

# Bellek İşlemleri



Gömülü Sistemler  
Laboratuvarı

# 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 }
```

# isaretciler (Pointers)

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

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
	0000	0x08BA
x	0000	0x08BC
y	0000	0x08BE
	0000	0x08C0
	0000	0x08C2
	0000	0x08C4
	0000	0x08C6

# isaretciler (Pointers)

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

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

x

y

p

# 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;
```

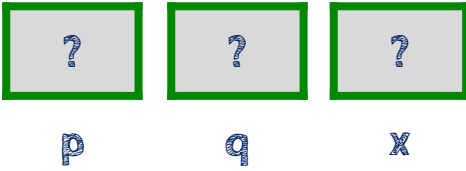


Diagram illustrating the initial state of memory. Three boxes are shown, labeled `p`, `q`, and `x` below them. Each box contains a question mark, indicating they are uninitialized.

```
p = &x;
```

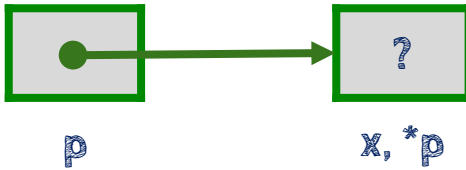


Diagram illustrating the state after `p = &x;`. The box labeled `p` now contains a dot and an arrow pointing to the box labeled `x`, which contains a question mark. The label `x, *p` is placed below the box `x`.

```
*p = 6;
```

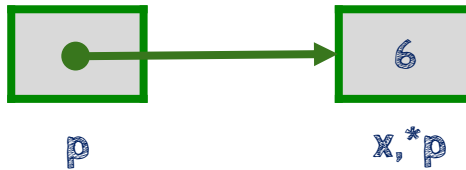


Diagram illustrating the state after `*p = 6;`. The box labeled `p` contains a dot and an arrow pointing to the box labeled `x`, which now contains the value 6. The label `x, *p` is placed below the box `x`.

```
p = new int;
```

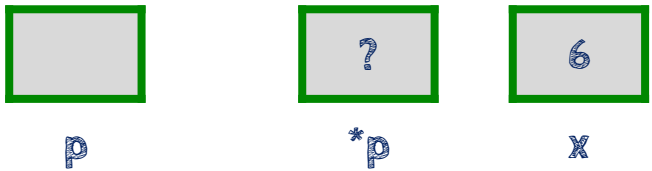


Diagram illustrating the state after `p = new int;`. The box labeled `p` is empty. The box labeled `*p` contains a question mark. The box labeled `x` contains the value 6.

```
*p = 7;
```

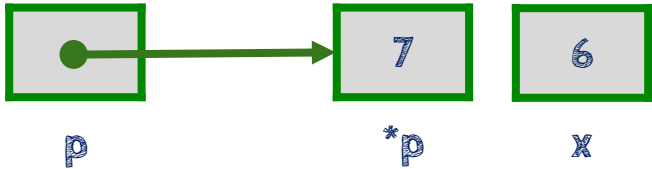
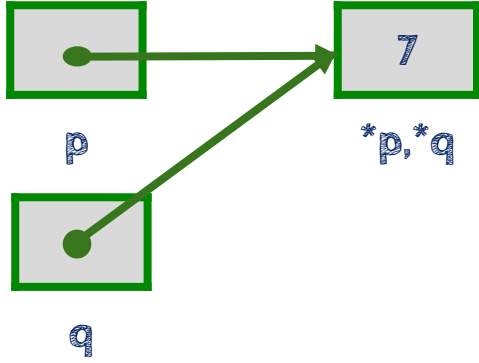


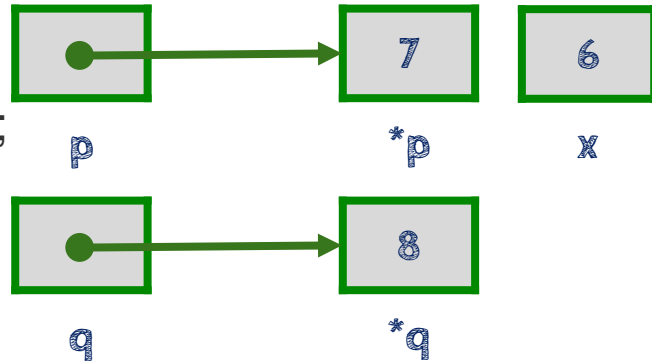
Diagram illustrating the state after `*p = 7;`. The box labeled `p` contains a dot and an arrow pointing to the box labeled `*p`, which contains the value 7. The box labeled `x` still contains the value 6.

