

İçindekiler:

Önsöz.....	5
Lecture-1.....	7
Lecture-2.....	13
Lecture-3.....	24
Lecture-4.....	33
Lecture-5.....	48
Lecture-6.....	60
Lecture-7 (General exercises).....	65
Lecture-8 (General exercises).....	72
Lecture-9 (Function exercises).....	76
Lecture-10 -P-.....	84
Lecture-11 (Pointer exercise).....	91
Lecture-12.....	101
Lecture-13 (valuable exercise of structs).....	111
Lecture-14.....	115
Lecture-15 (File Dynamics).....	127
Lecture-16 (File exercise).....	138
Lecture-17 – Dinamik Bellek Yönetimi.....	146
Lecture-18 (DBY examples).....	153
Lecture-19 – SEARCH and SORT Algorithms.....	168

ÖNSÖZ

Açıkçası, böyle bir şey üretme niyetim yoktu. Sadece Notion içerisinde notlar çıkardığım için belli bir düzende kayıtlı duran bir dijital defter gibiydi. İlk prototipin önsözünü kendim yüklenmek istedim o yüzden bu satırları yazıyorum. C programlama dilinin kendi açımdan biraz daha samimi kendim için aldığım notlarını kitaplaştırip kendi çalışmalarımda kullanmak ve ilk defa bir kitapçık tecrübesi edinmek için bu işe kalkıştım (Elimin altında kitap gibi durması kullanım açısından daha pratik geldi.).

Çok fazla yazım hatası, gereğinden fazla samimi ifadeler içermektedir. Belirttiğim gibi kendim için aldığım notlardı. Biri gelir isterde çoğaltılr diye altını çize çize belirtmek istedim. Sonuçta bu prototip bir basım, illaki ilerideki basımlarda önsöz ve içeriklerde değişiklik olacaktır.

Neyse tadını çıkartarak biraz da abartarak güzel bir çalışma oldu diye düşünüyorum. Bir istek oluşursa bundan herhangi bir kâr amacı gütme niyetinde filan da değilim. Bu yüzden okuyucunun çok büyük bir bekenti içerisinde girmemesini isterim.

Örnekleri ve içerikleriyle Can Boz'a teşekkürlerimi sunuyorum. Genel aksan ve görüntüler onun içeriğinden yararlanarak oluşturuldu.

Özellikle bu kitapçıyı oluşturmam için baskında bulunan arkadaşlarım Burak Doğan, Aleyna Demir, Seda İrem Kibrit'e teşekkür ederim. Böyle bir çalışma ortaya çıktıysa onların katkısı inkâr edilemez.

[MADE BY...]

Talha AYDIN

Lecture 1

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

→ C dilinin en güzel tarafı pişmiş ve tecrübelerle dolu bir dil olmas, bağımsız ve aynı zamanda taşınabilir olması bu bizim çalışmalarımız için muazzam bir fırsattır.

Bu noktada C hem derslerimiz hemde projelerimiz için çalışması çok iyidir.

Pythonu etkili bir kütüphane derleyicisi ve spesifik çalışmalar için birebir bir çağrıma merkezi görevi alırken genel hattı C üzerinde kurmak hem hız hem alan çalışması anlamda ve taşınabilirliğinde çok etkili bir hale gelecektir.

Genel bakışımız ve fikrimiz budur.

başlayalım.

Genel olarak ilk başta ekrana yazdırma kodunu bi inceleyelim.

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

int main() {

    printf("goodbye world");

    return 0;

}

çıktı>>
goodbye world
```

→ ilk yazılan include (dahil etmek) ile içeri aktarılan main iki kütüphanedir. Bunlar C nin ana iki kütüphanesidir ve olmazsa olmazlardır.

→ intmain() olayı belirttiğimiz bir alan bir bölgedir. Bu fonksiyon içine yazılan (köşeli parantezler ile nereyi bu main içinde çalıştıracağını belirtmiştık.) kısmını çalıştırır.

→ printf = python'daki print

→ return 0 komutu bir şeyi etkilemez sadece karşı tarafa hatanın olmadığını ifade eder. adından da anlaşıldığı gibi 0 hata dönüşü. yorum satırına alarakda çalıştırılabilir. yorum satırı **//** şeklinde ifade ediliyor.

⇒ printf kullanımı

printf türkçe karakter tehlikeli. Bu yüzden genelde yazılacaklarında ingilizce yazmak gerekiyor.

!!! birden fazla alt alta printf fonksiyonuda belirtsem run ettiğimde ard arda gelecek şekilde yazdırır. satır atlamasını istersem bunu **\n** ile belirtirim.

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

    printf("goodbye world \n");
    printf("goodbye PC");

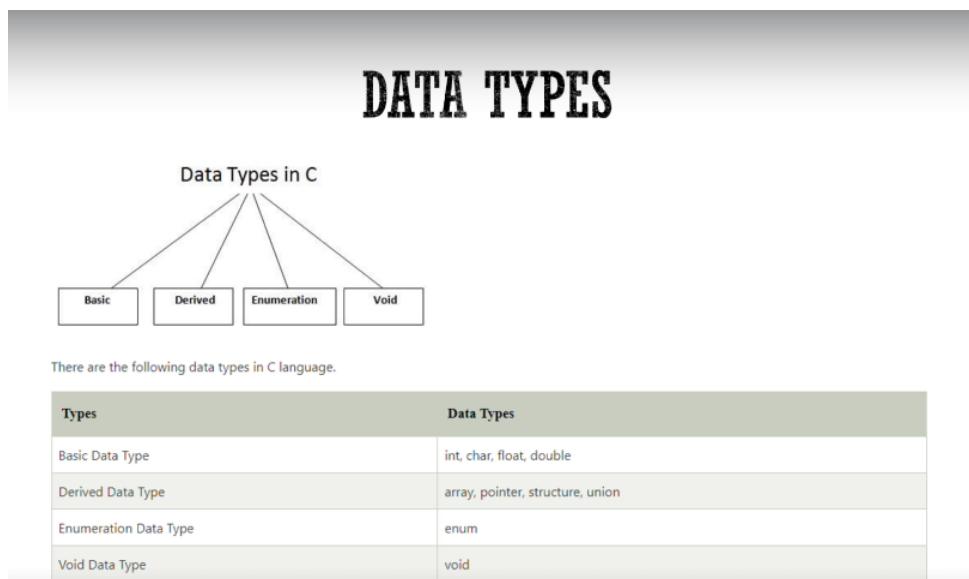
    return 0;
}

çıkıcı>>
goodbye world
goodbye PC

// \n komutu sayesinde ben bi alt satırda atabilirim
istediğimiz değişkeni python'daki gibi bastırabiliriz.
// çift tırnak ve benzeri işaretler bastırmak istersem bunu / tersslaş işaretini ile belirtmeliyim.
(bir kenarda kalsın lazım olur.)
```

!!! birden çok satırı yorum satırı haline getirmek istersem ben bunu /* */ arasına alarak yazarsam bu aralıktaki her şey yorum satırı haline gelir.

Datatypes



int (4 bytes) | %d
double (8 bytes) | %lf
float (4 bytes) | %f
char(1 byte) | %c

→ yandaki yüzdelik ifadeler printf() ile ekrana yazdırma için kullandığımız parametrelerdir.

!!! python'daki gibi direkt değişkeni verip yazdırma burada yapılamaz.

burada bir yazdırma için ürün yazdırma komutunu belirtmeli ve sonra değişkeni yazdırımıya geçmeliyim.

→ *int türünde printf kullanımı*

```

#include<stdio.h>
#include<stdlib.h>
/*
    int    | %d
    float  | %f
    double | %lf
    char   | %c
 */

int main() {
    int deger1,deger2 ;
    deger1=30 ;
    deger2=50 ;
    printf("%d\n%d",deger1,deger2);
    printf("\n goodbye world");
    return 0;

}
çıktı>>>

30
50
goodbye world

```

phyton ile yazım konusunda hafif benzerlikler olsada C nin manuel hareket kabiliyeti gerçekten çok iyi.

,,,,,,,,,,

→ *Float ve double tipinde printf kullanımı*

```

int main() {

    float deger1 ;
    double deger2 ;
    deger1=4.123455 ;
    deger2=79.4657 ;
    printf("%.2f\n%.3lf",deger1,deger2);
    printf("\nfloat ve double kullanımlarıydı'!'"');

    return 0;

}
çıktı>>>

4.12
79.466
float ve double kullanımlarıydı!

```

!!! bu tip ondalık sayı ifadelerinde virgülüden sonra kaç tane basamak sayısını bastırmak istedigimizi yukarıda gösterdiğim şekilde nokta koyup göstereceği basamak sayısını belirtebiliriz.

.....

sayı tipleriyle birlikte text bir ifade kullanmak istersem. % li belirteçlerle aynı tırnak içerisinde kullanarak yazılabilirim.

görelim,,,

```
int main() {  
  
    float deger1 ;  
    double deger2 ;  
    deger1=4.123455 ;  
    deger2=79.4657 ;  
    printf("float tipinde :::: %.2f \ndouble tipinde :::: %.3lf",deger1,deger2);  
  
    return 0;  
}  
  
çıktı>>  
  
float tipinde :::: 4.12  
double tipinde :::: 79.466
```

.....

→ *char tipinde printf kullanımı [çok değerli]*

::: char içinde tek bir karakter tutulur. 1 karakterden fazla mesela “ca” gibi bir girdi yapıldığında bu karakter dizisi olur.

```
int main() {  
  
    char deger3 ;  
  
    deger3 = 'J' ;  
  
    printf("char degerindedir::: %c ", deger3);  
    return 0;  
}
```

kalın belirttiğimiz yerin sebebini görelim.

```
int main() {  
  
    char deger3 ;  
    deger3 = 'Jale' ;  
  
    printf("char degerindedir::: %c ", deger3);  
  
    return 0;  
}  
  
çıktı>>>  
  
e
```

İşte buda ispatydı.

