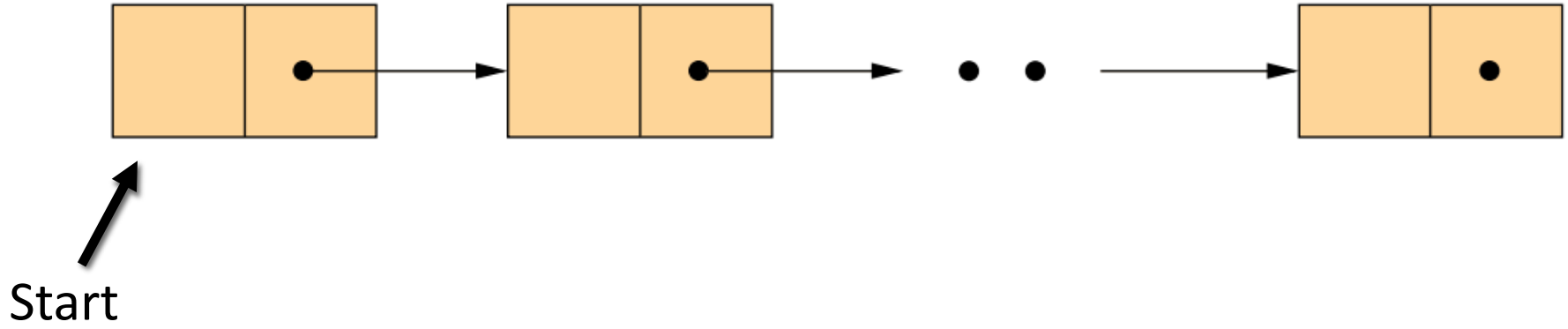
ساختمان داده ها و الگوریتم ها

لیست پیوندی یکطرفه

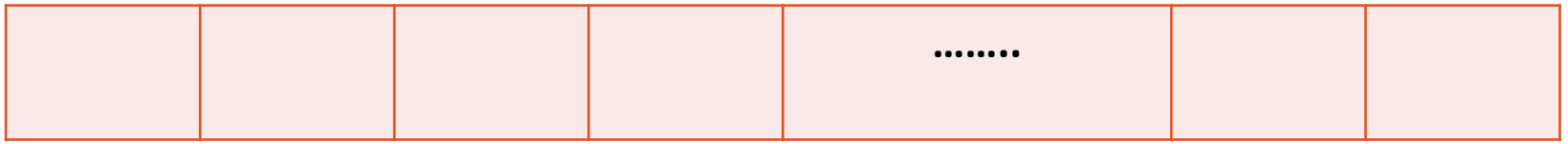
محمدحسین اولیائی

تعریف

LinkList:



Array:



اعمال اصلی

- $\textsc{Create-List}(L)$: **ایجاد یک لیست تهی** L
- $\textsc{Size}(L)$: **تعداد عناصر لیست را بر می‌گرداند**
- $\textsc{First}(L)$: **عنصر اول را برمی‌گرداند**
- $\textsc{isEmpty}(L)$: **مشخص می‌کند که آیا لیست خالی**
- $\textsc{Insert-First}(L, x)$: **درج عنصری با مقدار** x **در ابتدای** L
- $\textsc{Insert-After}(L, x, n)$: **درج عنصری با مقدار** x **پس از عنصر** n **در** L
- $\textsc{Delete-First}(L)$: **عنصر اول لیست** L **را حذف می‌کند**
- $\textsc{Delete-After}(L, n)$: **عنصر پس از عنصر** n **در** L **را حذف می‌کند**

```
class LinkList
{       private:
                struct node* start;
        public:
                LinkList();
                int IsEmpty();
                void insert_First(int);
                void insert_Last(int);
                void insert_after(struct node *,int d);
                struct node* search(int);
                void remove_First();
                void remove_Last();
                void remove_after(struct node *);
                void remove(int);//Your Homework!
                int ListSize();
                ~LinkList();
                void show();
};
```

```
struct node
{
        int data;
        struct node* link;
};
```

