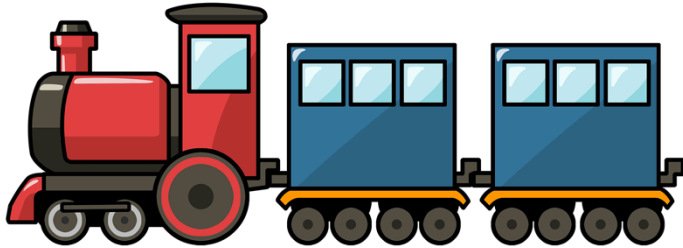


# Baglantılı Listeler

## Linked List



Suhaps  
SAHIN  
Onur GÖK

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

16-bit Hafıza	Adress
0000	0x08BA
0000	0x08BC
0000	0x08BE
0000	0x08C0
0000	0x08C2
0000	0x08C4
0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
x	0000	0x08BA
	0000	0x08BC
y	0000	0x08BE
	0000	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
x	0000	0x08BA
	0000	0x08BC
y	0000	0x08BE
p	0000	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0XDEAD	0x08BC
y	0000	0x08BE
p	0000	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0XDEAD	0x08BC
y	0XBEEF	0x08BE
p	0000	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0XDEAD	0x08BC
y	0XBEEF	0x08BE
p	08BC	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int x,y;
5     int *p;
6     x = 0xDEAD;
7     y = 0xBEEF;
8     p = &x;
9     *p = 0x100;
10    p = &y;
11    *p = 0x200;
12    return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0x100	0x08BC
y	0XBEEF	0x08BE
p	08BC	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6



# isaretciler (Pointers)

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x,y;
5      int *p;
6      x = 0xDEAD;
7      y = 0xBEEF;
8      p = &x;
9      *p = 0x100;
10     p = &y;
11     *p = 0x200;
12     return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0x100	0x08BC
y	0XBEEF	0x08BE
p	08BE	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x,y;
5      int *p;
6      x = 0xDEAD;
7      y = 0xBEEF;
8      p = &x;
9      *p = 0x100;
10     p = &y;
11     *p = 0x200;
12     return 0;
13 }
```

	16-bit Hafıza	Adress
	0000	0x08BA
x	0x100	0x08BC
y	0X200	0x08BE
p	08BE	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

```
int *p,*q;  
int x;
```




Diagram illustrating the initial state of memory. Three boxes are shown, labeled **p**, **q**, and **x**, each containing a question mark, indicating they are uninitialized.

```
p = &x;
```




Diagram illustrating the assignment `p = &x;`. The box labeled **p** now contains a pointer (represented by a dot) that points to the box labeled **x**, which contains a question mark. The label **x, \*p** is shown below the target box.

```
*p = 6;
```




Diagram illustrating the assignment `*p = 6;`. The box labeled **p** points to the box labeled **x**, which now contains the value **6**. The label **x, \*p** is shown below the target box.

```
p = new int;
```




Diagram illustrating the assignment `p = new int;`. The box labeled **p** now contains a pointer (represented by a dot) that points to a new memory location. The box labeled **\*p** contains a question mark, and the box labeled **x** contains the value **6**.

