داده ساختارها و الگوریتم ها

Data Structures

میلاد سلطانی

فهرست مطالب فصل چهارم

- مقدمه
- پیاده سازی لیست پیوندی
- اعمال لیست پیوندی
  - درج گره
  - حذف گره
- کاربردهای لیست پیوندی
- انواع ساختار لیست پیوندی
- اعمال لیست دو پیوندی
  - درج
  - حذف

لیست پیوندی

Linked List

مقدمه

- در فصل ها گذشته ساختمان داده های آرایه، پشته و صف را بررسی کردیم.
- این ساختمان‌های داده دارای یک اشکال عمده هستند و آن هم اینست که از یک حافظه ثابت برای آنها استفاده می شود. این عمل دو عیب دارد:

۱) ممکن است فقط بخشی از این فضا استفاده شده و بخش دیگر به هدر خواهد رفت.

۲) ممکن است در حین اجرای برنامه به فضای بیشتری برای ذخیره عناصر نیاز باشد که امکان افزایش حجم برای این ساختمان داده ها نیست و لذا سرریز اتفاق می‌اُفتد.

- ساختمان داده لیست پیوندی، تمام مشکلات ذکر شده را رفع می کند.

پیاده سازی لیست پیوندی

- لیست پیوندی را باید بصورت زیر در نظر گرفت :

لیست

مجموعه‌ای از عناصر داده‌ها:

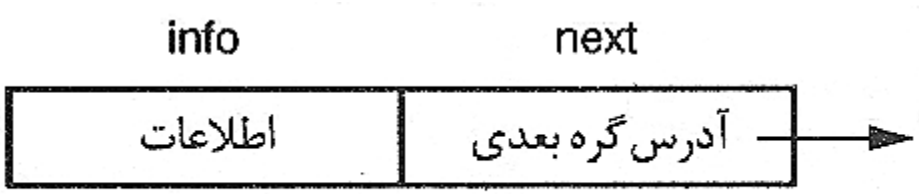
- مجموعه‌ای مرتب (دنباله متناهی) از عناصر داده‌ها است.

عملیات اصلی:

- **ایجاد لیست:** یک لیست خالی را ایجاد می‌کند.
- **تست خالی‌بودن:** بررسی می‌کند که آیا لیست خالی است یا خیر.
- **پیمایش لیست:** دستیابی و پردازش عناصر لیست.
- **درج:** مقداری را در هر نقطه لیست درج می‌کند.
- **حذف:** مقداری را از هر نقطه لیست حذف می‌کند.

ساختار لیست پیوندی

- به هر عنصر لیست پیوندی یک گره (node) گفته می شود.
- هر گره دارای دو فیلد است : فیلد اطلاعات و فیلد آدرس بعدی

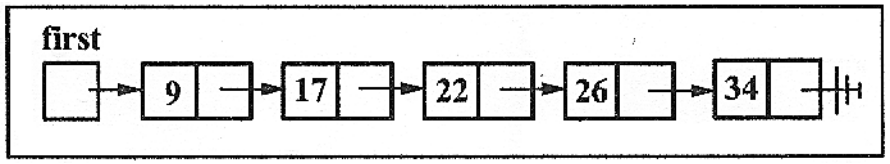


گره لیست پیوندی.

نکته ! چون هر گره لیست پیوندی آدرس گره بعدی را دارد، نیازی نیست گره های لیست پوندی در حافظه پست سر هم قرار گیرند.

نکته ! به فیلد آدرس اشاره گر هم گفته می شود.

نکته ! برای تشخیص انتهای لیست، فیلد آدرس آخرین گره تهی (null) می شود که با ⊣ نشان می دهیم.



پیاده سازی لیست پیوندی

- ساختار گره لیست پیوندی به کمک ساختمان ها تعریف می شود :

```
struct   node {
         int info;
         node *next;
};
```

نکته ! تعریف node یک تعریف بازگشتی است زیرا از خود node در داخل ساختمان استفاده شده است.

تعریف اشاره گر خارجی: اشاره گری است که برای دسترسی به لیست پیوندی استفاده می شود.

```
node *p1, *p2, *p3;
```

ایجاد گره جدید: برای ذخیر سازی عناصر بیشتر در حین اجرای برنامه.

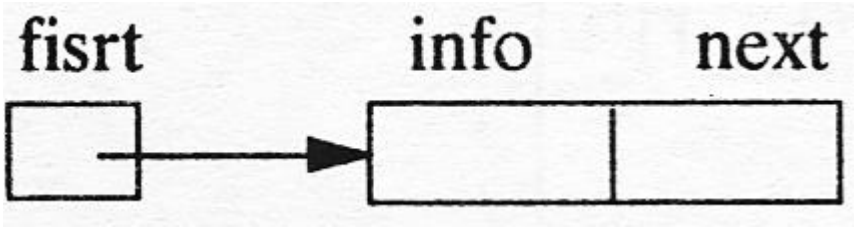
```
node *first;
first = (node *) malloc(sizeof(struct node));
```

پیاده سازی لیست پیوندی

- در دستورات ایجاد گره جدید :

  - تابع malloc() در زبان C برای اخذ حافظه از سیستم استفاده می شود.
  - دستور sizeof() در زبان C نشان دهنده اندازه مورد نیاز است.

در نهایت لیست پیوندی مانند زیر ایجاد می شود که first یک اشاره گر خارجی می باشد.



پیاده سازی لیست پیوندی

مراجعه به گره های لیست: مثلاً اگر p یک اشاره گر خارجی به یک گره از لیست باشد:

- ✓ برای مراجعه به فیلد آدرس آن گره p-> next را می نویسیم.
- ✓ برای مراجعه به فیلد اطلاعات آن گره p-> info را می نویسیم.

نمایش اشاره گر تهی: اشاره گر تهی با null مشخص شده و با ⊣⊢ نمایش داده می شود.

آدرس شروع لیست: فرض شده که آدرس شروع لیست پیوندی با اشاره گر خارجی first بیان می شود.

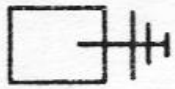
برگرداندن گره به حافظه سیستم: اگر به یک گره لیست پیوندی نیازی نباشد، برای جلوگیری از اتلاف حافظه آن را حذف و به حافظه سیستم برمی گردانیم. تابع free() در زبان C عمل آزاد کردن حافظه را انجام می دهد.

```
free(first);
```

پیاده سازی اعمال روی لیست پیوندی

- **ایجاد لیست پیوندی:** یک اشاره گر خارجی first ایجاد کرده و آنرا برابر با تهی قرار می دهیم.

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



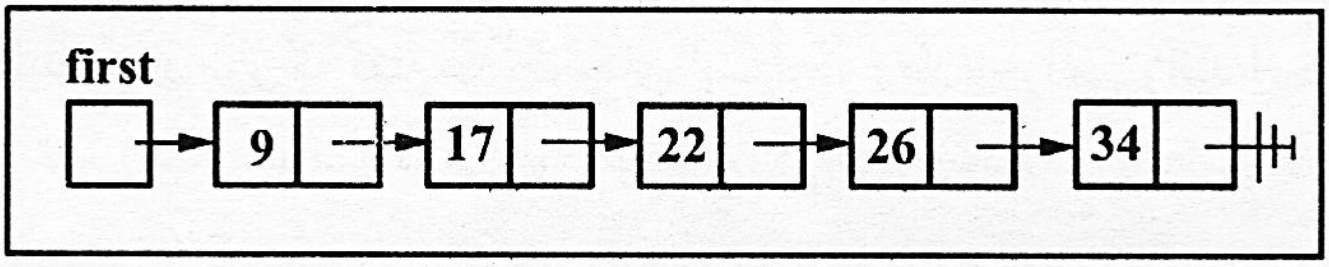
- **تست خالی بودن:** باید بررسی کنیم که آیا first برابر با تهی است یا نه !

```
if(first == NULL)
    /* لیست خالی است */
```