ایجاد لیست

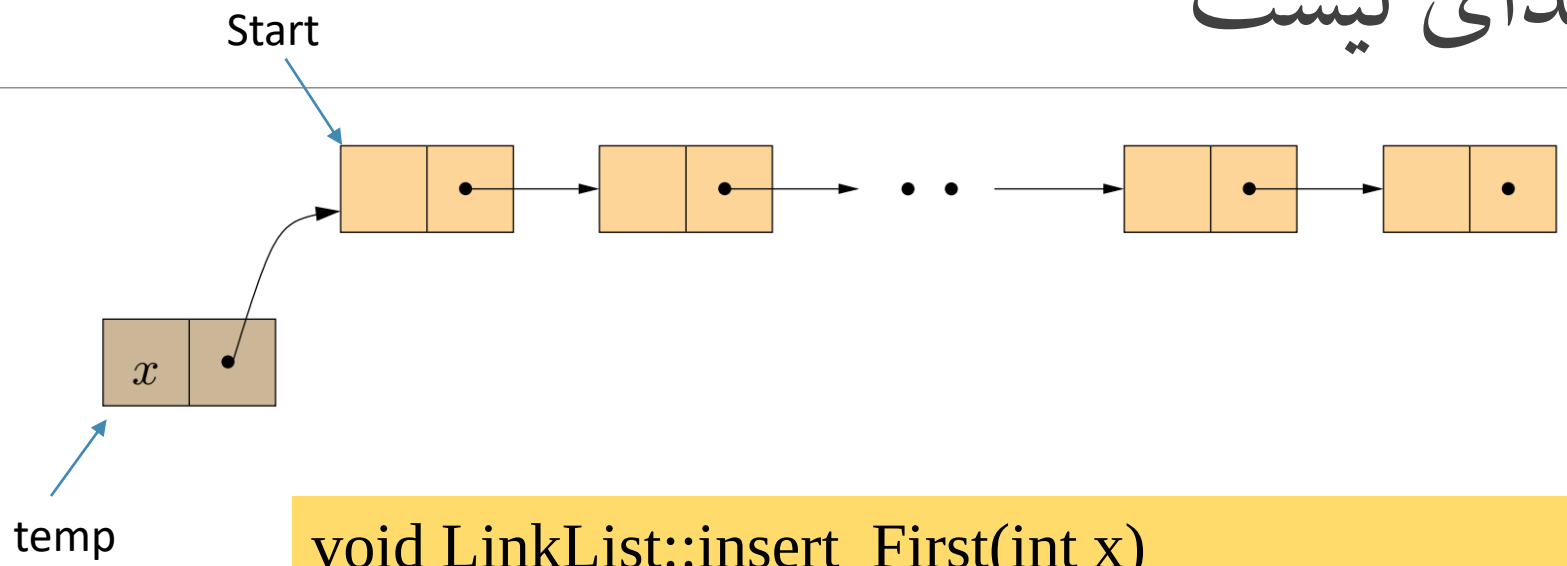
```
LinkList::LinkList()
{
        start=NULL;
}
```

LinkList Mylist;

بررسی خالی بودن

```
int LinkList::IsEmpty()
{
        if(start==NULL)
        return 1;
        return 0;
}
```