```
*p = 7;
```

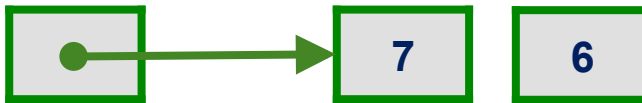
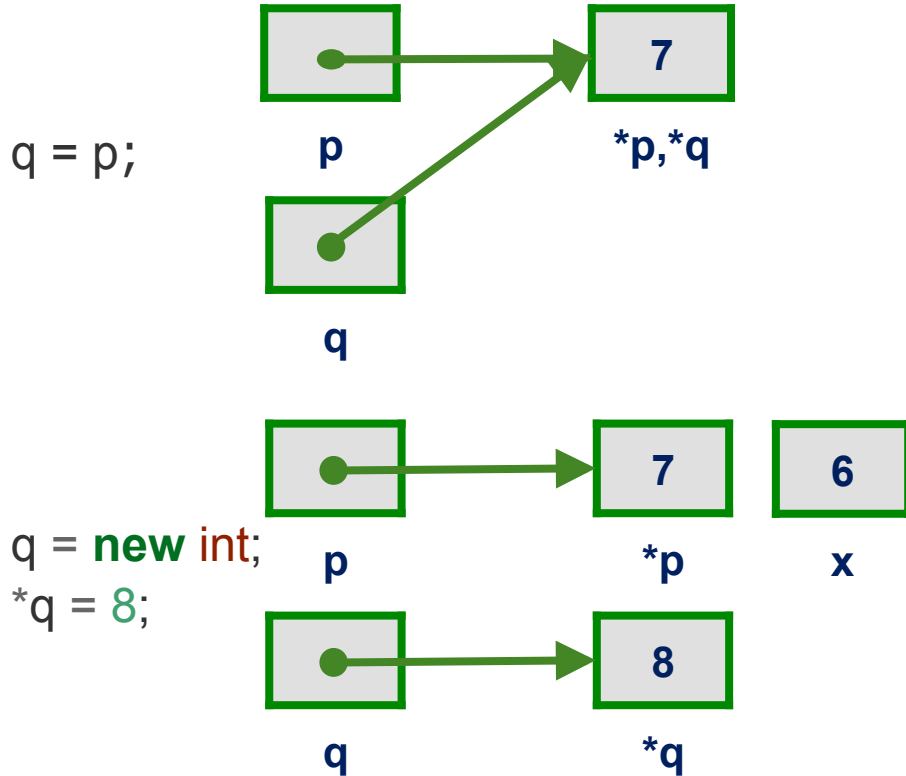
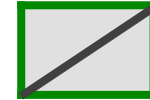


Diagram illustrating the assignment `*p = 7;`. The box labeled **p** points to the box labeled **\*p**, which now contains the value **7**. The box labeled **x** still contains the value **6**.

# isaretciler (Pointers)



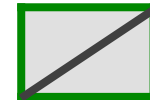
`p = NULL;`



`p`

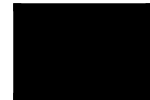
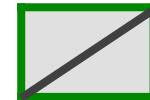
`x`

Delete `q;`  
`q = NULL;`



`p`

`x`



`q`

# il sıralaması



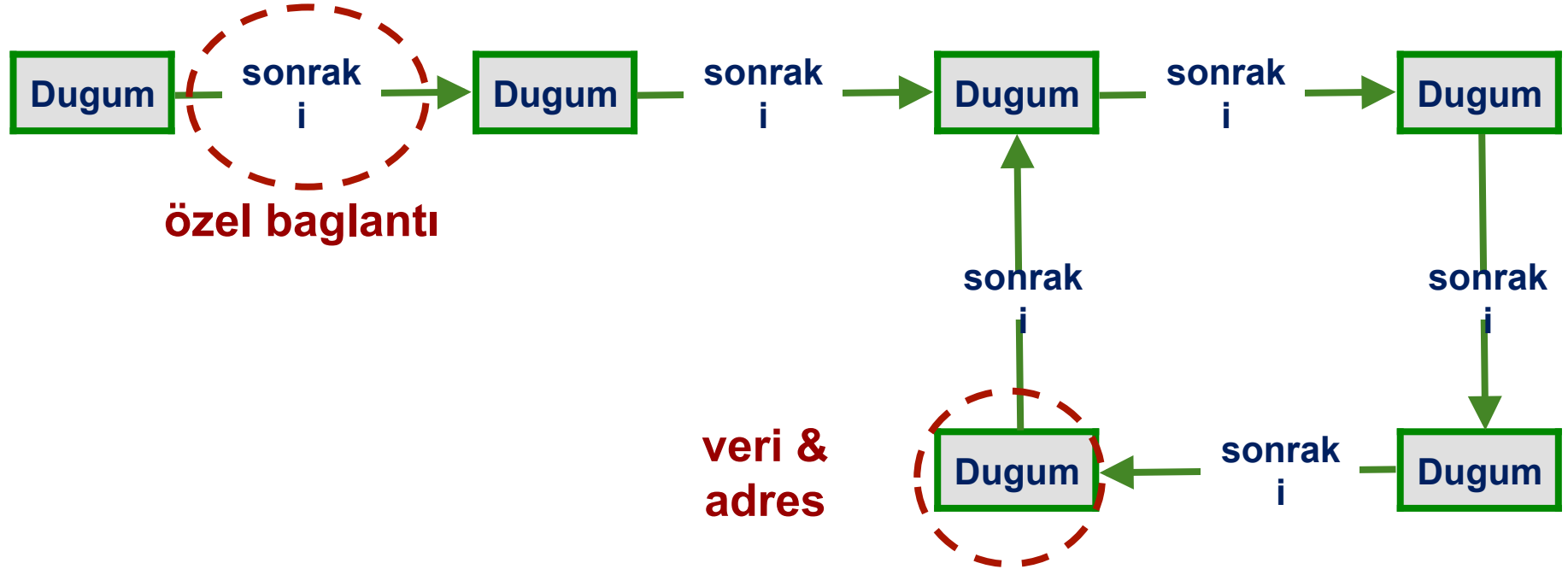
```
struct Dugum  
{  
    string il;  
    int sonraki;  
};
```

# il sıralaması

0	1	2	3	4	5	6	7	8	9
Isparta 9	Edirne 5	Mardin -1	Denizli 1	Hatay 0	Giresun 4	Bursa 8	Ankara 6	Çorum 3	Kocaeli 2

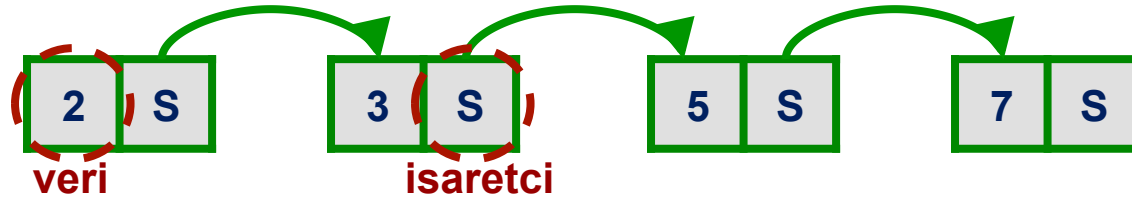
Adres	Bellek içeriği
0F20	Isparta   9
0F24	Edirne   5
0F28	Mardin   -1
0F2C	Denizli   1
0F30	Hatay   0
0F34	Giresun   4
0F38	Bursa   8
0F3C	Ankara   6
