

# VERİ YAPILARI

İsmail KADAYIF

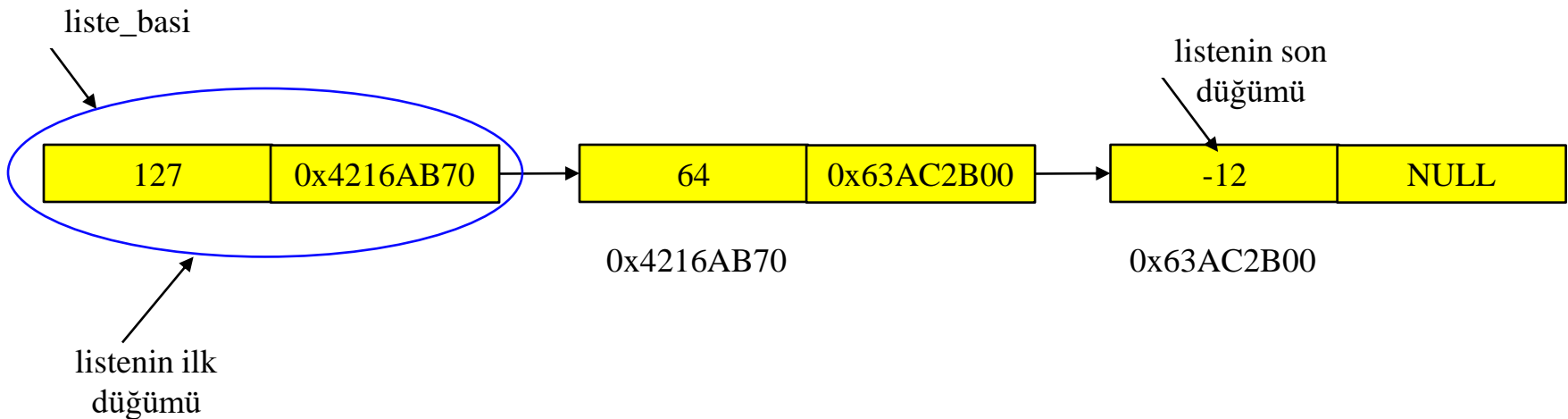
# LİSTELER

- Listeler düğümlerden (nodes) oluşur.
- Düğümler birbirlerine işaretçiler (pointers) üzerinden bağlanır.
  - ✓ Bir düğümden kendine komşu düğüme erişmek için o düğümün işaretçisinden yararlanılır.
- Bir listede işlem yapabilmemiz için o listenin ilk düğüme işaret eden bir işaretçiye ihtiyacımız vardır.
  - ✓ Liste başı (header)