,,,,,,,,,,

Lecture 2

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

Scarf olayı!

kullanıcıdan bir girdi istememizi ve bunu yazdırmanızı sağlayan komuttur.

Çalışma mantığı belirli bir adrese scarf ile atama yapmakta. Ve printf sağlayarak bunu çalıştırmayı sağlıyoruz.

örnek üzerinde görelim.

```
int main() {  
    int deger1;  
  
    printf("lutfen bir sayı giriniz::::");  
    scanf("%d", &deger1);  
    printf("aldığınız sayı %d", deger1);  
  
    return 0;  
}
```

burada dönen mantık kullanıcıdan klavye girdisini tarayıp (scanleyip) belirlediğimiz değişkenin adresine tanımlıyor. sonrada bu adrese tanıtılan değişkeni yeniden printf komutu ile yazdırıyor.

Scarf bu şekilde çalışıyor kısaca. Diğer type türlerindede uygun tanıma ve belirlemeyle ekrana yazdırılabilir.

.....

kapsayıcı bir örnek üzerinde birkaç noktadan bahsedeceğim.

```
int main() {  
  
    int deger1;  
    float deger2;  
    char deger3;  
  
    printf("lutfen bir karakter giriniz::::");  
    scanf(" %c", &deger3);  
  
    printf("lutfen bir sayı giriniz::::");  
    scanf("%d", &deger1);  
  
    printf("lutfen ondalik bir sayı giriniz::::");  
    scanf("%f", &deger2);  
  
    printf("\n%c aldığınız karakterdi ,,, %d aldığınız sayı idi ,,, %f aldığınız float tipti...", deger3,deger1,deger2);  
  
    return 0;  
}
```

→ Buradaki kodun doğru çalışması için yapılan bazı değişiklikler vardır.

!!!

Kritik nokta Char tipinin scanf ile kullanılmasına dair;;; char tipinde ifadeyi çağrıırken tırnak içinde boşlukla belirtirsek veya ilk önce char tipini istersek yaşanılacak problemi çözmiş oluruz. Buna kesinles dikkat edilmesi gerekiyor.

Sizeof

sizeof yapısı değişkenin türünün ne kadar yer kapladığını gösterir.

sizeof int değerindedir ve bildiğimiz gibi printf içinde bunu %d ile birlikte kullanıyoruz.

.....

```
int main() {  
  
    double deger3 ;  
  
    deger3= 1000,6 ;  
  
    printf("belirtilen degisken hafizada %d kadar yer tutar.\n", sizeof(deger3));  
  
  
    return 0;  
}
```

çıkıtı olarak ekrana 8 byte yer tuttuğunun bilgisini verecektir.

.....

değişkenler haricinde direkt type cinsinden verrekde hafızada ne kadar değer bulunduğu söyleyebilirim.

.....

Const ve define

Const yapısı sonradan değiştirilemez değerlerdir. İlk başta değer atandıktan sonra değer değişikliği yapılamaz.

main fonksiyonun içinde kullanmak yerine define ile main fonksiyonun dışında yukarıda belirtmek işleyebilirlik açısından daha etkili bir çözüm gibi duruyor.

.....

```
#include<stdio.h>  
#include<stdlib.h>  
#define T 7.78795
```

```

int main() {

    printf("%.2f\n", T);

    return 0;
}

/* yukarıda tanımladığım define yapısı ile değişkeni sabit bir atama yapabildim.*/
çıktı>>

7.79

virgülden sonra iki sayı belirtsin diye /f önüne .2 ile belirttik.

```

Aritmetik Oparatörler

pythondaki gibi oparatör kullanımı buradada benzerdir.

```

/*
ARİTMETİK OPARATÖRLER

+ , - , * , / , % , ++ , -- ,
*/

```

% → belli bir bölümde kalanı belirtir. (Hatta tek veya çift sorguları bu kalan mantığında tespiti yapılır.)

++ → bir arttırma

--, → bir azaltma

.....

görelim ;;;

```

int main()

{
    int x;
    int y;
    x=20;
    y=49;

    printf("%d\n", x+y);

    return 0;
}

çıktı>>>

69

```

veya sonucu farklı bir değişkene atayıpta yazdırabilirim.

```

int main()
{
    int x;
    int y;
    int sonuc;
    x=20;
    y=49;
    sonuc = x+y;
    printf("%d\n", sonuc);

    return 0;
}

```

direkt diğer operatörler de bu sisteme uygulanabilir.

`++` veya `--` operatörlerinde değişkenden önce konur yani `++x` veya `-x` olarak alınmalı, yoksa işleme almıyor.

Atama Operatörleri

değişkenleri tanımlarken kullandığımız operatörlerdir.

(equal meaning!)

eşitlikte python'dan öğrendiğimiz gibi sağdaki değer soldaki değerin

```

* assigment operator  *

=  > klasik atama
+= > değişkene sağdaki değeri direkt ekler.
-= > değişkene sağdaki değeri çıkararak azaltır.
*= > değişkeni sağdaki değeri çarptırır
/= > değişkeni sağdaki değeri böldürür
%=> değişkenin sağdaki değerin bölümünden kalanı değişkene eşitler.

```

.....

bölümünden kalanı görelim.

```

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

    int deger3 =25 ;

    deger3 %= 20 ;

    printf(" atanan yapılar sırayla,\n %d\n", deger3);

    return 0;
}çıkıcı>>>
atanan yapılar sırayla,
5

```

Comparasion Operators

karşılaştırma operatörleri aynı python'daki gibidir.

(çok bir şey yazmayacağım zaten nasıl kullanıldığını biliyoruz.)

```
=/*!hatırlatma*/ bu eşitse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.  
!=/*!hatırlatma*/ bu eşit değilse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.  
>/*!hatırlatma*/ bu büyükse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.  
</*!hatırlatma*/ bu küçükse eşitse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.  
>=/*!hatırlatma*/ bu büyük eşitse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.  
<=/*!hatırlatma*/ bu küçük eşitse demekti. Özellikle if else sorgularında çok işemize yarayacaktır.
```

if else yapısı kullanmadan bazı mantık sorguları yaptırırken boolean tipinde döndürme yaptığıunu unutmamalı.

formalite icabi...

```
#include<stdio.h>  
#include<stdlib.h>  
int main() {  
  
    int x , y ;  
    x=30;  
    y= 7;  
    x = x/5 ;  
  
  
    printf(" %d\n",x!=y);  
  
    return 0;  
}  
  
çitki>>>  
1
```

Logical operators [Mantıksel operatörler]

ve veya değil gibi yapılar matematikte olduğu gibi buradada var. şu şekildedir.

```
&& -> ve  
|| -> yada  
! -> değil  
  
-----  
klasik ve yada kurallırıdır. diğer dillerdeki ile aynı yapı.
```

printf() içinde tanımladıktan sonra filtreler gibi bu ifadeleri kullanabiliriz.

.....

görelim.

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

    int x ;
    x=30 ;
    printf(" %d\n", x>10 && x<40);
    return 0;
}
çıktı>>
1
/* buraya dikkat! ve sorusuya bir koşulu sağlattığımız için bu koşul doğru olursa bize doğru olduğunu (1
eğer bu koşul yanlış olsaydı yine aynı yapıda (0) döndürecekti.*/
```

değili yapısının kullanımı şöyledir.

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

    int x ;
    x=30 ;
    printf(" %d\n", !(x>10 && x<40));
    return 0;
}
```

...

If Else yapıları

aynı python'daki gibidir.

yinede hatırlatalım...

```
if(condition )
{
    ....kodlar....
}
```

→ condition kısmındaki koşul sağlanırsa yapı true dönecektir ve içindeki kodları çalışmaya alacaktır. ama false dörnerse kodlara uğramadan geçer.

küçük bir örnek görelim syntax için.

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

    int x,y ;

    x=30 ;
    y=40 ;

    if (!(x>y));
        printf("X, Y'den kucuk degildir.");
    return 0;
}

çıktı>>
X, Y'den kucuk degildir.
```

.....
ÇOK KRİTİK!

tek satır bi kod varsa süslü paranteze gerek yok ama birden fazla satır varsa süslü parantez kullanmak şarttır!

.....
→ if şartı sağlanmadığında direkt else kayacaktır.

.....
→ else if python'daki elif yapısıdır. else koymadan önce harici koşulları eklemek için else if yapısı kullanılabilir.
küçük bir örnek;:::

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

    int x,y ;

    x=30 ;
    y=40 ;

    if (x>y){

        printf("X, Y'den kucuk degildir.");
    }

    else if(x==y) {
        printf("X Y'ye eşittir.");
    }
}
```

```

    }

    else if(y>x) {

        printf("X Y'ye den kucuktur.");
    }

    return 0;
}

çıktı>>
X Y'ye den kucuktur.

```

sondada else yapısıyla biterebiliriz.

""

kullanıcıdan isteyecek bir örnek deneyelim.

```

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

    int x,y ;

    printf("litfen bir deger giriniz:::\n");
    scanf("%d", &x);
    printf("litfen bir deger giriniz:::\n");
    scanf("%d", &y);

    if (x>y){

        printf("X, Y'den kucuk degildir.");
    }

    else if(x==y) {
        printf("X Y'ye esittir.");
    }

    else {

        printf("X Y'ye den kucuktur.");
    }

    return 0;
}

```

!!! çok önemli

scanf yaparken '&' şu ifadeyi unutmamak gereklidir!

if else yapılarına örnekler

→ üç sayı girdisi isteyerek if yapısı kullanarak bize en büyük olanını buldurun bir senaryo düşünelim.

```

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

    int x,y,z ;

    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &x);
    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &y);
    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &z);

    if (x>y && x>z){
        printf("%d . en büyük sayıdır.", x);
    }
    if (y>x && y>z){
        printf("%d en büyük sayıdır..", y);
    }
    if (z>x && z>y){
        printf("%d en büyük sayıdır.", z);
    }
    if (x==y && x==z){
        printf("%d, %d, %d sayıları birbirine eşittir.", x,y,z);
    }

    return 0;
}

/*

```

bize şu fikri veriyor::: if yapısını ayrı ayrı şekilde birden çok [farklı] koşul yargılarda kullanırız
 birbirlerinden farklı ayrı yapılar oldukları çok önemli.
 else if yapılarında böyle değildir. açılan if koşul partinin içinde yer alır.

