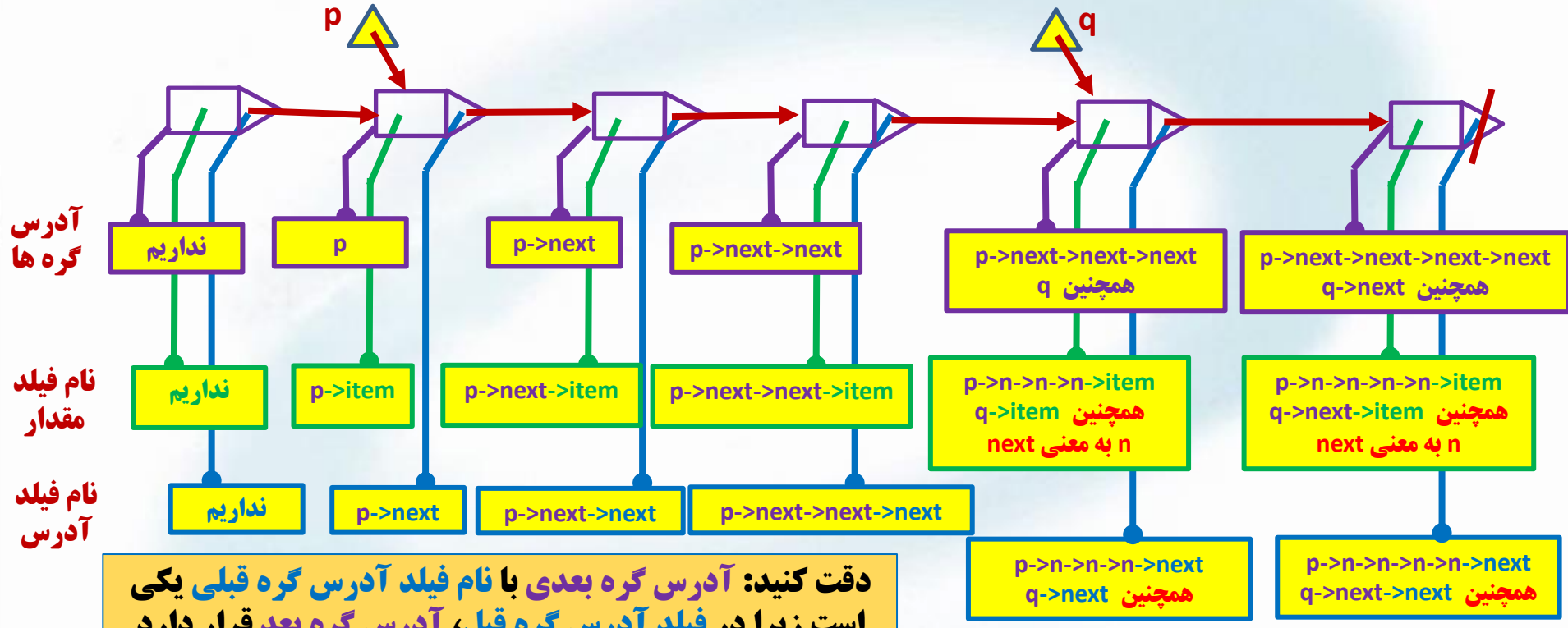
حذف عنصر بعد از q



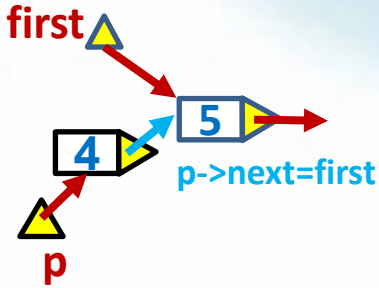
نکات کلیدی در خصوص دستورات لیست پیوندی – ادامه

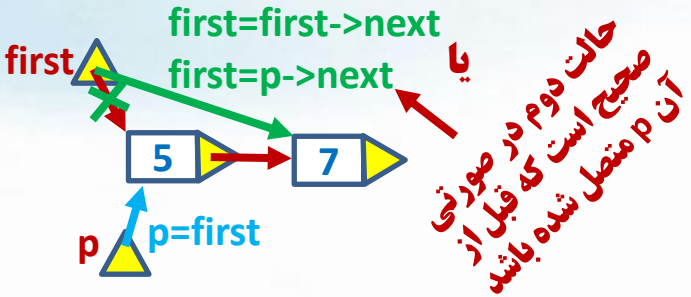
- برای نوشتن دستورات مربوط به هر فلش نیاز به یک تساوی داریم