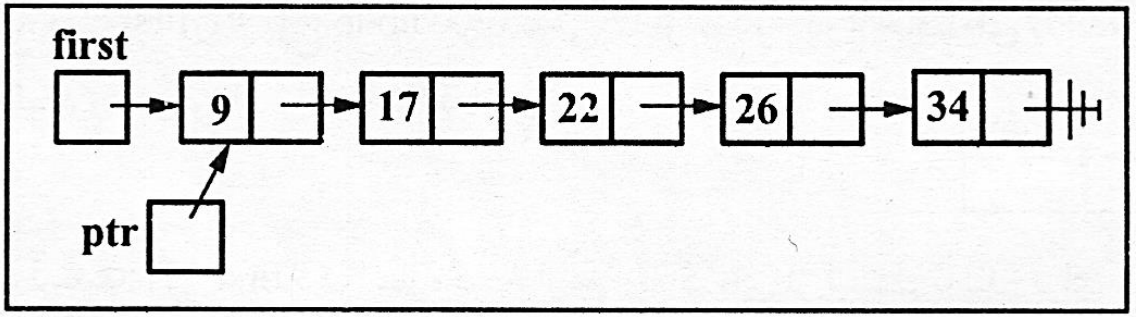
- **پیمایش لیست پیوندی:** یعنی به تمام عناصر لیست دسترسی داشته باشیم.

```
ptr = ptr -> next        (ptr به جایی اشاره می‌کند که next فعلی به آن اشاره می‌کند)
```

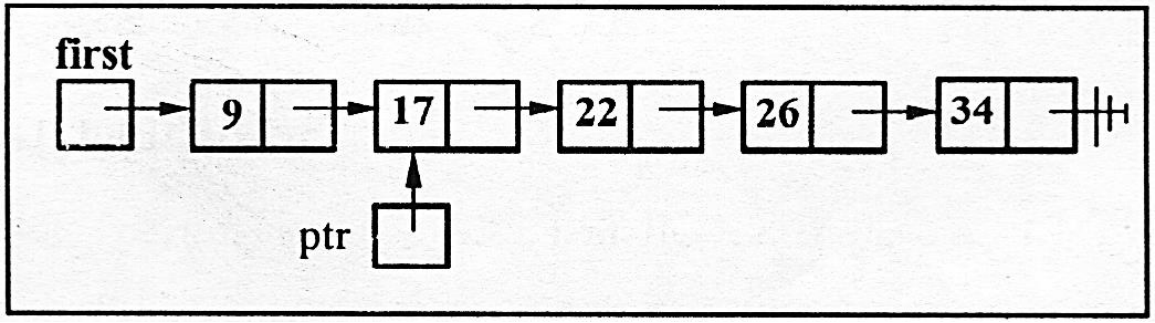
مثال (پیمایش لیست پیوندی): لیست زیر را فرض کنید.



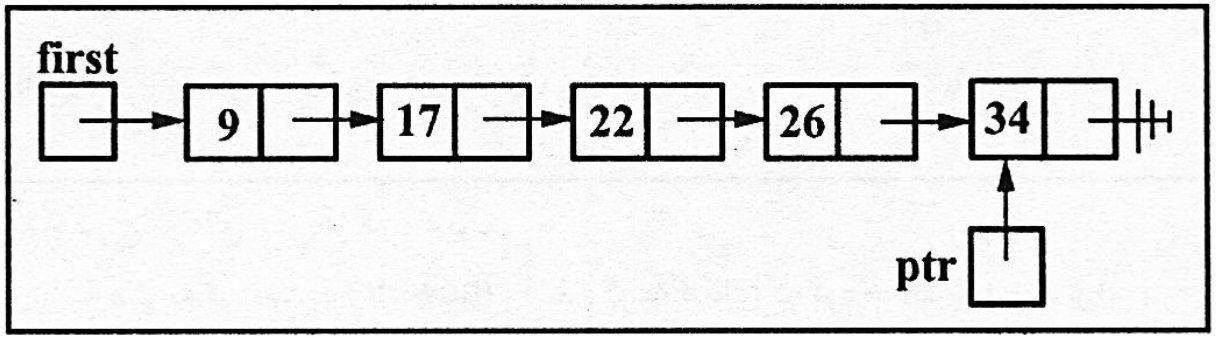
با دستور ptr = first اشاره گر ptr به first اشاره خواهد کرد.



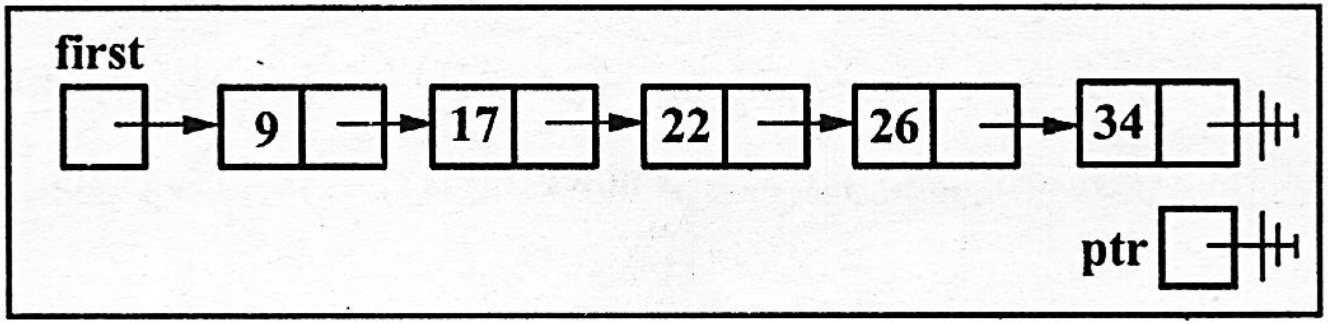
با دستور ptr = ptr -> next اشاره گر ptr بصورت زیر خواهد بود.



با تکرار دستور ptr = ptr -> next اشاره گر ptr بصورت زیر خواهد شد.



با آخرین تکرار دستور ptr = ptr -> next اشاره گر ptr بصورت زیر خواهد شد.



در انتها چون ptr برابر با تهی می شود، پیمایش لیست خاتمه می یابد.

پیاده سازی اعمال روی لیست پیوندی

- با توضیحات عنوان شده در مثال، الگوریتم پیمایش لیست پیوندی بصورت زیر است:

```
ptr = first;
while(ptr != NULL)
{
    process(ptr -> info);   // پردازش گره
    ptr = ptr -> next;      // گره بعدی
}
```

- **درج در لیست پیوندی:** ابتدا باید گره جدید ایجاد شود و اطلاعات لازم در فیلد info آن درج شود.

```
newPtr = (node *) malloc(sizeof(struct node));
```

درج گره در لیست پیوندی

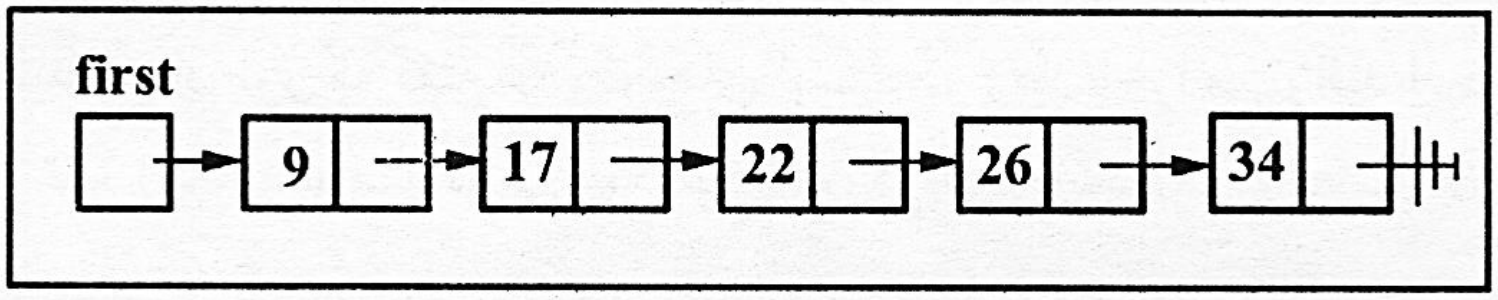
- پس از ایجاد گره و بارگذاری اطلاعات، نوبت به درج آن در لیست می رسد. برای درج دو حالت بوجود می آید:

۱. درج گره جدید بعد از گره ای در لیست
۲. درج گره در ابتدای لیست

۱) الگوریتم درج گره جدید بعد از گره ای در لیست

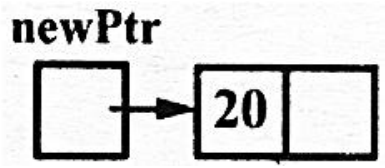
```
newPtr -> next = ptr -> next;
ptr -> next = newPtr;
```

مثال (درج گره جدید بعد از گره ای در لیست): لیست زیر را در نظر بگیرید.



