اصطلاح "گره" به‌تنهایی هنگامی که با لیست دارای سرلیست بکار گرفته شود اشاره به گرهٔ معمولی دارد نه سرگره. بدین ترتیب **گرهٔ اول** در یک لیست دارای سرلیست، گره‌ای است که بعد از سرگره قرار می‌گیرد و مکان گرهٔ اول **LINK[START]** است نه **START** به صورتی که با لیستهای پیوندی معمولی بکار گرفته می‌شد.

الگوریتم **5.11** که از یک متغیر اشاره‌گر **PTR** برای پیمایش یک لیست چرخشی با سرلیست استفاده می‌کند ذاتاً همان الگوریتم **5.1** است که یک لیست پیوندی معمولی را پیمایش می‌کند. اما اکنون این تفاوت وجود دارد که الگوریتم (۱) با **PTR = LINK[START]** شروع می‌شود نه با **PTR = START** و (۲) با **PTR = START** به پایان می‌رسد نه با **PTR = NULL**.

لیستهای چرخشی دارای سرلیست اغلب به جای لیستهای پیوندی معمولی بکار گرفته می‌شوند چون اغلب با استفاده از لیستهای دارای سرلیست بهتر و ساده‌تر بیان شده و پیاده‌سازی می‌شوند. این مطلب از دو خاصیت زیر از لیستهای چرخشی دارای سرلیست ناشی می‌شود:

(۱) اشاره‌گر پوچ مورد استفاده قرار نمی‌گیرد و از این‌رو تمام اشاره‌گرها دارای آدرس مجاز هستند.

(۲) هر گرهٔ (معمولی) یک گرهٔ پیشین دارد از این‌رو گرهٔ اول نمی‌تواند نیازمند حالت خاص باشد. مثال بعدی فایدهٔ این دو خاصیت را بیان می‌کند.

Algorithm 5.11: (Traversing a Circular Header List) Let LIST be a circular header list in memory. This algorithm traverses LIST, applying an operation PROCESS to each node of LIST.

```
1. Set PTR := LINK[START]. [Initializes the pointer PTR.]
2. Repeat Steps 3 and 4 while PTR ≠ START:
3.     Apply PROCESS to INFO[PTR].
4.     Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of Step 2 loop.]
5. Exit.
```

مثال ۱۹ـ۵

فرض کنید **LIST** یک لیست پیوندی در حافظه باشد و **ITEM** یعنی یک اطلاع خاص داده شده است.

(الف) الگوریتم **5.2**، **LOC** مکان گرهٔ اول در **LIST** را پیدا می‌کند که شامل **ITEM** است مشروط بر این که **LIST** یک لیست پیوندی معمولی باشد. الگوریتم زیر از این‌گونه الگوریتم است که در آن **LIST** یک لیست چرخشی با سرلیست است.

Algorithm 5.12: SRCHHL(INFO, LINK, START, ITEM, LOC)
LIST is a circular header list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST or sets LOC = NULL.

```
1. Set PTR := LINK[START].
2. Repeat while INFO[PTR] ≠ ITEM and PTR ≠ START:
       Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of loop.]
3. If INFO[PTR] = ITEM, then:
       Set LOC := PTR.
   Else:
       Set LOC := NULL.
   [End of If structure.]
4. Exit.
```

دو آزمونی که حلقهٔ جستجو را کنترل می‌کند (مرحلهٔ ۲ در الگوریتم 5.12) به صورت هم زمان در الگوریتم برای لیستهای پیوندی معمولی اجرا نمی‌شوند یعنی الگوریتم 5.2 اجازه ندارد از دستوری شبیه

Repeat while INFO[PTR]≠ITEM و PTR≠NULL

استفاده کند زیرا برای لیستهای پیوندی معمولی هنگامی که PTR = NULL باشد INFO[PTR] تعریف نشده است.

(ب) زیربرنامهٔ Procedure 5.9 مکان LOC گرهٔ اول N را پیدا می‌کند که شامل ITEM است و همچنین LOCP مکان گرهٔ قبل از N را وقتی LIST یک لیست پیوندی معمولی است پیدا می‌کند. هنگامی که LIST یک لیست چرخشی با سرلیست است زیربرنامه به صورت زیر است:

Procedure 5.13: FINDBHL(INFO, LINK, START, ITEM, LOC, LOCP)

```
1. Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]
2. Repeat while INFO[PTR] ≠ ITEM and PTR ≠ START.
       Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]
   [End of loop.]
3. If INFO[PTR] = ITEM, then:
       Set LOC := PTR and LOCP := SAVE.
   Else:
       Set LOC := NULL and LOCP := SAVE.
   [End of If structure.]
4. Exit.
```

به‌سادگی این زیربرنامهٔ Procedure در مقایسه با Procedure 5.9 توجه کنید. در اینجا نباید حالت خاصی را که ITEM در گرهٔ اول ظاهر می‌شود مورد توجه قرار دارد و از این‌رو می‌توان در یک زمان دو آزمونی را که حلقه را کنترل می کنند انجام داد.

(ج) الگوریتم 5.10 گرهٔ اول N را حذف می‌کند که وقتی ITEM یک لیست پیوندی معمولی است شامل LIST است. هنگامی که LIST یک لیست چرخشی با سرلیست است الگوریتم زیر این‌گونه است.

Algorithm 5.14: DELLOCHL(INFO, LINK, START, AVAIL, ITEM)

1. [Use Procedure 5.13 to find the location of N and its preceding node.]
   Call LINDBHL(INFO, LINK, START, ITEM, LOC, LOCP).
2. If LOC = NULL, then: Write: ITEM not in list, and Exit.
3. Set LINK[LOCP] := LINK[LOC]. [Deletes node.]
4. [Return deleted node to the AVAIL list.]
   Set LINK[LOC] := AVAIL and AVAIL := LOC.
5. Exit.

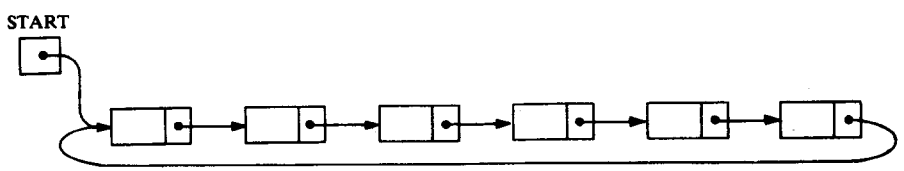
مجدداً نباید حالت خاصی را که ITEM در گرهٔ اول ظاهر می‌شود و به صورتی که در الگوریتم 5.10 عمل کردیم درنظر بگیریم.

توجه کنید : دو نوع دیگر از لیستهای پیوندی وجود دارد که بعضی وقتها در برخی از کتابهای ساختمان داده‌ها مطرح می‌شود:

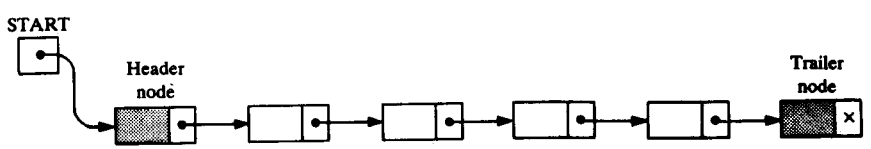
(۱) یک لیست پیوندی که گرهٔ آخرش به جای این که شامل اشاره‌گر پوچ باشد به گرهٔ اول اشاره می‌کند که به آن **لیست چرخشی یا حلقوی** می‌گویند.

(۲) یک لیست پیوندی که شامل سرگرهٔ خاصی در ابتدای لیست و هم‌گرهٔ انتهایی خاص دیگری در پایان لیست است.

شکل ۳۱ـ ۵ نمودارهای این لیستها را نشان می‌دهد.



(الف) لیست پیوندی چرخشی



(ب) لیست پیوندی با سرگرهٔ و گرهٔ انتهایی

شکل ۳۱ـ ۵

چندجمله‌ایها

لیستهای پیوندی دارای سرلیست غالباً برای ذخیرهٔ چندجمله‌ایها در حافظه مورد استفاده قرار

می‌گیرند. سرلیست قسمت مهم این نمایش را تشکیل می‌دهد زیرا به آن در نمایش چندجمله‌ای صفر نیاز داریم. این نمایشِ چندجمله‌ای‌ها در قالب یک مثال خاص ارائه می‌شود.

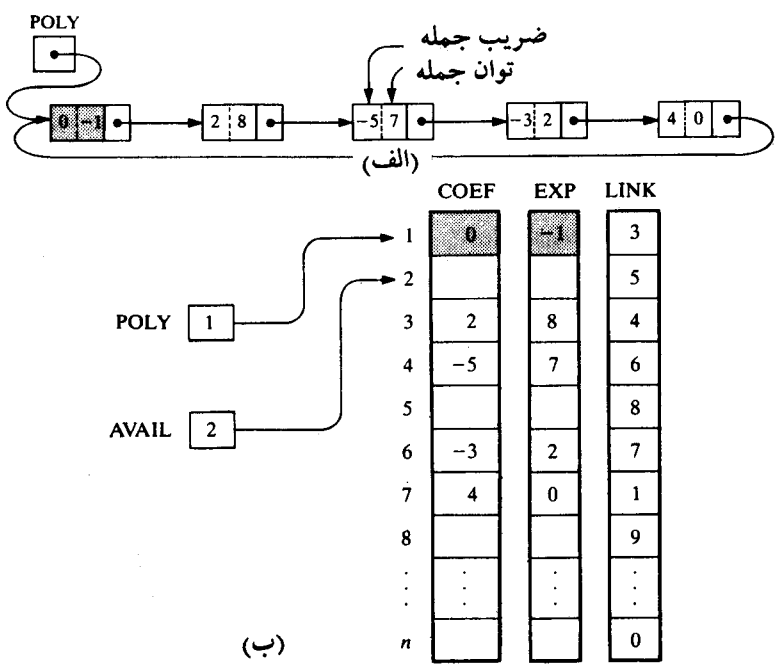
مثال ۲۰ـ ۵

فرض کنید $p(x)$ نمایش چندجمله‌ای یک متغیرهٔ زیر باشد که شامل چهار جملهٔ غیرصفر است:

$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4$$

آنگاه $p(x)$ را می‌توان به وسیلهٔ لیست دارای سرلیست مانند شکل ۳۲ـ ۵ (الف) نمایش داد که در آن هر گره متناظر با یک جملهٔ غیرصفر از $p(x)$ است به‌ویژه این که قسمت اطلاعات گره به دو فیلد تقسیم شده است که به ترتیب ضریب و توان جملهٔ متناظر آن را نمایش می‌دهند و گره‌ها باتوجه بـه نـزول درجه‌ها، پیوند خورده‌اند.

ملاحظه می‌کنید که لیست متغیر اشاره‌گر **POLY** به سرگره‌ای اشاره می‌کند که در فیلد توان آن یک عدد منفی، در اینجا ۱ـ، جایگزین شده است. اکنون نمایش آرایه‌ای لیست، مستلزم سه آرایهٔ خطی است که ما آنها را **COEF, EXP** و **LINK** نام‌گذاری کرده‌ایم. یک چنین نمایشی در شکل ۳۲ـ ۵ (ب) ظاهر شده است.



شکل ۳۲ـ ۵ $p(x) = 2x^8 - 5x^7 - 3x^2 + 4$

۱۰ـ۵ لیستهای دوطرفه

به لیستی که در بالا مورد بررسی قرار گرفت **لیست یکطرفه** می‌گویند. زیرا تنها از یک طرف می‌توان لیست را پیمایش کرد یعنی با شروع از **START** متغیر اشاره‌گر لیست که به گرهٔ اول یا سرگره اشاره می‌کند و با استفاده از فیلد اشاره‌گر بعدی **LINK** که به گرهٔ بعدی لیست اشاره می‌کند می‌توان لیست را تنها در یک جهت پیمایش کرد. علاوه براین، با معلوم بودن **LOC** مکان گرهٔ **N** در چنین لیستی، بلافاصله می‌توان به گرهٔ بعدی در لیست (با ارزیابی **LINK[LOC]**) دسترسی پیدا کرد. اما بدون پیمایش قسمتی از لیست نمی‌توان به گرهٔ قبلی دسترسی پیدا کرد. درحالت خاص معنی آن این است که باید قسمتی از لیست قبل از **N** را برای حذف **N** از لیست پیمایش کرد.

این بخش ساختمان یک لیست جدید موسوم به **لیست دوطرفه** را معرفی می‌کند که می‌توان آن را در دو جهت پیمایش کرد: در جهت متداول و معمولی به پیش‌رفتن، از ابتدای لیست تا انتهای آن یا درجهت عکس از انتهای لیست به ابتدای لیست. علاوه براین، با معلوم‌بودن **LOC** مکان یک گره **N** در لیست، اکنون می‌توان بلافاصله هم به گرهٔ بعدی و هم به گرهٔ قبل از آن در لیست دسترسی پیدا کرد. در حالت خاص معنی آن این است که می‌توان بدون پیمایش هر قسمت از لیست **N** را از لیست حذف کرد.

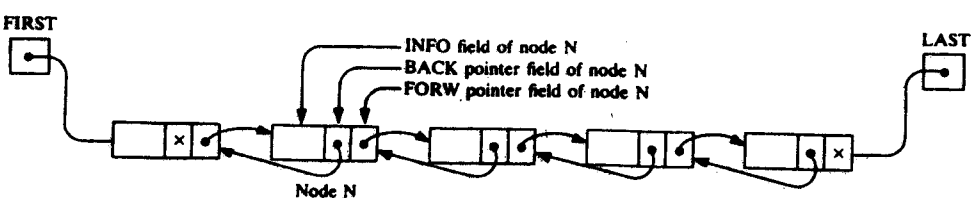
یک **لیست دوطرفه** یک مجموعهٔ خطی از عناصر داده‌ای بنام **گره‌ها** است که در آن هر گره **N** به سه قسمت تقسیم می‌شود:

(۱) یک فیلد اطلاعات **INFO** که شامل دادهٔ **N** است.

(۲) یک فیلد اشاره‌گر **FORW** که شامل مکان گرهٔ بعدی در لیست است.

(۳) یک فیلد اشاره‌گر **BACK** که شامل مکان گرهٔ قبل از آن در لیست است.

علاوه بر این لیست نیازمند دو متغیر اشاره‌گر است: **FIRST** که به گرهٔ اول و **LAST** به گرهٔ آخر در لیست اشاره می‌کند. شکل ۳۳ـ۵ شامل یک نمودار از چنین لیستی است. ملاحظه می‌کنید که اشاره‌گر پوچ **NULL** در فیلد **FORW** از گرهٔ آخر لیست و همچنین در فیلد **BACK** از گرهٔ اول لیست است.



شکل ۳۳ـ۵ لیست دوطرفه

ملاحظه می‌کنید که با استفاده از متغیر **FIRST** و فیلد اشاره‌گر **FORW**، می‌توانیم یک لیست دوطرفه را مانند قبل، از اول تا آخر لیست پیمایش کنیم. از طرف دیگر، با استفاده از متغیر **LAST** و فیلد اشاره‌گر **BACK** نیز می‌توانیم این لیست را از آخر تا اول لیست پیمایش کنیم.

فرض کنید **LOCA** و **LOCB** به ترتیب مکانهای دو گرهٔ **A** و **B** در یک لیست دوطرفه باشند. آنگاه روشی که اشاره‌گرهای **FORW** و **BACK** تعریف می‌شوند منتهی به نتایج زیر می‌شود:

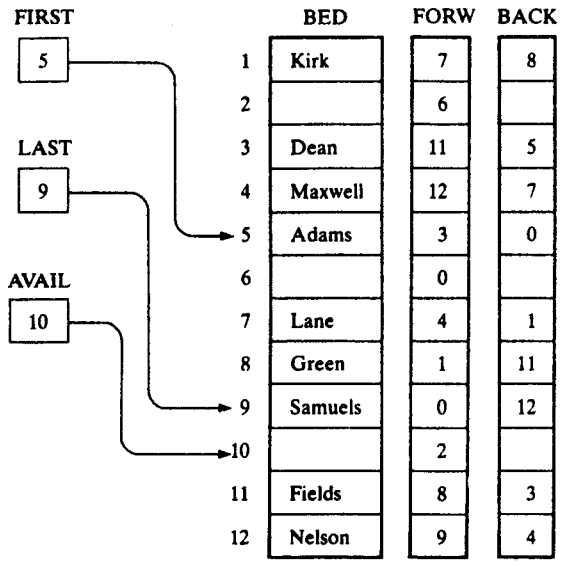
خاصیت اشاره‌گر: **FORW[LOCA] = LOCB** اگر و فقط اگر **BACK[LOCB] = LOCA**

از طرف دیگر این بیان که **B** بعد از گرهٔ **A** است با این بیان که گرهٔ **A** قبل از گرهٔ **B** قراردارد معادل است.

لیستهای دوطرفه را می‌توان به وسیلهٔ آرایه‌های خطی به همان صورتی که در مورد لیستهای یکطرفه بیان شد در حافظه نگهداری کرد با این تفاوت که اکنون به جای یک آرایهٔ اشاره‌گر **LINK** به دو آرایهٔ اشاره‌گر **FORW** و **BACK** نیاز داریم و به جای یک متغیر اشاره‌گر لیست **START** به دو متغیر اشاره‌گر لیست **FIRST** و **LAST** نیاز داریم. از طرف دیگر از آنجا که عمل حذف و اضافه کردن گره‌ها تنها در ابتدای لیست **AVAIL** انجام می‌شود لیست **AVAIL** از حافظهٔ موجود در آرایه‌ها نیز همانند لیست یکطرفه، با استفاده از **FORW** به عنوان فیلد اشاره‌گر نگهداری می‌شود.

مثال ۲۱ـ۵

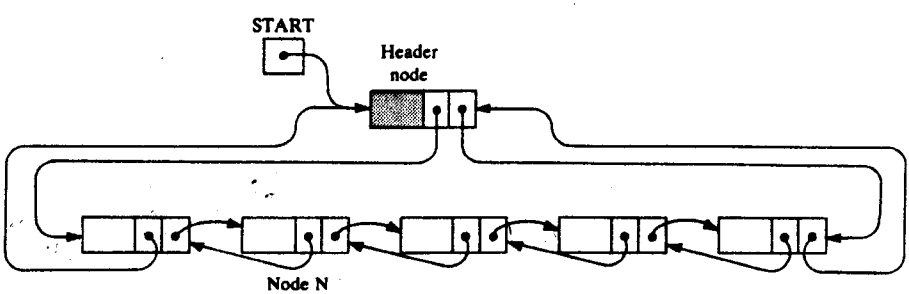
مجدداً داده‌های شکل ۹ـ۵ یعنی ۹ بیمار یک اطاق بیمارستان با ۱۲ تخت را درنظر بگیرید. شکل ۳۴ـ۵ چگونگی سازماندهی لیست الفبایی بیماران را در لیست دوطرفه نشان می‌دهد. ملاحظه می‌کنید که مقادیر **FIRST** و فیلد اشاره‌گر **FORW** به ترتیب دارای همان مقدار **START** و آرایهٔ **LINK** هستند. از این‌رو لیست را می‌توان همانند قبل به صورت الفبایی پیمایش کرد. ازطرف دیگر با استفاده از **LAST** و آرایهٔ اشاره‌گر **BACK**، لیست را می‌توان به ترتیب الفبایی عکس پیمایش کرد. یعنی **LAST** به **Samuels** فیلد اشاره‌گر **BACK** از **Samuels** به **Nelson**، فیلد اشاره‌گر **BACK** از **Nelson** به **Maxwell** و الی آخر اشاره می‌کند.



شکل ۳۴ـ ۵

لیستهای دوطرفه دارای سرلیست

مزایا و ویژگیهای خوب یک لیست دوطرفه و یک لیست چرخشیِ دارای سرلیست در یک لیست چرخشی دوطرفهٔ دارای سرلیست، به صورتی که در شکل ۳۵ـ ۵ نشان داده شده، جمع شده است. این لیست چرخشی است زیرا دو گرهٔ انتهایی آن به سرگرهٔ اول لیست اشاره می‌کنند.

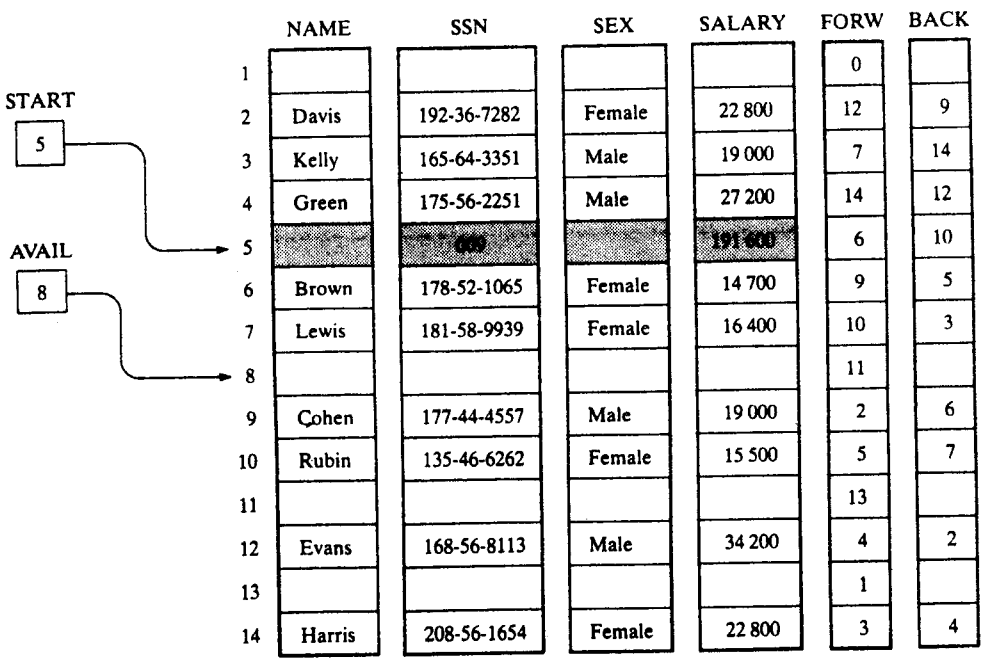


شکل ۳۵ـ ۵ لیست چرخشی دوطرفه با سرلیست

ملاحظه می‌کنید چنین لیست دوطرفه‌ای نیازمند تنها یک متغیر اشاره‌گر لیست **START** است که به سرگره اشاره می‌کند. زیرا دو اشاره‌گر سرگره به دوانتهای لیست اشاره می‌کنند.

مثال ۲۲ـ۵

فایل پرسنلی شکل ۳۰ـ۵ را درنظر بگیرید که به صورت یک لیست چرخشی با سرلیست سازماندهی می‌شود. داده‌ها را می‌توان در یک لیست چرخشی دارای سرلیست تنها با افزودن یک آرایهٔ دیگر **BACK** که مکان گره‌های قبلی را به دست می‌دهد سازماندهی کرد. چنین ساختمانی در شکل ۳۶ـ۵ به تصویر درآمده است که **LINK** به صورت **FORW** نام‌گذاری مجدد شده است. بار دیگر لیست **AVAIL** تنها به صورت یک لیست یکطرفه نگهداری می‌شود.



شکل ۳۶ـ۵

عملیات بر روی لیستهای دوطرفه

فرض کنید **LIST** یک لیست دوطرفه در حافظه باشد. در این قسمت عملیات پیمایش، جستجو، حذف و اضافه‌کردن عنصر روی **LIST** شرح داده می‌شود.

پیمایش : فرض کنید بخواهیم لیست **LIST** را برای پردازش دقیقاً یکبار پیمایش کنیم. در آن صورت اگر **LIST** یک لیست دوطرفهٔ معمولی باشد می‌توانیم از الگوریتم ۱ـ ۵ استفاده کنیم یا اگر **LIST** یک سرلیست داشته باشد می‌توانیم از الگوریتم ۱۱ـ ۵ استفاده کنیم. در اینجا هیچ مزیتی بین سازماندهی داده‌ها در لیست دوطرفه نسبت به لیست یکطرفه وجود ندارد.

جستجوکردن : فرض کنید اطلاعات **ITEM** ، یک مقدار کلیدی داده شده است و بخواهیم **LOC** مکان اطلاعات **ITEM** را در لیست **LIST** تعیین کنیم. در آن صورت اگر **LIST** یک لیست دوطرفهٔ معمولی باشد می‌توانیم از الگوریتم ۲ـ ۵ استفاده کنیم یا اگر لیست **LIST** یک سرلیست داشته باشد می‌توانیم از الگوریتم ۱۲ـ ۵ استفاده کنیم. در اینجا اگر این احتمال وجود داشته باشد که **ITEM** نزدیک انتهای لیست است در آن صورت مزیت اصلی آن این است که می‌توانیم جستجو برای **ITEM** را از انتهای لیست به ابتدای لیست انجام دهیم. برای مثال فرض کنید **LIST** لیستی از نامهای افراد باشد که به صورت الفبایی مرتب شده است. اگر **ITEM = Smith** آنگاه باید عمل جستجو را از انتهای لیست به طرف ابتدای لیست انجام دهیم اما اگر **ITEM = Davis** آنگاه باید جستجو در **LIST** از ابتدا لیست به طرف انتهای لیست انجام شود.

حذف کردن : فرض کنید **LOC** مکان گرهٔ **N** در لیست **LIST** داده شده است و فرض کنید بخواهیم **N** را از لیست حذف کنیم. فرض می‌شود که **LIST** یک لیست چرخشی دوطرفه با سرلیست است. توجه دارید که **BACK[LOC]** و **FORW[LOC]** به‌ترتیب مکان گره‌هایی هستند که قبل و بعد از گرهٔ **N** قرار دارند. بنابر این همانگونه که در شکل ۳۷ـ ۵ نشان داده شده است **N** با تغییر جفت اشاره‌گرهای زیر از لیست حذف می‌شود :

FORW[BACK[LOC]] := FORW[LOC] و **BACK[FORW[LOC]] := BACK[LOC]**

آنگاه گرهٔ حذف‌شدهٔ **N** با دستورهای جایگزینی زیر به لیست **AVAIL** برگردانده می‌شود :

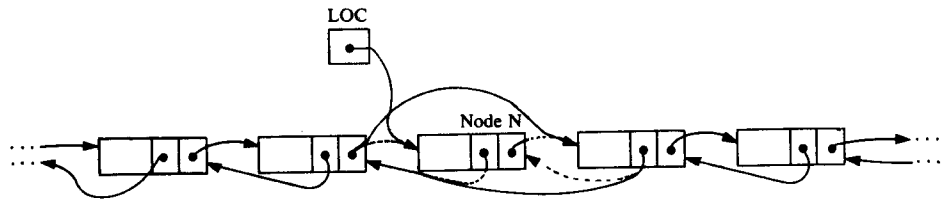
FORW[LOC] := AVAIL و **AVAIL := LOC**

بیان رسمی این الگوریتم به صورت زیر است :

Algorithm 5.15: DELTWL(INFO, FORW, BACK, START, AVAIL, LOC)

1. [Delete node.]
   Set FORW[BACK[LOC]] := FORW[LOC] and
   BACK[FORW[LOC]] := BACK[LOC].
2. [Return node to AVAIL list.]
   Set FORW[LOC] := AVAIL and AVAIL := LOC.
3. Exit.

در اینجا یک مزیت عمدهٔ لیست دوطرفه را مشاهده می‌کنید. اگر داده‌ها به صورت یک لیست یکطرفه سازماندهی شوند آنگاه برای حذف N باید لیست یکطرفه را پیمایش کنیم تا مکان گرهٔ قبل از N پیدا شود.



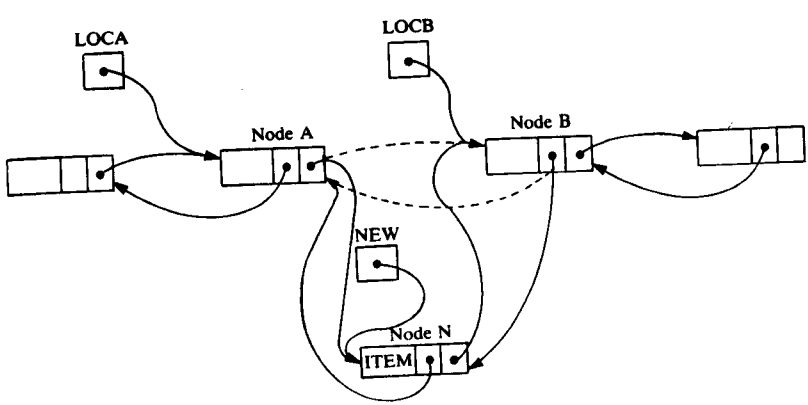
شکل ۳۷ـ ۵ حذف گرهٔ N

اضافه کردن: فرض کنید مکانهای LOCA و LOCB دو گرهٔ مجاور A و B در لیست LIST داده شده است و فرض کنید بخواهیم اطلاعات معلوم ITEM را بین گره‌های A و B اضافه کنیم. همانگونه که با یک لیست یکطرفه عمل کردیم، نخست گرهٔ اول N را از لیست AVAIL حذف می‌کنیم، از متغیر NEW برای نگهداشتن مکان یا آدرس آن استفاده می‌کنیم و آنگاه دادهٔ ITEM را در گرهٔ N کپی می‌کنیم. به عبارت دیگر قرار می‌دهیم :

NEW := AVAIL, AVAIL := FORW[AVAIL], INFO[NEW] := ITEM

حال همانگونه که در شکل ۳۸ـ ۵ نشان داده شده است، گرهٔ N با محتوای ITEM با تغییر چهار اشاره‌گر زیر در لیست اضافه می‌شود :

FORW[LOCA] := NEW, FORW[NEW] := LOCB
BACK[LOCB] := NEW, BACK[NEW] := LOCA



شکل ۳۸ـ ۵ اضافه کردن گرهٔ N

بیان رسمی این الگوریتم به صورت زیر است :

Algorithm 5.16: INSTWL(INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove node from AVAIL list and copy new data into node.]
   Set NEW := AVAIL, AVAIL := FORW[AVAIL], INFO[NEW] := ITEM.
3. [Insert node into list.]
   Set FORW[LOCA] := NEW, FORW[NEW] := LOCB,
   BACK[LOCB] := NEW, BACK[NEW] := LOCA.
4. Exit.

الگوریتم 5.16 فرض می‌کند که **LIST** شامل یک سرگره است. از این‌رو **LOCA** یا **LOCB** می‌توانند به سرگره اشاره کنند که در آن حالت N به عنوان گرهٔ اول یا گرهٔ آخر اضافه می‌شود. اگر **LIST** سرگره نداشته باشد آنگاه باید حالتی را درنظر بگیریم که **LOCA = NULL** و N به عنوان گرهٔ اول در لیست اضافه می‌شود و حالتی را درنظر بگیریم که **LOCB = NULL** و N به عنوان گرهٔ آخر در لیست اضافه می‌شود.

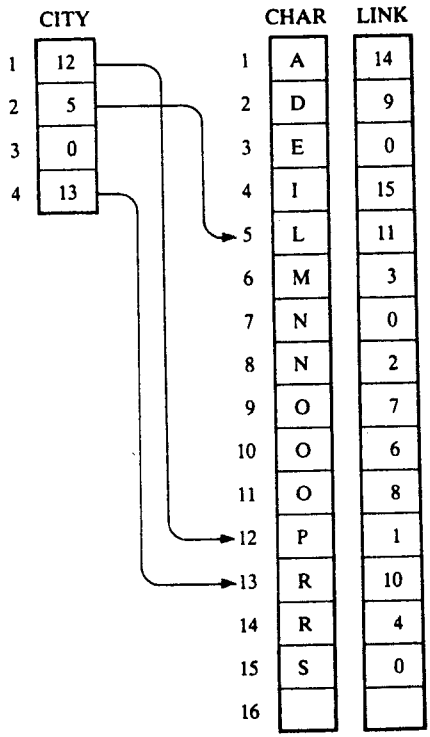
توجه کنید : در حالت کلی ذخیرهٔ داده‌ها به صورت یک لیست دوطرفه، که نیازمند حافظهٔ اضافی برای اشاره‌گرهای پیمایش از انتها به ابتدای لیست و زمان اضافی برای تغییر در اشاره‌گرهای اضافه شده، است که در مقایسه با لیست یکطرفه قابل توجه نیست و هزینهٔ زیادی دربر ندارد مگر آن که خواسته باشیم مکان گره‌ای را پیدا کنیم که قبل از گرهٔ معلوم N است مانند حالتی که هنگام عمل حذف در بالا انجام داده‌ایم.

مسأله‌های حل‌شده

لیستهای پیوندی

مسأله ۱ـ ۵ : رشته‌های کاراکتری ذخیره شده در چهار لیست پیوندی شکل ۳۹ـ۵ را تعیین کنید.

حل : در اینجا اشاره‌گر چهار لیست در آرایه **CITY** دیده می‌شود. با **CITY[1]** شروع می‌کنیم با تعقیب اشاره‌گرها، لیست را پیمایش می‌کنیم که درنتیجه آن رشتهٔ **PARIS** بدست می‌آید. با شروع از **CITY[2]** و پیمایش لیست رشتهٔ **LONDON** به دست می‌آید. چون **NULL** در **CITY[3]** دیده می‌شود از این‌رو لیست سوم خالی است و آن را با Λ رشتهٔ تهی نمایش می‌دهند. با شروع از **CITY[4]** و پیمایش لیست رشتهٔ **ROME** به دست می‌آید. به عبارت دیگر **PARIS**، **LONDON**، Λ و **ROME** چهار رشتهٔ موردنظر هستند.



شکل ۳۹ـ ۵

مسأله ۲ـ ۵ : لیست نامهای زیر (به ترتیب) در آرایه خطی INFO جایگزین می‌شوند.

Mary, June, Barbara, Paula, Diana, Audrey, Karen, Nancy, Ruth, Eileen, Sandra, Helen

یعنی **INFO[1] = Mary, INFO[2] = June,..., INFO[12] = Helen** . مقادیر فوق را در آرایهٔ **LINK** و یک متغیر **START** طوری جایگزین کنید که **INFO, LINK** و **START** تشکیل لیست الفبایی از نامها را بدهد.

حل : لیست الفبایی نامها عبارتند از :

Audrey, Barbara, Diana, Eileen, Helen, June, Karen, Mary, Nancy, Paula, Ruth, Sandra,

مقادیر **START** و **LINK** به صورت زیر بدست می‌آیند :

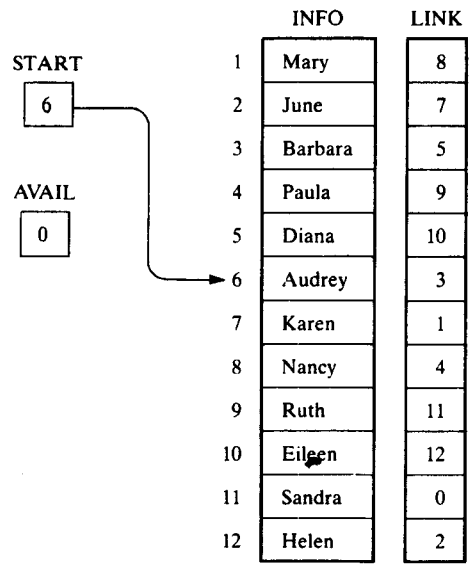
(الف) **INFO[6] = Audrey** ، بنابراین در **START = 6** جایگزین کنید.

(ب) **INFO[3] = Barbara** ، بنابراین در **LINK[6] = 3** جایگزین کنید.

(ج) **INFO[5] = Diana** ، بنابراین در **LINK[3] = 5** جایگزین کنید.

(د) INFO[10] = Eileen ، بنابراین در LINK[5] = 10 جایگزین کنید.

و الی آخر، چون INFO[11] = Sandra آخرین نام است LINK[11] = NULL جایگزین کنید. شکل ۴۰ـ ۵ ساختمان داده‌ای را نشان می‌دهد که در آن فرض شده است INFO برای تنها ۱۲ عنصر حافظه در اختیار دارد. قرار می‌دهیم AVAIL = NULL.



شکل ۴۰ـ ۵

مسأله ۳ـ ۵ : فرض کنید LIST یک لیست پیوندی در حافظه باشد. یک زیربرنامهٔ Procedure بنویسید که :

(الف) تعداد NUM دفعاتی را که ITEM داده شده در LIST ظاهر شده است پیدا کند.

(ب) تعداد NUM عناصر غیرصفر را در LIST پیدا کنید.

(ج) کلید معلوم K را به هر عنصر لیست LIST اضافه کنید.

حل : هر زیربرنامهٔ Procedure برای پیمایش این لیست از الگوریتم 5.1 استفاده می‌کند.

(الف)

```
Procedure P5.3A:  1. Set NUM := 0. [Initializes counter.]
                  2. Call Algorithm 5.1, replacing the processing step by:
                        If INFO[PTR] = ITEM, then: Set NUM := NUM + 1.
                  3. Return
```

(ب)

```
Procedure P5.3B:  1. Set NUM := 0. [Initializes counter.]
                  2. Call Algorithm 5.1, replacing the processing step by:
                        If INFO[PTR] ≠ 0, then: Set NUM := NUM + 1.
                  3. Return.
```

(ج)

Procedure P5.3C: 1. ، Call Algorithm 5.1, replacing the processing step by:
Set INFO[PTR] := INFO[PTR] + K.
2. Return.

مسأله ۴ـ۵: لیست الفبایی بیماران شکل ۹ـ۵ را درنظر بگیرید. تغییرات موردنیاز در این ساختمان داده را تعیین کنید تا (الف) **Walters** به لیست اضافه شود و آنگاه (ب) **Kirk** از لیست حذف شود.

حل: (الف) ملاحظه می‌کنید که **Walters** در تخت **10** خوابانده می‌شود که اولین تخت خالی است و **Walters** پس از **Samuels** به لیست اضافه می‌شود که آخرین بیمار لیست است. سه تغییر در فیلد اشاره‌گرها به صورت زیر است:

۱ـ **LINK[9] = 10** [اکنون **Samuels** به **Walters** اشاره می‌کند.]

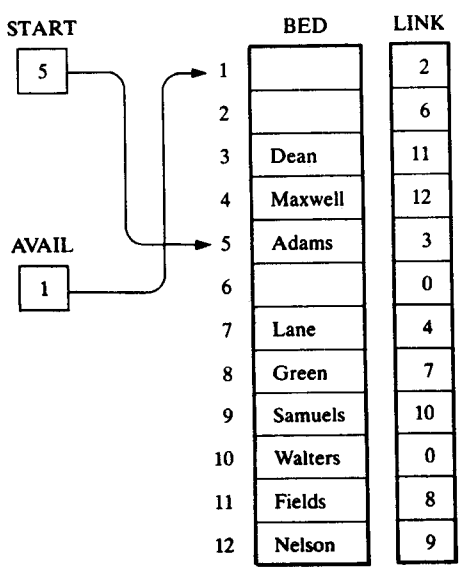
۲ـ **LINK[10] = 0** [اکنون **Walters** آخرین بیمار در لیست است.]

۳ـ **AVAIL = 2** [اکنون **AVAIL** به تخت بعدی خالی اشاره می‌کند.]

(ب) از آنجا که **Kirk** مرخص می‌شود، **BED[1]** اکنون خالی است. سه تغییر زیر در فیلد اشاره‌گرها باید داده شود:

LINK[8] = 7 **LINK[1] = 2** **AVAIL = 1**

بنابه تغییر اول، **Green**، که در آغاز قبل از **Kirk** بود اکنون به **Lane** اشاره می‌کند که در آغاز بعد از **Kirk** قرار داشت. تغییر دوم و سوم تخت خالی جدید را به لیست **AVAIL** اضافه می‌کند. تأکید می‌کنیم که قبل از انجام عمل حذف باید گرهٔ **BED[8]** را پیدا کنیم که در آغاز به گرهٔ حذف شدهٔ **BED[1]** اشاره می‌کرد. شکل ۴۱ـ۵ ساختمان دادهٔ جدید را نشان می‌دهد.



شکل ۴۱ـ۵

مسأله ۵ـ ۵ : فرض کنید LIST در حافظه باشد. الگوریتمی بنویسید که گرهٔ آخر را از لیست LIST حذف کند.

حل : گرهٔ آخر را تنها وقتی می‌توان حذف کرد که مکان عنصر قبل از گرهٔ آخر را بدانیم. بنابراین با استفاده از یک متغیر اشاره‌گر PTR لیست را پیمایش کنید و با استفاده از متغیر اشاره‌گر SAVE گرهٔ قبل از آن را نگهدارید. وقتی LINK[PTR] = NULL ، PTR به گرهٔ آخر اشاره می‌کند و در چنین حالتی SAVE به عنصر قبل از گرهٔ آخر اشاره می‌کند. حالتی را که LIST تنها یک گره دارد مستقلاً بررسی کنید چون SAVE تنها در صورتی قابل تعریف است که لیست ۲ یا چند عنصر داشته باشد. الگوریتم به صورت زیر است :

Algorithm P5.5: DELLST(INFO, LINK, START, AVAIL)

1. [List empty?] If START = NULL, then Write: UNDERFLOW, and Exit.
2. [List contains only one element?]
   If LINK[START] = NULL, then:
   (*a*) Set START := NULL. [Removes only node from list.]
   (*b*) Set LINK[START] := AVAIL and AVAIL := START.
   [Returns node to AVAIL list.]
   (*c*) Exit.
   [End of If structure.]
3. Set PTR := LINK[START] and SAVE := START. [Initializes pointers.]
4. Repeat while LINK[PTR] ≠ NULL. [Traverses list, seeking last node.]
   Set SAVE := PTR and PTR := LINK[PTR]. [Updates SAVE and PTR.]
   [End of loop.]
5. Set LINK[SAVE] := LINK[PTR]. [Removes last node.]
6. Set LINK[PTR] := AVAIL and AVAIL := PTR. [Returns node to AVAIL list.]
7. Exit.

مسأله ۶ـ ۵ : فرض کنید NAME1 یک لیست در حافظه است. الگوریتمی بنویسید که NAME1 را در لیست NAME2 کپی کند.

حل : نخست قرار دهید NAME2 := NULL تا لیست خالی تشکیل شود. آنگاه بااستفاده از یک متغیر اشاره‌گر PTR ، NAME1 را پیمایش کنید و هنگام ملاقات هر گرهٔ NAME1 ، محتوای آن INFO[PTR] را در یک گرهٔ جدید کپی کنید که در آن صورت به انتهای NAME2 اضافه می‌شود. از LOC برای نگهداری گرهٔ آخر NAME2 در خلال پیمایش استفاده کنید. شکل ۴۲ـ۵ ، PTR و LOC را قبل از اضافه‌شدن گرهٔ چهارم به NAME2 نشان می‌دهد.

لیستهای دارای سرلیست، لیستهای دوطرفه

مسأله ۷ـ ۵ : لیست‌های (چرخشی) دارای سرلیست از لیستهای یکطرفه شکل ۱۱ـ ۵ بسازید.

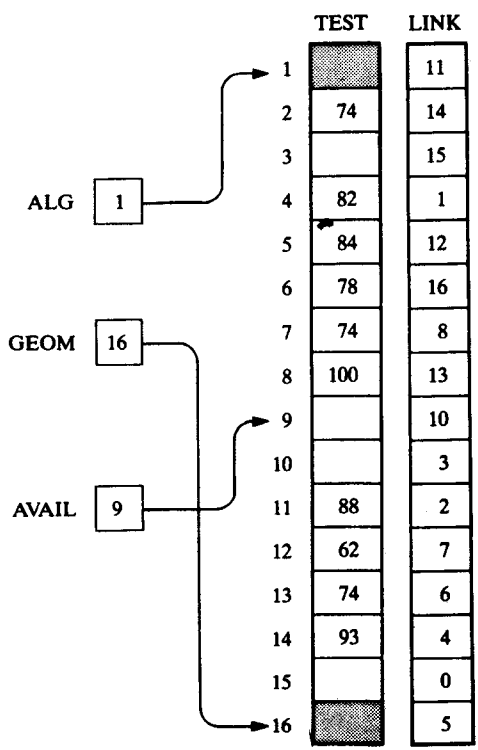
حل : **TEST[1]** را به عنوان سرگره برای لیست **ALG** و **TEST[16]** را به عنوان سرگره برای لیست **GEOM** انتخاب کنید. آنگاه برای هر لیست:

(الف) متغیر اشاره‌گر لیست را طوری تغییر دهید که به سرگره اشاره کند.

(ب) سرگره را طوری تغییر دهید که به گرهٔ اول لیست اشاره کند.

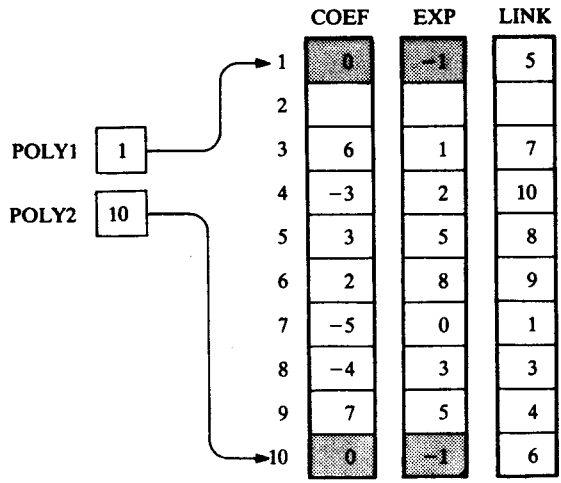
(ج) گرهٔ آخر را طوری تغییر دهید که از آخر به سرگره اشاره کند.

در پایان، لیست **AVAIL** را مجدداً سازماندهی کنید. شکل ۴۳ـ ۵ ساختمان دادهٔ تازه شده را نشان می‌دهد.



شکل ۴۳ـ ۵

مسأله ۸ـ ۵ : چندجمله‌ای **POLY1** و **POLY2** ذخیره شده در شکل ۴۴ـ ۵ را پیدا کنید.



شکل ۴۴ـ ۵

حل : کار را با **POLY1** شروع می‌کنیم. با تعقیب اشاره‌گرها، لیست را پیمایش می‌کنیم تا چندجمله‌ای به دست آید.

$$p_1(x) = 3x^5 - 4x^3 + 6x - 5$$

با شروع از **POLY2** ، و تعقیب اشاره‌گرها، لیست را پیمایش می‌کنیم تا چندجمله‌ای

$$p_2(x) = 2x^8 + 7x^5 - 3x^2$$

به دست‌آید. در اینجا **COEF[K]** و **EXP[K]** به ترتیب حاوی ضریب و توان یک جملهٔ چندجمله‌ای است. ملاحظه می‌کنید که در فیلد **EXP** سرگره‌هایی با مقدار 1 – جایگزین شده است.

مسأله ۹ـ ۵ : چندجمله‌ای **p(x, y, z)** را برحسب سه متغیر **x, y** و **z** درنظر بگیرید، جملات چندجمله‌ای **p(x, y, z)** به ترتیبِ حروف الفبا در کنار هم قرار می‌گیرند مگر آن که خلاف آن بیان شود. یعنی نخست جملات را برحسب درجه‌های نزولی **x** مرتب می‌کنیم و چنانچه جملاتی دارای درجهٔ **x** برابر باشند این جملات را برحسب درجه‌های نزولی **y** مرتب می‌کنیم و چنانچه جملاتی با درجهٔ **x, y** برابر باشند این جملات را برحسب درجه‌های نزولی **z** مرتب می‌کنیم. فرض کنید :

$$p(x, y, z) = 8x^2y^2z - 6yz^8 + 3x^3yz + 2xy^7z - 5x^2y^3 - 4xy^7z^3$$

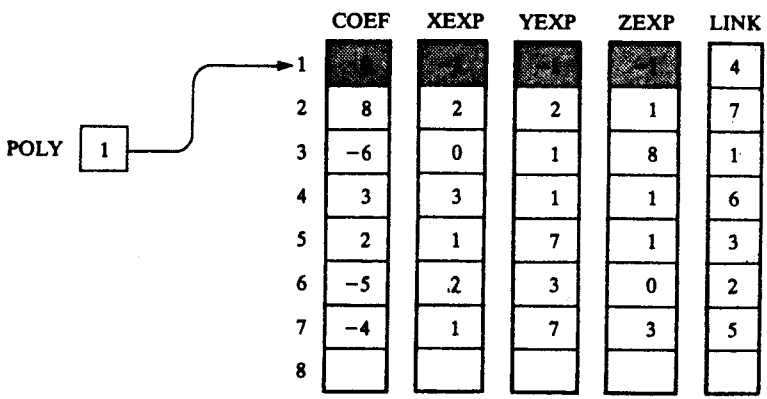
(الف) چندجمله‌ای را طوری بازنویسی کنید که جملات آن مرتب شده باشند.

(ب) فرض کنید جملات با ترتیب گفته شده در مسأله در آرایه‌های خطی **COEF** ، **XEXP** ، **YEXP** و **ZEXP** با گرهٔ اول **HEAD** ذخیره شده‌اند. مقادیر را در **LINK** طوری جایگزین کنید که لیست پیوندی شامل دنباله جملات چندجمله‌ای با ترتیب ذکر شده باشد.

حل : **(الف)** توجه دارید که $3x^3yz$ قبل از همه می‌آید چون بالاترین درجه را نسبت به **x** دارد. توجه دارید که $8x^2y^2z$ و $-5x^2y^3$ نسبت به x دارای درجهٔ یکسان هستند اما $5x^2y^3$ – قبل از $8x^2y^2z$ می‌آید چون درجهٔ y آن بیشتر است. و الی آخر، بالاخره داریم :

$$p(x, y, z) = 3x^3yz - 5x^2y^3 + 8x^2y^2z - 4xy^7z^3 + 2xy^7z - 6yz^8$$

(ب) شکل ۴۵ـ ۵ ساختمان دادهٔ موردنظر را نشان می‌دهد.



شکل ۴۵ـ ۵

مسأله ۱۰ـ ۵ : درصورت وجود، مزایای یک لیست دوطرفه را نسبت به لیست یکطرفه در هر یک از عملیات زیر بیان کنید.

(الف) با پیمایش لیست هر گره را پردازش کند.

(ب) با معلوم‌بودن **LOC** مکان یک گره، آن گره را حذف کند.

(ج) در یک لیست نامرتب عنصر معلوم **ITEM** را جستجو کند.

(د) در یک لیست مرتب عنصر معلوم **ITEM** را جستجو کند.

(ه) یک گره قبل از گره‌ای با مکان معلوم **LOC** اضافه کند.

(و) یک گره بعد از گره‌ای با مکان معلوم **LOC** اضافه کند.

حل : **(الف)** هیچ مزیتی ندارد.

(ب) مکان گرهٔ قبلی موردنیاز است. لیست دوطرفه شامل این اطلاعات است درحالی که با یک لیست یکطرفه باید لیست را پیمایش کنیم.

(ج) هیچ مزیتی ندارد.

(د) هیچ مزیتی ندارد مگر آن که بدانیم **ITEM** باید در انتهای لیست ظاهر شود که در آن حالت لیست از انتها به ابتدا پیمایش می‌شود. برای مثال اگر عمل جستجو را برای پیداکردن **Walker** در یک لیست الفبایی انجام دهیم می‌توان سریعتر لیست را از انتها به ابتدا پیمایش کرد.

(ه) مانند قسمت (ب) لیست دوطرفه کارایی بیشتری دارد.

(و) هیچ مزیتی ندارد.

توجه کنید : در حالت کلی، یک لیست دوطرفه مفیدتر از لیست یکطرفه نیست، مگر در شرایط خاص.

مسأله ۱۱ـ ۵ : فرض کنید **LIST** یک لیست (چرخشی) دارای سرلیست در حافظه باشد. الگوریتمی بنویسید که گرهٔ آخر را از **LIST** حذف کند. این مسأله را با مسأله ۵ـ ۵ مقایسه کنید.

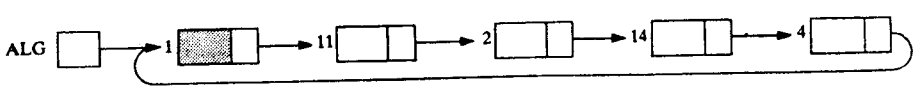
حل : الگوریتم همان الگوریتم **P5.5** است با این تفاوت که اکنون می‌توانیم حالت خاصی را که **LIST** تنها یک گره دارد حذف کنیم. به بیان دیگر، وقتی **LIST** خالی نیست بلافاصله می‌توانیم **SAVE** را تعریف کنیم.

Algorithm P5.11: DELLSTH(INFO, LINK, START, AVAIL)
This algorithm deletes the last node from the header list.

1. [List empty?] If LINK[START] = NULL, then: Write: UNDERFLOW, and Exit.
2. Set PTR := LINK[START] and SAVE := START. [Initializes pointers.]
3. Repeat while LINK[PTR] ≠ START: [Traverses list seeking last node.]
   Set SAVE := PTR and PTR := LINK[PTR]. [Updates SAVE and PTR.]
   [End of loop.]
4. Set LINK[SAVE] := LINK[PTR]. [Removes last node.]
5. Set LINK[PTR] := AVAIL and AVAIL := PTR. [Returns node to AVAIL list.]
6. Exit.

مسأله ۱۲ـ ۵ : با استفاده از لیستهای یکطرفه دارای سرلیست شکل ۴۳ـ ۵ لیستهای دو طرفه بسازید.

حل : لیست **ALG** را از ابتدا تا انتهای لیست پیمایش کنید تا نتیجهٔ زیر بدست آید:



ما نیازمند اشاره‌گرهای انتهای لیست هستیم. این اشاره‌گرها گره به گره محاسبه می‌شوند. برای مثال آخرین گرهٔ با مکان **LOC = 4** باید به گره قبل از گرهٔ آخر اشاره کند که دارای مکان **LOC = 14** است. از این‌رو

BACK[4] = 14

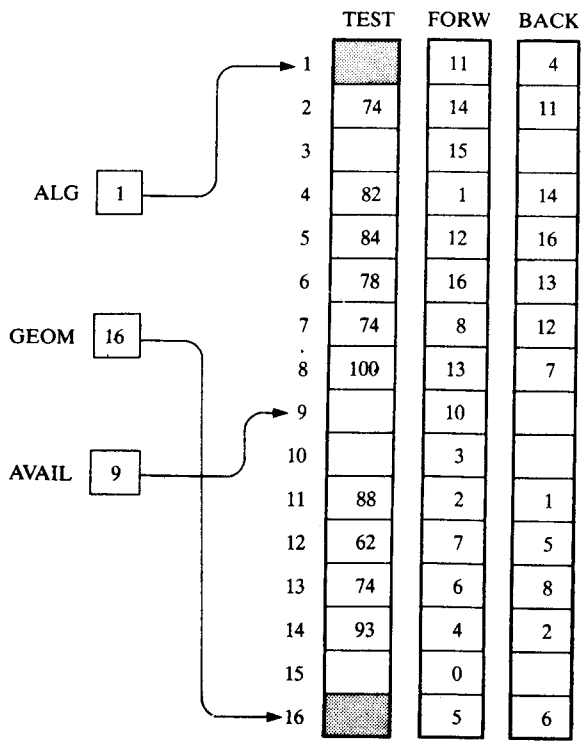
گرهٔ قبل از گرهٔ آخر با مکان (آدرس) **LOC = 14** باید به گرهٔ قبل از آن با مکان **LOC = 2** اشاره کند. از این‌رو

BACK[14] = 2

و الی آخر. سرگره با مکان آدرس **LOC = 1** باید به گرهٔ آخر با مکان **4** اشاره کند. از این‌رو

BACK[1] = 4

در مورد لیست **GEOM** روش مشابه‌ای انجام می‌شود. در شکل ۴۶ـ ۵ لیستهای دوطرفه به تصویر درآمده است. توجه دارید که هیچ تفاوتی بین آرایه‌های **LINK** و **FORW** وجود ندارد. به بیان دیگر، تنها محاسبهٔ آرایهٔ **BACK** موردنیاز است.



شکل ۴۶ـ ۵

مسأله‌های تکمیلی

لیستهای پیوندی

مسأله ۱۳ـ ۵ : شکل ۴۷ـ ۵ لیست بیماران پنج بیمارستان و شماره اتاقهای آنها را نشان می‌دهد.

(الف) مقادیر مربوط به **NSTART** و **NLINK** را به‌گونه‌ای پر کنید که تشکیل یک لیست الفبایی از نامها بدهد.
(ب) مقادیرمربوط به **RSTART** و **RLINK** را به‌گونه‌ای پر کنید که شماره اتاقها را به ترتیب ارائه دهد.

NSTART

RSTART

	NAME	ROOM	NLINK	RLINK
1	Brown	650		
2	Smith	422		
3	Adams	704		
4	Jones	462		
5	Burns	632		

شکل ۴۷ـ ۵

مسأله ۱۴ـ ۵ : شکل ۴۸ـ ۵ یک لیست پیوندی را در حافظه نشان می‌دهد.

START: 4

AVAIL: 3

	INFO	LINK
1	A	2
2	B	8
3		6
4	C	7
5	D	0
6		0
7	E	1
8	F	5

شکل ۴۸ـ ۵

(الف) دنباله کاراکترهای درون لیست را تعیین کنید.
(ب) فرض کنید **F** و آنگاه **C** از لیست حذف شده‌اند و بدنبال آن **G** به ابتدای لیست اضافه شده است. ساختمان نهایی لیست را تعیین کنید.
(ج) فرض کنید **C** و آنگاه **F** از لیست حذف شده‌اند و آنگاه **G** به ابتدای لیست اضافه شده است.

ساختمان نهایی لیست را تعیین کنید.

(د) فرض کنید G به ابتدای لیست اضافه شده است و آنگاه F و بدنبال آن C از این ساختمان حذف شده است. ساختمان نهایی لیست را تعیین کنید.

مسأله ۱۵ـ ۵ : فرض کنید **LIST** یک لیست پیوندی در حافظه و شامل مقادیر عددی باشد. برای هر یک از حالتهای زیر یک زیربرنامهٔ **Procedure** بنویسید که:

(الف) MAX ماگزیمم مقادیر لیست **LIST** را پیدا کنید.

(ب) MEAN میانگین مقادیر لیست **LIST** را پیدا کنید.

(ج) PROD حاصلضرب عناصر لیست **LIST** را پیدا کنید.

مسأله ۱۶ـ ۵ : عدد صحیح **K** داده شده است. یک زیربرنامهٔ **Procedure** بنویسید که عنصر **K** ام را از لیست پیوندی حذف کند.

مسأله ۱۷ـ ۵ : یک زیربرنامهٔ **Procedure** بنویسید که اطلاعات معلوم **ITEM** را به انتهای لیست اضافه کند.

مسأله ۱۸ـ ۵ : یک زیربرنامهٔ **Procedure** بنویسید که عنصر اول یک لیست را حذف کند و آن را بدون هیچ تغییری در مقدارهای **INFO** به انتهای لیست اضافه کند. تنها **START** و **LINK** قابل تغییر هستند.

مسأله ۱۹ـ ۵ : یک زیربرنامهٔ **Procedure SWAP(INFO, LINK, START, K)** بنویسید که بدون هیچ تغییری در مقدارهای **INFO** جای عناصر **K** ام و **K+1** ام لیست را عوض کند.

مسأله ۲۰ـ ۵ : یک زیربرنامهٔ **Procedure SORT(INFO, LINK, START)** بنویسید که بدون هیچ تغییری در مقدارهای **INFO** لیست را مرتب کند.

راهنمایی : از زیربرنامهٔ **Procedure SWAP** مسأله ۱۹ـ ۵ و مرتب‌کردن حبابی استفاده کنید.

مسأله ۲۱ـ ۵ : فرض کنید **AAA** و **BBB** لیستهای پیوندی مرتب شده با عناصر متمایز باشند. این دو لیست در **INFO** و **LINK** نگهداری می‌شوند. یک زیربرنامهٔ **Procedure** بنویسید که لیستها را بدون هیچ تغییری در مقادیر **CCC** تنها در یک لیست پیوندی **INFO** ترکیب کند.

مسأله‌های ۲۲ـ ۵ تا ۲۴ـ ۵ به رشته‌های کاراکتری مربوط می‌شوند که به صورت لیستهای پیوندی ذخیره شده‌اند و در هر گرهٔ آن تنها یک کاراکتر وجود دارد و از همان آرایه‌های **INOF** و **LINK** استفاده می‌کنند.

مسأله ۲۲ـ ۵ : فرض کنید **STRING** یک رشتهٔ کاراکتری در حافظه باشد.

(الف) یک زیربرنامهٔ **Procedure** بنویسید که **SUBSTRING(STRING, K, N)** را چاپ کند یعنی زیررشتهٔ **STRING** را که از کاراکتر **K** ام شروع می‌شود و طول **N** دارد چاپ کند.

(ب) یک زیربرنامهٔ **Procedure** بنویسید که یک رشتهٔ جدید **SUBKN** در حافظه ایجاد کند که در آن

SUBKN = SUBSTRING(STRING, K, N)

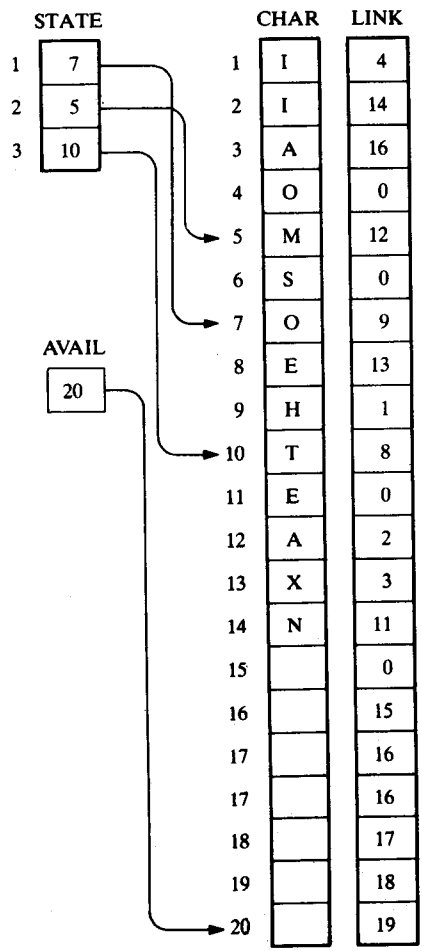
مسأله ۲۳ـ ۵ : فرض کنید **STR1** و **STR2** رشته‌های کاراکتری در حافظه باشند. یک زیربرنامهٔ

Procedure بنویسید که یک رشتهٔ جدید **STR3** ایجاد کند که از اتصال دو رشتهٔ **STR1** و **STR2** بوجود آمده است.

مسأله ۲۴ـ ۵ : فرض کنید **TEXT** و **PATTERN** رشته‌هایی در حافظه باشند. یک زیربرنامهٔ **Procedure** بنویسید که مقدار **INDEX(TEXT, PATTERN)** یعنی مکانی را که در آن **PATTERN** برای اولین بار به صورت یک زیررشتهٔ **TEXT** ظاهر می‌شود پیدا کند.

لیستهای دارای سرلیست، لیستهای دو طرفه

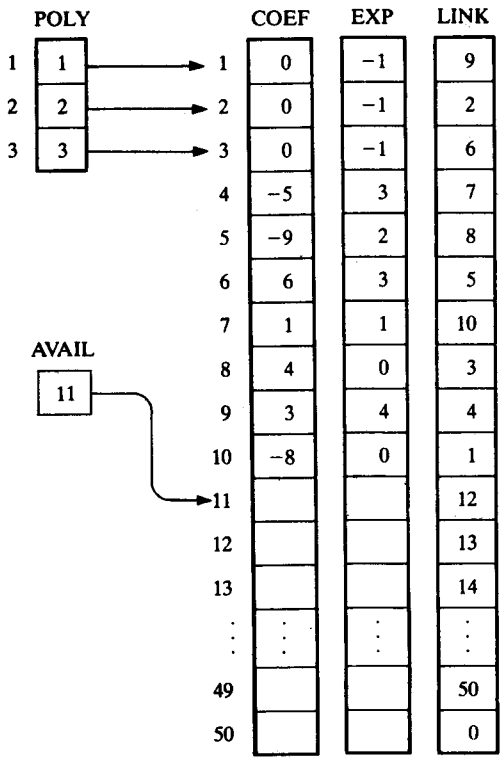
مسأله ۲۵ـ ۵ : رشته‌های کاراکتری در سه لیست پیوندی طبق شکل ۴۹ـ ۵ ذخیره شده‌اند.



شکل ۴۹ـ ۵

(الف) سه رشته را پیدا کنید. (ب) لیستهای چرخشی دارای سرلیست از لیستهای یکطرفه با استفاده از CHAR[20] ، CHAR[19] و CHAR[18] به عنوان سرگره‌ها تشکیل دهید.

مسأله ۲۶ـ ۵ : چندجمله‌ای‌های ذخیره‌شده در سه لیستِ دارای سرلیست شکل ۵۰ـ ۵ را تعیین کنید.



شکل ۵۰ـ ۵

مسأله ۲۷ـ ۵ : چندجمله‌ای زیر را درنظر بگیرید:

$$p(x, y, z) = 2xy^2z^3 + 3x^2yz^2 + 4xy^3z + 5x^2y^2 + 6y^3z + 7x^3z + 8xy^2z^5 + 9$$

(الف) این چندجمله‌ای را طوری بنویسید که جملات آن به ترتیب حروف الفبا باشند.

(ب) فرض کنید جملات این چندجمله‌ای به ترتیب نشان داده شده در اینجا در آرایه‌های موازی COEF ، XEXP ، YEXP و ZEXP که سرگره اولین گره است ذخیره شده‌اند. بنابراین به‌ازای K = 2, 3, ..., 9 COEF[K] = K داریم. مقادیر را در آرایهٔ LINK به گونه‌ای جایگزین کنید که لیست پیوندی شامل دنباله جملات با ترتیب ذکرشده باشد. مسأله ۹ـ ۵ را ببینید.

مسأله ۲۸ـ ۵ : یک زیربرنامهٔ **HEAD(INFO, LINK, START, AVAIL)** بنویسید که از یک لیست یکطرفهٔ معمولی یک لیست چرخشیِ دارای سرلیست بسازد.

مسأله ۲۹ـ ۵ : مجدداً مسأله‌های ۱۶ـ ۵ تا ۲۰ـ ۵ را به جای یک لیست یکطرفه معمولی با استفاده از یک لیست چرخشی دارای سرلیست حل کنید. ملاحظه می‌کنید که الگوریتم اکنون خیلی ساده‌تر شده است.

مسأله ۳۰ـ ۵ : فرض کنید **POLY1** و **POLY2** دو چندجمله‌ای یک متغیره باشند که با استفاده از همان آرایه‌های موازی **COEF** ، **EXP** و **LINK** به صورت لیستهای چرخشی دارای سرلیست ذخیره شده‌اند. یک زیربرنامهٔ **Procedure**

ADD(COEF, EXP, LINK, POLY1, POLY2, AVAIL, SUMPOLY)

بنویسید که مجموع **SUMPOLY** دو چندجمله‌ای **POLY1** و **POLY2** را پیدا کند (که با استفاده از **COEF** ، **EXP** و **LINK** نیز در حافظه ذخیره می‌شود).

مسأله ۳۱ـ ۵ : برای چندجمله‌ای‌های **POLY1** و **POLY2** مسأله ۳۰ـ ۵ یک زیربرنامهٔ **Procedure**

MULT(COEF, EXP, LINK, POLY1, POLY2, AVAIL, PRODPOLY)

بنویسید که حاصلضرب **PRODPOLY** چندجمله‌ای‌های **POLY1** و **POLY2** را پیدا کند.

مسأله ۳۲ـ ۵ : از لیستهای یکطرفه شکل ۴۹ـ ۵ با استفاده از **CHAR[19]** ، **CHAR[20]** و **CHAR[18]** به عنوان سرگره‌ها همانند مسأله ۲۵ـ ۵ ، لیستهای چرخشی دوطرفه با سرلیست بسازید.

مسأله ۳۳ـ ۵ : عدد صحیح **K** داده شده است. یک زیربرنامهٔ **Procedure** به صورت

DELK(INFO, FORW, BACK, START, AVAIL, K)

بنویسید که عنصر **K** ام را از لیست چرخشی دوطرفه با سرلیست حذف کند.

مسأله ۳۴ـ ۵ : فرض کنید **LIST(INFO, LINK, START, AVAIL)** لیست چرخشی یکطرفه با سرلیست در حافظه باشد. یک زیربرنامهٔ **Procedure** به صورت

TWOWAY(INFO, LINK, BACK, START)

بنویسید که مقادیر را در آرایهٔ خطی **BACK** جایگزین کند تا از لیست یکطرفه یک لیست دوطرفه بسازد.

برای مسأله‌های زیر برنامه بنویسید

مسأله‌های ۳۵ـ ۵ تا ۴۰ـ ۵ در ارتباط با ساختمان دادهٔ شکل ۵۱ـ ۵ هستند که از چهار لیست الفبایی مربوط به موکلین و وکیل مدافع‌های آنها تشکیل شده است.

مسأله ۳۵ـ ۵ : برنامه‌ای بنویسید که عدد صحیح **K** را از ورودی بخواند و در خروجی لیست موکلین وکیل مدافع **K** را چاپ کند. برای هر **K** برنامه را آزمایش کنید.

مسأله ۳۶ـ ۵ : برنامه‌ای بنویسید که نام و وکیل مدافع هر موکلی را که سنش برابر **L** یا بزرگتر از **L** است

	LAWYER	POINT
1	Davis	4
2	Levine	12
3	Nelson	21
4	Rogers	8

AVAIL
15

	CLIENT	AGE	LINK
1	Hall	35	16
2	Moss	28	13
3	Ford	47	25
4	Brown	54	22
5	Ginn	38	14
6	Pride	42	29
7			26
8	Berk	38	3
9	White	45	0
10			28
11	Todd	25	0
12	Dixon	32	24
13	Newman	46	6
14	Harris	42	30
15			7
16	Jackson	52	27
17			23
18	Roberts	40	0
19			0
20	Eisen	32	1
21	Adams	48	5
22	Cohen	36	20
23			19
24	Fisher	33	18
25	Graves	42	11
26			10
27	Parker	50	9
28			17
29	Singer	45	0
30	Lewis	28	2

شکل ۵ـ۵۱

چاپ کند. برنامه را با استفاده از (الف) L = 41 و (ب) L = 48 آزمایش کنید.

مسأله ۳۷ـ ۵ : برنامه‌ای بنویسید که نام LLL یک وکیل مدافع را از ورودی بخواند و در خروجی لیست وکلای مدافع موکلین را چاپ کند. برنامه را با استفاده از (الف) Rogers ، (ب) Baker ، (ج) Levine آزمایش کنید.

مسأله ۳۸ـ ۵ : برنامه‌ای بنویسید که NAME یک موکل را از ورودی بخواند و در خروجی نام موکل، سن و وکیل مدافع او را چاپ کند. برنامه را با استفاده از (الف) Newman ، (ب) Ford ، (ج) Rivers و (د) Hall آزمایش کنید.

مسأله ۳۹ـ ۵ : برنامه‌ای بنویسید که NAME موکل را از ورودی بخواند و رکورد موکل را از این ساختمان داده حذف کند. برنامه را با استفاده از (الف) Lewis ، (ب) Klein و (ج) Parker آزمایش کنید.

مسأله ۴۰ـ ۵ : برنامه‌ای بنویسید که رکورد یک موکل جدید را که از نام موکل، سن و وکیل مدافع او تشکیل شده است از ورودی بخواند و رکورد را در ساختمان داده اضافه کند. برنامه را با استفاده از (الف) Jones, 36, Levine و (ب) Olsen, 44, Nelson آزمایش کنید.

مسأله‌های ۴۱ـ ۵ تا ۴۶ـ ۵ در ارتباط با لیست الفبایی رکورد کارمندان شکل ۳۰ـ ۵ هستند که به صورت لیست چرخشی با سرلیست ذخیره شده‌اند.

مسأله ۴۱ـ ۵ : برنامه‌ای بنویسید که در خروجی لیست الفبایی رکورد تمام کارمندان را چاپ کند.

مسأله ۴۲ـ ۵ : برنامه‌ای بنویسید که نام NNN یک کارمند را از ورودی بخواند و در خروجی رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) Evans ، (ب) Smith و (ج) Lewis آزمایش کنید.

مسأله ۴۳ـ ۵ : برنامه‌ای بنویسید که شمارهٔ تأمین اجتماعی SSS یک کارمند را از ورودی بخواند و رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) 3351 – 64 – 165 ، (ب) 6262 – 46 – 136 و (ج) 5555 – 44 – 177 آزمایش کنید.

مسأله ۴۴ـ ۵ : برنامه‌ای بنویسید که عدد صحیح K را از ورودی بخواند و نام هر کارمند مذکر را وقتی K = 1 است یا نام هر کارمند مؤنث را وقتی K = 2 است در خروجی چاپ کند. برنامه را با استفاده از (الف) K = 2 ، (ب) K = 5 و (ج) K = 1 آزمایش کنید.

مسأله ۴۵ـ ۵ : برنامه‌ای بنویسید که نام NNN یک کارمند را از ورودی بخواند و رکورد این کارمند را از ساختمان داده حذف کند. برنامه را با استفاده از (الف) Davis ، (ب) Jones و (ج) Rubin آزمایش کنید.

مسأله ۴۶ـ ۵ : برنامه‌ای بنویسید که رکوردیک کارمند جدید را از ورودی بخواند و رکورد را در فایل اضافه کند. برنامه را با استفاده از (الف) Fletcher, 168 – 52 – 3388, Female, 21 000 و (ب) Nelson, 175 – 32 – 2468, Male, 19 000 آزمایش کنید.

توجه کنید : خاطرنشان می‌کنیم که هنگام اضافه کردن و حذف رکورد، رکورد سرگره را تازه کنید.

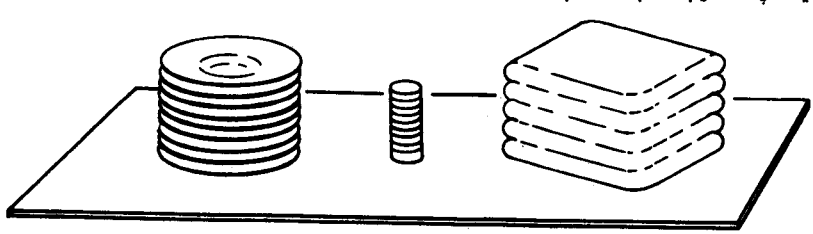
فصل ۶

پشته‌ها، صفها، زیربرنامه‌های بازگشتی

۱ـ ۶ مقدمه

لیستها و آرایه‌های خطی که در فصل‌های قبل مورد بررسی قرار گرفته‌اند، اجازه می‌دادند که عناصر را در هر مکانی از لیست، ابتدای لیست، انتهای لیست یا وسط لیست حذف یا اضافه کنیم. در علم کامپیوتر اغلب وضعیتهایی پیش می‌آید که می‌خواهیم در عمل اضافه‌کردن و حذف عناصر لیست محدودیت‌هایی ایجاد کنیم طوری که این عملیات تنها در ابتدا یا در انتهای لیست انجام شود نه در وسط لیست. دو ساختمان داده‌ای که برای این منظور مفید هستند عبارتند از : **پشته‌ها و صفها.**

یک پشته، یک ساختمان خطی است که در آن عمل اضافه‌کردن یا حذف عنصر تنها در یک انتهای آن امکان‌پذیر است. شکل ۱ـ ۶ سه مثال از این ساختمان داده را که در زندگی روزمره اتفاق می‌افتد نشان می‌دهد. یک پشته از بشقاب، یک پشته از سکه و یک پشته از حوله تا شده.



یک پشته از حولهٔ تاشده یک پشته از سکه یک پشته از بشقاب

شکل ۱ـ ۱۶