Pointerlar

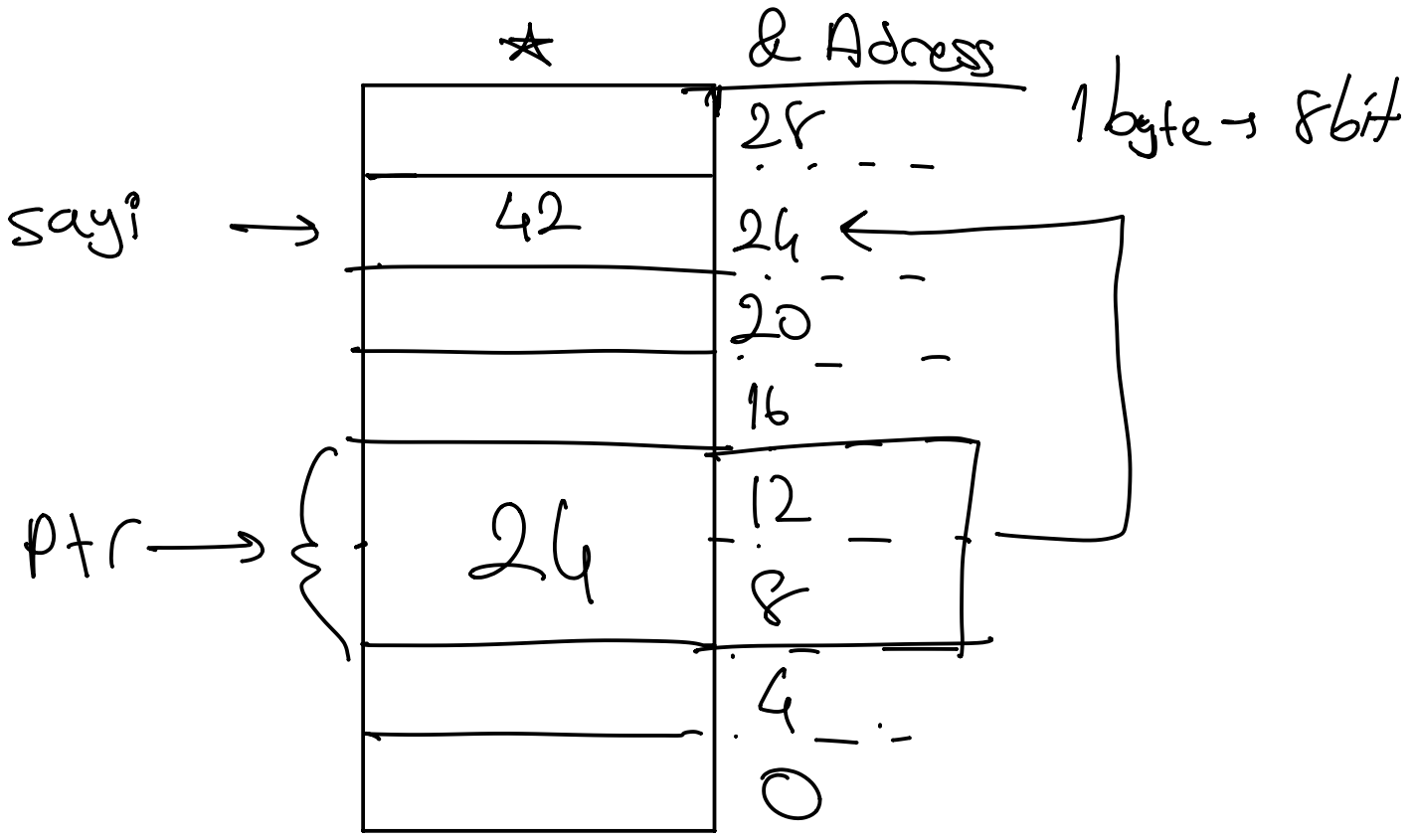
* → value of

& → address of

```
int sayi = 42;
int* ptr = &sayi
```

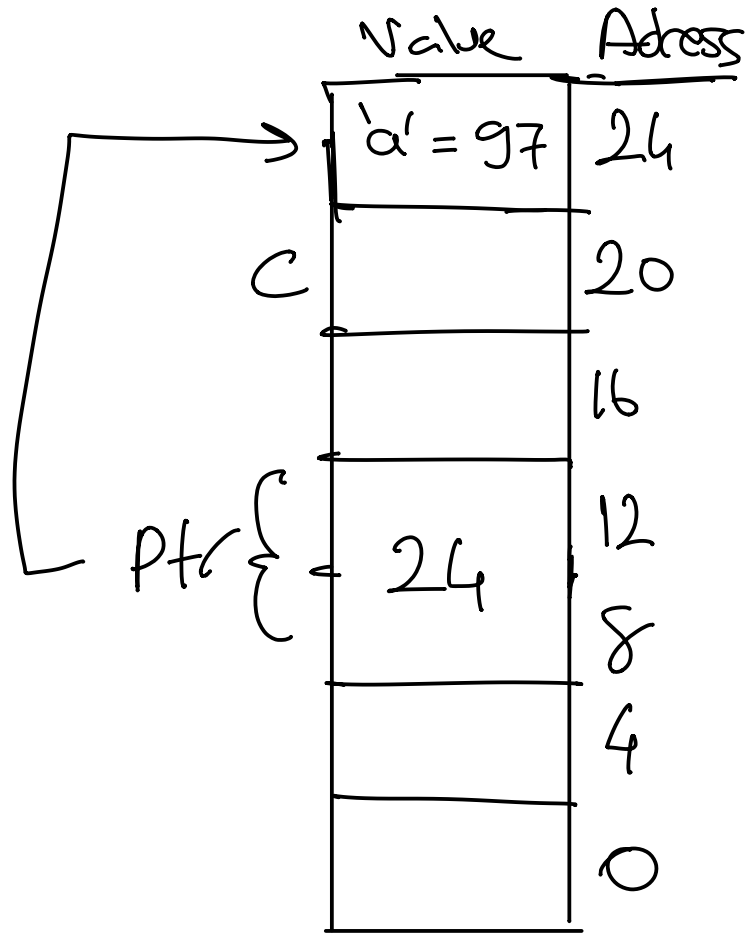
ptr sayının adresini gösteriyor

*ptr → sayının değerini gösterir



Bilgi

Bütün pointerlar 64 bit işlemci için 64 bittir. Böylece $2^{64}-1$ tane belleği adresleyebilir.




```
char c = 'a'
```
↓ 97 ASCII Value

```
char* ptr = &c
```

char, short, int, long

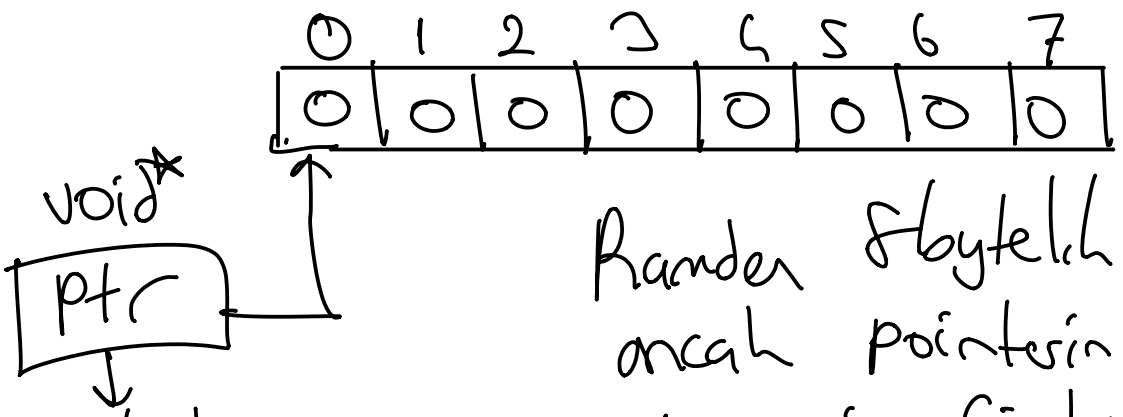
bütün veri tiplerinin pointeri 64bit bilgisayar için 8byte'dır Çünkü işlemci en fazla 8byteklık kısmı adresleyebilir.