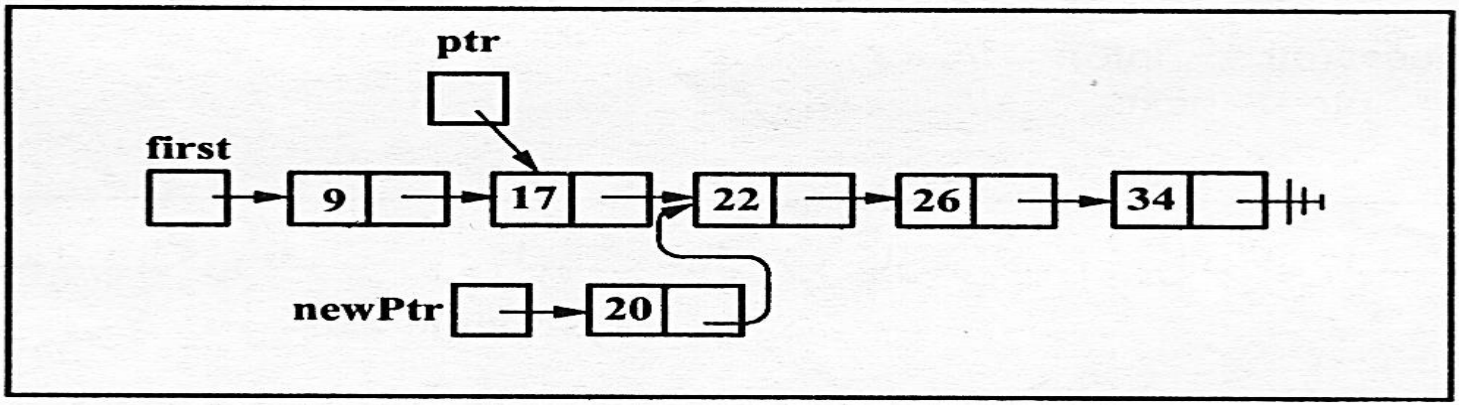
۱. گره جدید ایجاد کرده و آدرس آنرا در newptr گذاشته و بخش داده آنرا برابر با 20 قرار دهید.

```
newPtr = (node *) malloc(sizeof(struct node));
newPtr -> info = 20;
```



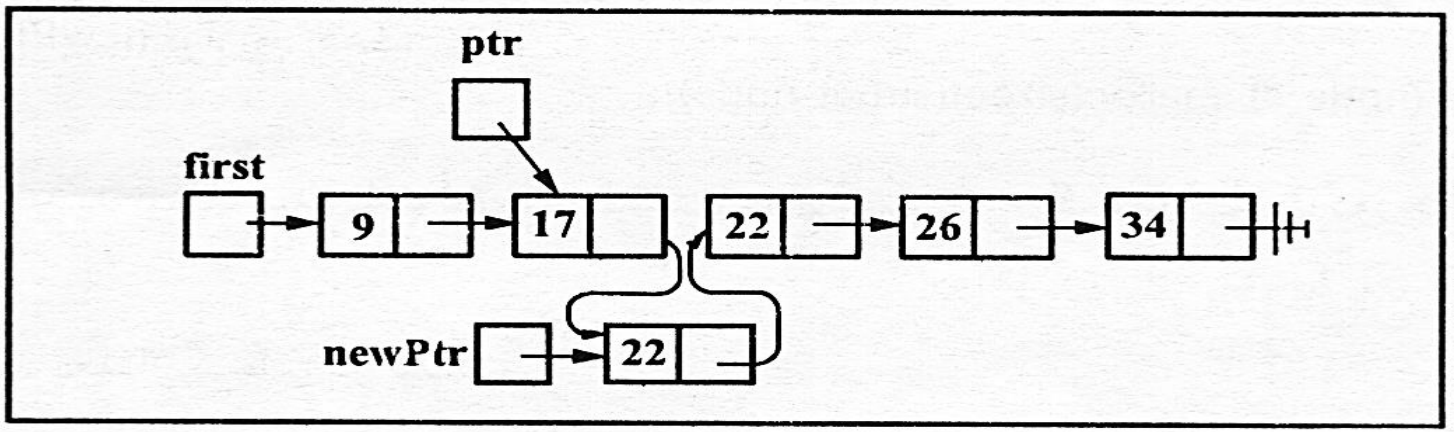
۲. فیلد آدرس گره ای که newptr به آن اشاره می کند با بخش آدرس گره ای که ptr به آن اشاره می کند برابر قرار دهید.

```
newPtr -> next = ptr -> next;
```



۳. فیلد آدرس گره ای که ptr به آن اشاره می کند برابر با newptr قرار دهید.

```
ptr -> next = newPtr;
```

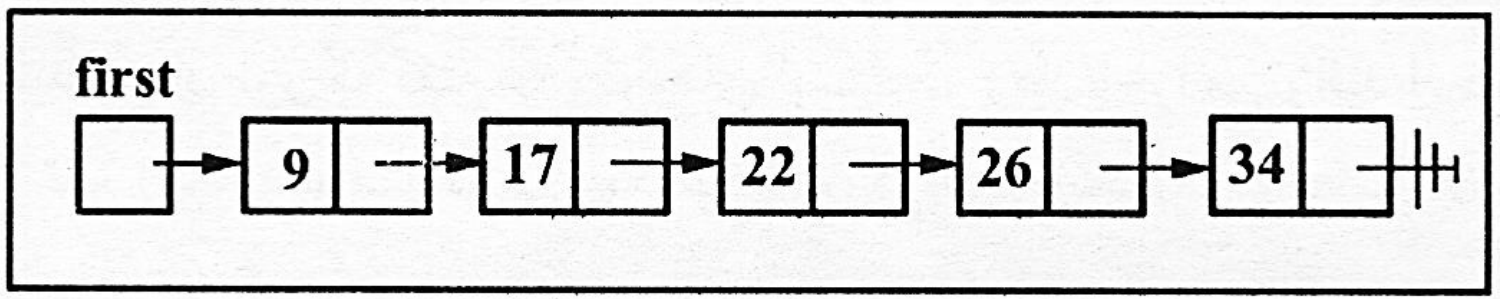


درج گره در لیست پیوندی

۲) الگوریتم درج گره جدید در ابتدای لیست

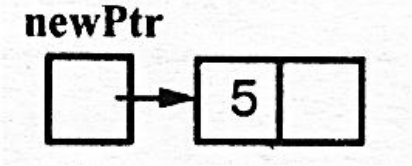
```
newPtr -> next = first;
first = newPtr;
```

مثال (درج گره جدید در ابتدای لیست): لیست پیوندی زیر را در نظر بگیرید.