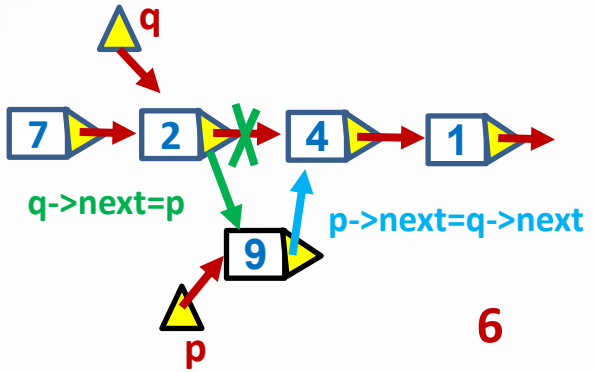
آدرس گره ای که نوک فلش با آن اشاره میکند = نام فیلدی که فلش از آن خارج شده



دقت کنید: آدرس گره بعدی با نام فیلد آدرس گره قبلی یکی است زیرا در فیلد آدرس گره قبل، آدرس گره بعد قرار دارد







توابع لیست پیوندی– افزودن به ابتدا و بعد از q

```
void addfirst(node * &first, type x)
{
   node *p;
   p=new();
   p->item=x;
   p->next=first;
   first=p;
}
```

ایجاد فیزیکی گره

اتصال منطقی

تابع افزودن عنصر x به ابتدای لیست پیوندی

پیچیدگی O(1)

همیشه ابتدا باید اشاره گر گره جدید را به لیست وصل کنیم بعد اشاره گر گره لیست به آن اشاره کند.

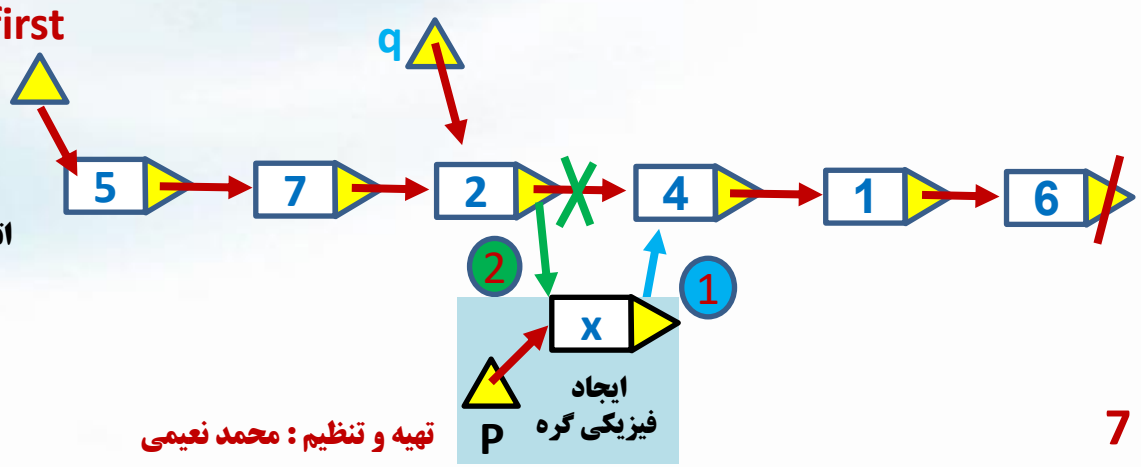
در ادامه بجای نوشتن تابع به صورت کامل فقط دستورات اتصال منطقی را مینویسیم.
قطعا قبل از اتصال منطقی باید دستورات ایجاد فیزیکی نوشته شود