0F40	Çorum   3
0F44	Kocaeli   2

# Baglantılı listeler



# Baglantılı listeler

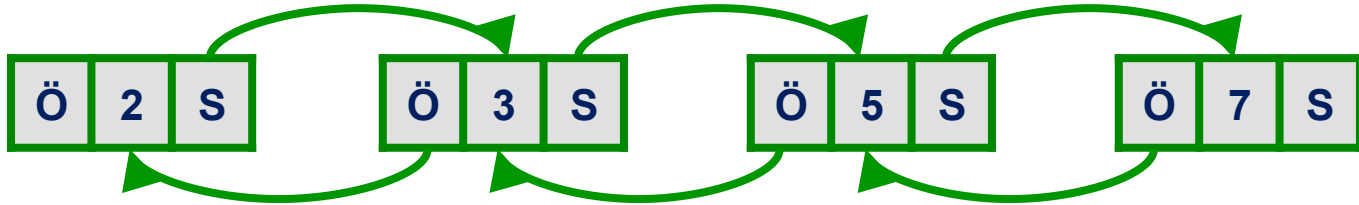
## Tek yönlü bağlantılı liste





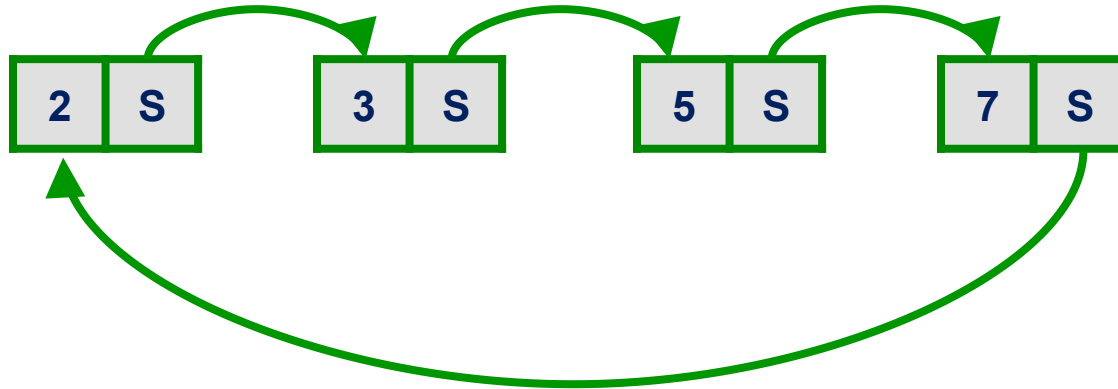
# Baglantılı listeler

Çift yönlü bağlantılı  
liste



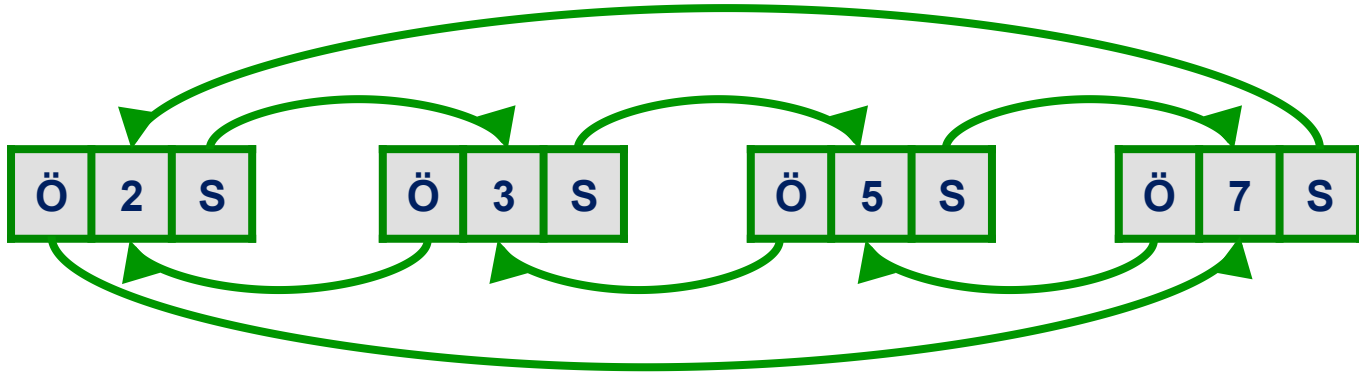
# Baglantılı listeler

Tek yönlü **çevrimsel** bağlantılı  
liste



# Baglantılı listeler

Çift yönlü **çevrimsel** bağlantılı  
liste



# Düğüm(Node)

C & C++

*Düğüm*

string veri;	int sonraki;
--------------	--------------

*// Düğüm*

**struct** Dugum

{

string veri;

int sonraki;

};



# il sıralaması

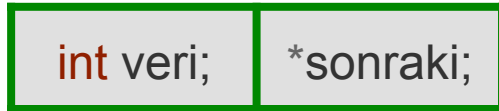
## Dinamik bellek kullanımı & Nesneye yönelik programlama

Adres s	Bellek icerigi
...	...
0F1C	
0F20	Isparta   9
0F24	Edirne   5
0F28	Mardin   -1
0F2C	Denizli   1
0F30	Hatay   0
0F34	Giresun   4
0F38	
0F3C	Bursa   8
0F40	Ankara   6
0F44	Çorum   3
0F48	Kocaeli   2
...	...

# Düğüm(Node)

C & C++

*Düğüm*



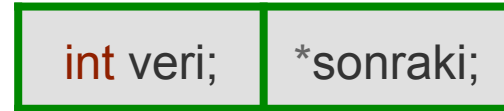
*// Düğüm*