# Liste Düğümü (Node)

```
struct hucre{  
    int içerik;  
    struct hücre *sonraki;  
}
```

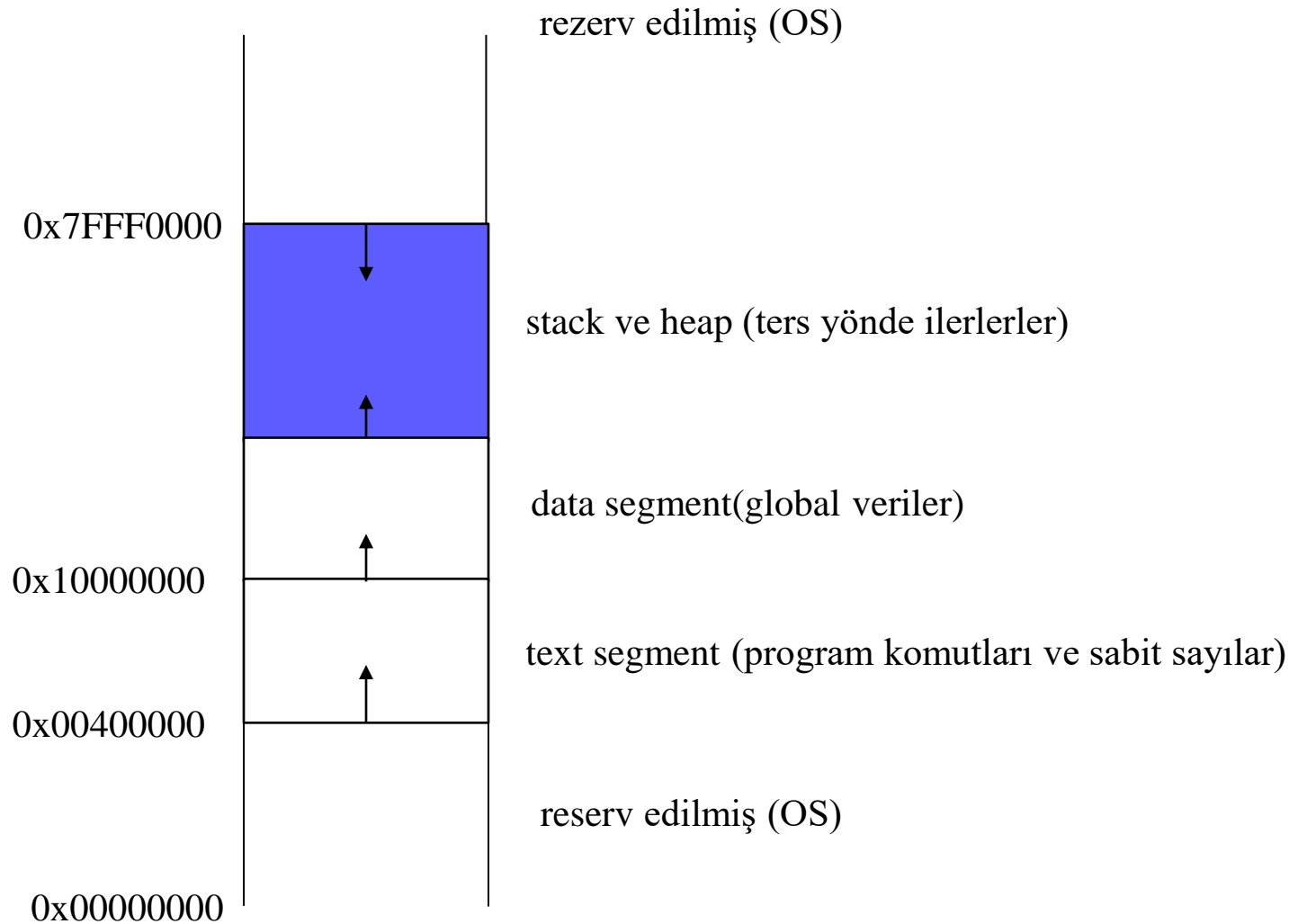
```
struct hucre{  
    char  *kimlik_no  
    char *ad;  
    char *soyad;  
    struct hücre *sonraki;  
}
```



# Dinamik Düğüm Oluşturma

- Listenin düğümleri dinamik olarak oluşturulur
- Bunun için C programlama dilinde malloc ve bunun türevi olan fonksiyonlar kullanılır
- Heap'ten uygun büyüklükte bir blok ayrılarak adresi geri döndürülür
- Bu blok bizim düğüme karşı gelen bloktur

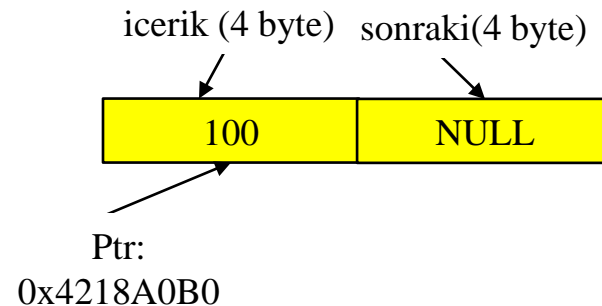
# Bellek Haritası



# Dinamik Düğüm Oluşturma

```
struct hucre{  
    int icerik;  
    struct hucre *sonraki;  
}
```

```
struct hucre *ptr;  
ptr= (struct hucre*)malloc(sizeof(struct hucre));  
// heaptan 8 byte lik boş bir blok ayırır (reserv eder)  
// ve bu 8 bytelik bloğun adresini geri dödürür  
ptr->icerik=100;  
ptr->sonraki=NULL;
```

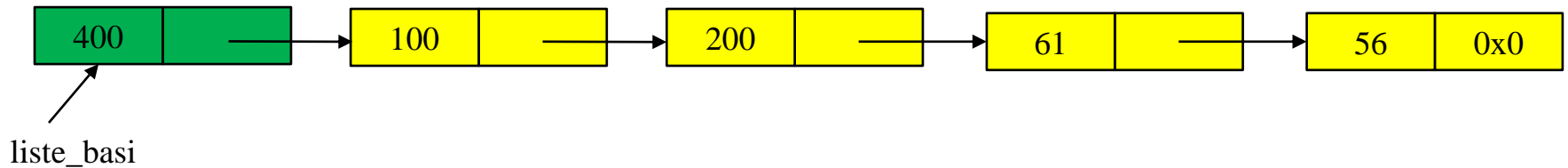
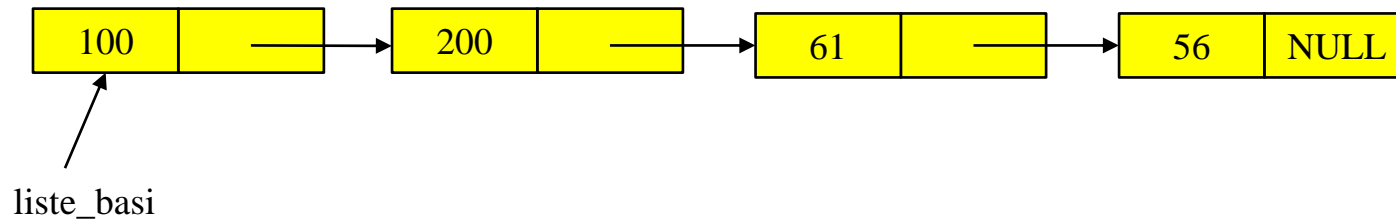


# Liste İşlemleri

➤ Çok çeşitli liste işlemleri olabilir. Bunların başlıcaları:

- ✓ Liste başına eleman ekleme
- ✓ Liste sonuna eleman ekleme
- ✓ Sıralı listeye ekleme
- ✓ Listeden eleman sil
- ✓ Liste sırala
- ✓ Liste ters çevir
- ✓ Liste yazdır

# Liste başına ekleme

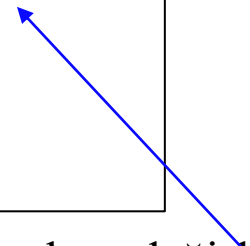




# Liste başına ekleme

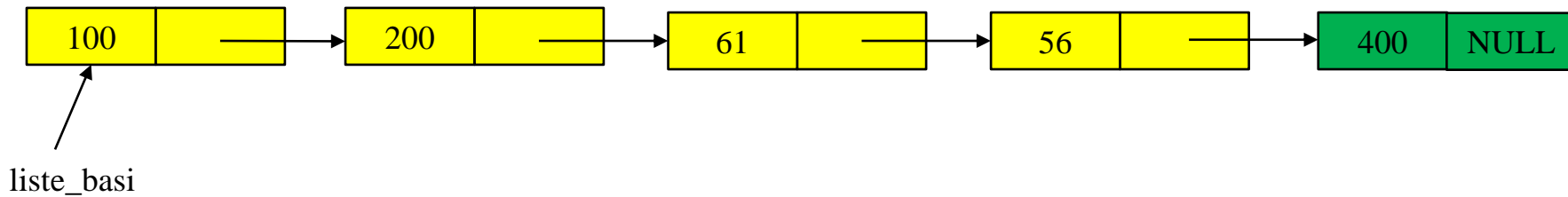
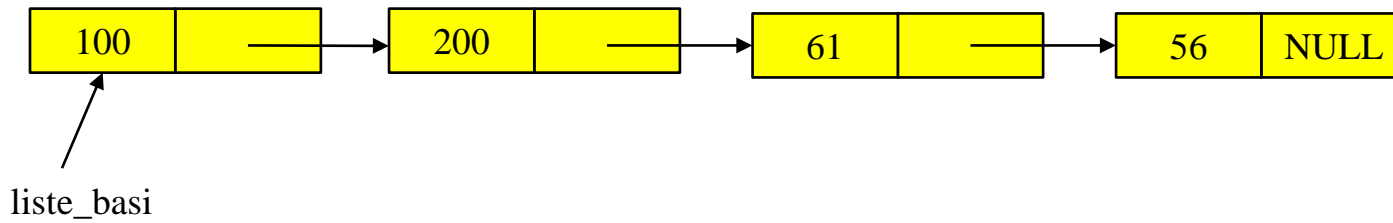
```
struct hucre* hucre_olustur(int icerik){
    struct hucre *a;
    a=(struct hucre*)malloc(sizeof(struct hucre));
    if(a==NULL){
        printf("Heap alanında yer ayrılamadı ...\n");
        exit(1);
    }
    a->icerik=icerik;
    a->sonraki=NULL;
    return a;
}
```

```
void liste_basina_ekle(int icerik,struct hucre **liste_basi){
    struct hucre* a=hucre_olustur(icerik);
    a->sonraki=*liste_basi;
    *liste_basi=a;
}
```