درج در ابتدای لیست

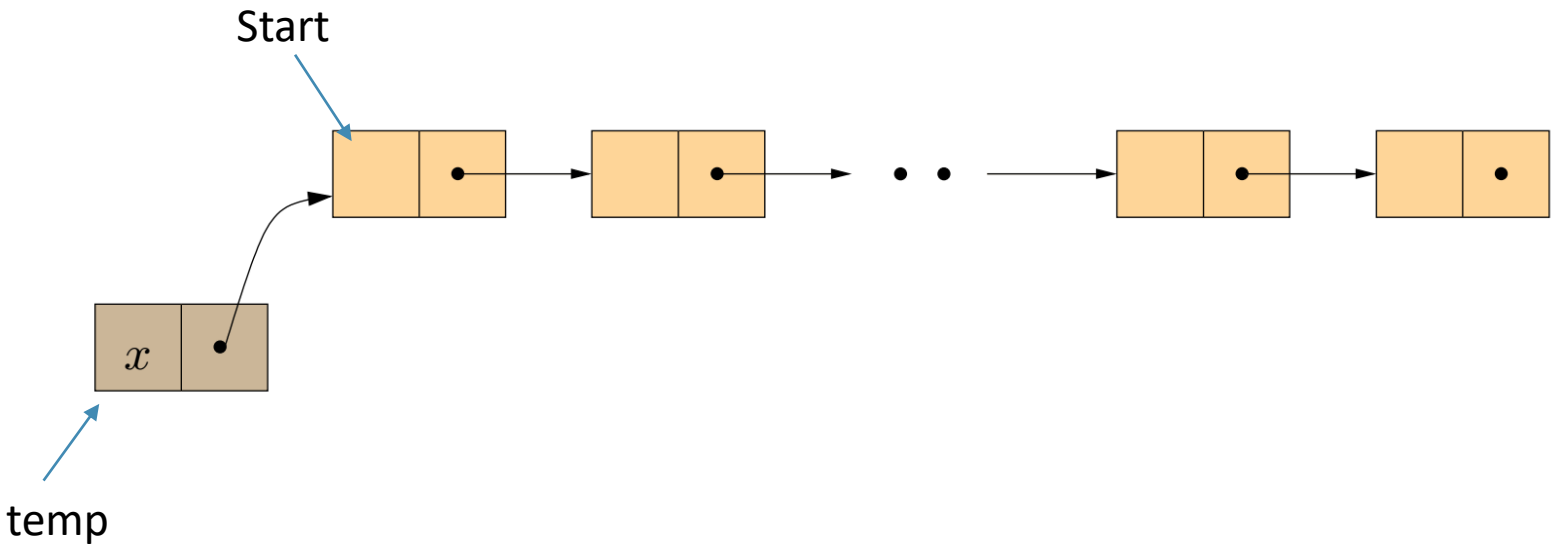



```
void LinkList::insert_First(int x)
{
        struct node *temp=new (struct node);
        temp->data=x;
        temp->link=NULL;
}
```

درج در ابتدای لیست

- دو حالت داریم:
  - لیست خالی می باشد
  - خالی نبوده و شامل تعدادی نود می باشد

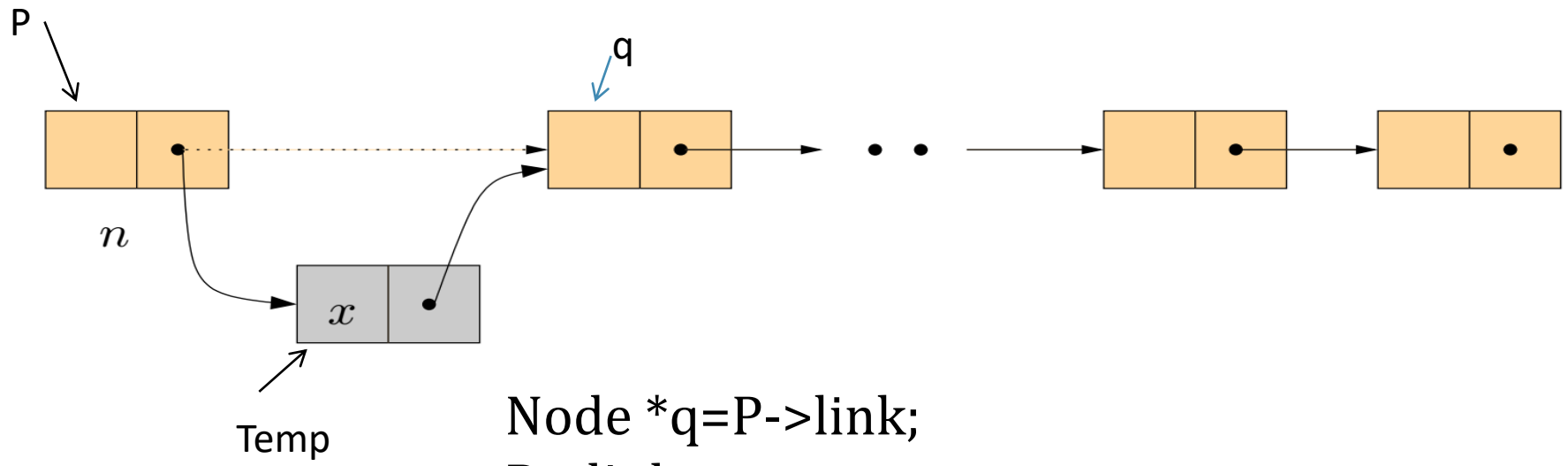
```
if (IsEmpty())
     start=temp;
else
{
   temp->link=start;
   start=temp;
}
```