```
struct Dugum
{
    int veri;
    struct Dugum *sonraki;
};
```

# Tek Yönlü Bağlantılı Liste

```
struct Dugum{  
    int veri;  
    struct Dugum *sonraki;  
};
```

*Düğüm*



```
struct Dugum* birinci = NULL;  
struct Dugum* ikinci = NULL;  
struct Dugum* ucuncu = NULL;
```

*birinci*



*ikinci*



*ucuncu*





# Tek Yönlü Bağlantılı Liste

```
birinci = (struct Dugum*)malloc(sizeof(struct Dugum));  
ikinci = (struct Dugum*)malloc(sizeof(struct Dugum));  
ucuncu = (struct Dugum*)malloc(sizeof(struct Dugum));
```

*birinci*



*ikinci*



*ucuncu*



# Tek Yönlü Bağlantılı Liste

birinci->veri = 1;  
birinci->sonraki = ikinci;

ikinci->veri = 2;  
ikinci->sonraki = ucuncu;

ucuncu->veri = 3;  
ucuncu->sonraki = NULL;



# Tek Yönlü Bağlantılı Liste

birinci->veri = 1;  
birinci->sonraki = ikinci;

ikinci->veri = 2;  
ikinci->sonraki = ucuncu;

ucuncu->veri = 3;  
ucuncu->sonraki = NULL;



# Tek Yönlü Bağlantılı

```
void printList(struct Dugum *n){  
    while (n != NULL){  
        printf(" %d ", n->veri);  
        n = n->sonraki;  
    }  
}
```



# Tek Yönlü Bağlantılı Liste

```
void printList(struct Dugum *n){  
    printf(" %d ", n->veri);  
    printf(" %d ", n->sonraki->veri);  
    printf(" %d ", n->sonraki->sonraki->veri);  
}
```



# Ekleme

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Dugum{
    int veri;
    struct Dugum *sonraki;
};
```

```
int main(){
    struct Dugum* yeni = NULL;

    return 0;
}
```

*yeni*



# Sona Ekleme

```
/* sona ekleme */
```

```
void sona_ekle(struct Dugum** yeni_ref, int yeni_veri)
```

```
{
```

```
/* 1. yeni dugum için yer al */
```

```
struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
```

```
struct Dugum *sonuncu = *yeni_ref;
```

```
/* 2. yeni dugume yeni veri */
```

```
yeni_dugum->veri = yeni_veri;
```

```
yeni_dugum->sonraki = NULL;
```

```
return;
```

```
}
```

```
sona_ekle(&yeni, 6);
```

*yeni*



# Sona Ekleme

```
/* sona ekleme */
```

```
void sona_ekle(struct Dugum** yeni_ref, int yeni_veri)
```

```
{
```

```
/* 1. yeni dugum için yer al */
```

```
struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
```

```
struct Dugum *sonuncu = *yeni_ref;
```

```
/* 2. yeni dugume yeni veri */
```

```
yeni_dugum->veri = yeni_veri;
```

```
yeni_dugum->sonraki = NULL;
```

```
/* 3. Eğer liste boşsa */
```

```
if (*yeni_ref == NULL)
```

```
{
```

```
    *yeni_ref = yeni_dugum;
```

```
    return;
```

```
}
```

```
return;
```

```
}
```

```
sona_ekle(&yeni, 6);
```

*yeni*





# Sona Ekleme

```
/* sona ekleme */
```

```
void sona_ekle(struct Dugum** yeni_ref, int yeni_veri)
```

```
{
```

```
/* 1. yeni dugum için yer al */
```

```
struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
```

```
struct Dugum *sonuncu = *yeni_ref;
```

```
/* 2. yeni dugume yeni veri */
```

```
yeni_dugum->veri = yeni_veri;
```

```
yeni_dugum->sonraki = NULL;
```

```
/* 3. Eğer liste boşsa */
```

```
if (*yeni_ref == NULL)
```

```
{
```

```
    *yeni_ref = yeni_dugum;
```

```
    return;
```

```
}
```

```
/* 4. Boş değilse son düğüme kadar git */
```

```
while (sonuncu->sonraki != NULL)
```

```
    sonuncu = sonuncu->sonraki;
```

```
return;
```

```
}
```

```
sona_ekle(&yeni, 6);
```

*yeni*



# Sona Ekleme

```
/* sona ekleme */
```

```
void sona_ekle(struct Dugum** yeni_ref, int yeni_veri)
```

```
{
```

```
/* 1. yeni dugum için yer al */
```

```
struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
```

```
struct Dugum *sonuncu = *yeni_ref;
```

```
/* 2. yeni dugume yeni veri */
```

```
yeni_dugum->veri = yeni_veri;
```

```
yeni_dugum->sonraki = NULL;
```

```
/* 3. Eğer liste boşsa */
```

```
if (*yeni_ref == NULL)
```

```
{
```

```
    *yeni_ref = yeni_dugum;
```

```
    return;
```

```
}
```

```
/* 4. Boş değilse son düğüme kadar git */
```

```
while (sonuncu->sonraki != NULL)
```

```
    sonuncu = sonuncu->sonraki;
```

```
/* 5. sonuncuya kadar gittik */
```

```
sonuncu->sonraki = yeni_dugum;
```

```
return;
```

```
}
```

```
sona_ekle(&yeni, 6);
```

*yeni*



# Basa Ekleme

basa\_ekle(&yeni, 7);

```
void basa_ekle(struct Dugum** birinci_ref, int yeni_veri)
{
    /* 1. yeni dugum için yer al */
    struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
    /* 2. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 3. Yeni dugum birinci dugum olmalı */
    yeni_dugum->sonraki = (*birinci_ref);
    (*birinci_ref) = yeni_dugum;
}
```

*birinci\_ref*



*yeni\_dugum*



# Basa Ekleme

basa\_ekle(&yeni, 7);

```
void basa_ekle(struct Dugum** birinci_ref, int yeni_veri)
{
    /* 1. yeni dugum için yer al */
    struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
    /* 2. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 3. Yeni dugum birinci dugum olmalı */
    yeni_dugum->sonraki = (*birinci_ref);
    (*birinci_ref) = yeni_dugum;
}
```

*birinci\_ref*



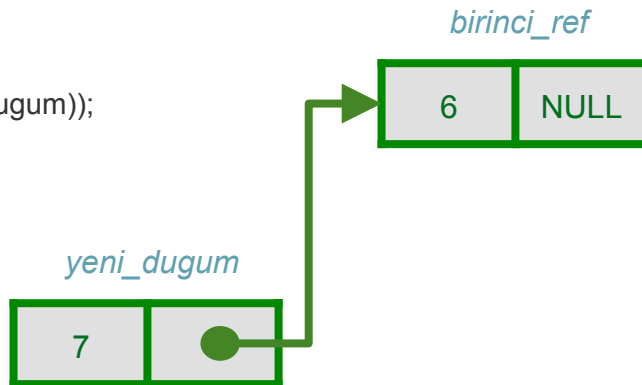
*yeni\_dugum*



# Basa Ekleme

basa\_ekle(&yeni, 7);

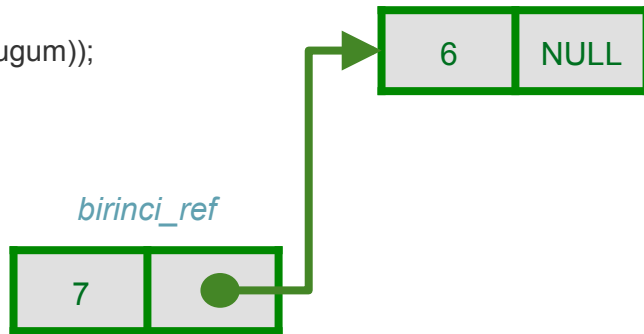
```
void basa_ekle(struct Dugum** birinci_ref, int yeni_veri)
{
    /* 1. yeni dugum için yer al */
    struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
    /* 2. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 3. Yeni dugum birinci dugum olmalı */
    yeni_dugum->sonraki = (*birinci_ref);
    (*birinci_ref) = yeni_dugum;
}
```



# Basa Ekleme

basa\_ekle(&yeni, 7);

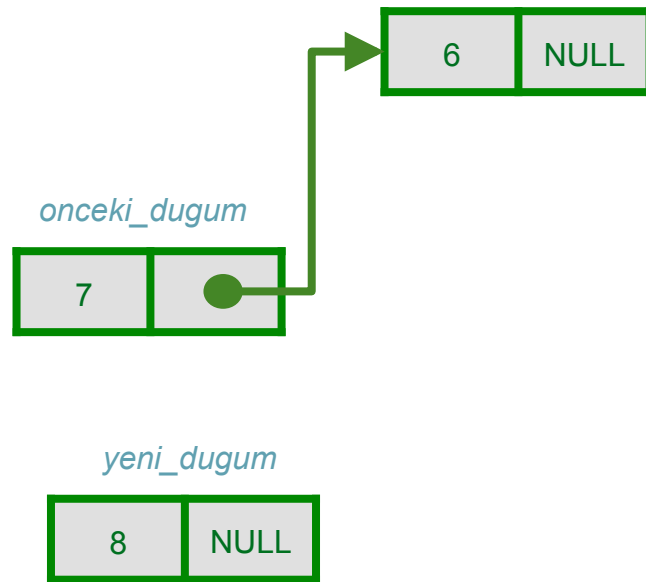
```
void basa_ekle(struct Dugum** birinci_ref, int yeni_veri)
{
    /* 1. yeni dugum için yer al */
    struct Dugum* yeni_dugum = (struct Dugum*) malloc(sizeof(struct Dugum));
    /* 2. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 3. Yeni dugum birinci dugum olmalı */
    yeni_dugum->sonraki = (*birinci_ref);
    (*birinci_ref) = yeni_dugum;
}
```