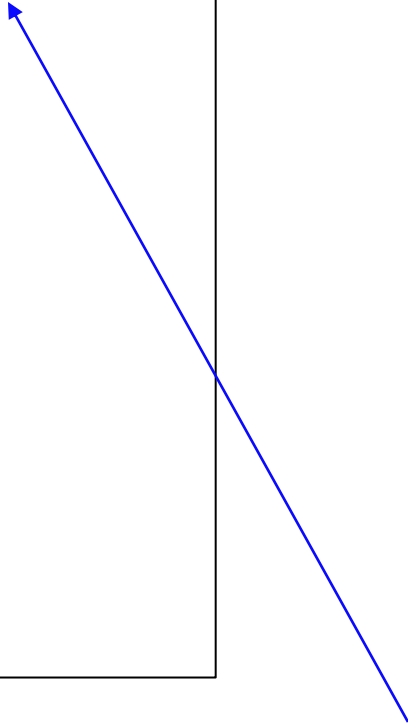
Not: Listenin ilk düğümünün adresini tutan liste\_bası değişkeni değişeceği için Bu değişkenin adresini parametre olarak fonksiyona yollamalıyız (struct hucre \*\*)

# Liste sonuna ekle



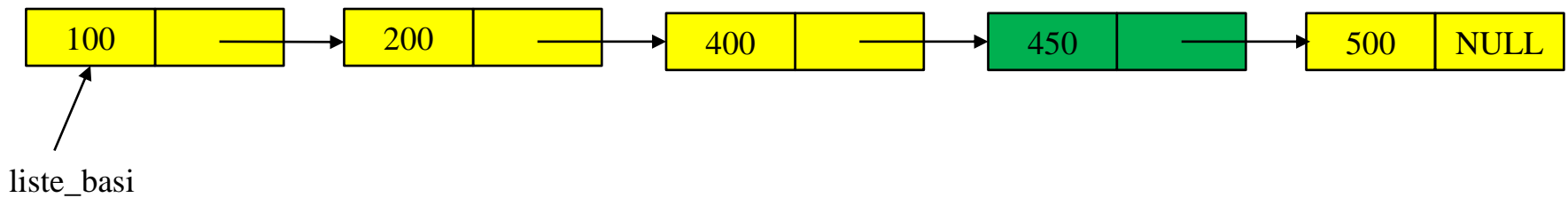
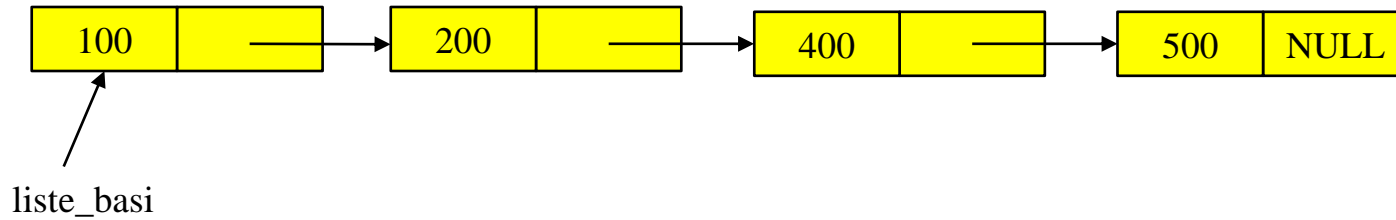
# Liste sonuna ekle

```
void liste_sonuna_ekle(int icerik, struct hucre **liste_basi){
    struct hucre* a=hucre_olustur(icerik);
    if(*liste_basi==NULL){
        a->sonraki=*liste_basi;
        *liste_basi=a;
    }
    else {
        struct hucre *x=*liste_basi;
        while(x->sonraki!=NULL){
            x=x->sonraki;
        }
        x->sonraki=a;
    }
}
```