# isaretciler (Pointers)

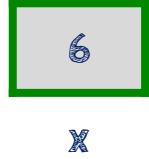
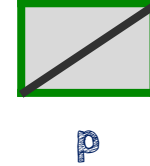
q = p;



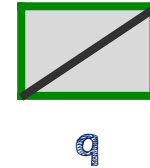
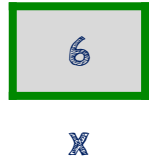
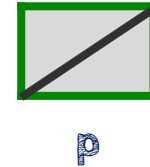
q = new int;  
\*q = 8;



p = NULL;



Delete q;  
q = NULL;



# Degiskenlerde Tür Dönüşümleri

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

```
#include <string.h>
```

```
int main() {
```

```
    float a = 3.5;
```

```
    int b = (int)a;
```

```
    printf("a: %f\n", a);
```

```
    printf("b: %d\n\n", b);
```

```
    return 0;
```

```
}
```

# Verilerin Bit İfadesi

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

```
int main() {
```

```
    int hesap = (256*256*256)*'t' + (256*256)*'s' + 256*'e' + 't';  
    printf("%d\n", hesap);
```

```
    return 0;
```

```
}
```

# İsaretcilerde Tür Dönüşümleri

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

```
#include <string.h>
```

```
int main() {
```

```
    int i;
```

```
    int dizi[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
    printf("dizi: ");
```

```
    for (i = 0 ; i < 10 ; i++)
```

```
        printf("%d ", dizi[i]);
```

```
    printf("\n\n");
```

```
    char *s = (char*)(dizi);
```

```
    strcpy(s, "test");
```

```
    printf("s: %s\n\n", s);
```

```
    printf("dizi: ");
```

```
    for (i = 0 ; i < 10 ; i++)
```

```
        printf("%d ", dizi[i]);
```

```
    printf("\n\n");
```

```
    return 0;
```

```
}
```



# malloc & free

```
#include <stdio.h>
#include <stdlib.h>
int main() {

    int *A;

    A = (int*) malloc( sizeof(int) );

    printf("A'nin gosterdigi adres: %p\n\n", A);

    *A = 123;
    printf("A'nin degeri: %d\n\n", *A);

    free(A);

    return 0;
}
```

# malloc & free

```
#include <stdio.h>
#include <stdlib.h>
int main() {

    int *A;
    A = (int*) malloc(10 * sizeof(int));

    if (A == NULL) {
        printf("HATA: bellek ayrilamadi\n");
        exit(1); // programi sonlandir
    }
    A[0] = 123;
    A[1] = 444;
    A[9] = 674;
    printf("%d\n", A[0]); // *A
    printf("%d\n", A[1]); // *(A+1)
    printf("%d\n", A[9]); // *(A+9)

    free(A);
    return 0;
}
```

# malloc & free

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i,N,*dizi_ptr;
    printf("eleman sayisini girin: ");
    scanf("%d", &N);
    dizi_ptr = (int*) malloc( N * sizeof(int) );
    if (dizi_ptr == NULL) {
        printf("HATA: bellek ayrilamadi\n");
        exit(1); // programi sonlandir
    }
    for (i = 0 ; i < N ; i++) {
        printf("sayi girin: ");
        scanf("%d", &dizi_ptr[i]);
    }
    printf("girilen sayilar:\n");
    for (i = 0 ; i < N ; i++)
        printf("%d\n", dizi_ptr[i]);
    free(dizi_ptr);
    return 0;
}
```

# malloc & free

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

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

```
int main() {
```

```
    /* sonsuz dongude serbest birakmadan malloc fonksiyonunu kullanmak
```

```
    UYARI: asagidaki kod calisitrilrsa bilgisayar kilitlenebilir
```

```
    */
```

```
    while (1) {
```

```
        int *a = (int*) malloc(100000);
```

```
    }
```

```
    return 0;
```

```
}
```

# malloc & free

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

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

```
int main() {
```

```
    int i;
```

*/\* ayrilmis alani gosteren pointera baska deger atayip alanin adresini kaybetmek. Programin git gide daha fazla bellek kullanmasina (memory leak) sebep olur. Bellek doldugunda bilgisayarın kilitlenmesine sebep olabilir.*

*\*/*

