

فصل دوم

لیست

لیست‌های پیوندی (Linked List)

به طور کلی برای ذخیره‌سازی انواع داده‌ها در حافظه اصلی (RAM) از دو نوع ساختار می‌توان استفاده کرد:

۱- آرایه‌ها

۲- لیست‌های پیوندی

آرایه

هر آرایه لیست پشت سرهمی از داده‌ها است که همگی داده‌ها از یک نوع بوده و ناحیه‌ای پیوسته از حافظه را در اختیار دارند.

نکات مهم در آرایه‌ها:

۱- تعداد عناصر هر آرایه همیشه محدود و مشخص و ایستا است و در تمام طول برنامه ثابت می‌باشد. (عیب)

۲- عملیات حذف و درج یک داده دلخواه در خانه‌ای از آرایه n عضو نیاز به شیفت دارد. (عیب)

عملیات	تعداد شیفت مورد نیاز	متوسط شیفت	مرتبه اجرایی
درج داده در خانه k ام	$n - k + 1$	$\frac{n + 1}{2}$	$o(n)$
حذف داده از خانه k ام	$n - k$	$\frac{n - 1}{2}$	$o(n)$

مثال:

1	2	3	4	5	6
10	20	30	40	50	60

حذف 40 از محل $k = 4$ ام از آرایه $n = 6$ تایی نیاز به $n - k = 6 - 4 = 2$ شیفت دارد.

1	2	3	4	5	6	7
10	20	30	40	50	60	

↑
70

درج 70 در محل $k = 4$ ام از آرایه $n = 6$ عضوی نیاز به $n - k + 1 = 6 - 4 + 1 = 3$ شیفت دارد.

دقت کنید: عملیات همراه با **شیفت عناصر** در شرایطی که n بزرگ باشد همواره **هزینه بالایی** دارد.

۳- عملیات جستجو به دنبال یک داده دلخواه در آرایه‌ها با یکی از دو روش زیر انجام می‌گیرد:

الف) جستجوی خطی (ترتیبی) در آرایه‌های مرتب یا نامرتب با زمان $O(n)$

ب) جستجوی دودویی (باینری) در آرایه‌های مرتب با زمان $O(\log_2 n)$ (مزیت)

مزیت در اینجا، امکان استفاده از **جستجوی دودویی** با زمان کمتر و سرعت بیشتر می‌باشد.

۴- دسترسی به داده‌های هر آرایه به صورت تصادفی و دلخواه به راحتی توسط اندیس آرایه انجام شده و دسترسی مستقیم و با زمان $O(1)$ خواهد بود. (مزیت)

۵ - روش‌های مختلفی برای مرتب‌سازی آرایه‌ها از قبیل: مرتب‌سازی حبابی - مرتب‌سازی انتخابی - مرتب‌سازی سریع - مرتب‌سازی درجی - ... وجود دارد. (مزیت)

لیست‌های پیوندی (Linked List)

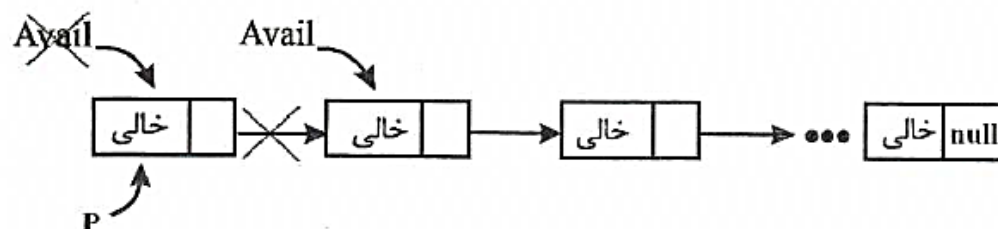
لیست پیوندی، دارای عناصر (رکوردها) مختلفی از حافظه پویا (Heap) بوده که لزوماً عناصر آن کنار هم قرار نگرفته‌اند.

لیست خانه‌های آزاد حافظه و اشاره گر (Avail)



آماده سازی و ایجاد و تخصیص یک گره جدید

از آنجایی که هر گره (node) یک رکورد از فضای پویای (Heap) می باشد ، برای ایجاد و تخصیص گره‌ای با اشاره گر دلخواه (مانند : p) از فضای آزاد و قابل دسترس (Available) با توجه به زبان‌های مختلف برنامه سازی می توان بصورت زیر عمل کرد:



-
- 1) if $Avail = Null$ then write 'overflow' and exit
 - 2) $p = Avail$, $Avail = Link(Avail)$
-

نحوه دسترسی به فیلدهای هر گره

در صورتی که p اشاره گر به گره ای دلخواه باشد ، برای دسترسی به فیلدهای گره

زبان پاسکال	زبان C	زبان الگوریتمی
$P \wedge .Data$	$P \rightarrow Data$	$Data[P]$
$P \wedge .Link$	$P \rightarrow Link$	$Link[p]$

در بعضی مراجع یا تست ها به جای $Data$ از کلمه $info$ استفاده می کنند.

مثال : در صورتی که گره با اشاره گر p به صورت

20	nil or null or 0
----	------------------

 باشد برای دسترسی به گره ها خواهیم داشت

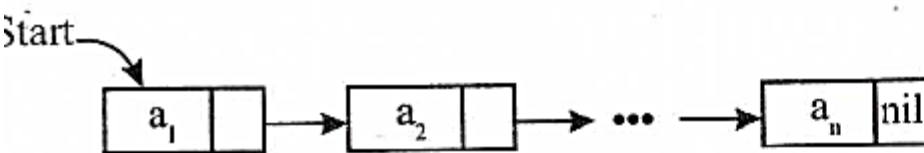
زبان الگوریتمی	زبان پاسکال	زبان C
$\text{Data}[P] = 20;$	$P.^{\wedge}.\text{Data} := 20;$	$P \rightarrow \text{Data} = 20$
$\text{Link}[P] = 0;$	$P.^{\wedge}.\text{Link} := \text{nil};$	$P \rightarrow \text{Link} = \text{null}$

0, nil, null یک مفهوم را دارند.

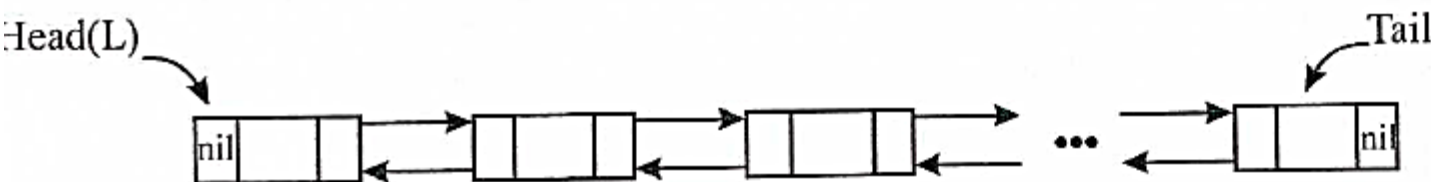
انواع لیست‌های پیوندی

لیست‌های خطی

۱- لیست پیوندی یک طرفه (خطی): در هر گره فقط یک فیلد اشاره‌گر وجود دارد که آن هم آدرس گره بعدی را دارد.



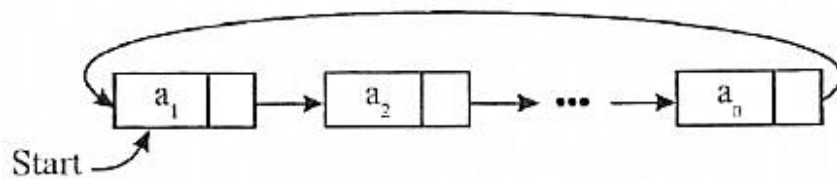
۲- لیست پیوندی دو طرفه (خطی): در هر گره دو فیلد اشاره‌گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی اشاره می‌کند.