درج در ابتدای لیست

```
void LinkList::insert_First(int d)
{
        struct node *temp=new (struct node);
        if(temp)
        {
          temp->data=d;
          temp->link=NULL;
          if (IsEmpty())
             start=temp;
          else
          {
            temp->link=start;
            start=temp;
          }
        }
}
```

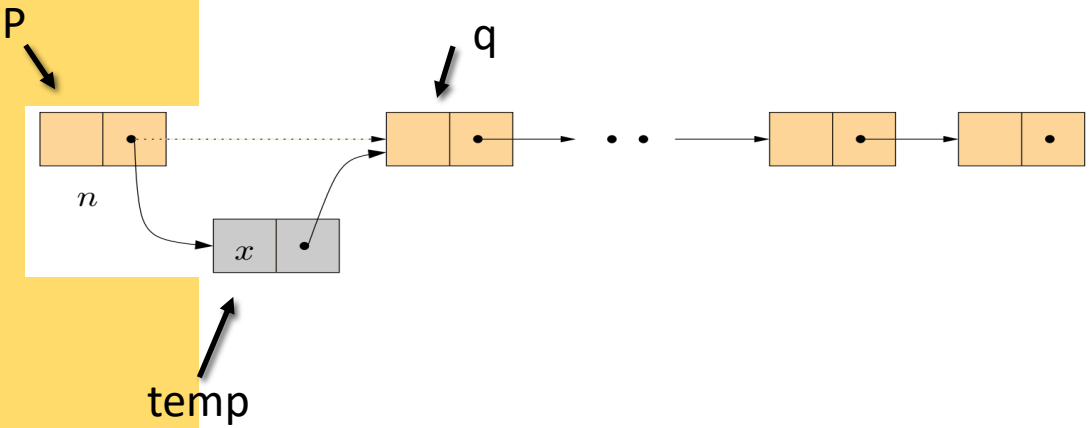
درج بعد از

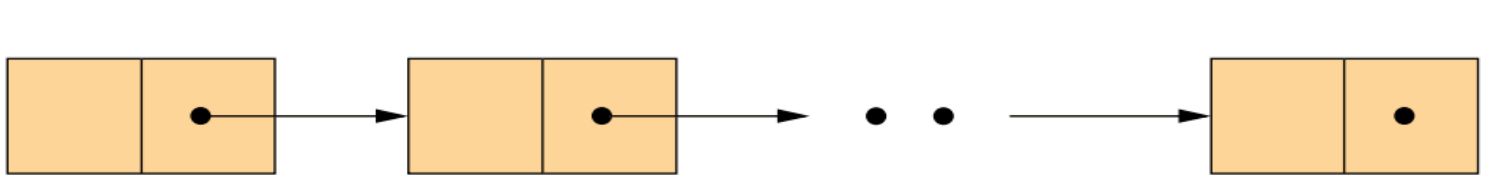



```
Node *q=P->link;
P->link=temp;
Temp->link=q;
```