# Araya Ekleme

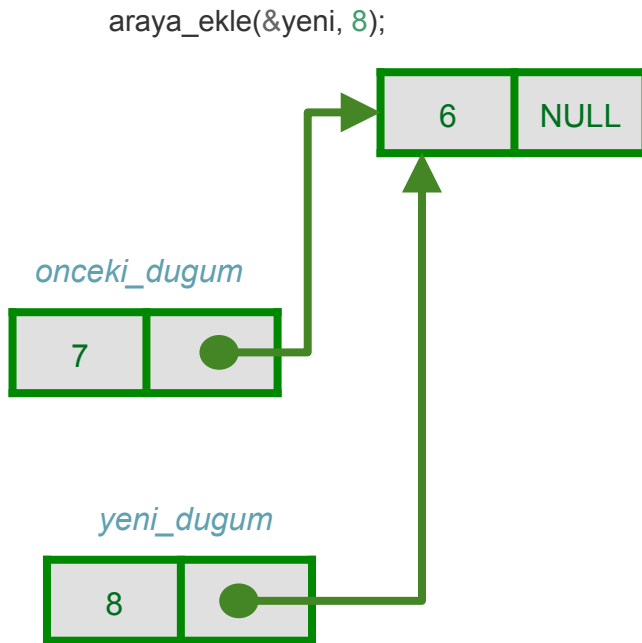
```
void araya_ekle(struct Dugum* onceki_dugum, int yeni_veri)
{
    /*1. onceki dugum NULL mu? */
    if (onceki_dugum == NULL)
    {
        printf("onceki dugum NULL olmamalı");
        return;
    }
    /* 2. yeni dugum için yer al */
    struct Dugum* yeni_dugum =(struct Dugum*) malloc(sizeof(struct Dugum));
    /* 3. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 4. yeni dugum araya al */
    yeni_dugum->sonraki = onceki_dugum->sonraki;
    onceki_dugum->sonraki = yeni_dugum;
}
```

araya\_ekle(&yeni, 8);



# Araya Ekleme

```
void araya_ekle(struct Dugum* onceki_dugum, int yeni_veri)
{
    /*1. onceki dugum NULL mu? */
    if (onceki_dugum == NULL)
    {
        printf("onceki dugum NULL olmamalı");
        return;
    }
    /* 2. yeni dugum için yer al */
    struct Dugum* yeni_dugum =(struct Dugum*) malloc(sizeof(struct Dugum));
    /* 3. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 4. yeni dugum araya al */
    yeni_dugum->sonraki = onceki_dugum->sonraki;
    onceki_dugum->sonraki = yeni_dugum;
}
```