.....
 bir de else if ,,, else yapılarını kullanarak görelim.

```

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

    int x,y,z ;

    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &x);
    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &y);
    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &z);

    if (x>y && x>z){


```

```

printf("%d . en büyük sayıdır.", x);

}
else if (y>x && y>z){

printf("%d en büyük sayıdır..", y);

}

else if (z>x && z>y){

printf("%d en büyük sayıdır.", z);

}

else{

printf("%d, %d, %d sayıları birbirine eşittir.", x,y,z);

}

return 0;
}

```

→ else harici bir koşul ile birlikte yer almaz. zaten if ve else if sağlanmadığı durumda yapmasını istediğimiz şeyi söyleriz, bir koşul ile ifade etmek mantıksızdır zaten.

.....

bir sayının tek mi çift mi olduğunu buldurun bir senaryo geliştirelim.

```

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

    int x;
    printf("lütfen bir sayı giriniz:::\n");
    scanf("%d", &x);

    if (x%2==0){

        printf("aldığınız sayı bir çift sayıdır. ");

    }

    else{

        printf("aldığınız sayı bir tek sayıdır.");
    }

    return 0;
}

```

.....

Kullanıcıdan aldığımız bir senaryo deneyelim.

```

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

int main() {
    int x,y;

```

```
printf("lutfen 2 sayi giriniz:::\n");
scanf("%d %d", &x,&y);

if (x>y){
    printf("%d buyuktur %d den", x,y);
}

else if (y>x) {
    printf("%d kucuktur %d den", x,y);
}

else {

    printf ("sayilar birbirine esittir.");

    return 0;
}
```

→ bir tane scanf içinde iki değeride kullanıcıdan isteyebiliriz. printf ile benzer bir kullanıma sahiptir diyebiliriz.

Lecture 3

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

İç içe if kullanımı

binevi filtreleme şeklinde çalışır.

başlangıçtaki if koşulunun sağlandığı ve daha da sıpesifik bir sorguya geçindiği zaman kapsanan bir if koşulunu daha oluşturmaya iç içe if kullanımını diyebiliriz.

görelim.

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

int main() {
    int x,y;

    printf("lutfen 2 say giriniz::::\n");
    scanf("%d %d", &x,&y);

    if (x >= y){
        if(x == y) {
            printf("%d esittir %d den", x,y);
        }
        else {
            printf("%d buyuktur %d den", x,y);
        }
    }
    else {
        printf("%d kucuktur %d den", x,y);
    }
    return 0;
}
```

!!!yukarıda göründüğü gibi ilk if koşulunun kapsadığı alanda daha ayrıntılı bir şeyi sorgulamak için iç içe if yapısını(buna else if ve else yapıları dahildir.) kullanma imkanım var.

Döngüler!

pythondan aşınalığımız var.

For döngüsü

→ for (initialization Statement ; test Expression ; update Statement)

initialization Statement ::> başlangıç değer ataması

test Expression ::> doğru olduğu sürece devam et koşulu

update Statement ::> yeni ve güncel değerin hazır süreci (burada ne yapmak istediğimizi belirtir ve test expression True döndürdüğü sürece bu loop devam eder.)

/* buradaki for açıklaması çok güzelmiş bulundurmak istedim. */

kısaca doğruluğu devam ettiği sürece loop alınacak ve bu sürekli olacaktır diyebiliriz.

görelim.

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

int main() {
    int x;
    for(x=1; x<=5; x++)
    {
        printf("%d\n", x);
    }
    return 0;
}

çıktı>>>
1
2
3
4
5
//    ++ ifadesi soldaki değeri bir arttırıyordu.
```

→ loop sürdüğü sürece her değer işleminde her adımda verilen ifadeyi yerine getirecektir. taki test expression False döndürenе kadar.(başlangıç değeride süreçte alınacaktır.)

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

int main() {
    int x,y,toplam;
    toplam = 0 ;
    printf("lutfen bir deger giriniz:::\n");
    scanf("%d", &x);

    for(y=1; x>=y ; y++ )

    {
        (toplam = toplam + y);
    }

    printf("%d", toplam);

}
```

burada dikkat edeceklerimiz çok nokta var.

- *değişken ekleyebiliriz ve özellikle başlangıçta o değişkenin kaça eşit olduğunu söylemeye unutmayalım.*
- *değişkeni önceki durumuna toplayabiliriz (aynı pythondaki gibi ($x=x+1$ vs.. gibi))*
- *testexpression önemli. mantıklı ve mantık kurallarına uygun olmalı. mantık hatalı bir yapı girildiğinde direkt for döngüsü içine girmiyor.*

teklik çiftlik bastırma üzerine;;

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

int main() {
    int x;

    for(x=1 ; x<=10 ; x = x+1){
        if (x % 2 == 1){

            printf("%d tek sayidir.\n", x);

        }
        else{

            printf ("%d cift sayidir.\n",x);
        }
    }

}

çıktı>>

1 tek sayidir.
2 cift sayidir.
3 tek sayidir.
4 cift sayidir.
5 tek sayidir.
6 cift sayidir.
7 tek sayidir.
8 cift sayidir.
9 tek sayidir.
10 cift sayidir.
```

bu for içinde if kullanımına bir örnekti.

ama ben sadece tek veya sadece çift sayı yazdırırmak istesem direkt for döngüsünün ayarlarında bir oynama yapmam gerekiyor demektir.>>>

tek sayı yazdırırmak için.

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

int main() {
    int x;

    for(x=1 ; x<=20 ; x +=2){

        printf("%d\n",x);
    }

}
```

```

}

printf("bunlar belirtilen araliktaki tek sayilardir.");
```

}

çıktı>>

1
3
5
7
9
11
13
15
17
19

bunlar belirtilen araliktaki tek sayilardir.

çift bir varyasyonu yapmak istesem bu sefer başlanıç değerini çift yapmam yeterli olacaktır.

→ burada *x* in artış değerini kafamıza göre arttırıp azaltabiliriz. Bunu illaki bi şeyleri yazdırırmak veya buldurmak anlamında yapmamıza gerek yok.

kaç kere loopa sokmak istediğimizi veya bu loop noktasında harici eklemeleri ve kullanımlarıda yapabilirim.

While döngüsü

→ while loopu için belirttiğimiz koşul doğru olduğu müddetçe while çalışacaktır.Python'daki gibi. (kullanım syntaxı if ve for ile aynı.)

```

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

int main() {
    int x;
    x=0;
    while(x<10){

        printf("%d\n",x);
        x += 2;

    }

    return 0;
}

// aritmetik ifadelere dikkat. doğru yazmak gerekiyor.
```

kullanıcıdan girilen değeri belirlediğimiz yere kadar çarpımlarını alan ve bunları yazdırın bir senaryo deniyelim.

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

int main() {
    int x,y=1,z=10;
    printf("bir sayı giriniz:::\n");
    scanf("%d", &x);

    while(z<=x*10){

        z = x*y;
        y += 1;
        printf("%d\n", z);

    }

    return 0;
}

çıktı>>
bir sayı giriniz:::
5
5
10
15
20
25
30
35
40
45
50
```

burada şunları anlıyoruz.

- değer atamak noktasında değişken atamaktan çekinmeyelim. ifadeyi farklı değişkenlerle abartmadığımız sürece sade bir kullanım oluşturabiliriz.
- başlangıç ataması atanın değer çok önemli olmasada atamak önemli. ister kullanıcı atasın ister sisteme dahil olsun yinede bu nokta önemli.

yukarıdaki örneği daha basitleştirebilriiz..

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

int main() {
    int x,y=1;
    printf("bir sayı giriniz:::\n");
    scanf("%d", &x);

    while(y<=10){

        printf("%d\n", x*y );
        y ++ ;

    }

}
```

```
    }

    return 0;
}
```

!!! matematiksel işlemi başka bir ifadeye atamak yerine direkt printf içerisinde bunu yazmak ve sonucu yazdırın daha pratik ve basit bir yöntem olabilir.

for döngüsü ile yaptığımız bir örneği while ilede tekrar edelim. (bu çift sayılar için ;; tek sayılar içinde 1 ile başlatmamız yeterli olacaktır.// benzer bir problemi if kullanarakta çözümleyebilirdik.)

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

int main() {
    int y=0;

    while(y<=10){

        printf("%d\n", y);

        y += 2;

    }

    return 0;
}
```

.....

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

int main() {
    int y=1;

    while (1==1){

        printf("%d\n", y);

    }

    return 0;
}
```

python'daki gibi ben döngüyü sürekli doğru olacak bir koşul verirsem, bu döngü sonsuza kadar devam edecektir. Bu farklı atak yöntemleri veya benzeri loop processleri için kullanılabilir.

Do while Döngüsü

→ [yap ve koşula bak] ifadesi diyebiliriz.

yani normal whiledan farkı ilk **bir çalışma** sonrası while döngüsüne bakmasıdır. önce ilk koşulu yerine getirir ve sonrasında döngüye göre hareket eder.

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

int main() {
    int y=0;

    do {
        printf("%d\n", y );
        y++;
    }while (y<=10);

    return 0;
}

çıktı>>>
0
1
2
3
4
5
6
7
8
9
10
```

DİKKAT! syntaxı normal while ifadesinden biraz farklı. Çıktıda da görüldüğü gibi ilk önce (do) kısmındaki ifadeyi bir kez yapıp while döngüsüne girdi.

ve sonrasında yukarıdaki ifadeyi döngü çerçevesinde gerçekleştirdi.

ve belli bi koşullamada yapabiliyim.

kullanıcıdan koşa bakmadan değer alabilirim, veya döngüyü çalıştırması için belli bi değerden sonra koşulunu sağlatabilirim. vs...

Break ve Continue

python'daki gibi break döngüden çıkıştırma.

Continue atlama diğer şekilde devam etme sağlar.

döngü içerisinde istemediğim bir sonucu geçmemde veya daha spesifik bir ifadeyi if ve continue yardımıyla geçebilir veya değiştirebilirim.

ikisinin bir arada olduğu bir kullanalım görelim.

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

    int x;
    x=1 ;

    for(x=1 ; x<=16 ; x +=1){
```

```

        if(x==8){
            continue;
        }

        if (x==10){

            break;
        }
        printf("%d\n",x);
    }

    return 0;
}

çıktı>>>
1
2
3
4
5
6
7
9
// kısaca continue o değere geldiği zaman direkt döngüde bir sonraki değere, break kullanıldığında direkt döngüyü sonlandırır.

```

Switch - Case yapısı

→ if, else if , else yapısının daha spesifik ve pratik bir kullanım şekli diyebiliriz.
daha pratik ve değişken odaklı bir yapıdır.

görelim.

```

int x =1 ;

switch(x){

    case 1:printf("x=1 olursa bu");
    break;
    case 2:printf("x=2 olursa bu");
    break;
    case 3:printf("x=3 olursa bu");
    break;
    case 4:printf("x=4 olursa bu");
    break;
    case 5:printf("x=5 olursa bu");
    break;
    case 6:printf("x=6 olursa bu");
    break;
    case 7:printf("x=7 olursa bu");
    break;
    default :printf("yukarıdaki koşullar sağlanmazsa bunu yap");
    break;

}

```

Kısaca değişkenin aldığı sonuca göre switchlerde farklı durumlar üretilebilir.

dikkat edilmesi gerekenler;;

- Break ile birlikte kullanım
- syntaxe dikkat (aynı döngü ve if lerdekdi gibi bi kullanım var.)

(! switch gerçekten kullanması ve buna yönelik sistem geliştirmesi çok pratik bir yapı.)

ÖNEMLİ;;; scanf de iki tane veya daha fazla değer alırsam. o kadar değer girip enterlamam lazımdır. sadece bir enterla aynı satırda işlem yapılamaz (mantığına aykırı bir kere!)

.....

→ switch kullanarak dört işlem üzerine odaklanalım.

(oparatörlerde bir char değeridir.)

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    float x,y ;
    char op ;
    printf("lutfen bir operator giriniz::: ");
    scanf("%c", &op);

    printf("lutfen 2 sayı giriniz::: ");
    scanf("%f%f", &x,&y);

    switch(op){

        // karakter kullanımı yaptığımız için tek tırnakla ('') ifad etmeyi unutma!!!!
        case '+': printf (" %.1f + %.1f = %.1f", x,y,x+y);
        break;
        case '*': printf (" %.1f * %.1f = %.1f", x,y,x*y);
        break;
        case '/': printf (" %.1f / %.1f = %.1f", x,y,x/y);
        break;
        case '-': printf (" %.1f - %.1f = %.1f", x,y,x-y);
        break;
        default : printf ("gecersiz oparator girişi");
        break;
    }
}
```

bu şekilde switch case kullanarak kişiden istediği değerde 4 işlemde çalışmasını sağlayabildik.

→ yeniden hatırlatma scanf yapısı hassas bir yapı kullanımına dikkat edilmeli.

→ eğer biz bir integer ve float toplamı vs deneseydik sonuç yuvarlanabilirdi. bunu önlemek içinde printf içindeki aritmetik değişkene hangi türde olduğunu (float), (int) tarzında belirtmemiz gerekebilirdi.

Lecture 4

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

Boolean tipi

python'daki gibi boolean tipi True ve False döndüren değerlerdir.

Bu tip C içindeki boolean kütüphanesinin çağırılması ile kullanılır.

ve bunları integer bir değer olarak çağrılmak istediğimizde bildiğimiz gibi 1 ve 0 değerlerinde döndürülecektir.

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

int main(){
    bool x = false;
    printf("%d",x);
    return 0;
}

çıktı>>
0

// buradaki eklenen kütüphaneye dikkat. ve ayrıca normal tiplerde olduğu gibi boolean
// tipindede int, float gibi tip ataması yapıyoruz!!!
```

Math kütüphanesi yardımcı fonksiyonlar.

math.h include edildikten sonra çalışan çok işe yarayan genel matematik fonksiyonlardır.

birçok farklı yardımcı fonksiyonlar bulunmaktadır.

- ceil (tavan değerine yuvarlama (float değerler için))
 - floor (taban değerine yuvarlama (float değerler için))
 - sqrt (sayıların karekökünü alır.)
 - pow -power- (sayıların üssünü almamızı sağlar. (syntaxe dikkat))
 - abs -absolute- (mutlak değer almamızı sağlar.)
- abs** ye has bi nokta integer bir sayı değeri olmak durumundadır.

syntaxları görelim.

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

int main() {
    printf("%f\n",ceil(4.2));
```

```

printf("%f\n", floor(4.9));
printf("%f\n", sqrt(64));
printf("%f\n", pow(3,4));
printf("%d\n", abs(67));

return 0;
}

çıktı>>
5.000000
4.000000
8.000000
81.000000
67
// abs (mutlak değer yapısı kullanırken integer formatında kullandığımızı unutmamalı. float tipinde bir kullanım yaparsak 0 sonucunu //geri kalanlar zaten yukarıda belirttiğim şekildeki özelliklerdir.

```

!!!!

Casting

→ bir sayıyı diğerine böldüğümüzde sonuç integer formatında geliyor. Mesela 5 i 2 ye böldüğümüzde sonuç 2.5 yerine 2 olarak geliyor (yukarıda bir divide örneği var içinde denenebilir.) işte burda çözüme casting yapısı yardım ediyor.

::: formatı integer tipi yerine floatta alsakta problemin sonucunu direkt bize vermiyor. çünkü benim bölme işleminde belirttiğim sayılar integer tipindedir. Doğal olaraka sistem sonucun integer formatında döndüreceğini düşünüyor. Benim bu şekilde istemediğimi sisteme söylemem (kastetmem) lazımdır.

casting methodu bu noktada yaptığımız işlemin sonucunun float tarzında olduğunu haber etmemiz gerekiyor şeklinde anlayabiliriz.

.....

görelim.

```

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

int main() {
    float sayi=(float)13/2;
    printf("%.1f",sayi);
    return 0;
}
// matematiksel ifadedenin sonucunu casting tarzında sonucun float olduğunu hatırlatmamızı sağlar.

```

ASCI Cods

Karakterleri ifade etmek için aski kodlarını kullanırız. 0 ve 255 arasında bir değer alır.

bit yapısı ve bilginin 0 ve 1 ile ifade ediliş şekli (hafızadas yer kaplaması) bu aski ifadeleri ile sağlanır.

küçük bir alıştırma yapabiliriz.

İfadelerin ASCII kodlarını yazdıralım.

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

int main() {
    printf("%d\t %d\t %d\n", 'a', 'B' , 'A' );
    return 0;
}

çıktı>>
97      66      65

// printf içinde karakterleri tek tırnakla ifade ettim ve ASCII karşılıklarını direkt olarak integer türünde bana yazdırdı.
```

for döngüsü kullanarak alfabetin ASCII kodlarını yazdırabiliriz.

Yukarıdaki genel ifadenin zıt şeklini for döngüsü içinde kullanabiliriz;;;

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

int main() {
    int i;
    for(i='a'; i<='z' ; i += 1){
        printf("%c\t", i);
    }
    return 0;
}

çıktı>>
a      b      c      d      e      f      g      h      i      j      k      l      m      n      o
p      q      r      s      t      u      v      w      x      y      z
```

→ Gayet incelenmesi gereken ve bize ASCII değerlerinin farklı türlerde karşılığını ifade eden güzel bir örnek.

DİZİLER

- Aynı tipten oluşan elemanların tek bir isimde ifade edilmesidir.
→ Örneğin ::: int dizi[10]

Tüm dizilerde bulunan ortak özellikler ;;

- Eleman tipi (int, float, char vs. belirtilir.)
- Dizinin ismi (boyutu belirtmeden önce kullanılır.)
- Dizinin boyutu [] içinde kullanılır. <bu boyut yapısı kac tane elemanın bulunduğu tanımlar.>

Hafıza boyutu n^* s hesapıyla yapılır. s değeri 2 byte'dır ve her eleman dizi içinde 2 byte ile ifade edilir. Dizi boyutuyla çarpıldığında zaten toplam byte vermiş oluyor.

!!!

dizide herhangi bir elemanı bastırabilmemiz için o elemanın dizideki yerini (pozisyonunu) belirtmemiz gereklidir.

Dizinin ilk elemanını bastırmanız için dizi[0] şeklinde belirtmek gereklidir.

=::python'daki index usulü gibi::=

→ **Tek boyutlu Diziler**

sayı dizileridir.

bi örnek görelim.

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

int main(){

    int dizi[10]={1, 4, 6, 78, 18, 34, 23, 67, 56, 98,};

    printf("%d", dizi[7]);


    return 0;
}

çıtkı>>
67
// 0 dan saymaya başlar ve aynı python'daki gibi index usulü şeklinde çağrıma yapılır. Dizi tanımlamasına dikkat !
// kafamızı takılan soruya gayet güzel bir yanıt oluşturdu.

// negatif şekilde index verme python'da varken burada yok. Onada dikkat.
```

⇒ ÖNEMLİ BİR ÖRNEK

- diziye ilk 10 çift sayıyı dahil eeden bir sistem senaryolayalım.

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

int main(){

    int a,i,dizi[10];

    // for döngüsü içinde başka bir değişken kullandığım anda yukarıda tanımlamasını vermem gereklidir unutulmamalı.
    for(i=0; i<10 ; i++){

        dizi[i]= i*2;
    }

    for(a=0 ; a<10 ; a++){

        printf("%d\n",dizi[a]);
    }

    return 0;
}
// indexi 0 dan olduğunu unutup 10 tane elemanı olan bir diziye [10] şeklinde belirtirsek program hata verecektir.
```

→ Dikkatimizi çekmesi gereken nokta dizi içeresine dahil ediş şekli. Dizinin belirtilen elemanını belirtilen değere eşitleyerek atama yapabiliyoruz!

haricinde dizi içine eleman atamak yerine aynı şekilde var olan elemanıda değiştirebiliriz.

bir örnek daha

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

int main(){

    int a;
    int dizi[3];

    // for döngüsü içinde başka bir değişken kullandığım anda yukarıda tanımlamasını vermem gereklidir unutulmamalı.
    for(int i=0; i<3 ; i++){

        scanf("%d", &dizi[i]);
    }

    printf("%d\t%d\t%d\t", dizi[0],dizi[1],dizi[2]);
    return 0;
}

//SCANF İÇİNDE BİR DAHA \n GİBİ BİR İFADE GİRME!
// her bir dizi ifadesi bir değişken özelliğini taşır. kısaca biz bunu direkt scanf girdisinde bulundurup işlem yaptırabiliriz.!!!
```

do while ile aynı örneği görelim.

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

int main(){

    int dizi[3];
    int i=0;

    // for döngüsü içinde başka bir değişken kullandığım anda yukarıda tanımlamasını vermem gereklidir unutulmamalı.
    do{

        scanf("%d", &dizi[i]);
        i += 1 ;
    }while(i<3);

    i=0;
    do{
        printf("%d\t",dizi[i]);
        i++;
    }while(i<3);

    return 0;
}
/* herhangi bir değer girmedigimizde dizinin boş şubelerinin içine 0 atıyor.*/
```

→ **İki Boyutlu Diziler**

- Matris oluşturmak için kullanılır.
- kodların hafızada tutulan kısmı doğrusal olmasına karşın çok boyutlu diziler doğrusal değildir.

o halde çok boyutlu dizileri doğrusal hale getirmek için dizinin dizisi haline getirmek gerekir.
(python'daki key, value ilişkisine benzer şekilde::; elimizdeki bir dizinin elemanları dizi formatında olabilir.)

yani M dizisinin 4 elemanı olduğunu varsayırsak ve 4. elemanın 5 elemanı olduğunu düşünelim.

M[3][4] dersm yukarıdaki ifadedeki 5. elemana karşılık gelen değeri yazdırımızı sağlamış olacaktır. (0 dan başlama kuralına dikkat!)

- Çok boyutlu dizi tanımlarken sonsuz uzunlukta bir matriste tanımlayabiliriz.
 - intmarice[][], ilk köşeli parantez ilk dizinin elemanlarının sayısını belirtirken, ikinici köşeli parantezde içteki liste şeklindeki elemanların sayısını belirtir. Bildiğimiz gibi direkt eleman sayısına eşitse direkt diziyi tanımlarken, index şeklinde bir kullanım varsa o zamanda belirtilen sayıyı veya değeri çağırma için kullanılır.

⇒ mateis matematikte küçük tablolar tarzı ifadelerdir diyebiliriz ve iki boyutlu dizilerde biz bunları satır ve sütün gibi ifade sağlıyoruz diyebiliriz.

m'ye n'lik bir dizey

$$a_{i,j} \xrightarrow{i \text{ değişir}} \downarrow j \text{ değişir} \left[\begin{array}{cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{array} \right] \xrightarrow{m \times n}$$

İşte bunu içe içe diziler şeklinde ifade edebiliyoruz.

İşlerin nasıl çalayıtığını görmek için küçük bir örnek;;

```
int main(){
    int dizi[3][3]={{65,6,7}, {13,23,44}, {24,25,29}};
    printf("%d", dizi[0][1]);
    return 0;
}
çıktı>>>
6
```

:::::: çok iyi bir örnek;;

```
int main(){
    int dizix[10][9];
    int i,j ;
    for(i=0; i<10; i +=1 ){
        for(j=0;j<9 ;j +=1){
            if(i==j){
```

```

        diziX[i][j]=1;
    }
    else{
        diziX[i][j]=0;
    }
}

for(i=0; i<10; i +=1 ){

    for(j=0;j<9 ;j +=1){

        printf("%d\t", diziX[i][j]);
    }
    printf("\n");
}

return 0;
}
/* buradaki birkaç noktaya dikkat
--> for döngüsünü iç içe kullanma sebebişimiz aslında indexlere değer ataması sağlayabilmek
içiçe iki tane dizi olduğu için iç içe iki for döngüsü oluşturduk.

--> diğer nokta ise printf lerin kullanımı. içteki for döngüsü sonrası biz printf kullandığımızda içerisindeki değerleri yanyana index s
Bu durumda direkt matris halini oluşturmuş olacağız.*/
çıktı>>>
1      0      0      0      0      0      0      0      0
0      1      0      0      0      0      0      0      0
0      0      1      0      0      0      0      0      0
0      0      0      1      0      0      0      0      0
0      0      0      0      1      0      0      0      0
0      0      0      0      0      1      0      0      0
0      0      0      0      0      0      1      0      0
0      0      0      0      0      0      0      1      0
0      0      0      0      0      0      0      0      1
0      0      0      0      0      0      0      0      0

```



buradaki birkaç noktaya dikkat

--> for döngüsünü iç içe kullanma sebebişimiz aslında indexlere değer ataması sağlayabilmek
içiçe iki tane dizi olduğu için iç içe iki for döngüsü oluşturduk.

--> diğer nokta ise printf lerin kullanımı. içteki for döngüsü sonrası biz printf kullandığımızda içerisindeki değerleri yanyana index
sayısına yazacak ve aşağıya büyük indexteki sayı kadar geçirecek.

Bu durumda direkt matris halini oluşturmuş olacağız.

3 Boyutlu sayı dizileri

2 boyutlu sayı dizilerinden (yani matrislerden) daha fazla sayıda içeren dizilere 3 boyutlu diziler diyoruz.

{genel olarak üniversitelerde gösterilmiyor ama önemli bir konu!}

syntax şekli:;;;

```

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

int main(){

    int dizi1[2][3][4]={{ {34,45,78,55}, {23,85,90,88}, {12,45,32,90} }, {{5,89,96,65}, {31,32,47,86}, {98,67,14,19}}};

    printf("%d\n", dizi1[1][0][1]);

    /* diğer dizi kurallarını uygun şekilde bir tanımlama fark bu sefer matrisleri indexliyor olmamız oluyor.
    (matrisleri aynı dizi hafızasında tutuyoruz kısaca) */

```

```
    return 0;
}

çıktı>>
89
```

biraz daha görsel şekilde anlayalım.

```
matris-1
#####
#      34 45 78 55      #
#      23 85 90 88      #
#      12 45 32 90      #
#####
#----->
#----->

matris-2
#####
#      5 89 96 65      #
#      31 32 47 86      #
#      98 67 14 19      #
#####
#----->
```

Karakter dizileri

→ char ile tanımlama yaptığımız ve %s (string cinsinde ifade ettiğimiz dizilerdir.)

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

int main(){

    char dizi1[]={'a', 'd', 'n', 'a', 'n', '\0'};
    /* ÇOK ÖNEMLİ!!! bu tip string tiplermelerinde \0 yazılmak zorundadır.*/

    char dizi2[]="adnan";
    printf("%s\n", dizi1);
    printf("%s", dizi2);

    return 0;
}
```

→ bu tip dizilerde \0 ile eğer karakter karakter tanımlama yaparsam gereklili olacaktır.
yani sıra bu iki farklı yazım şekliydi.

scanf ile kullanımını görelim.

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

int main(){

    char girdi[30];
```

```

printf("lutfen bi ifade giriniz:::  ");
scanf("%s", &girdi);
printf("\n\n\n%s", girdi);

return 0;
}
çıktı

lutfen bi ifade giriniz:::  dsfdasfsfd

dsfdasfsfd

```

→ görüldüğü gibi scanf kullanımında dizinin köşeli parantez içinde alabileceğim max karakter değerini belirtti. buradada dizideki her karakter bi indexe denk gelecektir. bunu değiştirdiğimiz bir senaryoda görelim.

```

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

int main(){

char girdi[]="ne diyosun ya";

girdi[0]='H';
printf("\n%s", girdi);

return 0;
}

```

>>> dikkat;:; eğer ben printf içinde bir karakter yazdırırmak istesem o zaman string ile değil char ile çağrırmam lazım. (yani %s yerine %c kullanmam gerekiyor.)

ve boşluktan sonra scanf ifadeyi görmeycektir.

burada gets ve fgets metodları devreye giriyor.

—String ifadeler için komutlar—

KARAKTER DİZİLERİİNDE KULLANILAN FONKSİYONLAR

Kullanılan belli başlı fonksiyonlar:`#include <string.h>`

- `strlen`: Bir karakter dizisinin boyutu
- `strcmp`: 2 karakter dizisini karşılaştırır.
- `strncmp`: Belirli bir n karakter limitine kara 2 diziyi karşılaştırır
- `strcpy`: bir karakter dizisini başka bir karakter dizisine kopyalar.
- `strncpy`: bir karakter dizisini başka bir karakter dizisine n karakter limite kadar kopyalar
- `strcat`: 2 karakter dizisini birlertirir.
- `strncat`: 2 karakter dizisini n karakter limitine kadar birleştirir.
- `strchr`: Karakter dizisinde bir karakteri arar.
- `strstr`: Karakter dizisinde bir alt karakter dizisi arar.