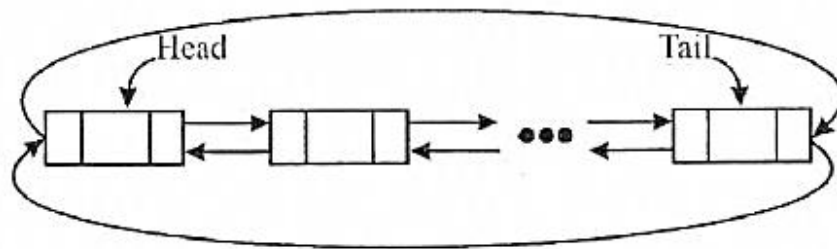
لیست‌های دوری (حلقوی)

- ۱- در لیست حلقوی یک طرفه اشاره‌گر **Link** در گره آخر آدرس گره اول را داشته و به آن اشاره می‌کند.
- ۲- در لیست حلقوی دو طرفه اشاره‌گر سمت راست (**Rlink**) در گره آخر به گره اول و اشاره‌گر سمت چپ (**Llink**) در گره اول به گره آخر اشاره می‌کند.

۱- لیست حلقوی یک طرفه



۲- لیست حلقوی دوطرفه



دقت کنید:

۱. بین لیست‌های دوری و غیردوری تفاوتی از نظر اتلاف حافظه وجود ندارد، با این وجود اتلاف حافظه در لیست‌های دو طرفه بیشتر از یک طرفه می باشد.
 ۲. یکی از مزایای مهم لیست‌های دوری یک طرفه نسبت به لیست‌های خطی یک طرفه این است که در لیست‌های دوری یک طرفه امکان رسیدن به گره قبلی از هر گره دلخواه با چرخش از انتها به ابتدا در زمان $O(n)$ وجود دارد اما در لیست‌های خطی یک طرفه این عمل امکان پذیر نیست.
- با این وجود در لیست‌های دو طرفه رسیدن از هر گره به گره ماقبل سریع تر و آسان تر از لیست‌های دوری انجام می شود. چون در هر گره یک اشاره گر **Link** به گره قبل وجود دارد و مانند لیست حلقوی نیازی به رفتن به انتهای لیست و برگشت به ابتدای لیست نخواهد بود و در زمان $O(1)$ انجام می شود.

جستجو در لیست پیوندی

جستجو در هر لیست پیوندی فقط و فقط بصورت ترتیبی امکان پذیر بودن و از ابتدای لیست آغاز می شود. مرتبه اجرایی جستجوی کلید دلخواه k در لیست پیوندی با n گره که اشاره گر **start** به ابتدای آن اشاره می کند برابر $O(n)$ است.

```
search(start, k)
```

```
if start  $\neq$  nil
```

```
p = start
```

```
while p  $\neq$  nil and data[p]  $\neq$  k
```

```
p = Link[p]
```

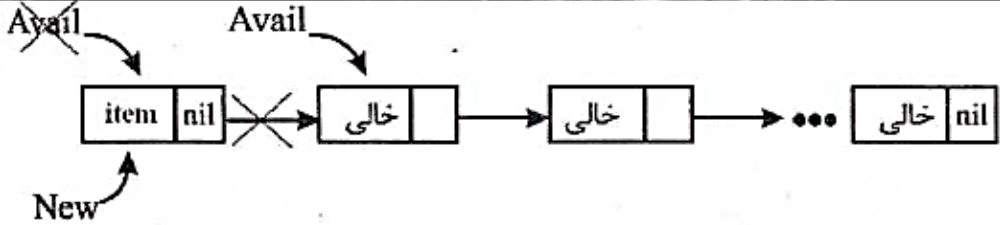
```
return p
```

درج گره در لیست پیوندی یک طرفه

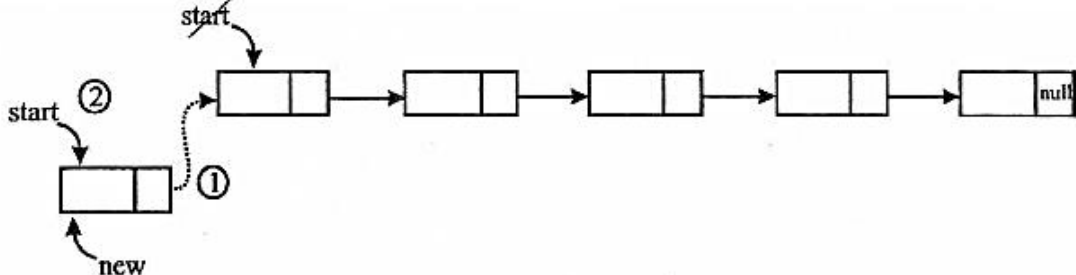
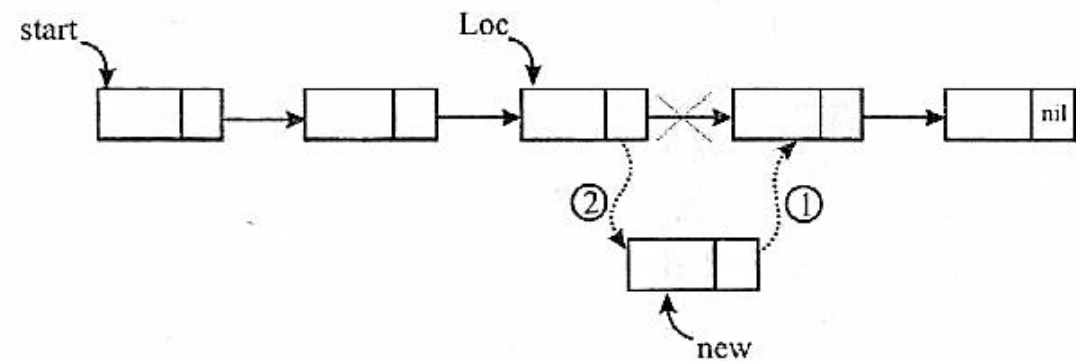
برای درج یک گره جدید در موقعیتی مشخص از یک لیست پیوندی ، باید ابتدا با پیمایش لیست به موقعیت قبل از محل درج رسیده و سپس گره مورد نظر را درج کرد.

می‌خواهیم گره جدید با اشاره گر **New** را بعد از گره معلوم با اشاره گر **Loc** از لیستی که اشاره گر **Start** به ابتدای آن اشاره می‌کند درج کنیم ، عملیات لازم عبارتند از:

آماده سازی گره جدید با اشاره گر New

الگوریتم	نمایش وضعیت
<pre> if Avail = nil then 'overflow' and 'exit' New = Avail , Avail = Link[Avail] data[New] = item Link[New] = nil </pre>	

درج گره جدید با اشاره گر New

الگوریتم	نمایش وضعیت
<p>if ($Loc = nil$) then</p> <p>1) $Link[New] = start$ \Rightarrow</p> <p>2) $start = New$</p>	 <p>الف) زمانی که موقعیت قبل از درج ($Loc = nil$) باشد یعنی گره جدید باید در ابتدا درج شود.</p>
<p>else</p> <p>1) $Link[New] = Link[Loc]$ \Rightarrow</p> <p>2) $Link[Loc] = New$</p>	 <p>ب) زمانی که موقعیت قبل از درج ($Loc \neq nil$) باشد یعنی گره جدید باید بعد از آن درج شود</p>

دقت کنید:

در صورتی که دستورات (1) ، (2) جابجا شود ، ادامه لیست از موقعیت (Loc) به بعد گم می شود.