Not: Liste boş ise ilk düğümünün adresini tutan liste\_bası değişkeni değişeceği için bu değişkenin adresini parametre olarak fonksiyona yollamalıyız (struct hucre \*\*)

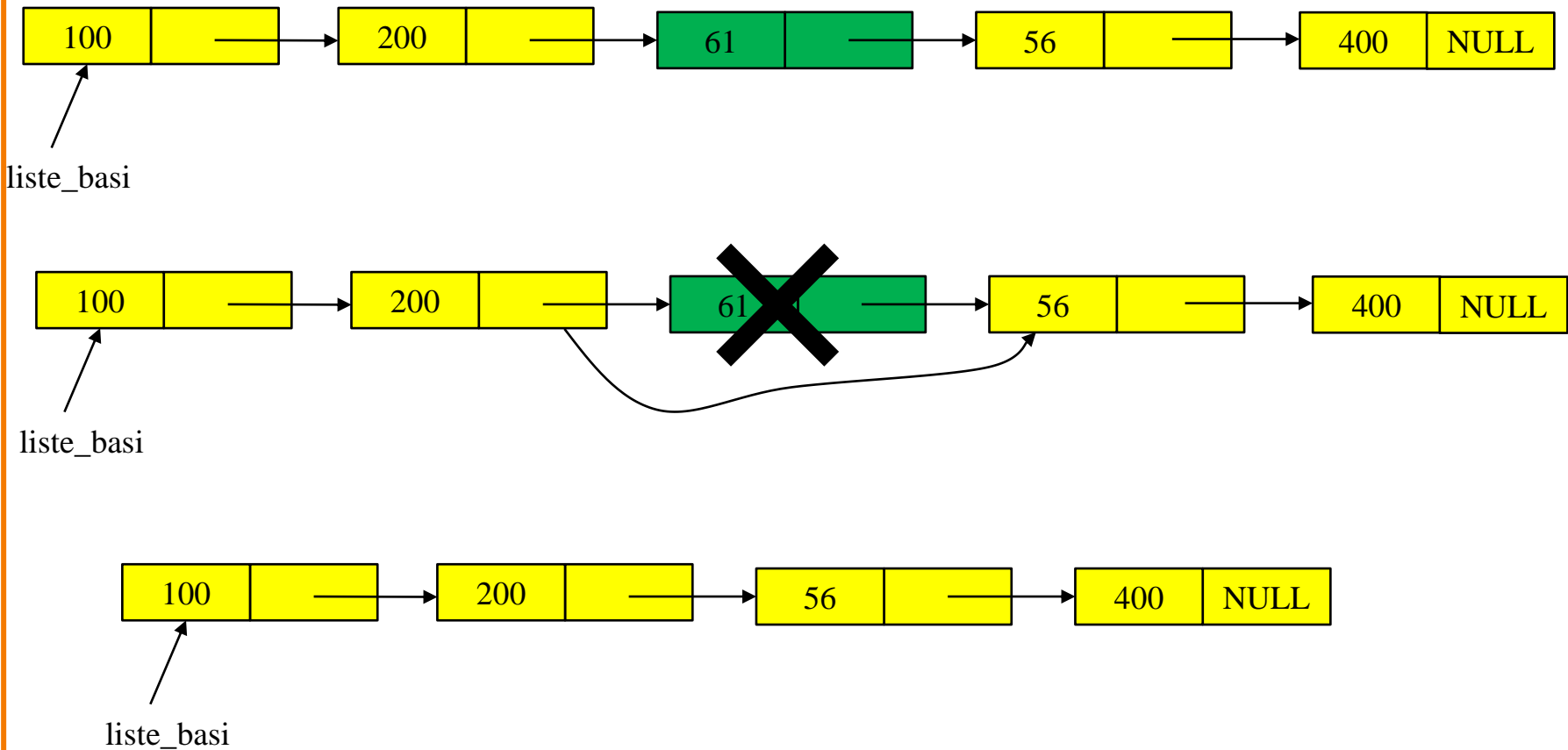
# Sıralı listeye ekleme



# Sıralı listeye ekleme

```
void liste_sirali_ekle(int icerik, struct hucre **liste_basi){
    struct hucre *a, *b, *eklenen;
    b=*liste_basi;
    while(b!=NULL && b->icerik <= icerik){
        if(b->icerik==icerik) return;
        a = b;
        b= b->sonraki;
    }
    eklenen=hucre_olustur(icerik);
    if(b==*liste_basi){
        eklenen->sonraki=*liste_basi;