→ bu fonksiyonlar ilerde işe yarayabilir. [özellikle linux toolarında manyak deli dehşet şeyler üretiyorlar.]

Karakter dizilerinde kullanılan fonksiyonlar:::

→ **strlen()**

- bu fonksiyon karakter dizilerinin uzunluğunun hesaplanması sağlar.

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

    char girdi[]{"ne diyosun ya"};

    printf("girdiginiz karakter dizisinin boyu::: %d", strlen(girdi));

    return 0;
}

çıkıcı>>
girdiginiz karakter dizisinin boyu::: 13
// DİKKAT uzunluk sorguladığımız için bunu integer cinsinden (%d) olarak belirtmiştim.
```

bu şekilde syntax yapısı var ve bu fonksiyonların hepsinde olduğu gibi karakter dizilerine özgüdür.



strlen komutu karakter dizilerinin char char hafızaya kaydedilirken sona konan \0 ifadesini umursamıyor. AMA uzunluk yazdırırmak için kullandığımız klasik sizeof kullanımında bu değer alınırı. Görelim....

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

int main(){
char girdi[]{"ne diyosun ya"};

printf("girdiginiz karakter dizisinin boyu::: %d", sizeof(girdi));

return 0;
}

çıkıcı>>
girdiginiz karakter dizisinin boyu::: 14
// burada \0 değerini alıyor.
```

→ **strcmp**

- İki değeri kıyaslamaya yarar. > ASKİ kodlarına göre iki değeri kıyaslar ve ASKİ koduna göre hangisi büyükse 1 , küçük olana -1 değerini döndürür.

görelim.

```

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>

int main(){
    int sonuc;
    char deneme1[40];"asdfghjk";
    char deneme2[40];"ASDFGHJK";

    sonuc=strcmp(deneme2,deneme1);

    printf("%d", sonuc);

    return 0;
}
çıktı>>
-1

```

⇒ strcmp kullanırken burada değişkenlerin hangi sırada belirtildiğinin önemi var. ilk yazılımı diğerine göre kıyaslayıp ona göre yazar. (1. değişken strcmp de daha büyük olsaydı eğer 1 döndürecek.)

- - aslında strcmp çok kritik bir noktadadır!, çünkü dışarıdan alan herhangi bir değişkeni sistem içerisindeki değişkenle asciiye göre tamamen kıyaslaması ileride kontrol alanında çok değerli olabilir.

→ **strncpy**

- strcmp den farkı n dir. yani....
belki bir karaktere kadar kıyaslama imkanı sağlar.

Önceki kodu kullanarak gösterelim.

```

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int main(){
    int sonuc;
    char deneme1[40];"ASDFGHJK";
    char deneme2[40];"asdFGHJK";

    sonuc=strncmp(deneme1,deneme2,3);

    printf("%d", sonuc);

    return 0;
}
çıktı>>
1

// dikkatimi şu çekti. değerler aynı olduğu zaman hafızada olan değeri yazdırıyor.
// eşitsizlik durumunda ona göre işlem yapıyor.

```

→ **strcpy**

- Bir dizinin içeriğine başka bir karakter dizisinin içine kopyalamaya yarar.

görelim.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int main(){
    int sonuc;
    char deneme1[40]="sdadsadsadasdwefw";
    char deneme2[80];

    strcpy(deneme2,deneme1);

    printf("%s", deneme2);

    return 0;
}

çıktı>>
sdadsadsadasdwefw

// kopyalanacak konum ilk, kopyalanan içerik sonra yazılır dikkat!
```

strcpy;

farklı şekilde bir diziden atama yapmak yerine direkt string ifadeyide kopyalamamızı sağlayabilir.

→ **strncpy**

- yine n farkı var,
- belli bir karaktere olan kısmı kopyalayacaktır.

görelim.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int main(){
    int sonuc;
    char deneme1[40]="sdadsadsadasdwefw";
    char deneme2[80];

    strncpy(deneme2,deneme1,4);

    printf("%s", deneme2);

    return 0;
}

çıktı>>>
sdad
```

→ “n” yapılı bunun gibi fonksiyonlarda hata yapısı oluşabiliyor. ()C içindeki buglardan kaynaklanıyor diyebiliriz. bunun çözümü için boşta olsa atanacak diziye değer vermek lazımdır.

—> deneme2 = ""; |||| şeklinde gayet yeterlidir.

→ **strcat**

- iki karakter dizisinide birleştirmeye yarar.
illa bir karakter dizisi olmak zorunda değil herhangi bi string ifadeyide birleştirmek mümkün.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int main(){
    int sonuc;
    char deneme1[80];
    char deneme2[80];

    strcpy(deneme1," sen kim oluyon");

    strcat(deneme1," \n - bilmem sen kim olyuon");

    printf("%s", deneme1);

    return 0;
}
çıktı>>
sen kim oluyon
- bilmem sen kim olyuon
```

→ **strncat**

- n parametresinin göründüğü gibi n karaktere kadar birleştirme yapacaktır.
yukarıdaki örnekten yararlanalım.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
int main(){
    int sonuc;
    char deneme1[80];
    char deneme2[80];

    strcpy(deneme1," sen kim oluyon");

    strncat(deneme1," \n - bilmem sen kim olyuon",10);

    printf("%s", deneme1);

    return 0;
}
çıktı>>
sen kim oluyon
- bilme
// belli bir karaktere kadar birleştirilmek istenen ifadeyi aldı ve printf ile yazdırdı.
```

gets() ve fgets()

- string bir ifadeyi scanf dışında bu iki yöntemlede alabiliyoruz.
ama şöyle bi durum var ki gets kullanımı tavsiye edilmiyor.
bunun yerine fgets ile kullanım öneriliyor.

→ fgets() için 3 parametre girilmeli.

standart input için!

```
fget(dizinin ismi , dizinin boyutu[sizeof...], stdin)
```

görelim..

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

int main(){

    int a;
    char dizi1[100];

    printf("lutfen bir deger giriniz:::");
    fgets(dizi1,sizeof(dizi1),stdin);
    printf("\n\n %s", dizi1);

    return 0;
}

çıktyı>>>
```

```
SSSSSSSSSSSSSSSSSS
```

→ **strrev()**

- adı üzerinde revers den geliyor (ters çevirmeye yarar.)
- ilk karakteri sona ve son karakteri başa alana kadar diziyi tersine çevirir.

: n li kullanımında anlaşıldığı gibi belli karaktere kadar tersine çevirecektir.

gözlemleyelim.(önceki örnekten yararlandım.)

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

int main(){

    int a;
    char dizi1[100];

    printf("lutfen bir deger giriniz:::");
    fgets(dizi1,sizeof(dizi1),stdin);
    printf("\n\n %s", dizi1);
    printf("\n\n %s", strrev(dizi1));

    return 0;
}

çıktı>>>
```

```
my name is Talha AYDIN
```

```
NIDYA ahlaT si eman ym
```

→ **strlwr()**

- adından anlaşılacağı gibi lower in kısaltmasıdır.
bir string ifadeyi küçük harf yapmaya yarar.

ne kadarının küçük olduğunu vs dikkat etmez , topunu küçük harf yapar.

aynı kullanım şekli. yeterince örnek yazdım. anlaşılabilir.



scanf string ifadede boşluk gördüğü zaman kesiyor o yüzden string ifadelerde fgets metodunu kullanmak daha yararlı olacaktır.

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

int main(){

    int a;
    char dizi1[100];

    printf("lutfen bir deger giriniz:::");
    fgets(dizi1,sizeof(dizi1),stdin);
    printf("\n\n    %s", dizi1);
    printf("\n\n    %s", strlwr(dizi1));

    return 0;
}

çıktı>>

MAHMUT ABEYYYYYY

mahmut abeyyyyy

// fgets in syntaxine dikkat! stdin kullandık çünkü input giriyoruz. Dosyalar için kullanım farklı olacaktır.
```

→ **strupr**

- strlwr in tam tersidir.
strupr hepsini büyük yapar.

bundan dolayı extra açıklama yapamayacağım.

.....

→ **strstr**

Pointer ile alakalı bir özellik.

belli bir string ifadeinden sonraki tüm kısmını alır.

pointer kullanımı ile bunu başka değişkene atayacaktır.

(binevi arama işlemini yapan bir fonksiyondur.)

Karakter dizileri ile ilgili iki örnek ;;

örnek açıklamları CAN BOZ c dersinde verildi.

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

int main(){

    char dizi1[50]="bonne ";
    char dizi2[50]="soirée";

    printf("%s", strncpy(dizi1,dizi2,3));

    /* şuna dikkat ki strcpy komutları direkt dizinin üzerine yapıştırır.
     * yapıştırılan dizinin içeriğinin direkt üzerine gelir.*/

    return 0;
}

çıktı>
soine --->
!!! dizi2 den aldığı ilk 3 karakteri
değişiklik yapmak istediğimiz dizininin ilk 3 karakterinde gördük
```

→ birleştirmek isteseydik strcat ile yapabilirdik.

bi iki boyutlu dizi hatırlartması

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

int main(){

    char dizi1[5][3]={{1,34,45},{54,78,99},{43,67,89},{45,78,23},{11,57,83}};

    printf("%d", dizi1[3][2]);

    return 0;
}

çıktı>>
23
TAMAMLANDI!
```

Lecture 5

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

FONKSİYONLARA GİRİŞSSSSSSSS

→ artık işleri farklı bir seviyeye çıkarma zamanı. Dilin ifade temeli fonksiyonlara geldiiiiiiik.
python'dan konuya hakimiz ama daha odaklı şekilde konuya dalalım.

fonksiyon neydi →

belirli işlemleri yapmamızı sağlayan kod bloklarını içeren yapılardır.

C de iki farklı fonksiyon tipi var.

geeriye değer döndürenler veya döndürmeyenler.

örnek üzerinde gösteririrsek bir tek mi çift mi srguusnu fonksiyonlaştıralım.

—
geriye değer döndürmeyen fonksiyonların birkaç kuralı var.

bunlar void ile belirtildi, belirlediğimiz fonksiyon isminden sonra parantez açılıp değişken türünü belirtip değişkeni yazarak belirtiriz.

aslında int main içerisinde ne yapıyorsak küçük çaplısını fonksiyonları oluşturmak için de kullanıyoruz diyebiliriz.