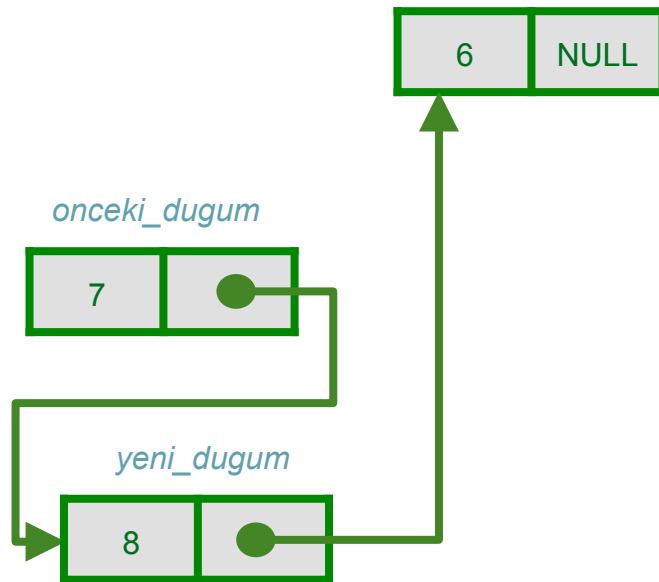




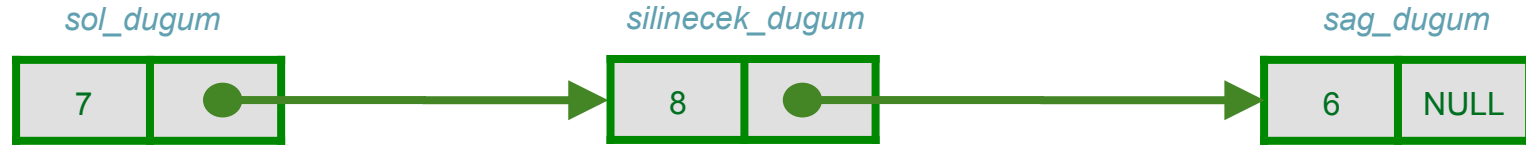
# Araya Ekleme

```
void araya_ekle(struct Dugum* onceki_dugum, int yeni_veri)
{
    /*1. onceki dugum NULL mu? */
    if (onceki_dugum == NULL)
    {
        printf("onceki dugum NULL olmamalı");
        return;
    }
    /* 2. yeni dugum için yer al */
    struct Dugum* yeni_dugum =(struct Dugum*) malloc(sizeof(struct Dugum));
    /* 3. yeni dugume yeni veri */
    yeni_dugum->veri = yeni_veri;
    /* 4. yeni dugum araya al */
    yeni_dugum->sonraki = onceki_dugum->sonraki;
    onceki_dugum->sonraki = yeni_dugum;
}
```

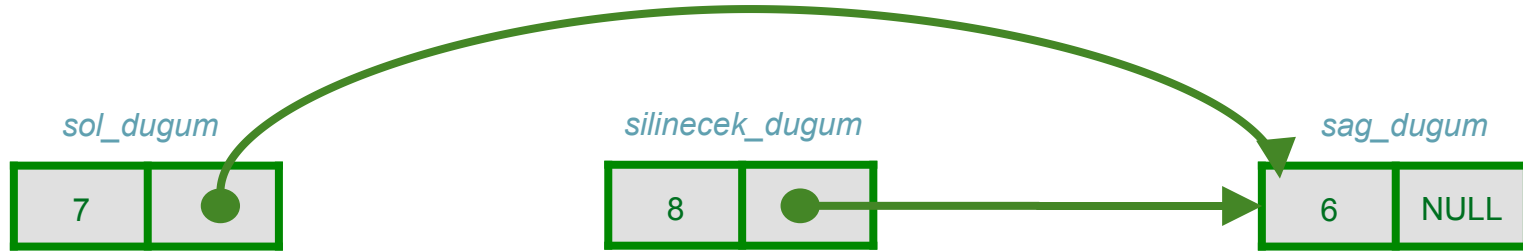
araya\_ekle(&yeni, 8);



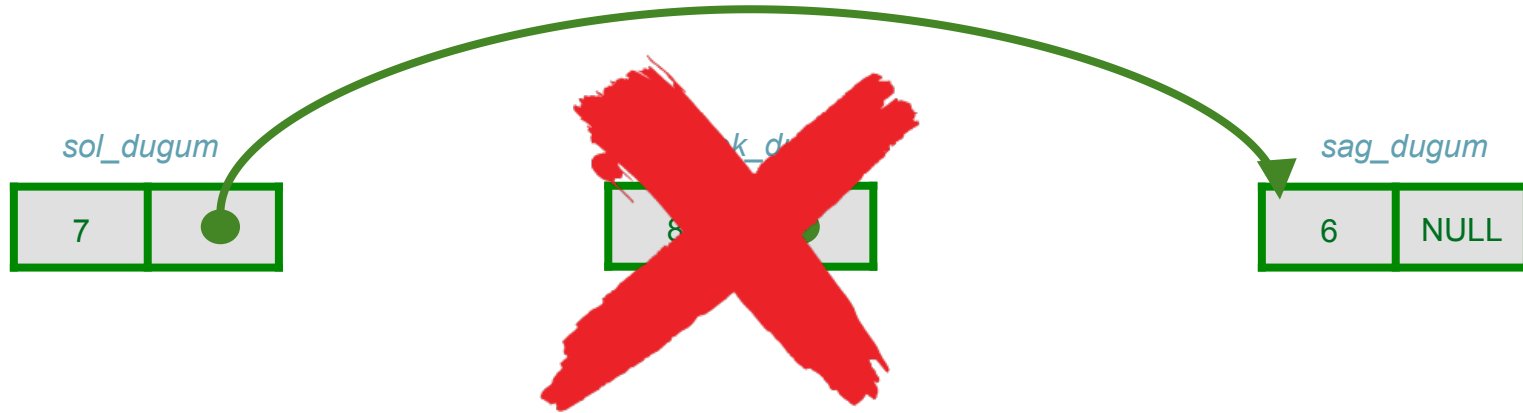
# Silme



# Silme



# Silme



# Silme



# Sorular