درج بعد از

```
void LinkList::insert_after(struct node *p, int d)
{
        if (p)
        {
                struct node *q,*temp;
                q=p->link;
                temp=new (struct node);
                if(temp)
                {
                        temp->data=d;
                        p->link=temp;
                        temp->link=q;
                }
        }
}
```





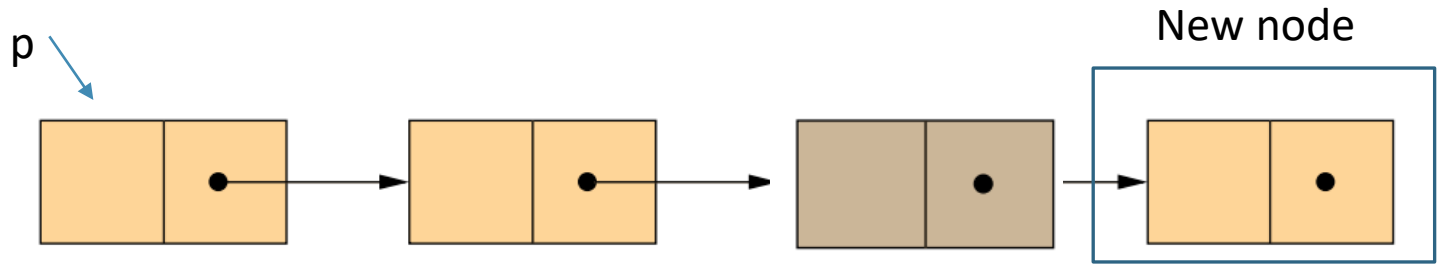
اندازه لیست

```
int LinkList::ListSize()
{
      if(IsEmpty())
            return 0;
      int L=0;
      struct node *p=start;
      while(p)
      {
         L++;
         p=p->link;
      }
       return L;
}
```

نمایش لیست

```
void LinkList::show()
{
   if(!IsEmpty())
   {
          struct node *p=start;
          cout<<"\n";
          while(p)
          {
              cout<<p->data<<"\t";
              p=p->link;
          }
    }
    else
        cout<<"List is Empty";
}
```

درج در انتها



p=p->link

```
While(p)
{
P=p->link;
}
```

درج در انتها

```
void LinkList::insert_Last(int d)
{
   struct node *temp=new (struct node);
   if(temp)
   {
      temp->data=d;
      temp->link=NULL;
      if (IsEmpty())
          start=temp;
      else
     {
          struct node *next;
          next=start;
          while(next->link)
                next=next->link;

          next->link=temp;
      }
    }
}
```

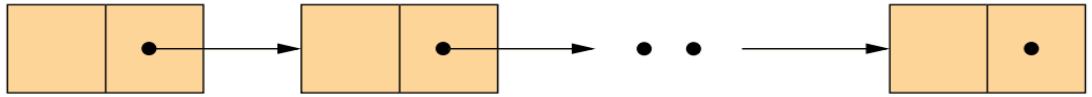
جستجو

```
struct node* LinkList::search(int d)
{
        if(!IsEmpty())
        {
                struct node *temp=start;
                while(temp)
                {
                        if(temp->data==d)
                        {
                                return temp;
                        }
                        temp=temp->link;
                }
        }
        return NULL;
}
```