دستورات افزودن گره p به بعد از گره q

ایجاد فیزیکی گره

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

اتصال منطقی



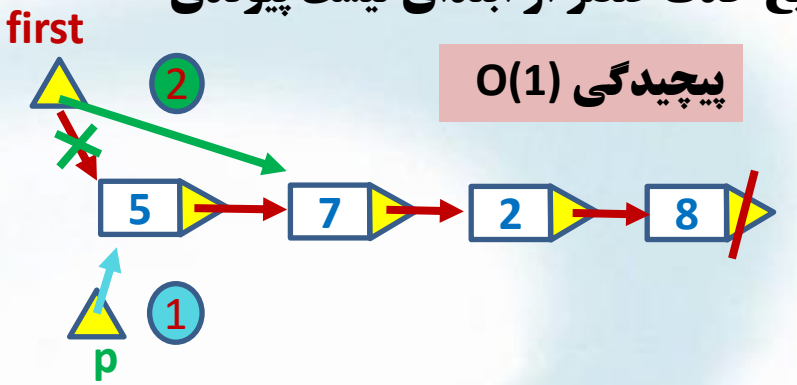
توابع لیست پیوندی – حذف از ابتدا و بعد از q

```
type delfirst(node * &first)
{
   node *p;
   type x;
   if(first==NULL)
       cout<<"empty";
   else
    {
      p=first;          1
      first=first->next; 2
      x=p->item;
      free(p);
      return (x);
    }
}
```

حذف منطقی

حذف فیزیکی گره

تابع حذف عنصر از ابتدای لیست پیوندی



پیچیدگی O(1)

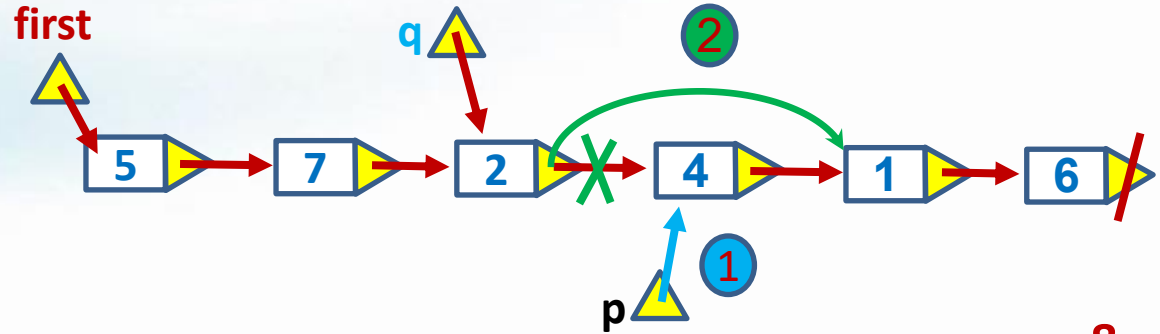
در ادامه بجای نوشتن تابع به صورت کامل فقط دستورات حذف منطقی را مینویسیم.
قطعا بعد از حذف منطقی باید دستورات حذف فیزیکی نوشته شود

دستورات حذف گره بعد از گره q (برای حذف یک گره باید آدرس عنصر قبل را حتما داشته باشیم)

```
p=q->next;          1
q->next=p->next;    2
```

حذف منطقی

حذف فیزیکی گره



توابع لیست پیوندی – جستجو – چاپ – معکوس

تابع چاپ عناصر یک لیست

```
void printlist(node * first)
{
   node *q;
   q=first;
   while(q!=NULL)
       {
           cout<<q->item;
           q=q->next;
       }
}
```

پیچیدگی O(n)

تابع چاپ عناصر یک لیست از آخر به اول به صورت بازگشتی

```
void printlist_rec(node * first)
{
   if (first!=NULL)
      {
          printlist_rec(first->next);
          cout<<first->item;
      }
}
```

پیچیدگی O(n)

تابع یافتن محل عنصر x در لیست پیوندی

```
node* find(node * first, int x)
{
   node *q;
   q=first;
   while(q!=NULL)
       {
           if (q->item.key==x)
               return (q);
           q=q->next;
       }
   return (NULL);
}
```

پیچیدگی O(n)

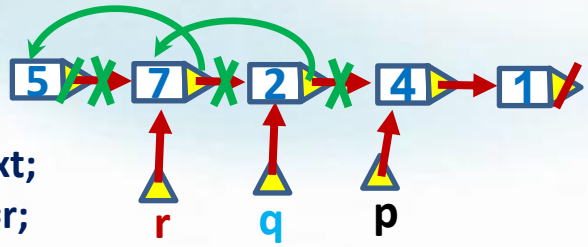
در مقایسه امکان مقایسه رکورد با هم وجود ندارد.
اگر type از نوع داده های پایه (Primitive) باشد میتوان در قسمت if خود داده را مقایسه کرد.
ما برای مقایسه از فیلد key استفاده کردیم که حالتی کلی تر و مربوط به رکورد ها می باشد.
x را در این حالت جستجو از نوع int فرض کرده ایم.