```
    int *b = (int*) malloc(1000*sizeof(int));
```

```
    int x = 10;
```

```
    b = &x;
```

```
    return 0;
```

```
}
```

# malloc & free

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

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

```
int main() {
```

```
    int i;
```

```
    /* Bir onceki hata buyuk bir dongude ise kilitlenmeye sebep olabilir.
```

```
    UYARI: asagidaki kod calistirilrsa bilgisayar kilitlenebilir
```

```
    */
```

```
    for (i = 0 ; i < 1000000000 ; i++) {
```

```
        int *c = (int*) malloc(1000*sizeof(int));
```

```
        int x = 10;
```

```
        c = &x;
```

```
    }
```

```
    return 0;
```

```
}
```

# malloc & free

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

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

```
int main() {
```

```
    /* aynı alani birden fazla kere serbest bırakmak
```

```
       programın sonlanmasına veya beklenmeyen bir davranış
```

```
       sergilemesine sebep olur
```

```
    */
```

```
    int *d = (int*) malloc(1000*sizeof(int));
```

```
    free(d);
```

```
    free(d);
```

```
    free(d);
```

```
    return 0;
```

```
}
```

# malloc & free

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

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

```
#include <string.h>
```

```
int main() {
```

```
    char *p_dizi;
```

```
    p_dizi = malloc(5 * sizeof(char));
```

```
    strcpy(p_dizi, "test");
```

```
    printf("%s\n\n", p_dizi);
```

```
    p_dizi = realloc(p_dizi, 100 * sizeof(char));
```

```
    strcat(p_dizi, " 123456789123456789");
```

```
    printf("%s\n", p_dizi);
```

```
    printf("stringin boyutu: %d\n", strlen(p_dizi));
```

```
    printf("bellegin boyutu: 100\n\n");
```

```
    int karakter_sayisi = strlen(p_dizi)+1; // +1 sonlandirma karakteri
```

```
    p_dizi = realloc(p_dizi, karakter_sayisi * sizeof(char) );
```

```
    printf("%s\n", p_dizi);
```

```
    printf("stringin boyutu: %d\n", strlen(p_dizi));
```

```
    printf("bellegin boyutu: %d\n", strlen(p_dizi)+1);
```

```
    free(p_dizi);
```

```
    return 0;
```

```
}
```



# Dosya Sistemi



# fopen()

**FILE** \*fopen(**const char** \*filename, **const char** \*mode)

\*filename: dosya yolu

\*mode: açma modu

# Text Dosyası

**FILE** \*fopen(\*dosya\_yolu, \*acma\_modu)

*// dosya\_yolu*

Windows

“d:\test.txt”

Linux

“/home/kullanici/test.txt”

# Text Dosyası

**FILE** \*fopen(\*dosya\_yolu, \*acma\_modu)

*// acma\_modu*

- ❖ “r” : okuma
- ❖ “w” : yazma (dosyanın içeriğini silip baştan yazar, yoksa oluşturur)
- ❖ “a” : ekleme (dosya sonuna yazar, yoksa oluşturur)
- ❖ “r+” : okuma ve güncelleme (dosya yoksa açmaz, hata verir)
- ❖ “w+” : yazma ve güncelleme (dosyanın içeriğini silip açar)
- ❖ “a+” : ekleme ve güncelleme

# ilk karakteri okuma

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

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

```
int main() {
```

```
    FILE * dosya = fopen("test.txt", "w");
```

*// dosyayi ac*

```
    if (dosya == NULL) {
```

*// dosyaya erisilemiyorsa NULL*

*olur*

```
        printf("dosya acilamadi\n");
```

```
        exit(1);
```

```
    }
```

```
    char c = fgetc(dosya);
```

```
    if (c == EOF) {
```

*// dosya sonu kontrolu*

```
        printf("dosya sonu, karakter yok\n");
```

*// dosyanin sonuna gelindiyse EOF (-1) degeri*

*okunur.*

```
    } else {
```

```
        printf("okunan karakter: %c\n", c);
```

*// dosyada ilk karakteri okunur.*