حذف از ابتدا

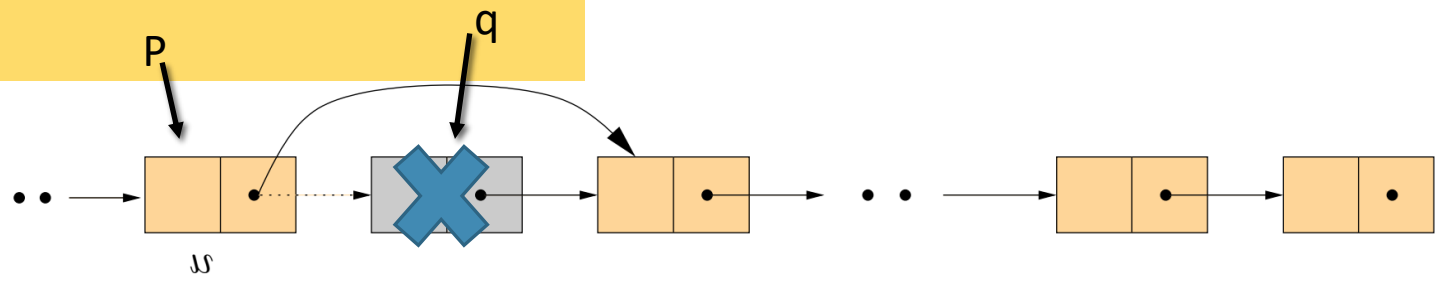
```
void LinkList::remove_First()
{
        if(!IsEmpty())
        {
                struct node *q;
                q=start->link;
                delete(start);
                start=q;
        }

}
```



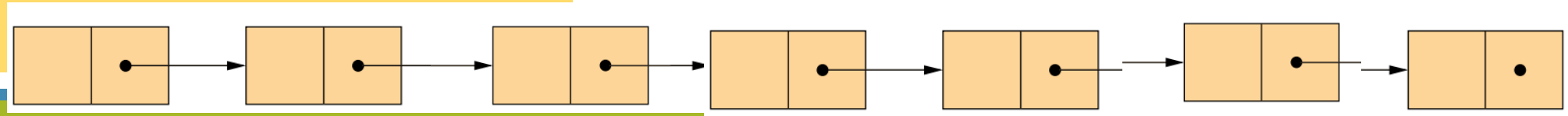
حذف بعد از

```
void LinkList::remove_after(struct node *p)
{
        struct node *q;
        if(p && p->link)
        {
                q=p->link;
                p->link=p->link->link;
                delete(q);
        }
}
```



حذف از انتهای لیست

```
void LinkList::remove_Last()
{    if(!IsEmpty())
            { if(!start->link)
                        {  delete(start);
                           start=NULL;
                        }
                        else
                        { struct node *p,*q;
                        p=start;
                        q=p->link;
                        while(q->link)
                        {           p=q;
                                    q=q->link;
                        }
                        delete(q);
                        p->link=NULL;
                        }
            }
}
```

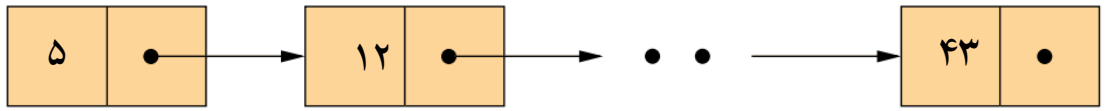


حذف کامل لیست

```
LinkList::~LinkList()
{
        if(!IsEmpty())
        {
                struct node *p=start,*q;
                while(p)
                {
                        q=p->link;
                        delete(p);
                        p=q;
                }
        }
}
```

مثال:

تابعی طراحی کنید که آدرس شروع یک لیست پیوندی شامل اعداد مرتب را دریافت کرده و بصورتی در آن درج نماید که لیست مرتب باقی بماند



```
Void insert( node * start, int x){
node * temp=new(node);
temp->link=null;
temp->data=x;
if (! start || s->data>x)
{     temp->link=start;
       start=temp;
}
node *p=start;
while(p->link && p->link->data<x)
        p=p->link;
temp->link=p->link;
P->link=temp;
}
```

تمرین

تابعی بنویسید که آدرس شروع دو لیست پیوندی را دریافت کرده و آنها را به یکدیگر متصل نماید: