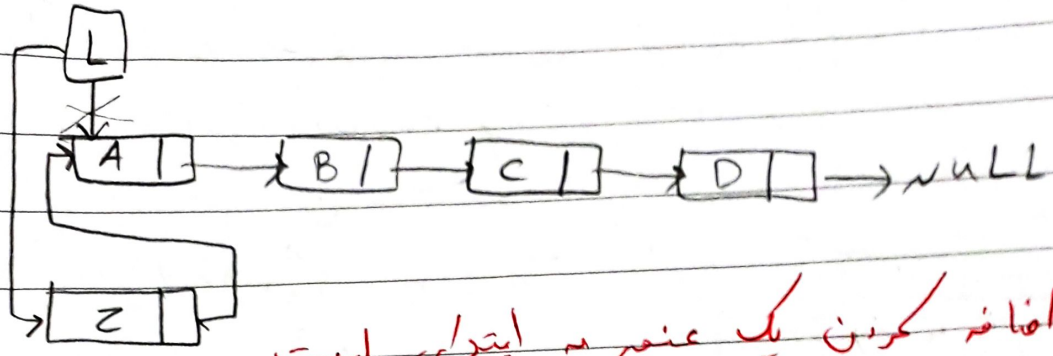
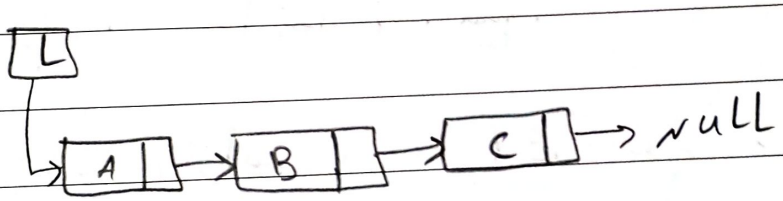


برای اضافه کردن یک عنصر به ابتدای لیست پیوندی یک راه:

ابتدا node اول که آدرس را به node عنصر جدید وصل می‌کنیم یعنی $L^{\wedge}.next$ بجای اشاره به A به Z اشاره کند سپس $Z^{\wedge}.next$ به A اشاره کند.



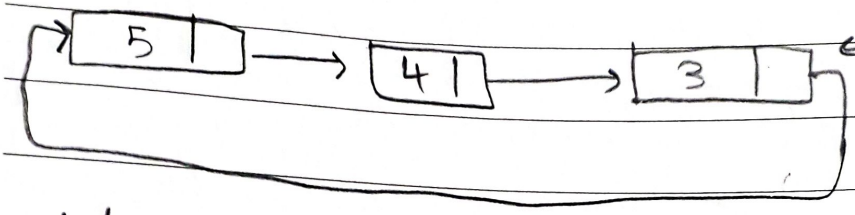
برای حذف عنصر آخر: باید تا انتهای لیست پیش برویم و پوینتر عنصر یکی مانده به آخر را به null وصل کنیم سپس عنصر آخر را حذف کنیم ($P_{n-1}^{\wedge}.next$ را به null وصل می‌کنیم و P_n را $dispose$ می‌کنیم)

اگر لیست خالی بود null را return می‌کنیم

اگر لیست فقط یک عنصر داشت هم null را return می‌کنیم

۲) لیست پیوندی حرفه‌ای چیزی به نام ابتدا یا انتها ندارد یعنی عناصر به صورت حرفه‌ای در کنار هم قرار می‌گیرند و در واقع یک لیست فعال است که انتهایش به ابتدایش اشاره می‌کند یعنی $p_n^{\wedge}.next$ به p_1 اشاره می‌کند.

اضافه کردن المنت به انتهای لیست حرفه‌ای:



برای اضافه کردن یک المنت به لیست پیوندی حرفه‌ای ابتدا باید یک node جدید با دیتای مورد نظر بسازیم و بعد از حافظه را به آن اختصاص دهیم.