```
int main(){  
    int sayı;  
    printf("bir sayı giriniz::::: ");
```

```

scanf("%d", &sayi);

if(sayi%2 == 0){

    printf("\n\nGirdiginiz deger çifttir.");

}

else{
    printf("\n\nGirdiginiz deger tektir.");
}

return 0;
}

```

elimde böyle bir ifade var.

0 zaman

```

void tekmiciftmi(int deger){
    if(deger%2 == 0){

        printf("\n\nGirdiginiz deger çifttir.");

    }

    else{
        printf("\n\nGirdiginiz deger tektir.");
    }

}

int main(){

    int sayi;
    printf("bir sayı giriniz:::: ");
    scanf("%d", &sayi);

    if(sayi%2 == 0){

        printf("\n\nGirdiginiz deger çifttir.");

    }

    else{
        printf("\n\nGirdiginiz deger tektir.");
    }

    tekmiciftmi(sayı);
    return 0;
}

```

syntax bu şekilde.



buraya dikkat.

bu void ile geri döndürmesiz bir yapı ama şu dikkatimi çektı.

int mainin değişkeni ile fonksiyonu isminin yanında (aşağıdaki gibi çağrılığında çalışıyor. direkt bu satırda geldiğinde fonksiyona gelmiş gibi davranış taaaak diye çalıştırıyor.)

fonksiyon içinde verdigimiz değişkenin yerine geçiyor, main içerisinde verdigimiz asıl değişken.

ki zaten fonksiyonun istediğimiz değerlerde çalışmasını sağlayan özellik bu.

ve çağrıırken sırayla hangi değişkenin hangi değişkene karşılık geldiğini zaten normal virgül kuralıyla yapıyoruz.

şimdisi bi de geri döndürmeli fonksiyonlara bakalım.

fonksiyonu bir değişkene atayarak döndürmeli ve işleme dahil yapılar kurabiliriz.

burada önceki tipten farklı kullandığımız birkaç farklı yazım var.

görelim

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

int tekmiçiftmi(int deger){
    if(deger%2 == 0){

        return 1;
    }

    else{
        return 0;
    }
}

int main(){

    int sayı;
    printf("bir sayı giriniz:::: ");
    scanf("%d", &sayı);

    if(sayı%2 == 0){

        printf("\n\nGirdiginiz deger çifttir.");
    }
}
```

```

else{
    printf("\n\nGirdiginiz deger tektir.");
}

int sonuc=tekmiciftmi(sayi);

if(sonuc== 1){
    printf("\n\nbasarili1");

}
else{

    printf("\n\nbasarili2");
}

return 0;
}

```

→ return komutu geri döndürmeli fonksiyonlar için değerli . return yaptığımızda direkt fonksiyona o değere döndürüyor diyebiliriz. geri döndürmeli fonksiyonlara özgü diyebiliriz.

kısaca şunu yaptık.

fonksiyonun integer deger dondureceğini bu yüzden integer tipinde oladığını söylelik ve bunu yine integer tipinde belirttiğimiz değişkene atadık.

bu sayede fonksiyon sonucu direkt değişkene atanmış oldu ve sonraki adımlarında bu değişkene klasik sorgularımızı (aynı normal yazılardaki gibi devam ettirdik.)

fonksiyonu main içerisinde çağrırmak için özellikle isim yazıp ana değişkenleri (artık hangilerini fonksiyon içinde kullanmak istiyorsak) parantez içinde kullandığımızı unutmayaşalım.

ÇOOOK ÖNEMLİ! BURAYI zamanla güzelce anlayalım.

fonksiyon kullanarak bir senaryo kuralım.

```

//görev şu :::
// iki girilen tam sayının 4 işlemini fonksiyon kullanarak veya kullanmadan yazdırma.
void dortislem(int x, int y){
    printf("\n%d * %d = %d (fk)",x,y, x*y);
    printf("\n%d + %d = %d (fk)",x,y, x+y);
}

```

```

printf("\n%d / %d = %d (fk)",x,y, x/y);
printf("\n%d - %d = %d (fk)",x,y, x-y);

}

int main(){
    int sayi1,sayi2;
    printf("lutfen bir deger giriniz::: ");
    scanf("%d", &sayi1);
    printf("lutfen bir deger giriniz::: ");
    scanf("%d", &sayi2);

    // fonksiyon kullanmadan

    printf("\n%d * %d = %d",sayi1,sayi2, sayi1*sayi2);
    printf("\n%d + %d = %d",sayi1,sayi2, sayi1+sayi2);
    printf("\n%d / %d = %d",sayi1,sayi2, sayi1/sayi2);
    printf("\n%d - %d = %d",sayi1,sayi2, sayi1-sayi2);

    // fonksiyon kullanarak

    dortislem(sayi1,sayi2);

    // (fk) fonksiyon kullanarak çağrırdıklarımız.
}

```

çıktılar >>

```

99 * 34 = 3366
99 + 34 = 133
99 / 34 = 2
99 - 34 = 65
99 * 34 = 3366 (fk)
99 + 34 = 133 (fk)
99 / 34 = 2 (fk)
99 - 34 = 65 (fk)

```

bunu %f (float cinsinden de yapabiliirdik. [hatırlatma %.2f deki .2 gibi kullanımlar float cinsinin . sonrası ne kadarının görüneceğini belirtmek için konur.])

güzel bir nokta.

[printf in % atamalına önceden direkt scanf değerini yazdırabildiğimiz gibi direkt fonksiyonunuda yazdırabiliyoruz. aşırı iyi!]

geri döndürmeli fonksiyonlar için gerçekten güzel bir yöntem.

(geri döndürmel ifonksiyonlar gerçekten çok değerli. void fonksiyonlarından daha esnek yapıları olduğu söylenebilir.)

GLOBAL ve LOCAL Değişken Mantığı

diğer dillerde parent child ilişkisi olarak da bilinir.

yani bir fonksiyon içindeki bir değişken tanımı o fonksiyona özel olması gibi (local) daha kapsayıcı, fonksiyon dışı değişkenlerde olabilir. (global.)

ve içiçe atanmış bir değişkende → farklı değerler aynı değişkene atanmış olsun..

hangisi yazdırırlı veya alınır diye sorulduğu zaman hangisi daha local(özel)se o değer direkt çıktı olarak alınır.

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

int main(){
    int x=5;
    {
        int x=10;
        printf("%d", x);
    }

    return 0;
}

// görüldüğü gibi...
> hangi ifade daha özelse o ifade yazılmış oldu.
```

→ evet basit gibi görünüyor ama son derece önemli bir yapı!

Fonksiyonlarda globalm değişken !

LOCAL VE GLOBAL MANTIĞI FONKSİYONLAR İÇİN ÇOK KRİTİK!

bir örnek üzerinde belirtmek istiyorum.

```

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

void f1(int a){
    a=24;
    printf("%d", a);

}

int main(){
    int x=5;

    printf("%d", x);
    {

        int x=10;
        printf("\n\n%d\n\n", x);

        f1(x);

        printf("\n\n%d", x);
    }

    return 0;
}

çıkıcı>>
5

10

24

10

```

→ evet...

birkaç noktaya yeniden bakalım.

- local değişken mantığına dikkat çıktılarında gayet hangi değer nasıl atanmış anlaşılıyor.
- önceden belirttiğimiz gibi int main içinde birkaç satır sonra çağrıdığımız fonksiyon main kısmın değişkenini kendi içinde yapıyor ve gidiyor. AYNEN FONKSİYON KURALLARINDA OLDUĞU GİBİ.
- kısaca fonksiyonun kendi dedeğişkeni oluyor. local olarak ona ait. diğer int main'in değişkeninden bağımsız hareket ediyor.)

fonksiyon içinde tanımladığımız bir değişkeni sistemin herhangi bir yerinde dikkate almamıza gerek yok kısaca.

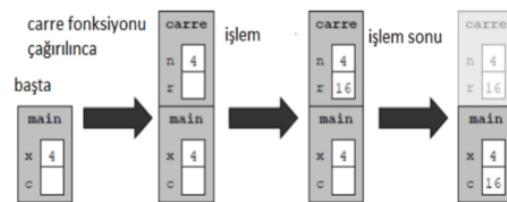
kafamız karışmasın gayet mantık kuralları içinde bir yapısı var fonksiyonların.

değişkenin değeriyle ve değişkenin adresi ile işlemler.

DEĞİŞKENİN DEĞERİYLE İŞLEM

```
int carre(int n)
{
    int r = n * n;
    return r;
}

Bu fonksiyonun çağırılması
int main()
{
    int x=4,c;
    c = carre(x);
    printf("%d",c);
    return 0;
}
```



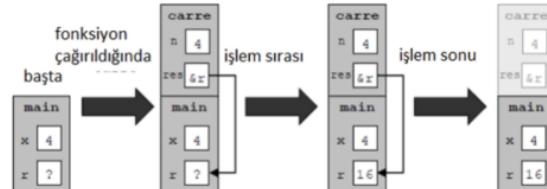
Bu işlem sırasında değişkenin değeri fonksiyona gönderilir ve fonksiyon bu değere göre işlem yaparak sonucu dönderir ancak bu işlemler sırasında main fonksiyonu içindeki değer değişmez buna değişkenin değeriyle işlem denir.

değişkenin değeriyle olanb işlemi zaten geri döndürmeli fonksiyonlarda görmüştük.

DEĞİŞKENİN ADRESİYLE İŞLEM