Malloc

Malloc dinamik veri gerektiğinde kullanılır. Derleme anında ne kadar bellek gerektiği net bilinmez.

```
void* ptr = malloc(8);
```

Ramden 8byte iste ve tahsis edilen kısmı ptr'a ata



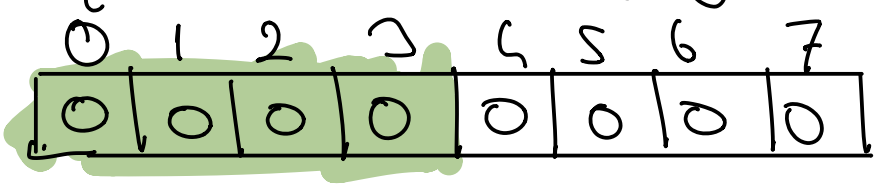
Ramden 8byteluk yer istedik ancak pointerın gösterdiği veriye erişemeyiz. Çünkü gösterdiği addresten ne kadar okuyacağını belirtmedik (void*)

örnek olarak

```
int* tam = ptr; // diyelim.
```

eğer *tam yaparsak ilk 4byte'ını okur

Çünkü int 4byte.



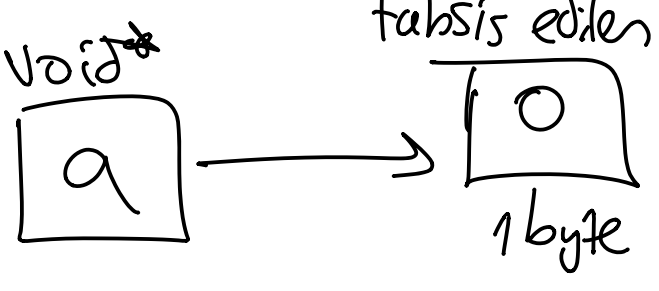
ancak geriye kalan 4byte boşa gitmiş olur.

bu sebep ile çoğunlukla kullanacağımız yer kadar bellek tahsis ederiz.

Örneğin;

```
void* a = malloc(1) yada malloc(sizeof(char));
```

(bir veri tipinin boyutunu verir.)



*a yaparak dataya erişemezsiniz. Çünkü void bir tip belirtmez.

char büyüklüğünde tahsis ettiğimiz için char* 'a casting işlemi yapacız

char* karakter = a;

büyüklükle *karakter yaparak orda bulunan veriye erişiriz.

<type>* <degisken_ismi);

tipler aslında gösterdiği adresten nekadar okuyacağını belirtir.

Aslında dolaylı yoldan void* 'in herhangi bir tipden veriyi tuttuğunu söylemiş oluyoruz.

int a; long b; char c;