۱. گره جدید ایجاد کرده و آدرس آنرا در newptr گذاشته و بخش داده آنرا برابر با 5 قرار دهید.

```
newPtr = (node *) malloc(sizeof(struct node));
newPtr -> info = 5;
```

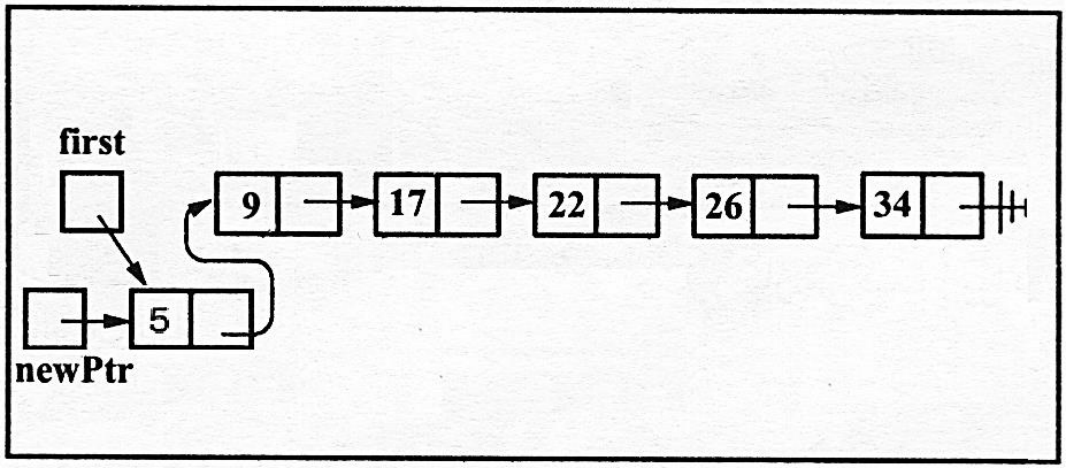


۲. فیلد آدرس گره جدید را برابر با first قرار دهید تا به ابتدای لیست اشاره کند.

```
newPtr -> next = first;
```

۳. first را تغییر دهید تا به ابتدای لیست جدید اشاره کند.

```
first = newPtr;
```



حذف گره از لیست پیوندی

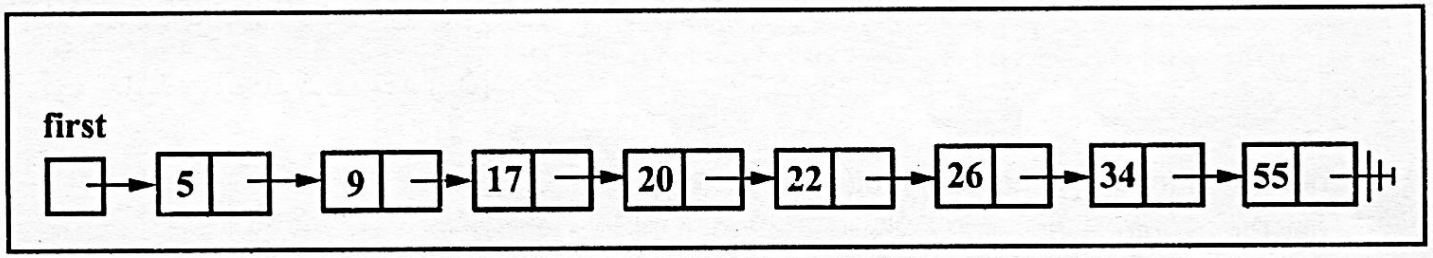
- برای حذف گره از لیست پیوندی دو حالت بوجود می آید:

۱. حذف گره ای که قبل از آن گره ای وجود دارد
۲. حذف گره از ابتدای لیست

۱) الگوریتم حذف گره که قبل از آن گره ای وجود دارد

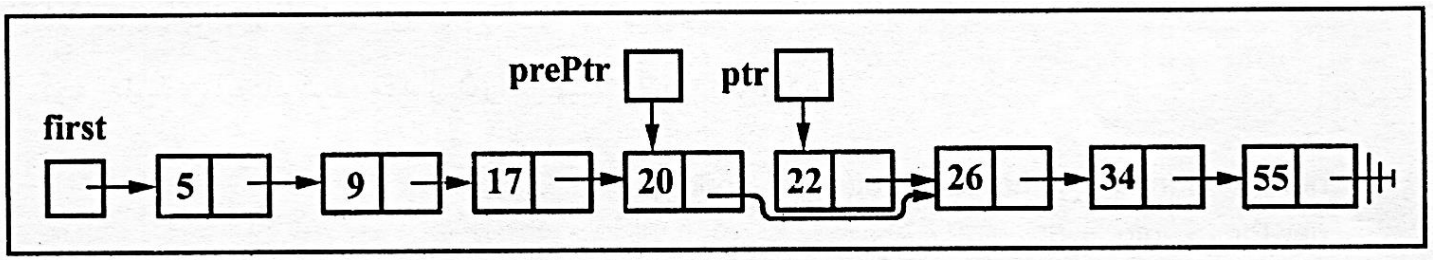
```
prePtr -> next = ptr -> next;
free(ptr);
```

مثال (حذف گره که دارای گره قبل از خود است): لیست زیر را در نظر بگیرید.



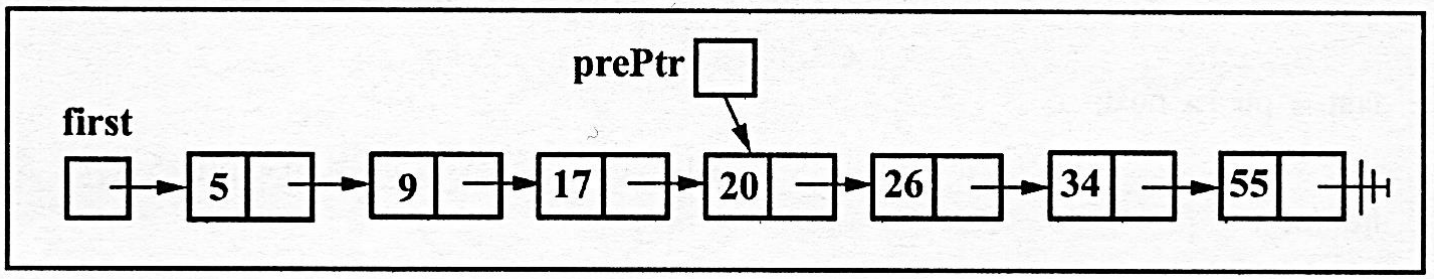
۱. فیلد آدرس preptr را برابر با فیلد آدرس گره ptr قرار دهید.

```
prePtr -> next = ptr -> next
```



۲. گره ای که ptr به آن اشاره می کند به مخزن حافظه بر گردانید.

```
free(ptr);
```

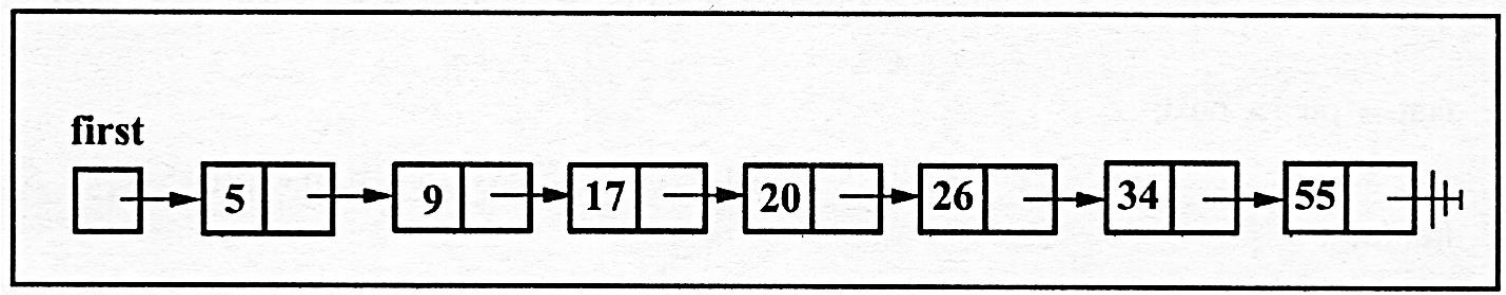


حذف گره از لیست پیوندی

۲) الگوریتم حذف گره از ابتدای لیست

```
ptr = first;
first = ptr -> next;
free(ptr);
```

مثال (حذف گره از ابتدای لیست): لیست زیر را در نظر بگیرید.



۱. ptr به جایی اشاره کند که first به آن اشاره می کند.