سپس اگر لیست خالی باشد یعنی Last یا head برابر null باشد. آن وقت node باید به طور حرفه‌ای به خودش اشاره کند

اگر لیست خالی نباشد باید $new^{\wedge}.next$ را به عنصر head اشاره دهیم و $last^{\wedge}.next$ را به عنصر new اشاره دهیم

در آخر هم $last = new$ قرار دهیم

باشرط اینکه به صورت pointer داده شده:

۳) مقایسه پیچیدگی زمانی عملیات insert، delete، retrieve
در لیست های پیوندی و آرایه ای

	Insert	Delete	retrive
Array List	O(n)	O(n)	O(1)
Linked List	O(1)	O(n)	O(1)

(4) برای پیاده‌سازی یک لیست پیوندی باید از node اول شروع پیمایش کنیم و دیتای هر نود را با x مورد نظر مقایسه کنیم اگر برابر بود آن را برمی‌گردانیم.

در بهترین حالت: x مورد نظر ما در خانه اول است و در شروع پیمایش آن را پیدا می‌کنیم که پیچیدگی زمانی برابر با $O(1)$ است

در بدترین حالت: x مورد نظر ما در آخر ذخیره شده و پیچیدگی زمانی برابر $O(n)$ است.

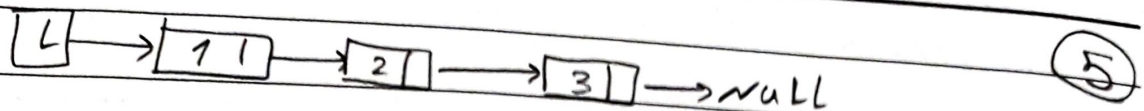
```
Struct node{
    int data  مقدار گره
    node* next;  اشاره‌گر به گره بعدی
};

int search(node* head , int target) {

    node* current = head;  اشاره‌گر به لیست
    int index=0;  موقعیت گره جاری

    while (current != nulptr) {  تا زمانی به انتهای لیست نرسیدیم

        if (current -> data == target){
            return index;  موقعیت گره پیدا شده
        }
        current = current->next;  به گره بعدی برویم
        index++;
    }
    return -1;  اگر مقدار پیدا نشد
}
```



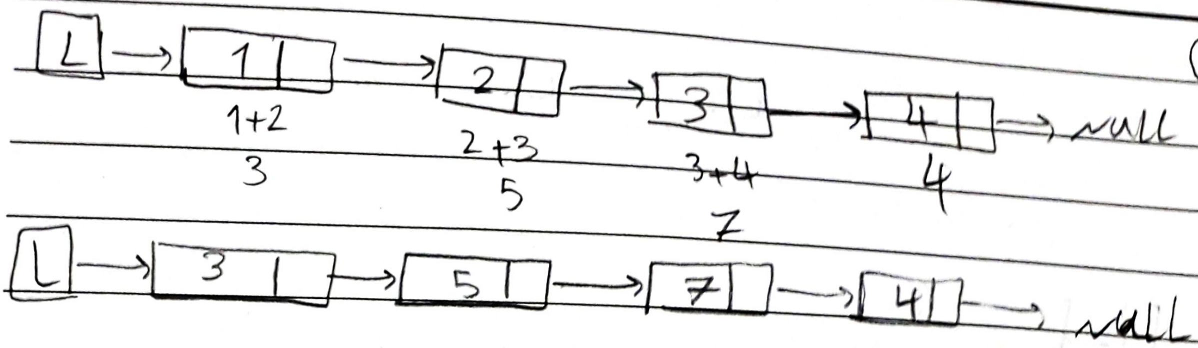
(5)

```
Print Rev(ref L : List)
{
    if (L==null) {
        return;
    }
    else{
        Print Rev(L^.Next);
        cout << L^.Data;
    }
}
```

(6)

```
make null (ref L : List){
    if(L==null){ return}
    else {
        make null (L^.next);
        dispose (L);
        L = null
    }
}
```

۷



8) اگر لیست عنصر نداشته باشد یا یک عنصر داشته باشد معکوس آن خودش است.

ابتدا عنصر اول لیست را از بقیه لیست جدا کرده و عنصر اول

sublist دوم را با p نشان میدهیم. سپس در هر قدم عنصر اول باقیمانده لیست
اصلی را از آن جدا کرده و به لیست L اضافه می کند.

در واقع در هر مرحله نود فعلی به نود قبلی تغییر داده می شود.

سپس پوینتر ها به نودهای بعدی می روند و در آخر Head را به آخرین node

انتقال می کند.