void* ptr = &a, &b, &c } hepsi doğru

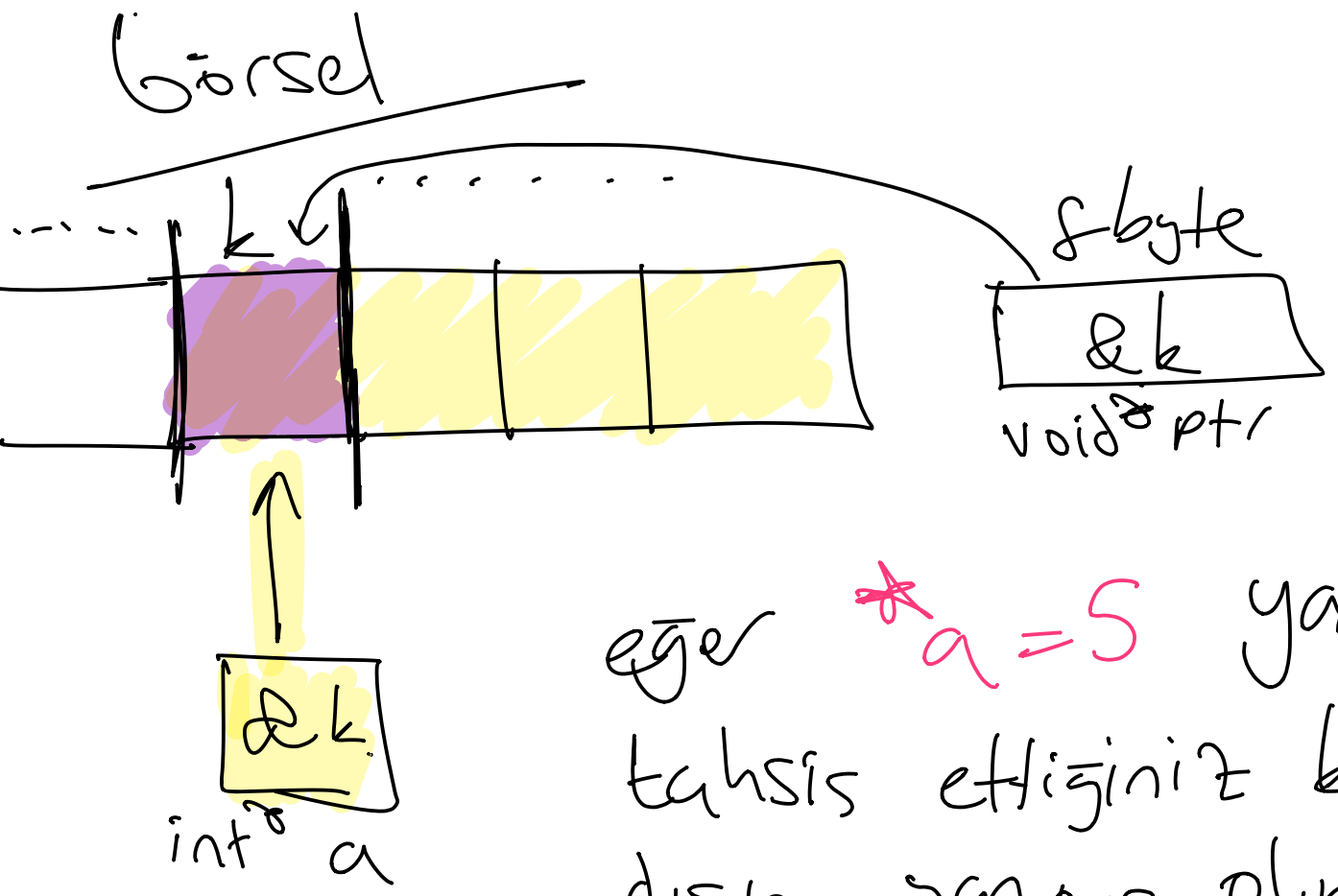
ancak ptr tipini yazılımcı olarak bilmelisiniz yoksa erişmeniz gereken yerlere erişebilirsiniz.

Örneğin

char k; void* ptr = &k;

unutup int* a = ptr; yanlış tipe casting yaparsanız