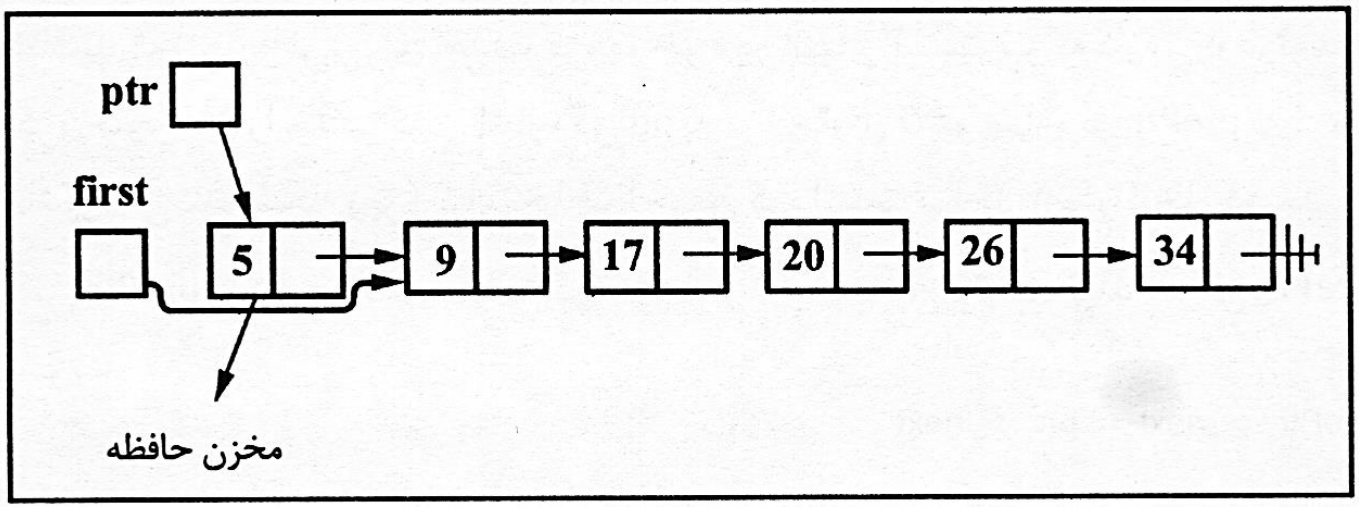
```
ptr = first;
```

۲. first به جایی اشاره کند که فیلد آدرس ptr به آنجا اشاره می کند.

```
first = ptr -> next;
```

۳. گره ای را که ptr به آن اشاره می کند به مخزن حافظه بر گردانید.

```
free(ptr);
```



کاربرد لیست پیوندی

پیاده سازی چند جمله ای ها

- چند جمله ای یک متغیره بر حسب X بصورت زیر تعریف می شود:

$$P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

- در این چند جمله ای a_i ها ضرایب هستند و بزرگترین توان X را درجه چند جمله ای می گویند.

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

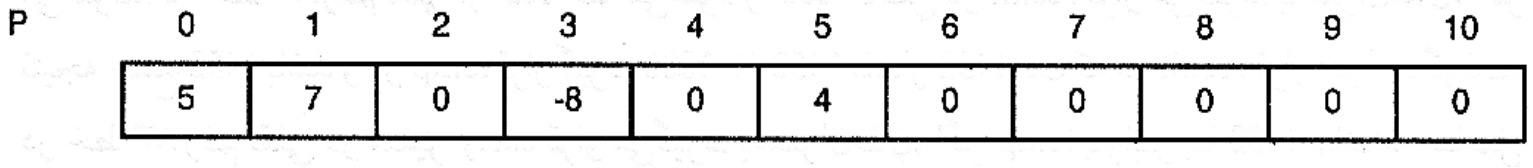
- چند جمله ای را می توان بصورت لیستی از ضرایب در نظر گرفت:

$$P(x) = 5 + 7x + 0x^2 - 8x^3 + 0x^4 + 4x^5 + 0x^6 + 0x^7 + 0x^8 + 0x^9 + 0x^{10}$$

(5, 7, 0, -8, 0, 4, 0, 0, 0, 0, 0)

پیاده سازی چند جمله ای ها

- لیست ضرایب چند جمله ای می تواند با آرایه بصورت زیر ذخیره شود:
  - توان های جملات چند جمله ای به عنوان اندیس آرایه در نظر گرفته می شود.
  - ضرایب جملات متناظر با توان هر جمله، در سلول آرایه با همان اندیس ذخیره می شود.



- اگر درجه چند جمله ای از حد بالای آرایه بیشتر نباشد و تعداد جملات با ضریب صفر زیاد نباشند، می توان از آرایه برای نمایش چند جمله ای استفاده کرد.

- اگر در یک چند جمله ای تعداد ضرایب صفر آن زیاد باشد، به آن چند جمله ای اسپارس (خلوت) گفته می شود.

پیاده سازی چند جمله ای ها

- به عنوان مثال برای نمایش چند جمله ای خلوت زیر بایستی یک آرایه به طول ۱۰۰ در نظر گرفته شود که ۹۸ عنصر آن صفر و ۲ عنصر آن غیر صفر خواهد بود.

$$Q(x) = 5 + x^{99}$$

$$Q(x) = 5 + 0x + 0x^{2} + \ldots + 0x^{98} + 1x^{99}$$

- اگر در چنین چند جمله ای هایی فقط ضرایب غیر صفر ذخیره شود، در مصرف حافظه صرف جویی خواهد شد. برای این کار جملات را با جفت های **(توان ، ضریب)** نمایش خواهیم داد.

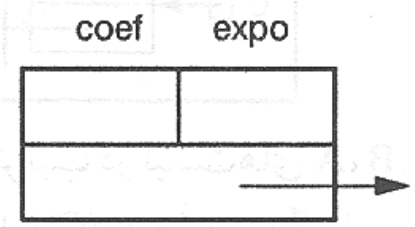
$$P(x) = 5 + 7x - 8x^{3} + 4x^{5} \Leftrightarrow ((5, 0), (7, 1), (-8, 3), (4, 5))$$

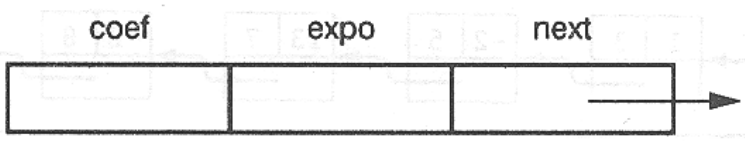
$$Q(x) = 5 + x^{99} \Leftrightarrow ((5, 0), (1, 99))$$

پیاده سازی چند جمله ای ها

- برای نمایش جفت های در نظر گرفته شده از لیست پیوندی استفاده می شود که دارای ساختار زیر است:

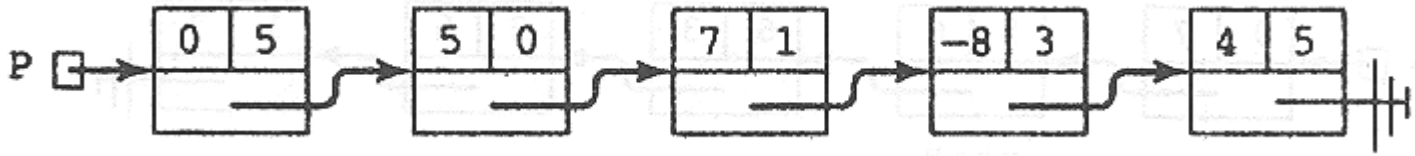
```
struct   node {
         int coef;
         int expo;
         node  *next;
};
```





- در این ساختار coef برای ضریب و expo برای توان استفاده می شود و next به گره بعدی اشاره می کند.

