ALGORİTMA VE PROGRAMLAMA II

HAFTA#4

YZM 1106

Celal Bayar Üniversitesi Hasan Ferdi Turgutlu
Teknoloji Fakültesi

Genel Bakış…

- Bellek ve Adresleme
- İşaretçi Kavramı
  - Adresleme
  - İşaretçi Değişkenleri Bildirmek ve Değişkenlere Atama Yapmak
  - NULL İşaretçiler
  - İşaretçi Zinciri
  - İşaretçi Aritmetiği
    - Değişken Adresinin Arttırılması
    - Değişken Değerinin Arttırılması
    - İşaretçi İşlemlerinde ++ ve – operatörlerinin kullanımı

3. BÖLÜM

İşaretçilere Giriş

Bellek ve Adresleme

- Bilgisayarın ana belleği (**RAM**) sıralı kaydetme **gözlerinden** oluşmuştur.
- Her göze bir adres atanmıştır.
- Bu adreslerin değerleri 0 ila belleğin sahip olduğu üst değere bağlı olarak değişebilir.
- Örneğin 1GB bir bellek,
  - 1024*1024*1024 = 1.073.741.824 adet gözden oluşur.

Bellek ve Adresleme (devam…)

- Bir programlama dillinde, belli bir tipte değişken tanımlanıp ve bir değer atandığında, o değişkene dört temel özellik eşlik eder:

1. değişkenin adı
2. değişkenin tipi
3. değişkenin sahip olduğu değer (içerik)
4. değişkenin bellekteki adresi

Bellek ve Adresleme (devam…)

- Değişken türlerinin bellekte kapladığı alanlar:

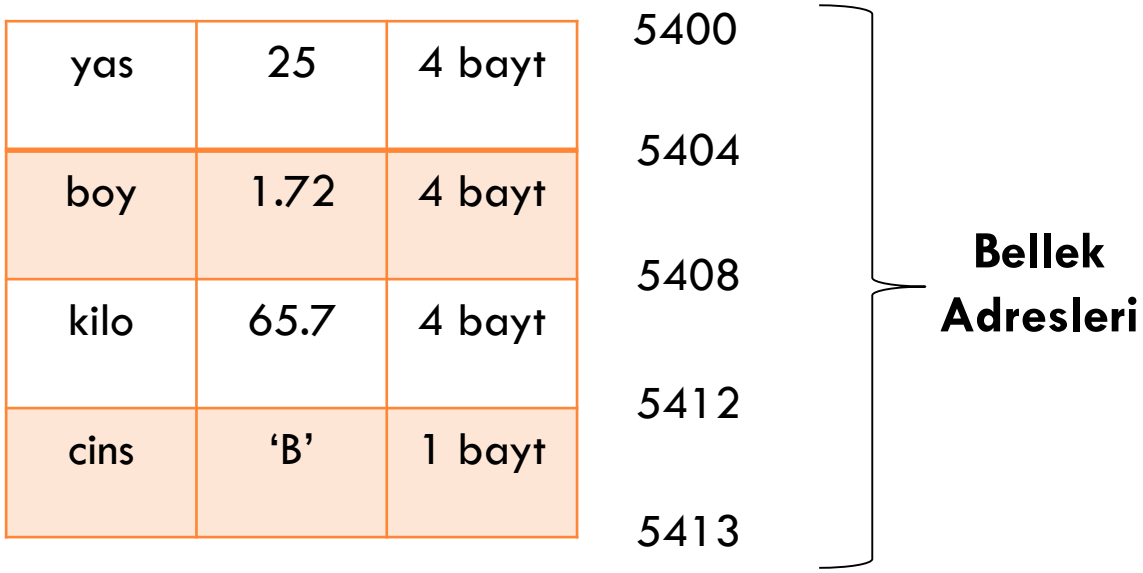
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf( "char
  printf( "short
  printf( "int
  printf( "long
  printf( "unsigned c
  printf( "unsigned s
  printf( "unsigned i
  printf( "unsigned l
  printf( "float
  printf( "double
  printf( "long doubl
  return 0;
}
```

```
char               : 1 bayt
short              : 2 bayt
int                : 4 bayt
long               : 4 bayt
unsigned char      : 1 bayt
unsigned short     : 2 bayt
unsigned int       : 4 bayt
unsigned long      : 4 bayt
float              : 4 bayt
double             : 8 bayt
long double        : 12 bayt
```

Örnek: Bellek ve Adresleme


```
int yas = 25;
float boy = 1.72;
float kilo = 65.7;
char cins = 'B';
```



Örnek: Bellek ve Adresleme (devam…)

- Atama deyimlerine göre hücrelerin adreslere yerleşimi otomatik olarak gerçekleşir.
  - **yas** değişkeni 5400, 5401, 5402, 5403 adreslerini kapsadığından
  - **boy** değişkeni 5404 adresinden başlar ve sırasıyla 5405, 5406 ve 5407 adreslerini işgal eder.
  - **kilo** değişkeni bu nedenle 5408 adresinden başlar ve 5409, 5410, 5411 adreslerini kapsar.
  - **cins** değişkeni ise 5412 adresini tutar.

Bellek ve Adresleme (devam…)

- Örnekte belirtilen kod bloğundaki değişkenler bilgisayar tarafından belirli bir adreste saklanır. Bu yönteme **implicit (örtük)** adresleme denir.
- İşaretçi değişken kullanılarak, işaretçilere verilerin bellekte saklandığı bellek hücrelerinin başlangıç adresleri atanır. Bu yönteme ise **explicit (açık)** adresleme denir.

Bellek ve Adresleme (devam…)

- **Örnek:**

```
int tam = 33;
```

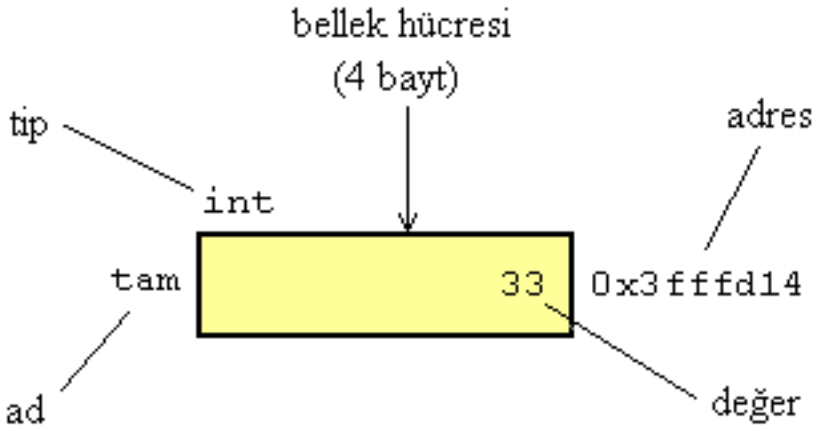
- Bu değişken için, int tipinde bellekte (*genellikle herbiri 1 bayt olan 4 bayt büyüklüğünde*) bir hücre ayrılır ve o hücreye **33 sayısı** ikilik (binary) sayı sitemindeki karşılığı olan 4 baytlık (32 bitlik) karşılığı aşağıdaki gibi yazılır.

00000000 00000000 00000000 00100001

Bellek ve Adresleme (devam…)

- **Örnek:**

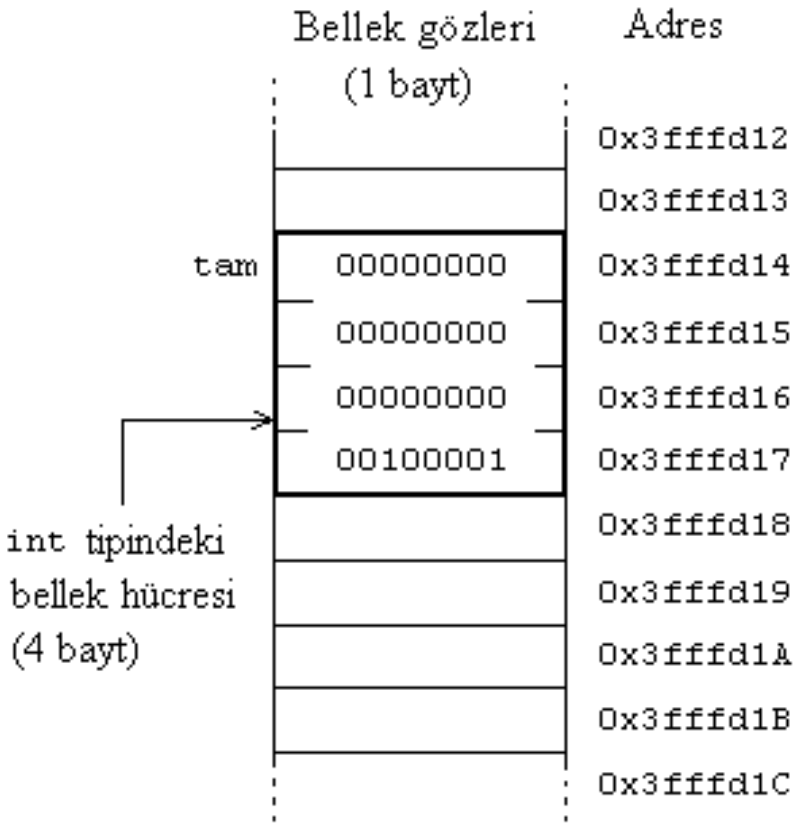
```
int tam = 33;
```



- Bellek adresleri genellikle onaltılık (hexadecimal) sayı sisteminde ifade edilir.
- 0x3fffd14 sayısı onluk (decimal) sayı sisteminde **67108116** sayına karşılık gelir. Bunun anlamı, **tam** değişkeni, program çalıştığı sürece, bellekte **67108116. - 67108120**. numaralı gözler arasındaki **4 baytlık** hücreyi işgal edecek olmasıdır.

Bellek ve Adresleme (devam…)

- **tam** adlı değişkenin bellekteki gerçek konumu ve ikilik düzendeki içeriği aşağıdaki gibidir:



Bellek ve Adresleme (devam…)

- Değişkenin saklı olduğu adres, **&** karakteri ile tanımlı adres operatörü ile öğrenilebilir.
- Bu operatör bir değişkenin önüne konursa, o değişkenin içeriği ile değil adresi ile ilgileniliyor anlamına gelir.

Örnek: Bellek ve Adresleme (devam…)


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

int main()
{
    int tam = 33;
    printf("icerik: %d \n", tam);
    printf("adres: %p \n", &tam);
    return 0;
}
```

```
icerik: 33
adres: 0028FF0C
```

İşaretçi (Pointer) Kavramı

- C dili, bir değişkenin adresinin bir başka değişkende saklanmasına izin verir. Bu değişkene işaretçi denir.
- İşaretçi denmesinin sebebi ilgili değişkenin adresini işaret etmesinden yani göstermesinden kaynaklanır.
- Diğer bir deyişle **işaretçi**, bir değişkenin adresini içeren **başka bir değişkendir**.

İşaretçi (Pointer) Kavramı (devam…)

- İşaretçiler, C'nin yönetilmesi en zor yetenekleri arasındadır.
- Programların referansa göre çağırma yapmasını sağlarlar.
- İşaretçiler sayesinde; Bağlı listeler (**Linked List**), Sıralar (**Queue**), Yığınlar (**Stack**) ve Ağaçlar (**Tree**) gibi büyüyüp küçülebilen dinamik veri yapılarının oluşturulması ve yönetilmesi sağlanır.

İşaretçi (Pointer) Kavramı (devam…)

- Bir işaretçi, diğer değişkenler gibi, sayısal bir değişkendir.
- Bu sebeple kullanılmadan önce program içinde bildirilmelidir. İşaretçi tipindeki değişkenler aşağıdaki gibi tanımlanır:

```
tip_adı *isaretci_adı;
```

- Burada **tip_adı** herhangi bir C veri türü olabilir. Değişkenin önündeki * karakteri yönlendirme (indirection) operatörü olarak adlandırılır ve bu değişkenin **veri değil** **bir adres bilgisi** tutacağını işaret eder.

İşaretçi (Pointer) Kavramı (devam…)

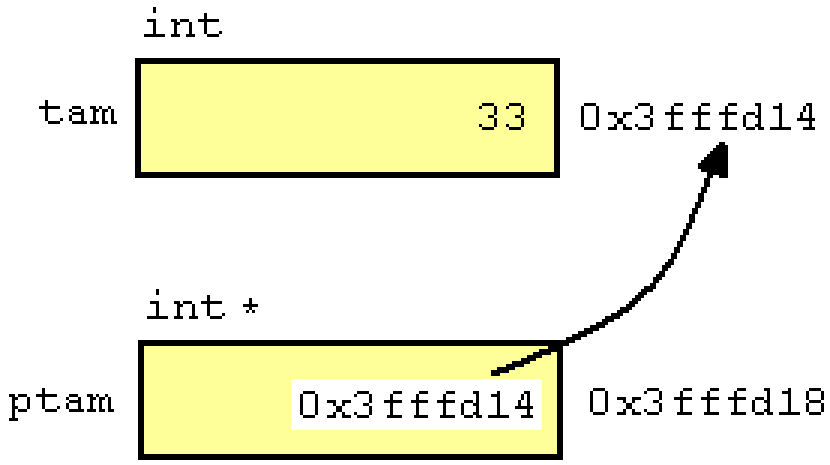
- Örnek:

```
char  *kr;              /* tek bir karakter için */
int   *x;               /* bir tamsayı için */
float *deger, sonuc;    /* deger işaretçi tipinde, sonuc sıradan bir gerçel
                           değişken */
```

Yukarıda bildirilen işaretçilerden; **kr** bir karakterin, **x** bir tamsayının ve **deger** bir gerçel sayının bellekte saklı olduğu yerlerin adreslerini tutar.

İşaretçi (Pointer) Kavramı (devam…)


```
int *ptam, tam = 33;
          .
          .
          .
ptam = &tam;
```



- Bir işaretçiye, bir değişkenin adresini atamak için **& (adres)** operatörünü kullanırız.
- **ptam** işaretçisi **tam** değişkeninin saklandığı adresi tutacaktır.

Örnek: İşaretçi Gösterimi


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

int main()
{
    int tam = 33;
    int *ptam;

    ptam = &tam;

    printf("tam icerik = \t %d \n",tam);
    printf("tam adres = \t %p \n", &tam);
    printf("tam adres = \t %p \n", ptam);

    return 0;
}
```

```
tam icerik =      33
tam adres  =      0028FF08
tam adres  =      0028FF08
```

Örnek: İşaretçi Gösterimi ve Değer Değiştirme


```c
int main()
{
    int tam = 33;
    int *ptam;

    ptam = &tam;

    printf("&tam   = %p\n",&tam);
    printf("ptam   = %p\n",ptam);
    printf("\n");

    printf("tam    = %d\n",tam);
    printf("*ptam  = %d\n",*ptam);
    printf("\n");

    *ptam = 44;       // tam = 44 ataması

    printf("tam    = %d\n",tam);
    printf("*ptam  = %d\n",*ptam);
    printf("\n");

    printf("&tam   = %p\n",&tam);
    printf("ptam   = %p\n",ptam);

    return 0;
}
```

```
&tam   = 0028FF08
ptam   = 0028FF08

tam    = 33
*ptam  = 33

tam    = 44
*ptam  = 44

&tam   = 0028FF08
ptam   = 0028FF08
```

Örnek: İşaretçi Gösterimi ve Değer Değiştirme

(devam..)

- **tam** adlı değişkenin içeriğine **ptam** işaretçisi kullanılarak da erişilebilir.
- Bunun için program içinde **ptam** değişkeninin önüne yönlendirme operatörü (*****) koyulmuştur.
- Yani ***ptam**, **tam** değişkeninin adresini değil içeriğini tutar. Buna göre:

```
*ptam = 44;
```

- komutuyla, **ptam'ın** adresini tuttuğu hücreye 44 değeri atanır.

Örnek: İşaretçi Gösterimi ve Değer Değiştirme

(devam..)

- Özetle
  - ***ptam** ve **tam**, **tam** adlı değişkenin içeriği ile ilgilidir.
  - **ptam** ve **&tam**, **tam** adlı değişkenin adresi ile ilgilidir.
  - ***** yönlendirme ve **& (adres)** operatörüdür.

NULL İşaretçi

- NULL işaretçiler C'de ve birçok kütüphanede sıklıkla kullanılırlar.
- Amaç, işaretçiye herhangi bir değer atanmadığının kontrol edilebilmesine olanak sağlamaktır.
- NULL işaretçi **bir sabittir** ve hemen hemen tüm dillerde **0'ı değerine sahiptir**.

NULL İşaretçi (devam…)


```
int main()
{
    int *p;
    p = NULL; //Commentle ve sonucunu gor

    printf("\np isaretcisinin adresi = %p\n\n", p);

    if (p)
        printf("p isaretcisi NULL degil...\n");
    else
        printf("p isaretcisi NULL...\n");
    return 0;
}
```

```
p isaretcisinin adresi = 00000000

p isaretcisi NULL...
```

NULL İşaretçi (devam…)

- Tüm işletim sistemlerinde 0. adrese erişim yasaklanmıştır.
- Bu adres, işletim sistemi tarafından kullanılır.
- NULL işaretçinin adresinin 0 olmasının sebebi, işaretçinin herhangi bir **adresi göstermediğini** ifade etmektir.

NULL İşaretçi (devam…)

p = NULL; // ptr bellekte hiç bir hücreyi göstermiyor (0 adresi)

*p = 8 // HATA! NULL işaretçisinin gösterdiği yere bir değer atanamaz. **Önceki örnekte deneyelim ve programı çökertelim ☺**

İşaretçi Zinciri

- Genelde bir işaretçi bir değişkenin adresini gösterir.
- Ancak *bir işaretçiye işaret eden başka bir işaretçi* tanımlanabilir.
- Bu duruma **işaretçi zinciri** denir.
- Tanım aşağıdaki gibi yapılır:

```
int **pp;
```

İşaretçi Zinciri (devam…)

- *p işaretçisi val değişkeninin **adresini gösetrmektedir**.
- **pp işaretçisi *p işaretçisinin adresini göstermektedir.

İşaretçi Zinciri (devam…)

Örnek

- Tam sayı tipinde **val** isimli bir değişken tanımlayınız ve değerini **1903** yapınız.
- **val** değişkeninin adresini gösteren ***p** işaretçisi tanımlayınız.
- ***p** işaretçisinin adresini gösteren ****pp** işaretçisi tanımlayınız.
- Gerekli işaretçi atamalarını yapınız.
- Bu 3 değişkeni kullanarak ekrana değerleri ve adresleri yazdırınız.

İşaretçi Zinciri (devam...)


```
int main()
{
    int *p;
    int **pp;
    int val = 1903;

    //p isatecisine val adresi atanıyor
    p = &val;
    //pp isatecisine p isaretcisinin adresi atanıyor
    pp = &p;

    printf("Degerler:\n***********\n");
    printf("val = %d\n", val);
    printf("*p = %d\n", *p);
    printf("**pp = %d\n", **pp);
    printf("\nAdresler:\n***********\n");
    printf("&val = %d\n", &val);
    printf("p = %d\n", p);
    printf("&p = %d\n", &p);
    printf("pp = %d\n", pp);

    return 0;
}
```

İşaretçi Aritmetiği

- İşaretçiler kullanılırken, bazen işaretçinin gösterdiği **adres taban alınıp**, o adresten **önceki** veya **sonraki** adreslere erişilmesi istenebilir.
- Bu durum, işaretçiler üzerinde, aritmetik operatörlerin kullanılmasını gerektirir.
- İşaretçiler üzerinde yalnızca
  - toplama (+),
  - çıkarma (-),
  - bir arttırma (++) ,
  - bir eksiltme (--)

  operatörleri işlemleri yapılabilir

İşaretçi Aritmetiği (devam…)

- Aşağıdaki gibi üç tane gösterici bildirilmiş olsun:

```
char   *kar;→   10000 (0x2710)
int    *tam;→   20000 (0x4e20)
double *ger;→   30000 (0x7530)
```

- Buna göre aşağıdaki atama işlemlerinin sonucu ne olmalıdır?
  - kar++;
  - tam++;
  - ger++;

İşaretçi Aritmetiği (devam…)

- Bir işaretçiye ekleme yapıldığında, o anda tuttuğu adres ile eklenen sayı **doğrudan toplanmaz**.
- Böyle olsaydı, bu atamaların sonuçları sırasıyla 10001, 20001 ve 30001 olurdu.
- Gerçekte, işaretçiye bir eklemek, işaretçinin gösterdiği yerdeki veriden hemen sonraki verinin adresini hesaplamaktır
- Buna göre atama işlemlerinin sonucu:
  - kar++; → 10001 (0x2711)
  - tam++; → 20004 (0x4e24)
  - ger++; → 30008 (0x7538)

İşaretçi Aritmetiği (devam…)

- Genel olarak, bir işaretçiye n sayısını eklemek veya çıkarmak, *bellekte gösterdiği veriden sonra veya önce gelen n. elemanın adresini hesaplamaktır.*

- Buna göre aşağıdaki atamalar şöyle yorumlanır.

```
kar++;          /* kar = kar +   sizeof(char)*/
tam = tam + 5;/* tam = tam + 5*sizeof(int)*/
ger = ger - 3; /*ger = ger - 3*sizeof(double)*/
```

Örnek: İşaretçi Aritmetiği


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

int main()
{
    char   *pk, k = 'a';
    int    *pt, t = 22;
    double *pg, g = 5.5;

    pk = &k;
    pt = &t;
    pg = &g;

    printf("Onceki  adresler: pk= %p  pt= %p   pg= %p \n", pk, pt, pg);
    printf("Onceki degerler: *pk= %c  *pt= %d   *pg= %f \n\n", *pk, *pt, *pg);

    pk++;
    pt--;
    pg = pg + 10;

    printf("Sonraki adresler: pk= %p  pt= %p   pg= %p \n", pk, pt, pg);
    printf("Sonraki degerler: *pk= %c  *pt= %d   *pg= %f \n\n", *pk, *pt, *pg);

    return 0;
}
```

```
Onceki  adresler: pk= 0028FF03  pt= 0028FEFC   pg= 0028FEF0
Onceki degerler: *pk= a  *pt= 22   *pg= 5.500000

Sonraki adresler: pk= 0028FF04  pt= 0028FEF8   pg= 0028FF40
Sonraki degerler: *pk= @  *pt= 2686864   *pg= -0.000000
```

KAYNAKLAR

- N. Ercil Çağıltay ve ark., C DERSİ PROGRAMLAMAYA GİRİŞ, Ada Matbaacılık, ANKARA; 2009.
- Milli Eğitim Bakanlığı "Programlamaya Giriş ve Algoritmalar Ders Notları", 2007
- C Programlama Dili, Şerafettin ARIKAN
- Problem Solving and Program Design in C, Hanly, Koffman
- http://www.AlgoritmaveProgramlama.com

Algoritma ve Programlama

İYİ ÇALIŞMALAR…