*a = 5 yaptığınız zaman 1 byte'lık kısmın dışına yazmış olursunuz



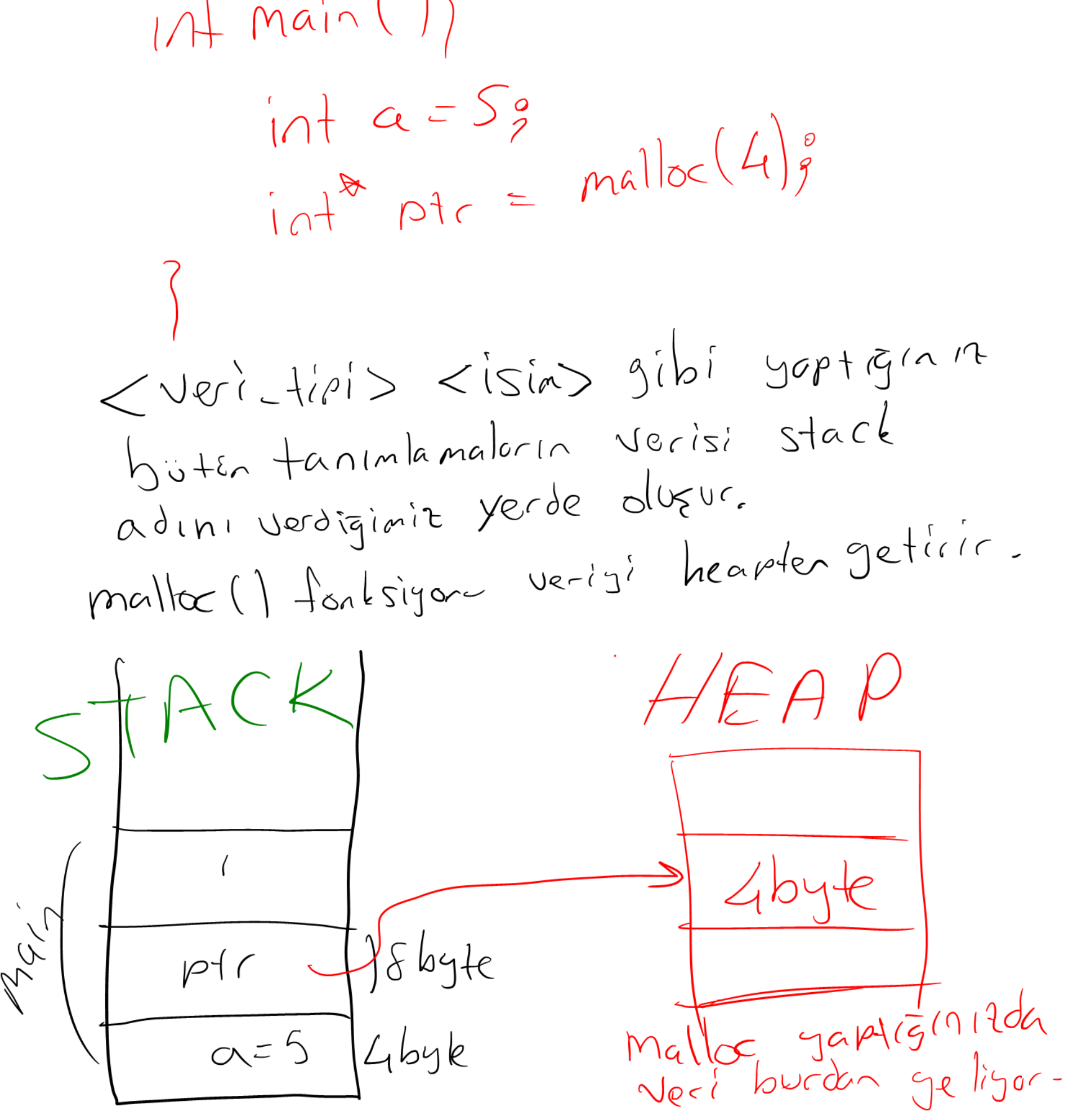
eğer *a=5 yaparsanız tahsis ettiğiniz kısmın dışına yazmış olursunuz. Çünkü int 4 byte'lık okur/yazar

heap ve stack bellek

```
int main(){
    int a=5;
    int* ptr = malloc(4);
}
```

<veri_tipi> <isim> gibi yaptığınız bütün tanımlamaların verisi stack adını verdiğimiz yerde oluşur.

malloc() fonksiyonu veriyi heap'ten getirir.



Structlar

Structlar veri tiplerini paketleyip yeni veri tipleri oluşturmanızı sağlar.

Ör:

```
Struct Ogrenci{
    int no;
    int sinif;
    char isim[32];
};
```

4 byte
4 byte
32 byte
) 40 byte

```
Struct Ogrenci ogr;
```

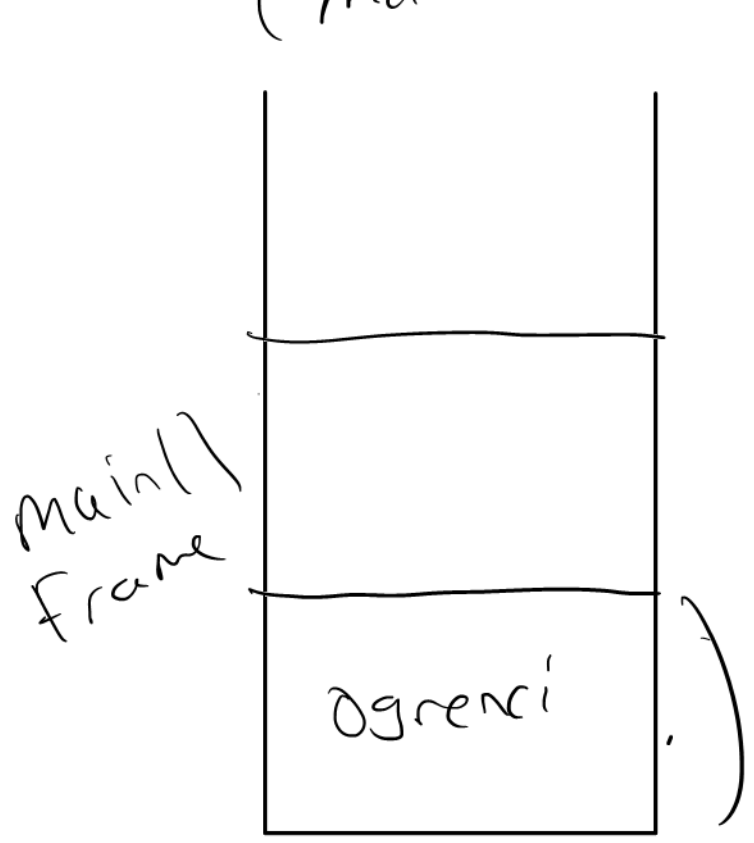
veya

```
Struct Ogrenci *ogr_ptr = malloc(sizeof(Struct Ogrenci));
```

yapacak struct verisi oluşturabiliriz

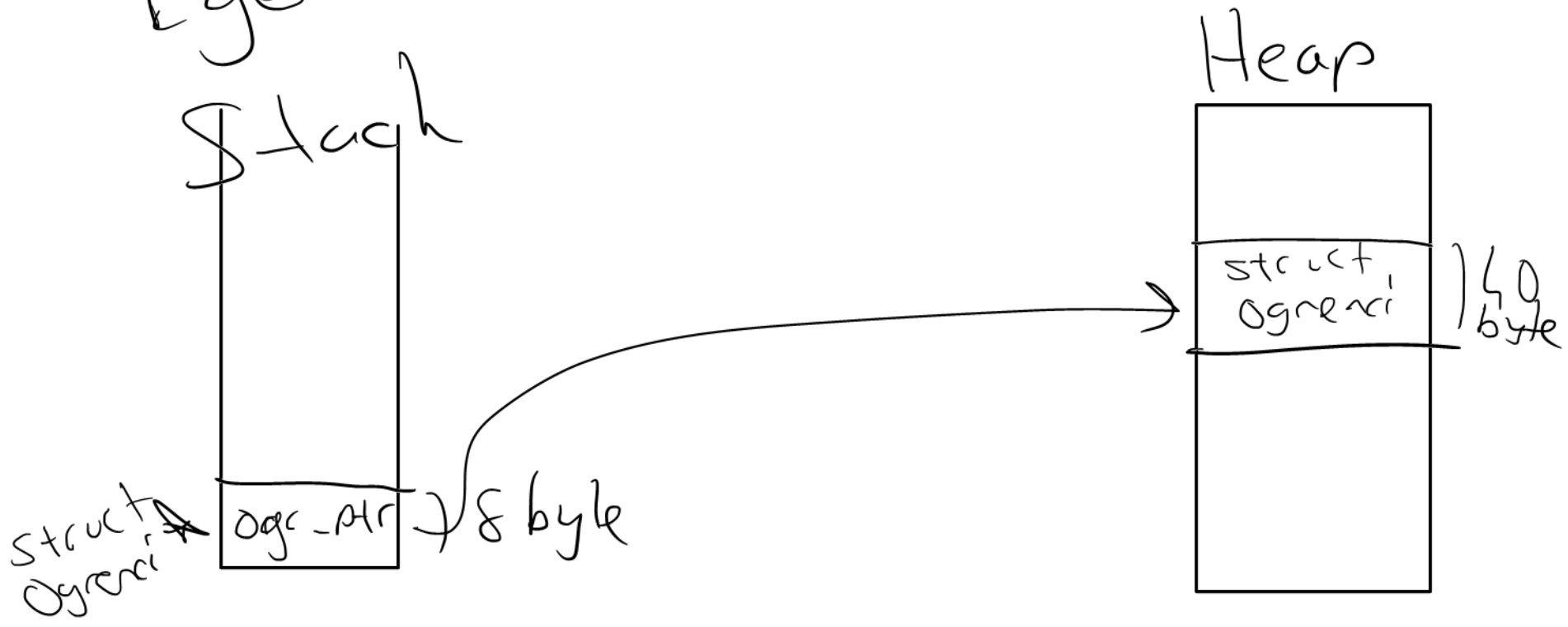
Heap ve Stack Mantığını Pekiştirmek için tekrar görselleştirelim.

Eğer normal şekilde struct tanımlasaydınız. (malloc kullanmadan)



) sizeof(Struct Ogrenci) kadar yer stackde kaplar.

Eğer malloc kullanırsanız.



Structa Veriye Erişim

pointer kullanarak erişmek için

ptr -> yas kullanmalısınız.

VEYA

(*ptr).yas

pointerin gösterdiği yere git ve ↑ Eriş

pointersız erişim

ogrenci.yas doğrudan nokta ile erişirsiniz.