$P(x) = 5 + 7x - 8x^3 + 4x^5 \Leftrightarrow ((5, 0), (7, 1), (-8, 3), (4, 5))$



ساختارهای دیگر لیست پیوندی

- اشکال دیگری از لیست پیوندی که می تواند مورد استفاده قرار گیرد به شرح زیر است:

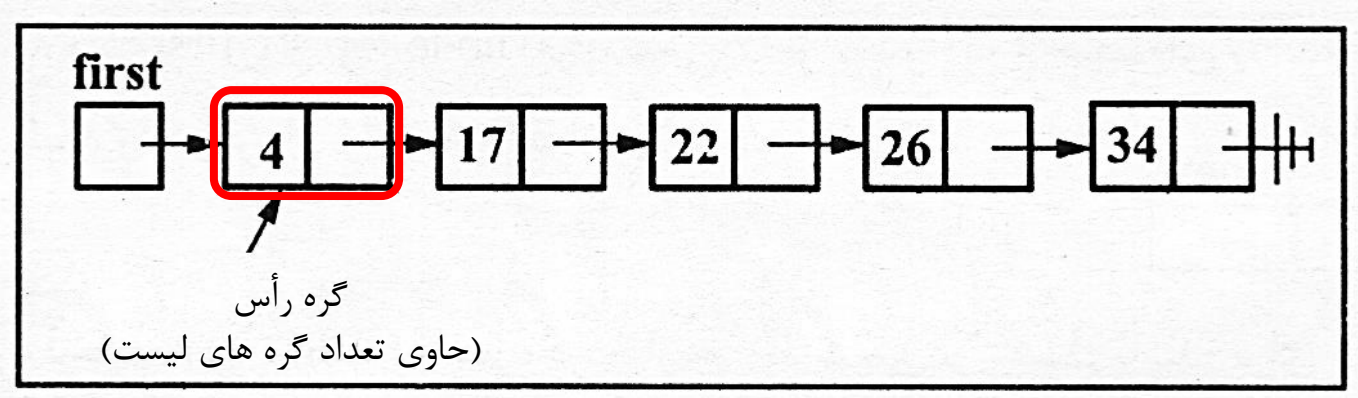
✦لیست های با گره رأس (گره ابتدایی) و گره انتهایی

✦لیست های حلقوی

✦لیست های دو پیوندی

لیست های با گره رأس و گره انتهایی

- گره اضافی که در ابتدای لیست قرار می گیرد و عضوی از لیست محسوب نمی شود، گره رأس (Head Node) نامیده می شود.
- فیلد اطلاعات مربوط به گره رأس معمولاً برای نگهداری اطلاعات کلی در مورد لیست بکار می رود.

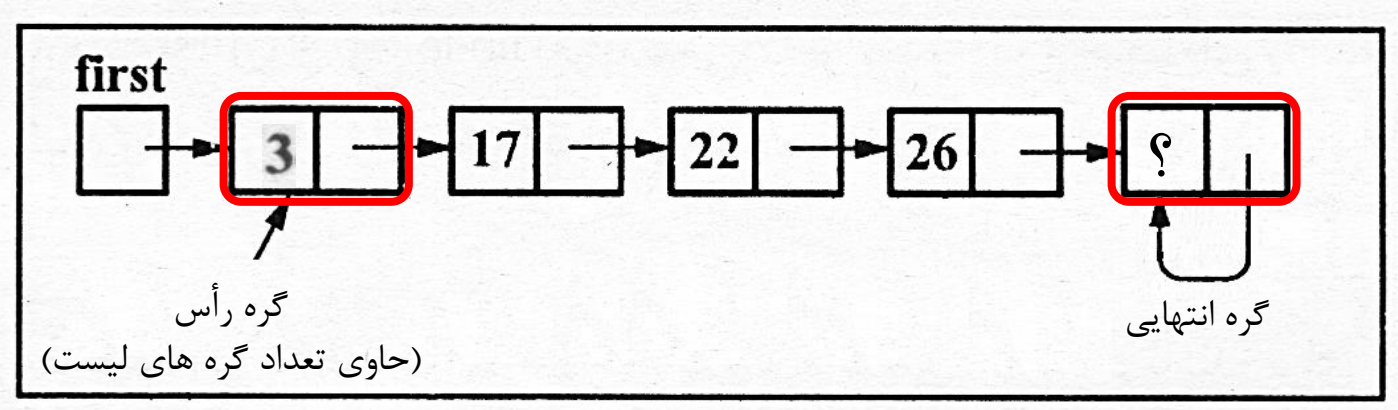


لیست های با گره رأس و گره انتهایی

- ویژگی های این گونه از لیست ها عباتند از:

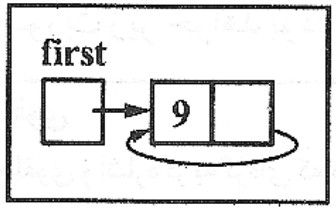
- گره رأس باعث می شود که هر گره لیست دارای یک گره قبلی باشد. به این ترتیب، حتی لیست خالی نیز دارای گره رأس می باشد.

- گره انتهایی باعث می شود که هر گره لیست دارای یک گره بعدی باشد. فیلد آدرس گره انتهایی معمولاً به خودش اشاره می کند.

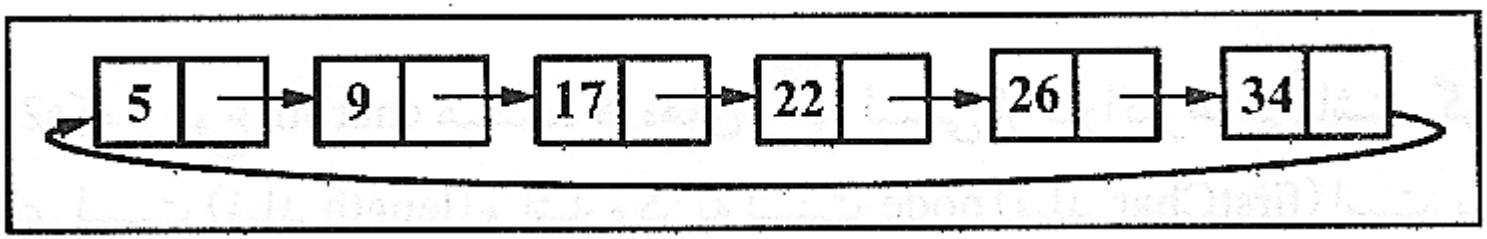


لیست پیوندی حلقوی

- اگر در لیست پیوندی، فیلد آدرس آخرین گره به اولین گره لیست اشاره کند، به آن لیست پیوندی حلقوی می گویند.



- در هنگام ساخت لیست پیوندی خطی، اشاره گر first همیشه به ابتدای لیست اشاره می کرد. اما بهتر است در لیست پیوندی حلقوی، اشاره گر first به انتهای لیست اشاره کند.



ایجاد لیست پیوندی حلقوی

الگوریتم ایجاد لیست پیوندی حلقوی

ورودی: اشاره‌گر ابتدای لیست حلقوی (first)

خروجی: لیست حلقوی

```
newPtr = (struct node*) malloc(sizeof(struct node));
newPtr -> info = item;
if(first == NULL)  /* لیست خالی است */
    newPtr -> next = newPtr;
else {
    newPtr -> next = first -> next;
    first -> next = newPtr;
}
first = newPtr;
```

حذف از لیست پیوندی حلقوی

- علاوه بر لیست خالی، در لیستی دارای یک گره نیز باید حالت خاصی در نظر گرفته شود، زیرا در این حالت، پس از حذف گره، لیست خالی می شود.

الگوریتم حذف گره از لیست حلقوی

ورودی: اشاره‌گر به ابتدای لیست حلقوی و اشاره‌گر به گره‌ای که گره بعد از آن باید حذف شود

خروجی: لیستی که گره‌ای از آن حذف شده است

```
if(first == NULL)
    /* لیست خالی است */
else
{
    ptr = prePtr -> next;
    if(ptr == prePtr)     /* فقط یک گره وجود دارد */
        first = null;
    else
        prePtr -> next = ptr -> next;
    free(ptr);
}
```

پیمایش لیست پیوندی حلقوی

- در لیست پیوندی خطی، اشاره گری را برای پیمایش به انتهای لیست حرکت می دهند تا به تهی تبدیل شود. ولی در لیست حلقوی ابتدا کنترل می شود که لیست خالی نباشد، سپس لیست پیمایش می شود.

الگوریتم پیمایش لیست حلقوی

ورودی: اشاره‌گر به ابتدای لیست

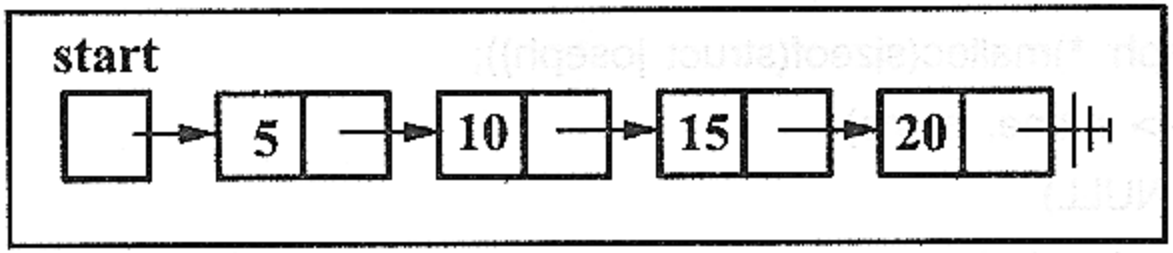
خروجی: محتویات لیست حلقوی

```
if(first != NULL)     /* اگر لیست خالی نیست */
{
    ptr = first;
    do {
        process(ptr -> data);
        ptr = ptr -> next;
    } while(ptr != first);
}
```

لیست های پیوندی مرتب

- لیستهایی هستند که بر اساس یک فیلد از اطلاعات موجود در گره، بصورت صعودی یا نزولی مرتب هستند.

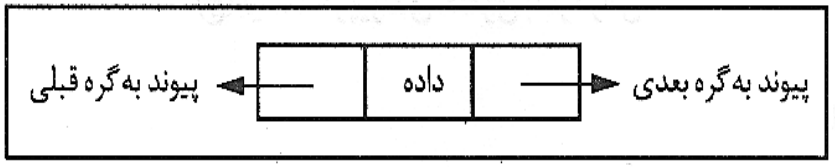
مثال: لیست زیر بصورت صعودی مرتب است.



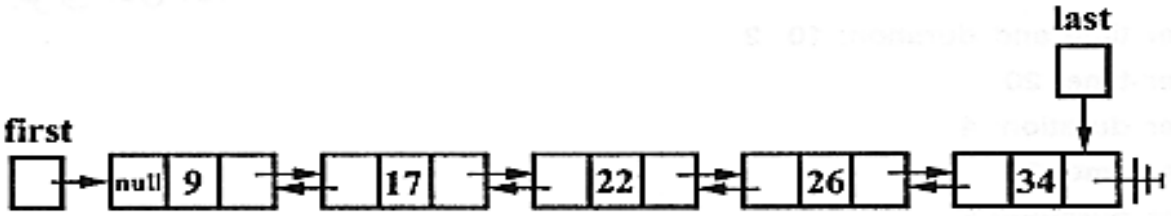
لیست های دو پیوندی

- شکل دیگری از لیست های پیوندی به نام لیست دوپیوندی وجود دارد که هر گره آن دارای دو پیوند است. یک پیوند به گره بعدی و یک پیوند به گره قبلی اشاره می کند.

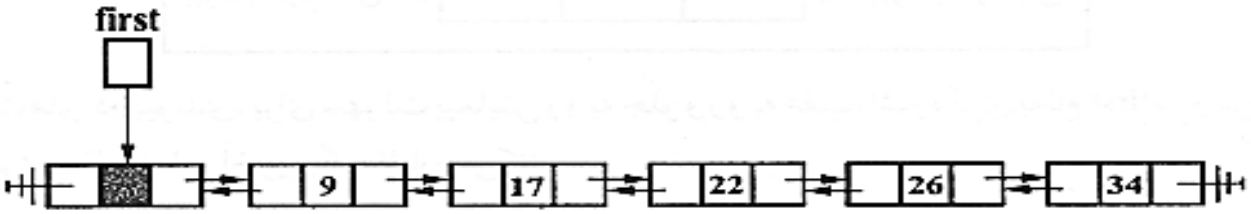
```
struct   node {
         struct node *left;
         int info;
         struct node *right;
};
```



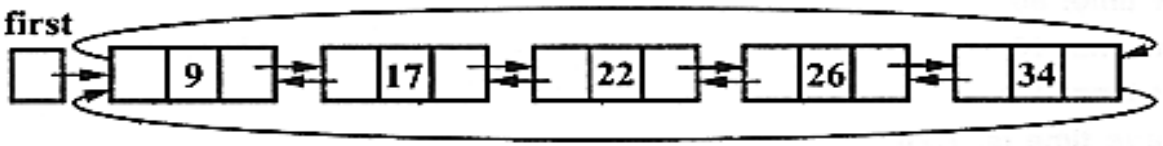
- در لیست دو پیوندی، جهت سهولت پیمایش، اشاره گر first به اولین گره و اشاره گر last به آخرین گره اشاره می کنند.



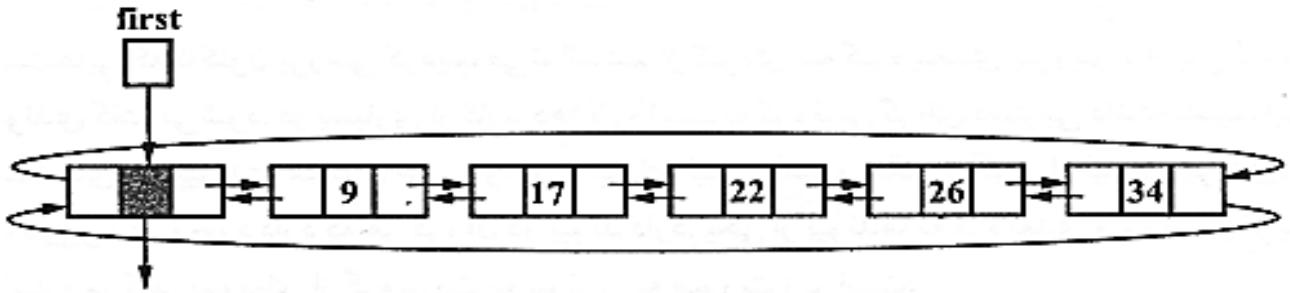
(الف) لیست دوپیوندی.



(ب) لیست دوپیوندی باگره رأس.

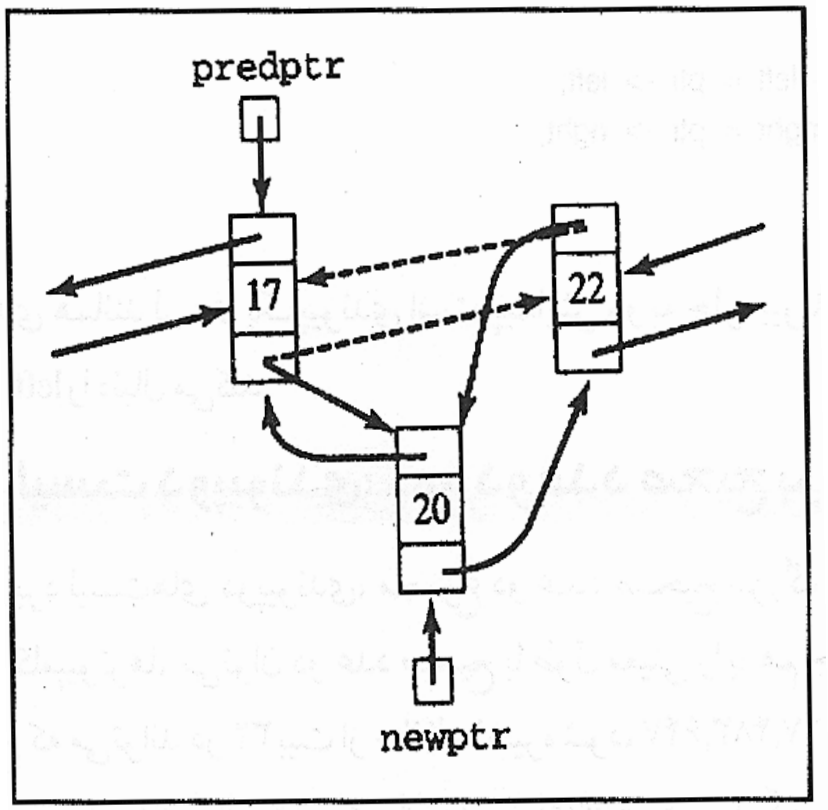


(ج) لیست دوپیوندی حلقوی.