```
    }
```

```
    fclose(dosya);
```

*// dosyayi kapat*

```
    return 0;
```

```
}
```

# birden fazla karakter okuma

```
#include <stdio.h>
#include <stdlib.h>
#define DOSYA_YOLU "test.txt"

int main() {

    FILE * dosya;
    char c;
    // dosyayi ac
    if ((dosya = fopen(DOSYA_YOLU, "r")) == NULL) {
        printf("dosya acilamadi!\n");
        exit(1);
    }

    // dosya sonuna gelene kadar dongu calissin
    while ((c = fgetc(dosya)) != EOF) {
        printf("karakter: %c (%d)\n", c, c);
    }

    // dosyayi kapat
    fclose(dosya);
    return 0;
}
```

# fgets()

`char *fgets(char *str, int n, FILE *stream)`

\*str: Okunan “string”i tutan işaretçi

n: Okunacak maksimum karakter sayısı(boş karakter dahil)

\*stream: Okunacak string

# fgets()

`char *fgets(char *str, int n, FILE *stream)`

İşlem başarılı ise, \*str işaretçisi döner

Dosya Sonu ile karşılaşıldığında ve/veya hiçbir karakter okunmazsa, \*str işaretçisinin içeriği değişmeden kalır ve boş işaretçisi (Null Pointer) döndürülür.

Bir hata oluşursa, boş işaretçi(Null Pointer) döner.



# formatlı okuma

```
#include <stdio.h>
#include <stdlib.h>
#define DOSYA_YOLU "test.txt"

int main() {
    satir_satir_oku();
    printf("\n-----\n");
    formatli_oku();
    return 0;
}
```

# formatlı okuma

```
void satir_satir_oku() {  
    FILE * dosya;  
    char buf[100];  
  
    if ((dosya = fopen(DOSYA_YOLU, "r")) == NULL) {  
        printf("dosya acilamadi!\n");  
        exit(1);  
    }  
  
    int satir_sayisi = 0;  
    // satir satir okuma islemi  
    while ( fgets(buf, 100, dosya) != NULL ) {  
        satir_sayisi++;  
        printf("%d. satir: \"%s\"\n", satir_sayisi, buf);  
    }  
}
```

# fscanf()

```
int fscanf(FILE *stream, const char *format, char *buf)
```

\*stream: String'in okunduğu yeri tutan işaretçi

\*buf: Okunan string'in saklandığı yeri tutan işaretçi

\*format: Okunan karakterin nasıl tutulacağı

# formatlı okuma

```
void formatli_oku() {  
    FILE * dosya;  
  
    if ((dosya = fopen(DOSYA_YOLU, "r")) == NULL) {  
        printf("dosya acilamadi!\n");  
        exit(1);  
    }  
  
    char buf[100];  
  
    fscanf(dosya, "%99s", buf); // max 99 harfli kelime oku  
    printf("okunan kelime: %s\n", buf);  
  
    fscanf(dosya, "%99s", buf);  
    printf("okunan kelime: %s\n", buf);  
  
    fclose(dosya);  
}
```

# Dosyada okunacak yeri belirleme

```
#include <stdio.h>
#include <stdlib.h>
#define DOSYA_YOLU "test.txt"
int main() {
    FILE * dosya;
    if ((dosya = fopen(DOSYA_YOLU, "r")) == NULL) {
        printf("dosya acilamadi!\n");
        exit(1);
    }
    fseek(dosya, 0, SEEK_END);
    long int len = ftell(dosya);
    printf("dosyanin boyutu: %ld byte\n", len);
    fseek(dosya, 0, SEEK_SET);
    long int gosterge = ftell(dosya);
    printf("%ld. indexteki karakter okunacak\n", gosterge);
    char c = fgetc(dosya);
    printf("okunan karakter: %c\n\n", c);
    gosterge = ftell(dosya);

    printf("%ld. indexteki karakter okunacak\n", gosterge);
    c = fgetc(dosya);
    printf("okunan karakter: %c\n\n", c);
    fseek(dosya, len-2, SEEK_SET);
    c = fgetc(dosya);
    printf("sondan iki onceki karakter: %c\n", c);
    fclose(dosya);
    return 0;
}
```

*// dosyanin sonuna atla  
// gostergein bulunduğu byte numarasini oku*

*// dosya basina atla  
// bulunduğu byte numarasini oku*

*// bulunduğu byte numarasini*

*// sondan iki önceki karaktere atla*

*// dosyayi kapat*

oku

# Sorular