نتایج

- ۱- پیمایش لیستی با n گره برای رسیدن به موقعیت قبل از درج زمانی معادل $O(n)$ صرف می کند.
- ۲- برای درج هر گره جدید نیاز به ۲ عمل جایگزینی است که زمانی معادل $O(1)$ صرف می کند.

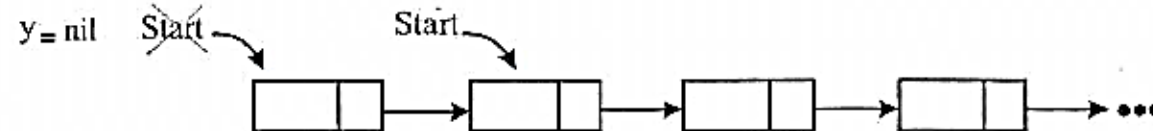
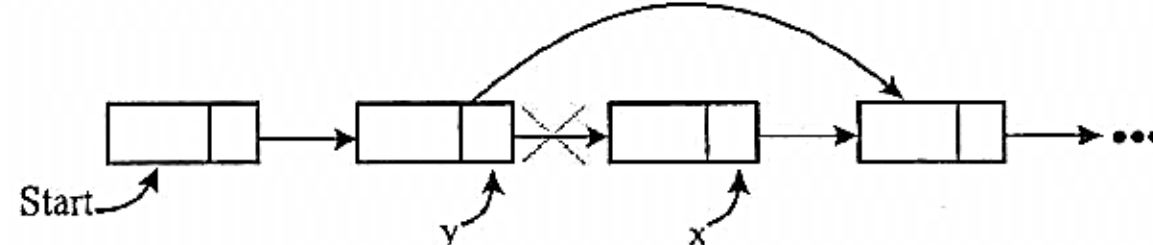
الگوریتم بالا در زبان‌های برنامه‌سازی مختلف می‌تواند بصورت زیر باشد:

زبان پاسکال	زبان C
<pre> if Loc = nil then Begin New^.link := start; start := New; End else Begin New^.Link := Loc^.Link; Loc^.Link := New; End </pre>	<pre> if (Loc == null) then { New → link = start; start = New; } else { New → Link = Loc → Link; Loc → Link = New; } </pre>

حذف گره از یک لیست پیوندی (یک طرفه)

برای حذف گره‌ای دلخواه از موقعیتی مشخص از یک لیست پیوندی، باید ابتدا با پیمایش لیست به موقعیت قبل از محل حذف رسیده، سپس گره مورد نظر را حذف کرد.

می‌خواهیم گره‌ای دلخواه با اشاره گر x را که اشاره گر y به گره قبل از آن اشاره می‌کند حذف کنیم.

الگوریتم	نمایش وضعیت
<p>if ($y = \text{nil}$) then</p> <p>$\text{start} = \text{link}[\text{start}]$ \Rightarrow</p>	 <p>الف) زمانی که موقعیت قبل از حذف ($y = \text{nil}$) باشد یعنی گره اول لیست باید حذف شود.</p>
<p>else</p> <p>$\text{Link}[y] = \text{Link}[x]$ \Rightarrow</p>	 <p>ب) زمانی که موقعیت قبل از حذف ($y \neq \text{nil}$) باشد یعنی گره با اشاره گر (x) باید حذف شود</p>

نتایج

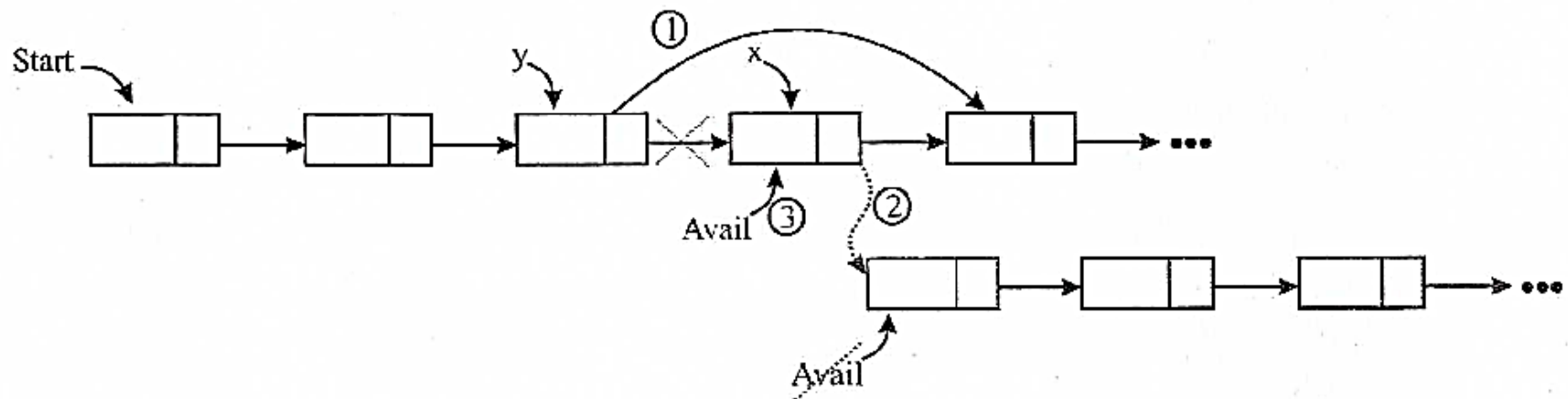
۱- پیمایش لیستی با n گره برای رسیدن به موقعیت قبل از حذف زمانی معادل $O(n)$ صرف می‌کند.

۲- برای حذف هر گره نیاز به ۱ عمل جایگزینی است.

۴- الگوریتم بالا در زبان‌های برنامه‌سازی مختلف می‌تواند بصورت زیر باشد:

زبان پاسکال	زبان C
<pre> if y = nil then start := start^.link; else y^.Link := x^.Link;</pre>	<pre> if (y == null) then start = start → link; else y → Link = x → Link;</pre>

تبصره: در صورتی که بخواهیم گره حذف شده با اشاره گر x را آزاد کنیم (به ابتدای لیست در دسترس با اشاره گر **Avail** اضافه کنیم)، از دستورات ۲ و ۳ به شرح زیر استفاده می کنیم. این دستورات معادل توابع `free, dispose` در زبانهای پاسکال و C هستند.

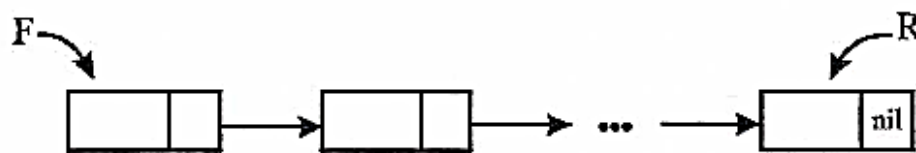


$$1) y.^{\text{link}} = x.^{\text{link}}$$

$$2) x.^{\text{link}} = \text{Avail}$$

$$3) \text{Avail} = x$$

مثال ۱: یک لیست خطی یک طرفه با دو اشاره گر F و R که به ترتیب به عنصر اول و آخر لیست اشاره می کنند پیاده سازی شده است. هزینه کدامیک از اعمال زیر وابسته به تعداد عناصر لیست است؟
(کارشناسی ارشد - دولتی ۸۰)



(۱) حذف اولین عنصر

(۲) حذف آخرین عنصر

(۳) درج یک عنصر در انتهای لیست

(۴) درج یک عنصر در ابتدای لیست

حل: گزینه ۲ درست است.