(د) لیست دوپیوندی حلقوی باگره رأس.

درج گره جدید در لیست دو پیوندی

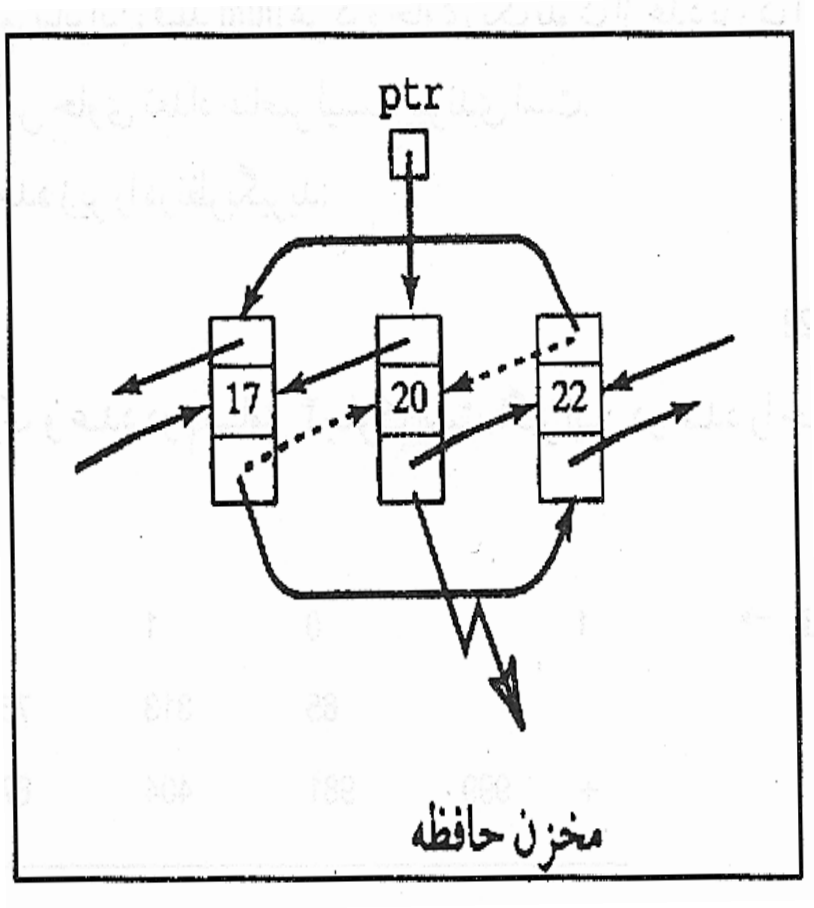


- برای درج یک گره در لیست دو پیوندی از دستورات زیر استفاده می کنیم :

```
newPtr -> left = predPtr;
newPtr -> right = predPtr -> right;
predPtr -> right -> left = newPtr;
predPtr -> right = newPtr;
```

شکل روبرو نمونه ای از درج گره در لیست دو پیوندی را نشان می دهد.

حذف گره از لیست دو پیوندی



- برای حذف یک گره از لیست دو پیوندی دستورات زیر استفاده می شود :

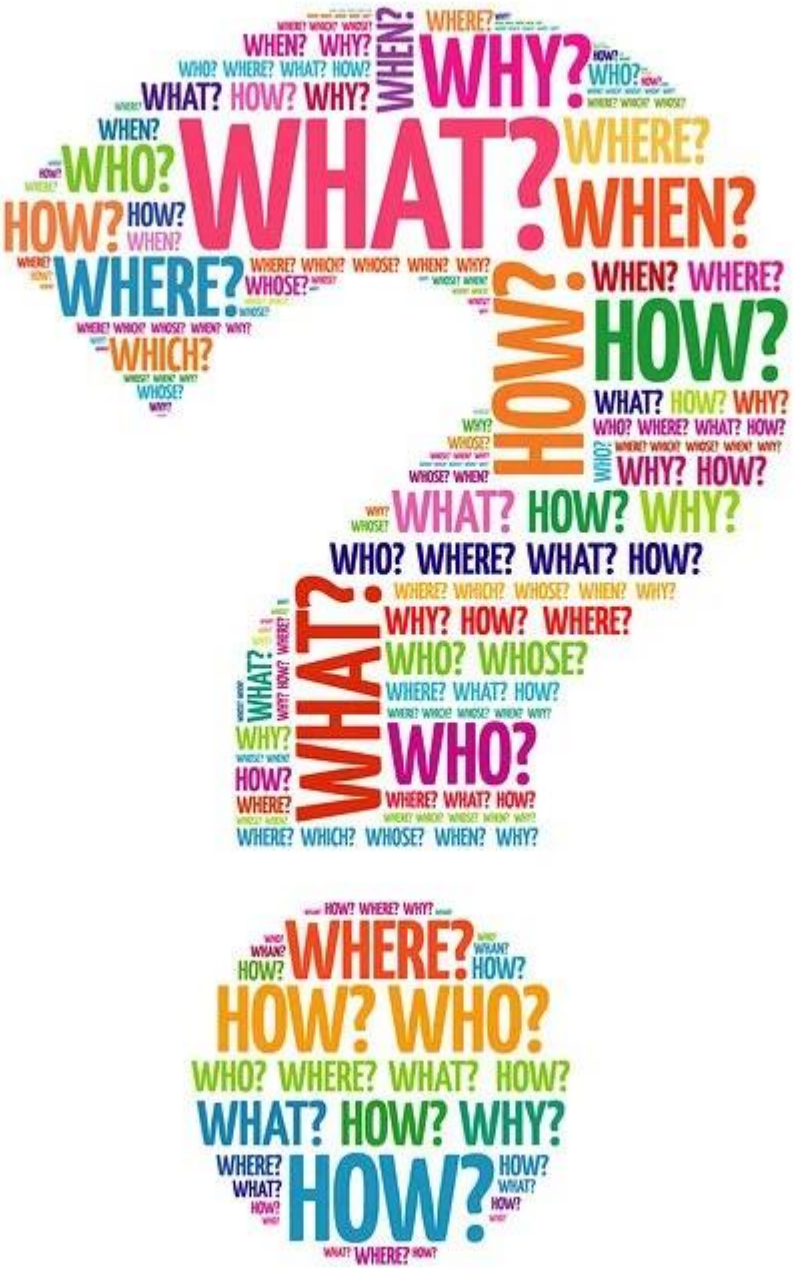
```
ptr -> right -> left = ptr -> left;
ptr -> left -> right = ptr -> right;
free(ptr);
```

شکل روبرو نمونه ای از درج گره در لیست دو پیوندی را نشان می دهد.

کاربرد لیست های دو پیوندی

- جمع دو عدد صحیح بزرگ
- **کنفرانس دانشجویی**

- نمایش ماتریس های اسپارس (خلوت)
- **کنفرانس دانشجویی**


WHERE?
WHEN? WHY?
WHO? WHERE? WHAT? HOW?
WHAT? HOW? WHY?
WHEN?
WHO?
WHAT?
WHERE?
HOW?
HOW?
WHEN?
WHERE?
WHERE? WHICH? WHOSE? WHEN? WHY?
WHOSE?
WHEN? WHERE?
HOW?
WHAT? HOW? WHY?
WHO? WHERE? WHAT? HOW?
WHY? HOW?
WHICH?
WHOSE?
WHAT? HOW? WHY?
WHO? WHERE? WHAT? HOW?
WHERE? WHICH? WHOSE? WHEN? WHY?
WHY? HOW? WHERE?
WHO? WHOSE?
WHERE? WHAT? HOW?
WHAT?
WHY?
HOW?
WHERE?
WHO?
WHERE? WHAT? HOW?
WHERE? WHICH? WHOSE? WHEN? WHY?
HOW? WHERE? WHY?
WHERE?
HOW?
HOW? WHO?
WHO? WHERE? WHAT? HOW?
WHAT? HOW? WHY?
WHERE?
WHAT?
HOW?
HOW?
HOW?
WHAT?
HOW?
WHAT? WHERE? HOW?