        *liste_basi=eklenen;
    }
    else {
        a->sonraki=eklenen;
        eklenen->sonraki=b;
    }
}
```

# Listeden düğüm silme

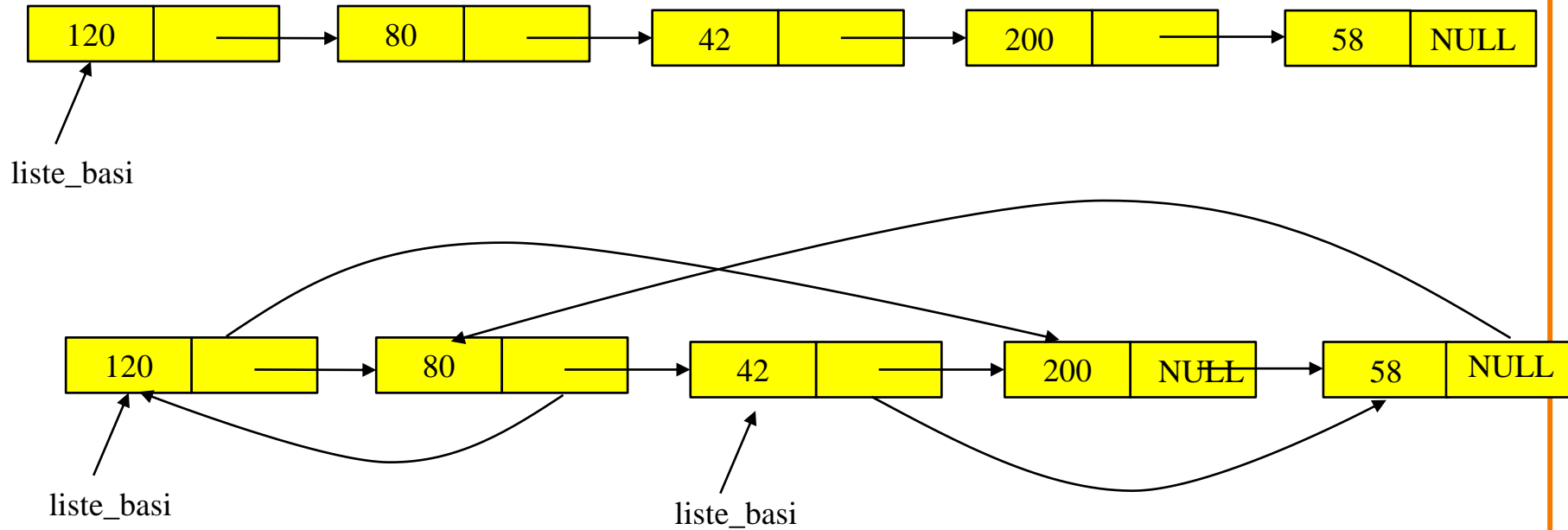


# Listeden düğüm silme

```
void liste_eleman_sil(int silinen, struct hucre **liste_basi){
    struct hucre *temp=*liste_basi;
    struct hucre *once;

    while(temp!=NULL && temp->icerik!=silinen){
        once = temp;
        temp=temp->sonraki;
    }
    if(temp==NULL) return;
    else {
        if(temp==*liste_basi) *liste_basi=(temp->sonraki);
        else once->sonraki=temp->sonraki;
        free(temp);
    }
}
```