برای درج و حذف گره از ابتدا (با استفاده از اشاره گر F) و درج گره در انتها (با استفاده از اشاره گر R) نیاز به پیمایش و وابسته به تعداد گره های لیست نیست. اما برای حذف گره آخر ابتدا باید به کمک پیمایش در زمان $O(n)$ به گره ما قبل آخر رسیده تا بتوان عمل حذف را انجام داد.

(ارشد ۸۴ - IT)

مثال ۲: کدام یک از عبارتهای زیر نادرست است؟

$$(۱) \quad O(n!) = O(n^n)$$

$$(۲) \quad O(10^6) < O(n) < O(n \cdot \log_2 n)$$

(۳) هر الگوریتم از مرتبه $\theta(n)$ ، از مرتبه $O(n^2)$ نیز هست.

(۴) حذف عنصر آخر در یک لیست زنجیره‌ای یک طرفه، با داشتن اشاره‌گرهای first و last از مرتبه $O(1)$ است.

حل: گزینه ۴ درست است.

برای حذف گره آخر ابتدا باید به کمک پیمایش در زمان $O(n)$ به گره ما قبل آخر رسیده تا بتوان عمل حذف را انجام داد.

پیمایش لیست‌های پیوندی خطی

پیمایش گره‌های یک لیست پیوندی خطی یک‌طرفه از ابتدای لیست انجام شده و به دو صورت بازگشتی و غیربازگشتی می‌تواند انجام گیرد.

الگوریتم غیربازگشتی

```

if start  $\neq$  nil then
{
    p := start
    while(p  $\neq$  nil)
    {
        Visit(Data[p]);
        p = Link[p]
    }
}

```

دقت کنید:

۱- شرط $start \neq nil$ در ابتدا تهی نبودن لیست پیوندی را کنترل می‌کند.

۲- در شرط حلقه while اگر از شرط $Link[p] \neq nil$ استفاده کنیم پیمایش لیست تا قبل از گره آخر انجام شده و بارسیدن اشاره گر p به گره آخر از حلقه خارج می‌شویم.

۳- پیمایش لیست پیوندی با n گره از مرتبه اجرایی $O(n)$ است.

الگوریتم بازگشتی

الف (پیمایش از ابتدا تا انتها

```
Show (L:List)
{
  if (L ≠ nil)
  {
    Visit(Data[L]);
    Show(Link[L]);
  }
}
```

ب (پیمایش از انتها تا ابتدا

```
Show (L:List)
{
  if (L ≠ nil)
  {
    Show(Link[L]);
    Visit(Data[L]);
  }
}
```

لیست پیوندی حلقوی (Circular Linked List)

یک لیست پیوندی خطی یک طرفه که اشاره گر Link در گره آخر به جای مقدار null آدرس گره اول لیست را نگهداری کرده و به آن اشاره می کند.

مزیت لیست حلقوی بر لیست خطی

مزیت لیست حلقوی (دوری) بر لیست خطی یک طرفه این است که در لیست حلقوی بدون نیاز به هیچ حافظه اضافی با داشتن آدرس یک گره دلخواه امکان دسترسی به گره قبلی وجود دارد (با پیمایش $n - 1$ گره در لیست با n گره از مرتبه $O(n)$ ولی در لیست خطی یک طرفه این امکان وجود نداشته و دسترسی به گره های قبل از یک گره فقط از طریق آدرس شروع لیست وجود دارد.

پیمایش لیست حلقوی

پیمایش یک لیست حلقوی در صورتی که start اشاره‌گری به ابتدای لیست حلقوی باشد به شرح زیر خواهد بود:

```
if start ≠ nil then
{
  p = start;
  repeat
    Visit ( Data[p] );
    p = Link[p];
  until p = start;
}
```

عملیات در لیست‌های حلقوی

عملیات در لیست‌های حلقوی مانند عملیات در لیست‌های خطی است با این تفاوت که باید شرط پایان حلقه‌ها و نحوه اصلاح اشاره‌گر گره آخر باتوجه به دوری بودن لیست تغییر یابد.

مثال: فرض کنید a اشاره‌گر به انتهای یک لیست حلقوی و یک لیست غیرحلقوی باشد و بخواهیم گره x را بعد از آن درج کنیم.

لیست خطی

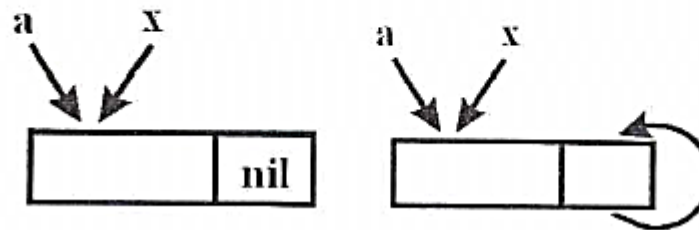
```
if (a ≠ nil) then
{
  Link[x] = Link[a];
  Link[a] = x;
}
else
{
  a = x;
  Link[x] = nil;
}
```

← در صورتی که لیست تهی باشد. در صورتی که لیست تهی باشد →

لیست دوری

```
if (a ≠ nil) then
{
  Link[x] = Link[a];
  Link[a] = x;
}
else
{
  a = x;
  Link[x] = x;
}
```

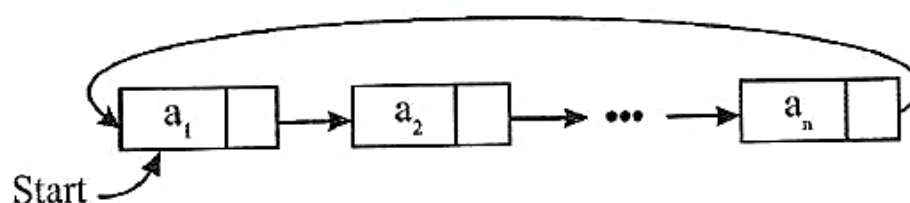
دیده می‌شود برای درج درحالتی که لیست تهی نیست دو الگوریتم شبیه هم عمل می‌کنند اما زمانی که لیست تهی باشد دو وضعیت متفاوت به صورت زیر بعد از درج گره به وجود می‌آید:



لیست دوری و اشاره گر Start

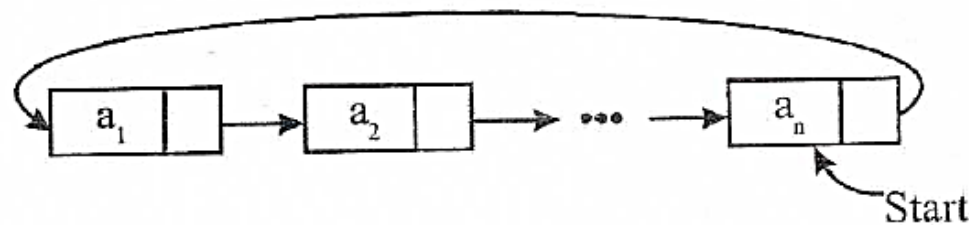
در صورتی که اشاره گر Start در لیست های دوری به گره اول یا آخر اشاره کند وضعیت های متفاوتی از نقطه نظر زمان انجام بعضی از عملیات ها بوجود می آید که به شرح زیر آن ها را بررسی می کنیم.