تابع معکوس کردن لیست

```
void invers(node * &first)
{
   node *p,*q,*r;
   p=first;
   q=NULL;
   while(p!=NULL)
       {
         r=q;
         q=p;
         p=p->next;
         q->next=r;
       }
   first=q;
}
```

پیچیدگی O(n)

در هر دور از while
r q p به ترتیب آدرس سه خانه پشت سر هم هستند



توابع لیست پیوندی – افزودن به لیست مرتب

```
void addsortlist(node * &first, type x)
{
   node *p,*b,*q;
   p=new();
   p->item=x;
   q=first;
   while(q!=NULL)
       {
           if (q->item.key>x.key)
               break;
           b=q;
           q=q->next;
       }
   if (q==first)
     {
        p->next=first;
        first=p;
     }
   else
     {
        p->next=q;
        b->next=p;
     }
}
```

ایجاد فیزیکی

b جای q می آید

q یک خانه به جلو

x از اولین عنصر کوچکتر است

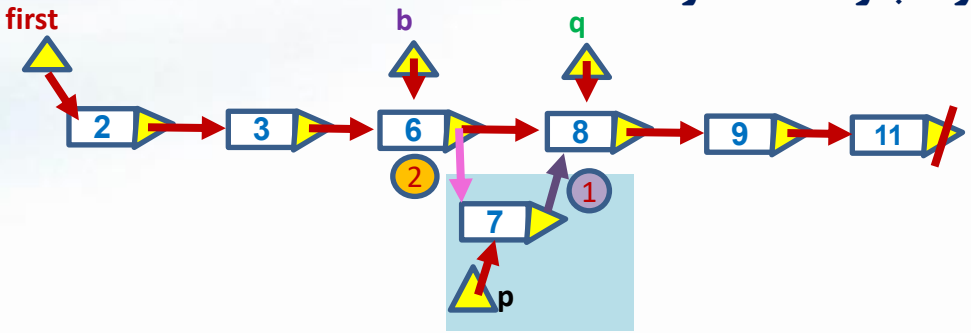
تابع افزودن عنصر در مکان خود در یک لیست پیوندی مرتب — **پیچیدگی O(n)**

باید از ابتدای لیست حرکت کرد. وقتی به اولین عنصری که بزرگتر از x بود رسیدیم باید قبل از آن گره x را اضافه کنیم. از آنجایی که لیست پیوندی مانند خیابان یک طرفه است امکان برگشت به عقب وجود ندارد.

برای همین اشاره گر b هنگام حرکت q به جلو، جای q را گرفته بعد q حرکت میکند و عملا b خانه قبل از q می باشد.ما عنصر را به خانه قبل از q (بعد از b) اضافه میکنیم.

شکل حالتی که عنصر از همه کوچکتر باشد یا لیست خالی باشد و x بخواهد ابتدای لیست اضافه شود

شکل حالتی که عنصر جایی بعد از گره b بخواهد اضافه شود.



توابع لیست پیوندی – حذف عنصر x از لیست

تابع حذف عنصر در یک لیست پیوندی

پیچیدگی O(n)

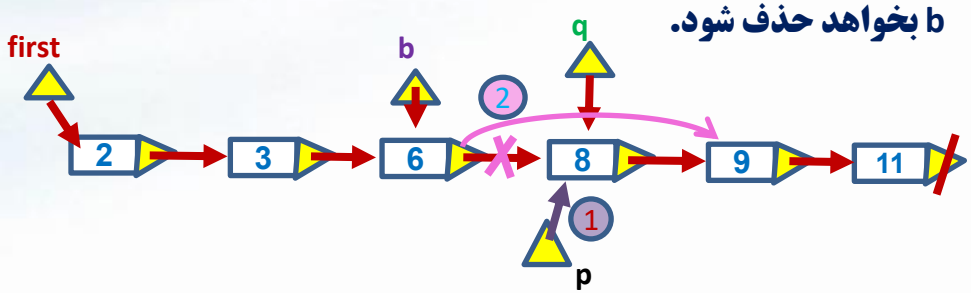
در حذف ابتدا باید مکان عنصر را پیدا کنیم.
برای حذف هر عنصر باید آدرس خانه قبل را داشته باشیم تا اشاره گر آن را به گره بعد وصل کنیم.
با شیوه ای شبیه صفحه قبل b خانه قبل از q خواهد بود و گره q باید حذف شود.

```
type del_list(node * &first, int x)
{
   node *p,*b,*q;
   q=first;
   while(q!=NULL)
       {
           if (q->item.key==x.key)
               break;
           b=q;
           q=q->next;
       }
   if (q==NULL)
       cout<<"not found";
   else
     {
        if (q==first)
         {
            p=first;          1
           first=p->next;     2
         }
        else
         {
           p=q;               1
           b->next=q->next;   2
         }
       type y=p->item;
       free(p);
       return (y);
     }
}
```

حذف فیزیکی

شکل حالتی که عنصر اول لیست بخواهد حذف شود

شکل حالتی که عنصر بعد از گره b بخواهد حذف شود.



ساخت پشته و صف با استفاده از لیست پیوندی

پشته با لیست پیوندی

از آنجایی که در پشته ورود و خروج داده ها از یک سمت است کافیست برای پیاده سازی پشته تابع افزودن به ابتدای لیست و حذف از ابتدای لیست را به عنوان push و pop در نظر بگیریم. در لیست پیوندی پر بودن پشته نداریم مگر حافظه سیستم تمام شود و شرط خالی بودن هم می شود first==NULL

صف با لیست پیوندی

در صف ورود و خروج داده ها از دو سمت متفاوت انجام می پذیرد حالا باید ببینیم ابتدای لیست را ابتدای صف قرار دهیم یا انتهای لیست را. برای این تصمیم باید پیچیدگی عملیات ها را بررسی کنیم.

- افزودن و برداشتن از ابتدای لیست پیچیدگی O(1) دارد.
- افزودن به انتهای صف در صورتی که لیست یک اشاره گر اضافی برای تعیین آدرس آخرین عنصر لیست داشته باشد دارای پیچیدگی o(1) خواهد بود.
- حذف از انتهای لیست همیشه دارای پیچیدگی O(n) است زیرا ما آدرس خانه قبل را جهت حذف اخرین عنصر میخواهیم و باید لیست را از ابتدا پیمایش کنیم تا به خانه قبل از آخرین خانه لیست برسیم و داشتن آدرس خانه آخر کمکی نمیکند.

با توجه به پیچیدگی های عنوان شده باید افزودن از انتهای لیست و برداشتن از ابتدای لیست انجام شود تا پیچیدگی هر دو عملیات o(1) شود.

- اشاره گر های لیست هم f جهت ابتدا و r جهت انتها بوده و مقدار اولیه آنها f=r=NULL می باشد

دستورات افزودن به صف

ایجاد فیزیکی گره p

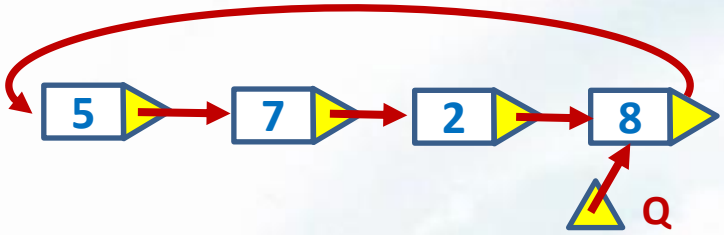
```
if (f==NULL)  صف خالی است
  f=r=p;  تک عنصر هم اولین است هم آخرین
else
 {
    r->next=p;  p بعد از آخرین عنصر قرارمیگیرد
    r=p;  p آخرین عنصر می شود
 }
```

دستورات حذف از صف

```
if (f==NULL)
   cout<<"empty";
else
 {
    p=f;
    if (f==r)  تک عنصر در صف
       f=r=NULL;  صف خالی می شود
     else
       f=f->next;  نفر بعد اولین عنصر می شود
    حذف فیزیکی گره p
 }
```

لیست حلقوی – صف حلقوی

برای دسترسی به آخرین عنصر در لیست پیوندی یا باید کل لیست را پیمایش کنیم (پیچیدگی o(n)) یا یک متغیر اضافه در انتهای لیست قرار دهیم. در ادامه لیست حلقوی را برای طراحی صف معرفی میکنیم.
در صف حلقوی اشاره گر گره آخر به اولین گره لیست اشاره میکند و اشاره گر لیست به آخرین عنصر

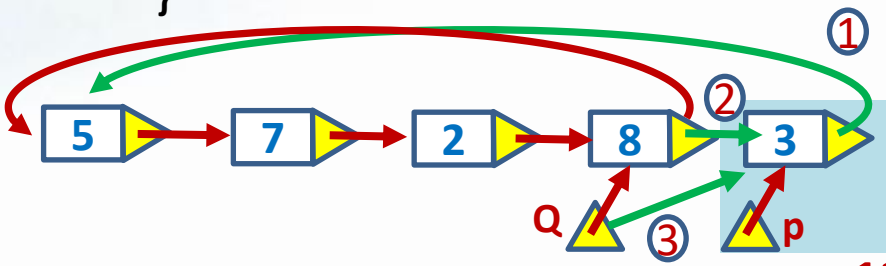


عنصر آخر صف Q
عنصر ابتدای صف Q->next

دستورات افزودن به صف حلقوی

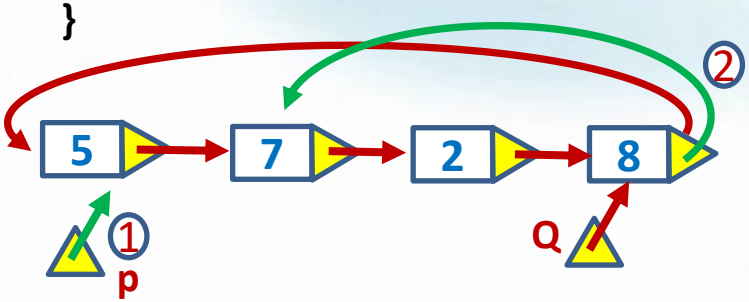
ایجاد فیزیکی گره p

```
if (Q==NULL)  صف خالی است
  {
    p->next=p;  تک عنصر (عنصر بعد از خودش خودش است)
    Q=p;
  }
else
  {
     p->next=Q->next;          ①
     Q->next=p;  p بعد از آخرین عنصر قرارمیگیرد ②
     Q=p;  p آخرین عنصر می شود ③
  }
```



دستورات حذف از صف حلقوی

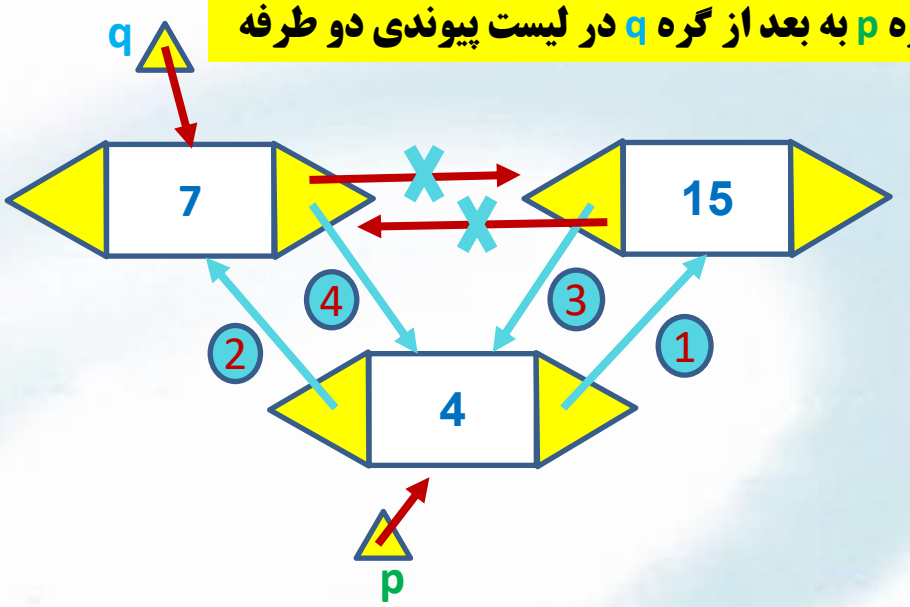
```
if (Q==NULL)
   cout<<"empty";
else
  {
     p=Q->next;  ①
     if (Q->next==Q)  تک عنصر در صف
        Q=NULL;  صف خالی می شود
      else
        q->next=Q->next->next;  ②
     حذف فیزیکی گره p
  }
```



توابع لیست پیوندی دو طرفه

دستورات افزودن گره p به بعد از گره q در لیست پیوندی دو طرفه

```
p->next=q->next;  1
p->back=q;        2
q->next->back=p;  3
q->next=p;        4
```



دستورات حذف گره با آدرس p در لیست پیوندی دو طرفه

```
p->next->back=p->back;  1
p->back->next=p->next   2
```