# Liste sıralama





# Liste sıralama

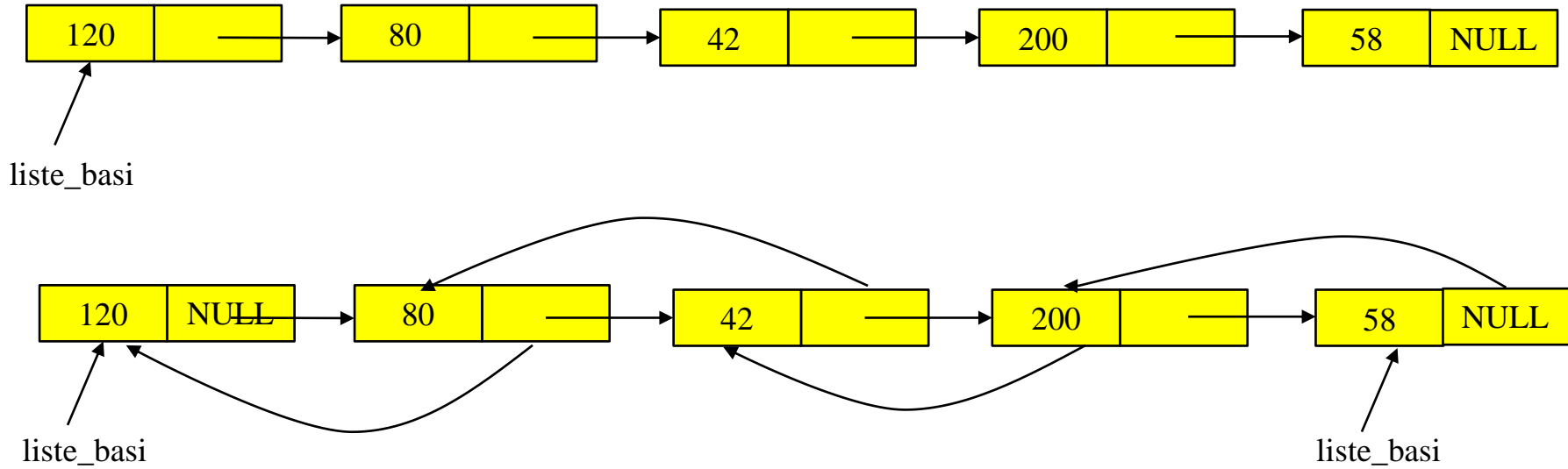
```
void liste_sirala(struct hucre **liste_basi){
    struct hucre *a,*b,*c,*d;

    if(*liste_basi == NULL || (*liste_basi)->sonraki==NULL) return;

    d=(*liste_basi)->sonraki;
    (*liste_basi)->sonraki=NULL;

    while(d!=NULL){
        c=d;
        d=d->sonraki;
        b=*liste_basi;
        while(b!=NULL && b->icerik < c->icerik){
            a=b;
            b=b->sonraki;
        }
        if(b==*liste_basi){
            c->sonraki=*liste_basi;
            *liste_basi=c;
        }
        else {
            a->sonraki = c;
            c->sonraki = b;
        }
    }
}
```

# Liste ters çevir



# Liste ters çevir

```
void liste_ters_cevir(struct hucre **liste_basi){  
  
    struct hucre *a,*b;  
  
    a=NULL;  
    while(*liste_basi!=NULL){  
        b=*liste_basi;  
        *liste_basi=(*liste_basi)->sonraki;  
        b->sonraki=a;  
        a=b;  
    }  
    *liste_basi=a;  
}
```

# Çift Yönlü Liste

- Tek yönlü listede düğümlerinde tek bir işaretçi vardı
  - ✓ sonraki
- Çift yönlü liste düğümlerinde iki tane işaretçi mevcuttur
  - ✓ ileri
  - ✓ geri
- Çift yönlü listede düğümlerinde iki farklı işaret bulundurulması
  - ✓ Avantaj: Liste işlemlerini kolaylaştırır
  - ✓ Dezavantaj: Liste düğümlerinin alan ihtiyacı (bellek ihtiyacı) artar

# Çift Yönlü Liste

```
struct ciftYonluDugum{  
    int icerik;  
    struct ciftYonluDugum *ileri;  
    struct ciftYonluDugum *geri;  
}
```

geri	120	ileri
------	-----	-------