الف) اشاره گر Start به ابتدای لیست اشاره کند:



عملیات	مرتبۀ اجرایی (زمان)	توضیحات
درج در ابتدا	$O(n)$	باید به انتهای لیست رفته تا گره جدید را در ابتدای لیست درج کنیم.
حذف از ابتدا	$O(n)$	باید به انتهای لیست رفته تا گره ای را از ابتدای لیست حذف کنیم.
درج در انتها	$O(n)$	باید به انتهای لیست رفته تا گره جدید را در انتهای لیست درج کنیم.
حذف از انتها	$O(n)$	باید به گره ماقبل آخر رفته تا گره آخر را حذف کنیم.

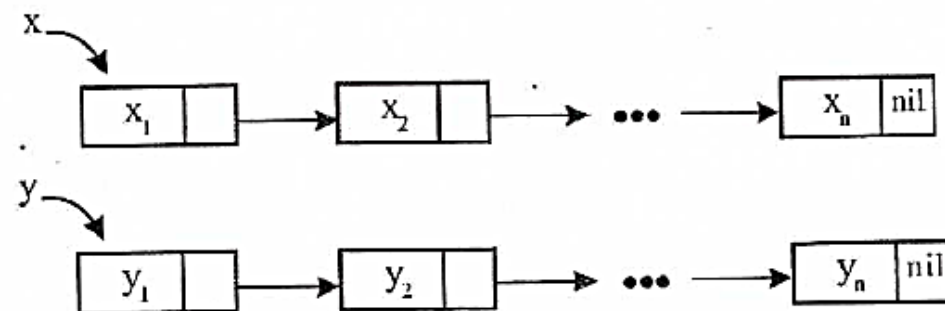
ب) اشاره گر Start به انتهای لیست اشاره کند:



عملیات	مرتبه اجرایی (زمان)	توضیحات
درج در ابتدا	$O(1)$	درج در انتها همان درج در ابتدا خواهد بود.
حذف از ابتدا	$O(1)$	اشاره گر Start در انتها به راحتی گره را از ابتدا حذف می کند.
درج در انتها	$O(1)$	درج در انتها به سادگی انجام خواهد شد.
حذف از انتها	$O(n)$	باید به گره ماقبل آخر رفته تا گره آخر را حذف کنیم.

اتصال (Concat) لیستهای پیوندی

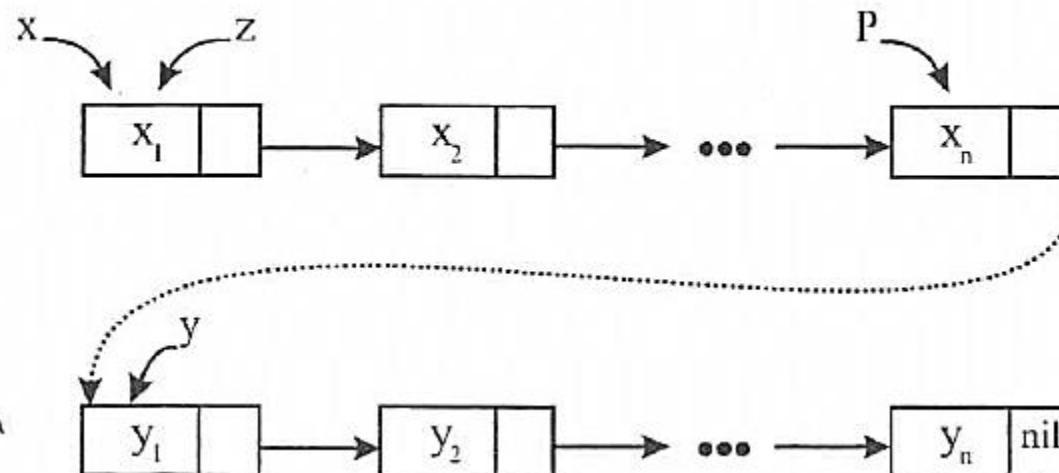
اگر دو لیست پیوندی x و y به شرح زیر وجود داشته باشند، با استفاده از قطعه برنامه زیر می توان این دو لیست را به هم متصل کرده و اشاره گر z را به ابتدای لیست حاصل از اتصال دو لیست x , y اشاره داد.



```

if x = nil then z := y
else
  begin
    z := x;
    if y <> nil then
      begin
        p := x;
        while p^.link <> nil do
          p := p^.link;
          p^.link := y;
        end;
      end;
  end;
end;

```



نکات

- ۱- در شرط $x = nil$ اگر x تهی باشد z به y اشاره خواهد کرد در غیر این صورت در قسمت $else$ ، $z := x$ شده و z به ابتدای لیست x اشاره خواهد کرد.
- ۲- در قسمت $else$ بعد از آن که $z := x$ انجام شد، با فرض آن که y تهی نباشد (در شرط $y < > nil$)، اشاره گر p در حلقه $while$ ، لیست x را پیمایش کرده تا روی گره آخر قرار گیرد، در این حالت از حلقه خارج شده و با جمله $p^.link := y$ ، اشاره گر (link) گره آخر x به گره اول y اشاره کرده و الگوریتم پایان می پذیرد.
- ۳- عملیات اتصال دو لیست x, y از مرتبه $O(n)$ است.

لیست‌های پیوندی دوطرفه (Doubled Linked List)

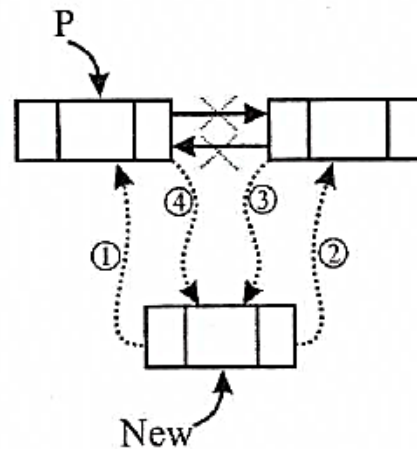
در لیست‌های پیوندی دوطرفه هر گره به کمک دو اشاره‌گر Left (آدرس گره قبلی) و Right (آدرس گره بعدی) می‌تواند به گره قبلی و بعدی اشاره کند، در نتیجه با داشتن آدرس هر گره به راحتی می‌توان به گره قبلی یا بعدی دسترسی پیدا کرد.



درج در لیست‌های دویپوندی

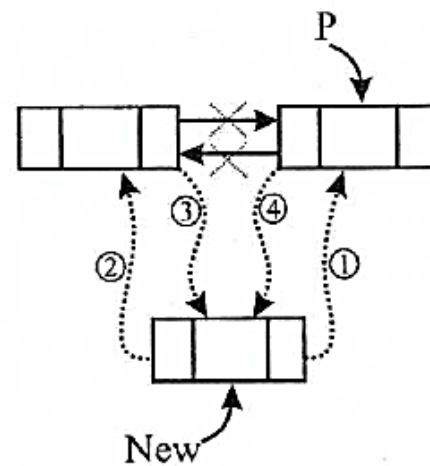
با داشتن آدرس یک گره دلخواه مانند P در یک لیست دویپوندی می‌توان گره جدید با اشاره گر New را در سمت چپ یا راست آن درج کرد. برای اینکار به 4 عمل جایگزینی نیاز داریم.
درج گره با اشاره گر New سمت راست P :

- 1) $Left[New] = p$
- 2) $Right[New] = Right[p]$
- 3) $Left[Right[p]] = New$
- 4) $Right[p] = New$;

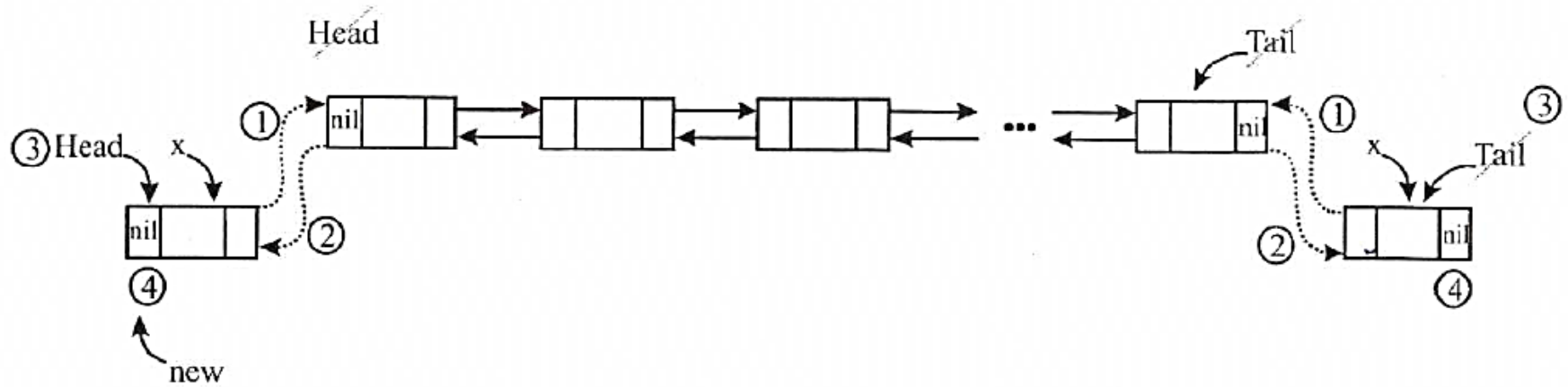


درج گره با اشاره گر New سمت چپ P :

- 1) **Right**[New] = p
- 2) **Left**[New] = Left[p]
- 3) **Right**[Left[p]] = New
- 4) **Left**[p] = New;



درج گره با اشاره گر x در ابتدا یا انتهای لیست:



- 1) **Right**[x] = Head
- 2) if Head \neq nil
 then **Left**[Head] = x
- 3) Head = x
- 4) **Left**[x] = nil;

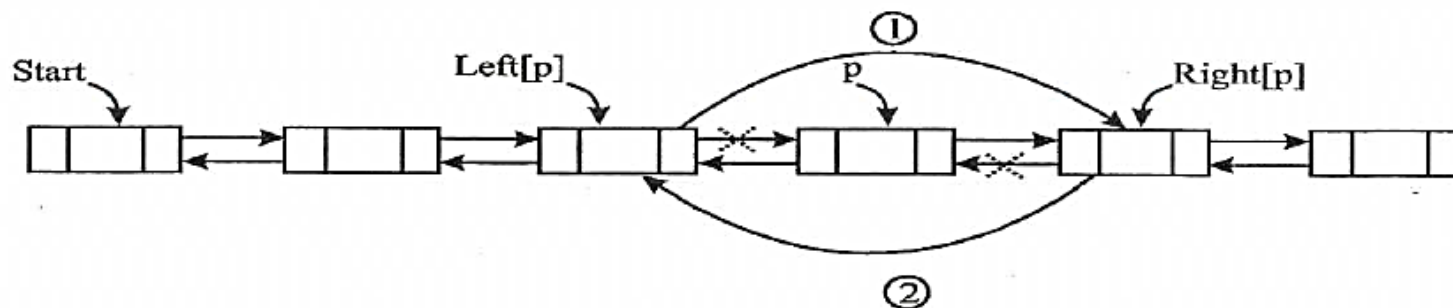
- 1) **Left**[x] = Tail
- 2) if Tail \neq nil
 then **Right**[Tail] = x
- 3) Tail = x
- 4) **Right**[x] = nil;

عمل حذف در لیست‌های دو پیوندی

برای حذف گره‌ای دلخواه از یک لیست دوپیوندی (دو طرفه) در صورتی که اشاره گر p به گره مورد نظر اشاره کند، با جایگزینی آدرس بصورت زیر می توان گره مورد نظر را حذف کرد :

$$1) \text{Right}[\text{Left}[p]] = \text{Right}[p]$$

$$2) \text{Left}[\text{Right}[p]] = \text{Left}[p]$$



الگوریتم بالا در زبان‌های برنامه سازی مختلف می‌تواند بصورت زیر باشد:

زبان پاسکال	زبان C
1) $p^{\wedge}.\text{Left}^{\wedge}.\text{Right} = p^{\wedge}.\text{Right}$	1) $p \rightarrow \text{Left} \rightarrow \text{Right} = p \rightarrow \text{Right}$
2) $p^{\wedge}.\text{Right}^{\wedge}.\text{Left} = p^{\wedge}.\text{Left}$	2) $p \rightarrow \text{Right} \rightarrow \text{Left} = p \rightarrow \text{Left}$

معکوس کردن لیست پیوندی

الگوریتم غیر بازگشتی

ابزار مورد نیاز:

برای معکوس کردن یک لیست پیوندی با هر تعداد گره تنها به 3 اشاره گر (p, q, r) نیاز داریم.

```
p = start;
q = null;
while p ≠ null do
begin
1) r = q;
2) q = p;
3) p = p^.link;
4) q.link = r;
end;
start = q;
```

مرتبه اجرایی

زمان لازم برای معکوس کردن لیست پیوندی با n گره برابر با $O(n)$ خواهد بود.

شروع الگوریتم:

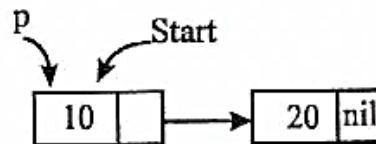
p، به گره اول لیست اشاره می کند.

q، null می شود.

r، null می شود.

P = start

q = nil

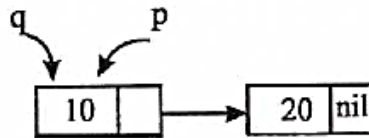


تکرار اول حلقه :

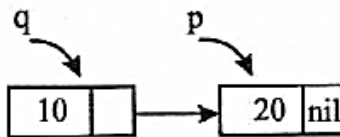
1) $r = q$

$r = \text{nil}$

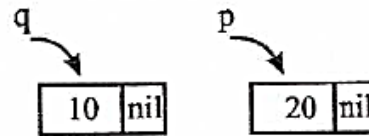
2) $q = p$



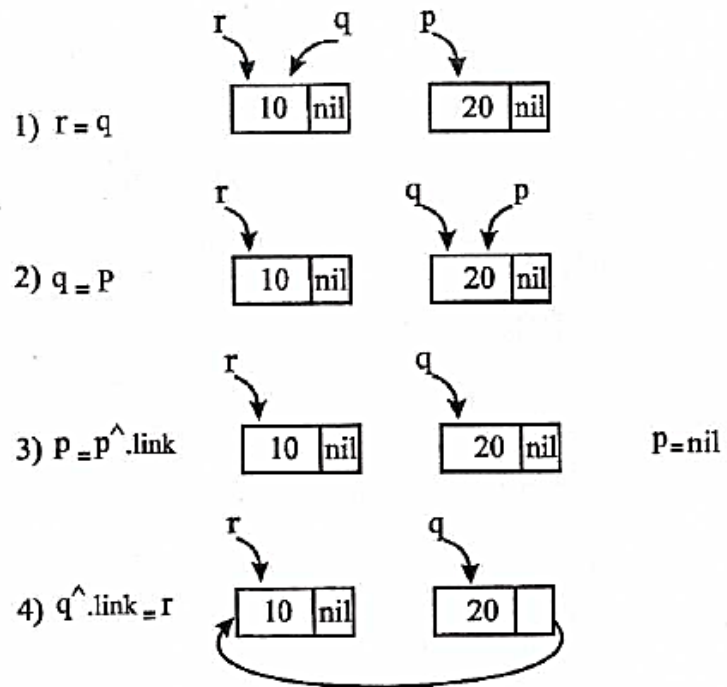
3) $p = p^{\wedge}.\text{link}$



4) $q^{\wedge}.\text{link} = r$



تکرار دوم حلقه:

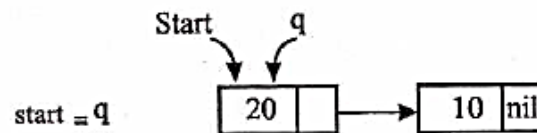


پایان الگوریتم:

p ، null می شود.

q ، به ابتدای لیست معکوس شده اشاره می کند.

r ، به یک گره بعد از q اشاره می کند.



الگوریتم بازگشتی

```
reverse(s : list)
begin
if (s = nil) then return(nil)
else
return(cons(reverse(tail(s)), head(s)));
end;
```

cons (x , y) : لیست y را به انتهای لیست x متصل می کند.

خصوصیات لیست‌های پیوندی

مزایا:

- ۱- تخصیص پویا و متناسب برای داده‌ها برخلاف آرایه‌ها که تخصیص ایستا و محدود دارند.
- ۲- عملیات درج و حذف بدون نیاز به شیفت با $O(1)$ انجام می‌شود برخلاف آرایه‌ها که نیاز به شیفت برای این عمل دارند با $O(n)$.

معایب:

- ۱- اتلاف حافظه نسبت به آرایه‌ها به علت استفاده از فیلد اشاره‌گر بیشتر است.
- ۲- تنها روش جستجو، جستجوی خطی است و با زمان $O(n)$ برخلاف آرایه‌ها که جستجو دودویی را نیز دارند با زمان $O(\log_2 n)$.
- ۳- دسترسی به هر داده ترتیبی و از ابتدای لیست است با زمان $O(n)$ برخلاف آرایه‌ها که امکان دسترسی تصادفی با زمان $O(1)$ را دارند.

فرض کنید $x = (x_1, x_2, \dots, x_n)$ و $y = (y_1, y_2, \dots, y_m)$ دو لیست پیوندی خطی ساده باشند. مرتبه زمانی الگوریتم اتصال دو لیست در

(مهندسی فناوری اطلاعات (IT) - آزاد ۸۵)

لیست z چیست؟ $z = (x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m)$

$O(n)$ (۴)

$O(nm)$ (۳)

$O(n + m)$ (۲)

$O(m)$ (۱)

- گزینه «۴» با توجه به لیست پیوندی Z که ابتدا لیست x و بعد لیست y آمده است، کافی است فقط لیست x پیمایش شده و لیست y به انتهای آن اضافه شود، پس فقط n گره را باید پیمایش کنیم، بنابراین مرتبه زمانی آن $O(n)$ خواهد بود.

۱۰- برای معکوس کردن یک لیست پیوندی ساده (زنجیر) حداقل به چند اشاره گر نیاز داریم؟

(مهندسی کامپیوتر - آزاد ۸۶)

5 (۴)

3 (۳)

4 (۲)

2 (۱)

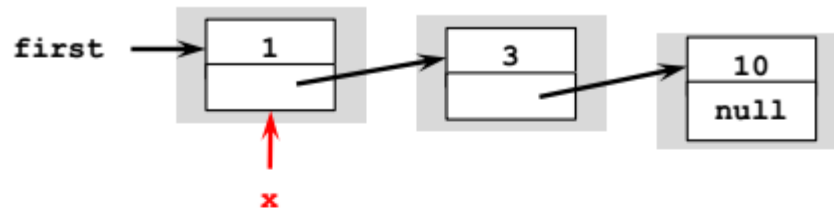
گزینه «۳» با توجه به الگوریتم معکوس کردن یک لیست پیوندی ساده که در این فصل ذکر شده، و نکته ۱۶ برای این کار به سه اشاره گر نیاز می باشد.

ادغام دو لیست پیوندی مرتب در یک لیست جدید

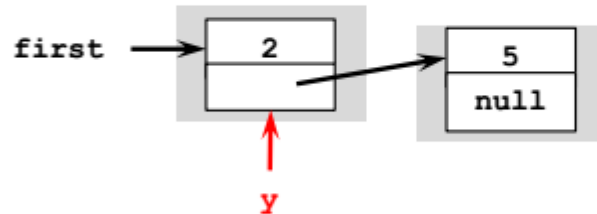
```
public static LinkedList<Integer> merge(LinkedList<Integer> L1, LinkedList<Integer> L2)
{
    LinkedList<Integer> L = new LinkedList<Integer>();
    Node x = L1.first();
    Node y = L2.first();
    while (x != null || y != null)
    {
        String item;
        if (x == null) { item = y.item; y = y.next; }
        else if (y == null) { item = x.item; x = x.next; }
        else if (y.item < x.item) { item = y.item; y = y.next; }
        else { item = x.item; x = x.next; }
        L.addLast(item);
    }
    return L;
}
```

ادغام دو لیست پیوندی مرتب در یک لیست جدید

L1



L2

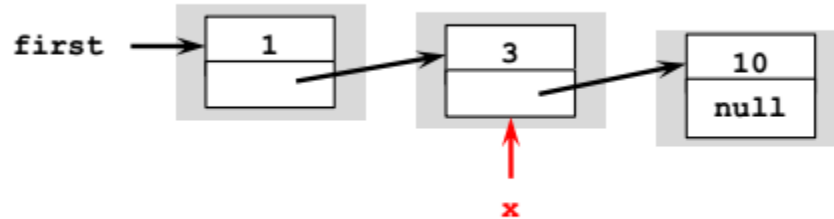


L

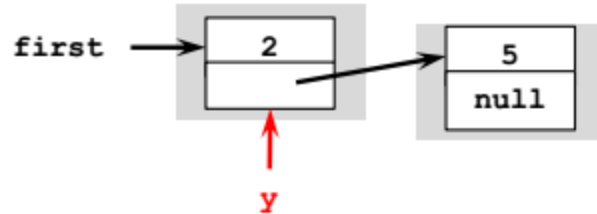


ادغام دو لیست پیوندی مرتب در یک لیست جدید

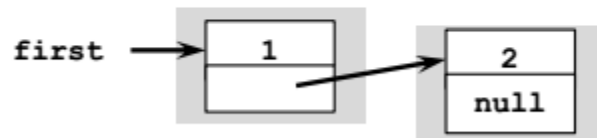
L1



L2

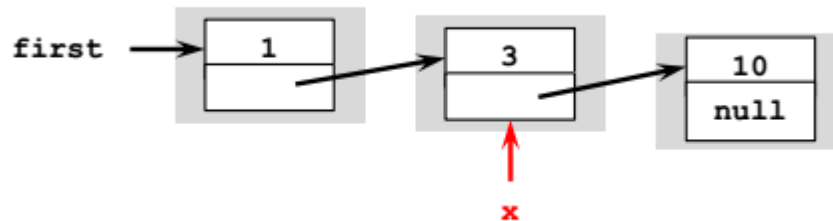


L

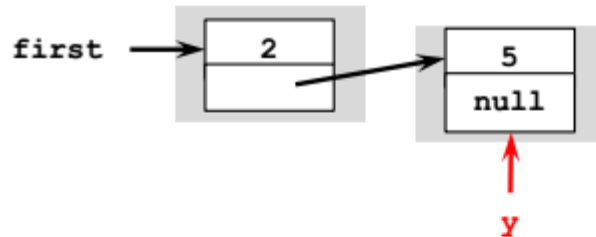


ادغام دو لیست پیوندی مرتب در یک لیست جدید

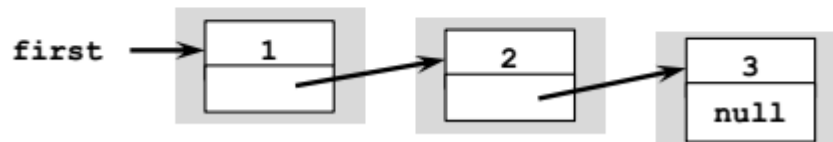
L1



L2

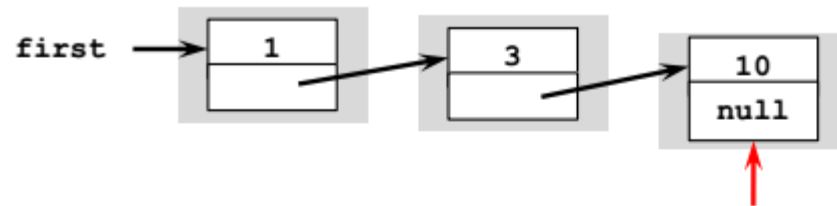


L

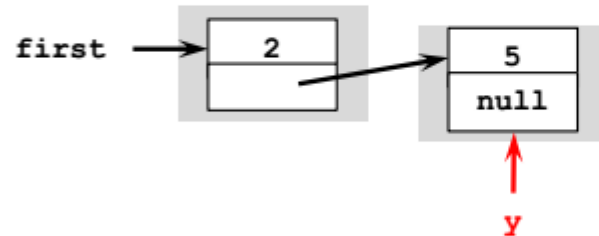


ادغام دو لیست پیوندی مرتب در یک لیست جدید

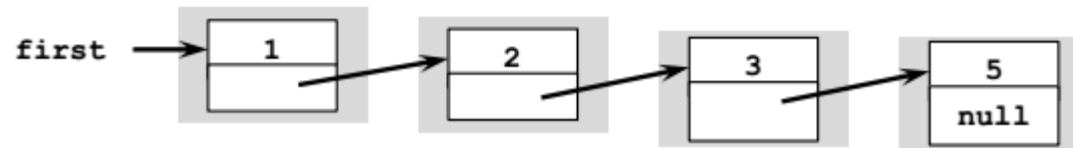
L1



L2

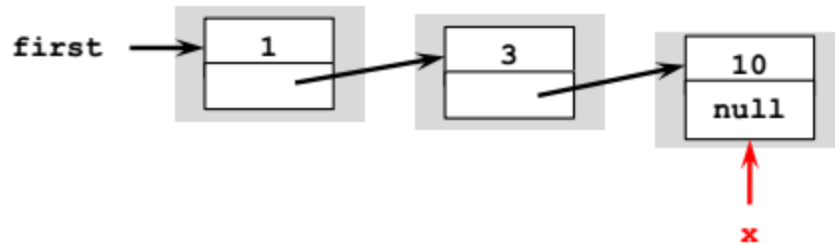


L

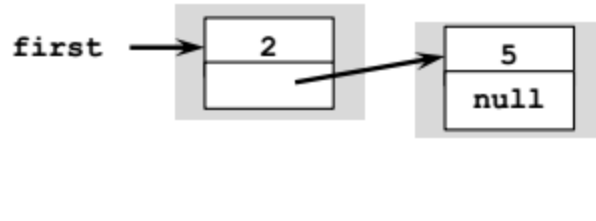


ادغام دو لیست پیوندی مرتب در یک لیست جدید

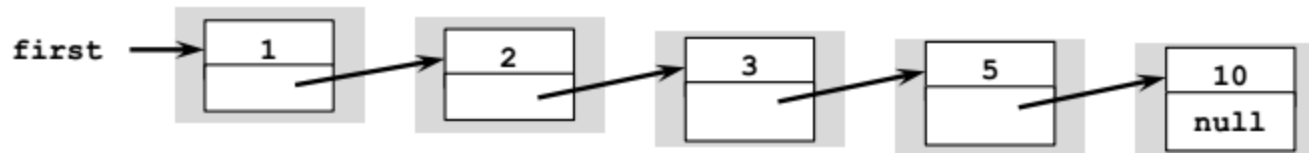
L1



L2

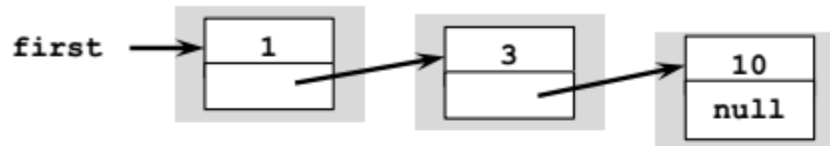


L

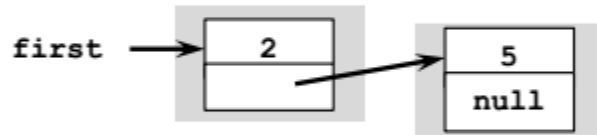


ادغام دو لیست پیوندی مرتب در یک لیست جدید

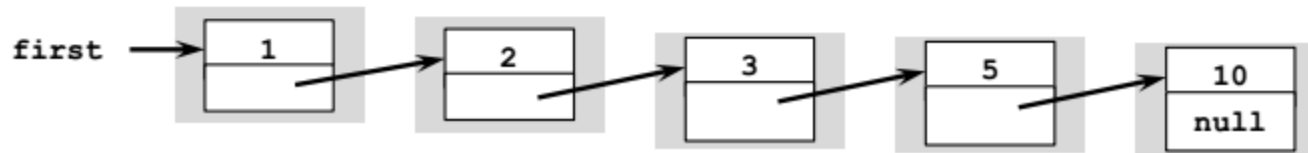
L1



L2



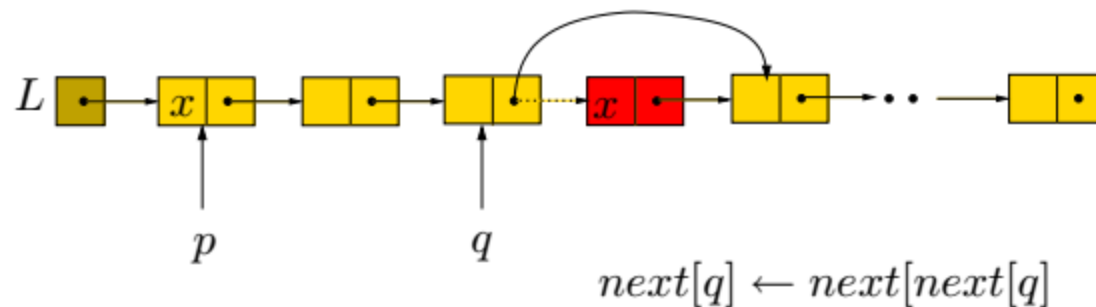
L



↑
x

↑
y

عملیات دیگر بر روی لیست‌ها: حذف عناصر تکراری در یک لیست



PURGELIST (L)

▷ همه‌ی عناصر یکسان را جز یکی حذف می‌کند

```

1   $p \leftarrow \text{FIRST}(L)$ 
2  while  $p \neq \text{null}$ 
3      do  $q \leftarrow p$ 
4          while  $\text{next}[q] \neq \text{null}$ 
5              do if  $\text{element}[p] = \text{element}[\text{next}[q]]$ 
6                  then DELETE-AFTER ( $L, q$ )
7                  else  $q \leftarrow \text{next}[q]$ 
8       $p \leftarrow \text{next}[p]$ 

```