```
void carre(int n, int* res)
{
    *res = n * n;
}

int main()
{
    int x = 4;
    int r;
    carre(x,&r);
    printf("%d",r);
    return 0;
}
```



gayet güzel anlatıyor. Nasıl çalıştığını dair.

.....

arada fazla bir değişken kullanmadan adresle değişken ataması çok iyi.

→ fonksiyonu main içersinde çağrıırken fonksiyonun sonucunun atanmasını istediğim bu sefer direkt fonksiyon değil adreslediğim değişken oluyor.

bu şekilde void fonksiyonlar kadar sağılıklı ve çok daha kanalize bir şekilde yeni değişkene yönlendirme yapabiliyorum demek oluyor bu.

return yazmama da gerek kalmadı ve bu değer artık global olmuş oldu.

neden? çünkü zaten main fonksiyonu içinde yazmıştım ve fonksiyonu kullanarak sadece scanfteki gibi bir adresleme yapmış oldum.

bu sefer kullanıcının yazdığı bir değere değil kendi yazdığım bir fonksiyonu kullanarak bir adresleme yaptım.

görelim.

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

void f1(int a,int*atama){
    a=12;
    *atama=a*a;
    // aslında burada fonksiyon içindeki
    // kullanacağım değişkeni yıldızla belirtiyorum ki asıl yapıyla karışmasın.

}

int main(){
    int x=5;
    int r;

    f1(x,&r);

    printf("\n\n%d",r);
    printf()

    return 0;
}

çıktı>
144
```

→ unutmadan, fonksiyonda yazdığımız x normal değeri hala korunuyor. sadece fonksiyon bitene kadar fonksiyonun kendi değişkeni (a) ya eşit oluyor (ki fonksiyonun ana sisteme bağlı çalışması için;;;;) sonrasında hala normal x değerinde kalıyor. çünkü x değeri main içinde tanımlandığı gibi global, fonksiyon içinde ise local .

→ eğer yıldızlı değerle işlem yaparsam normal olarak devam eder. değişkenin yıldızına dokunulmaz, mesela çarpma işleminde filan (*deger)*(^deger) olur.

→ yıldızlı şekilde olan fonksiyon içindeki konuma ben main içindeki bir değeri adresleyebilirim. kısaca fonksiyondan değeri almak yerine fonksiyona değeri yazarım.

çok kompleks bir yazım oldu. sade bir örnek üzerinde gösterelim.

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

void f1(int *adres1,int *adres2){

    *adres2=*adres1**adres1;

}

int main(){

    int x=5;
    int r;

    f1(&x,&r);

    printf("\n%d",r);

    return 0;
}

çıktı>>
25
// yeni olan değerleri fonksiyona adresleyip, fonksiyonda çalışıp direkt sisteme geri döndürebilirim.
EHEHEHEHEHHEHEHEHE

// kurallara uygun olarak sıra ile adresleniyor.
```

iste

adreslemenin normal süreçten(aşağıda onuda yazdım) farkı bu.

cok daha net bence

ki pointer meyzusunu içeren bu adresleme mekanizması ileride çok kritik bir rol alabilir.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
// main içindeki değişkenlerle bağlantılı olan noktaları parantez ile belirtiriz.
// yoksa diğerlerini parantez içinde kullanmak hata verdirir.
int f1(int a){
    int sonuc;
    a=12;
```

```
sonuc=a*a;
return sonuc;

}

int main(){
    int x=5;
    int r;
    r=f1(x);
    printf("\n\n%d",r);
    return 0;
}
```

Lecture 6

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

C PREPROCESSOR DIRECTIVES

→ koda makro atama sağlar.

Başka şeylere transform etmeyi eklemeyi ve daha geniş çapta şeyler yaptırmayı sağlar.

ve değerli.

bunlar

```
#include  
#define  
#undef  
#ifdef  
#ifndef  
#if  
#else  
#elif  
#endif  
#error  
#pragma
```

Macro

- object-like Macros

```
#define PI 3.14
```

(bunu görmüşük direkt bir değişkene bir değer eşitlemesi sağlıyor. pi değişkeni 3.14 e eşitmiş.)

- Function-like Macros

```
#define circleArea(r) (PI*r*r)
```

(bude fonksiyon şekilde atama. direkt fonksiyonu define ile belirtip r paremtersiyle kullanmış oldu.)

şimdi bu macroların syntaxini görelim.

```
#include<stdio.h>  
#include<stdlib.h>  
#define PI 3.14  
#define circleArea(r) (PI*r*r)
```

```

int main(){

    float alan,pr;

    printf("lutfen bir yaricap degeri giriniz;;;  ");
    scanf("%f", &pr);

    alan=circleArea(pr);

    printf("girdiginiz yaricaptaki dairenin alani::: %f", alan);

    return 0;

}

çıktı>>
lutfen bir yaricap degeri giriniz;;;  6.78
girdiginiz yaricaptaki dairenin alani::: 144.340790

```

→ gerçekten böyle durumları makro ile direkt sisteme atamak avantaj oluşturuyor.

Predefined Macros

adından mütevelliit hali hazırda C kütüphnaesinde bulunna Macrolardır.

Predefined Macros

Here are some predefined macros in C programming.

Macro	Value
<code>__DATE__</code>	A string containing the current date.
<code>__FILE__</code>	A string containing the file name.
<code>__LINE__</code>	An integer representing the current line number.
<code>__STDC__</code>	If follows ANSI standard C, then the value is a nonzero integer.
<code>__TIME__</code>	A string containing the current time.

.....

hemen syntaxini görelim.

```

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>

int main(){

    printf("file : %s\n", __FILE__);

```

```

printf("Date : %s\n", __DATE__);
printf("Time : %s\n", __TIME__);
printf("Line : %d\n", __LINE__);
printf("STDC : %d\n", __STDC__);

return 0;

}

çikti>>
file : C:\Users\taayd\OneDrive\Masaüstü\WORK-T\Dev-Documents\for C\C-base.cpp
Date : Feb 22 2023
Time : 18:37:11
Line : 13
STDC : 1

```

→ her makroyu ayrı ayrı ele alma zamanı.

⇒ **include**

C içindeki ayrı kütüphaneleri kendi sistemimizin içine dahil etmek için kullandığımız keyworddur.

(math kütüphnesi string kütüphanesi vs bunlar include ile aktarılır.)

⇒ **define**

söylemiştik.. define ile iki farklı şekilde makro atayabiliyorduk.

yukarıdaki örnekdede zaten anlaşıyor. bir tanımlama(fonksiyonda olur bir değerde olur.) yapmamızı sağlar.

⇒ **undef**

bu keyword adından da mütevellit define ile tanımlanmış iki tip makroyuda kaldırımızı sağlar.

yani define silmek yerine bunu yazmak garip ama genede dursun.

belki güvenlikte vs işimize yarayabilir.

⇒ **ifdef**

define yapısına bağlı bir if else soru sistemidir.

ve if koşulunun çalışması için define ile değişken tanımlanmak zorundadır.

aksi halde ifdef te else hali belirtilmişse onu gerçekleştirir.

endif ile bitirilmesi gereken bir keyword yapısı diyebiliriz.

görelim bi syntaxı

```

#include<stdio.h>
#include<stdlib.h>
#define P 3.14

int main(){

```

```

#define PI
    printf("sadasdsad");
#else
    printf("dsadsadasdsadasdsadsadsad");
#endif

    return 0;

}
çıktı>>
dsadsadasdsadasdsadsad

// yukarıdaki keywordler gibi yukarıda değil bu int main içerisinde belirtilir.

```

→ evet.... yukarıdaki define ile uyuşmayan bir değer verildiğinde else belirtmişsek ona gitdiyor buda kanıtı oldu.

.....

güzel bir örnek var.

:: define ile bağlı bir değişkene defif yapısı ile işlem yaptıralım . değer yoksa kullanıcı bir değer gırsın.

görelim senaryoyu;;;

```

#include<stdio.h>
#include<stdlib.h>
#define d 2

int main(){

    int m;
    #ifdef di
        printf("%d",d);
    #else
        printf("deger girinizzz::: ");
        scanf("&d", &m);
    #endif

    return 0;

}

// define le uyuşmadığ için else yapısına gitti.
//ama biz else ile belirtmiş olmuyoruz direkt ifdef yapısından çıkacaktı.
// endif her türlü bitiriste kullanıyoruz.

```

⇒ #if#else

bir fark yok yine #endif yapısıyla tanımlanan define parametresine bağlı genel macro kullanımı. bir önceki ifdef ile arasında mantık olarak hiçbir fark yok.

⇒ #error

süreç bazı noktalarda uymuyorsa istenmeyen bir durumu hata fırlatma şeklinde modifiye edebilirim.

.....

bunu if else ile olan bi yapıda göstereyim.

```

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

```

```
#define pp3

int main(){

#if !pp || pp > 5
#error pp is not defined or bigger than 5
#endif

return 0;

}

-> aşağıda gördüm erroru
// neden peki; if sorusu çerçevesinde ya tanımlanmamışsa ya da 5 ten büyük değilse hata mesajı vermesini istedim.
// bunuda direkt #if yapısını kullanarak yaptık.
```

(string ifade ile #if i doldururken ya da ifadesine ve !(değil) ifadesine DİKKAT!)

bunları biz SQL çerçevesinde de kullanıyor ve işlem yapıyorduk.

Lecture 7(General exerxices)

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

güzel örneklerde dikkat çekici olanlara odaklandığım bir lecture.

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

int main(){
    int a,b;
    printf("lutfen iki deger giriniz;;;; ");
    scanf("%d", &a,&b);
    // if else yapısını sürekli kullanarak çok uzun bir yol oluşturuyor. && ve || operatörlerini kullanarak örneği görelim.

    if (a>0 && b>0 || a<0 && b<0){
        printf("\n\t 1");
    }
    else if(a<0 && b>0 || a>0 && b<0){

        printf("\n\t -1");
    }
    else{
        printf("\n\t 0");
    }

    return 0;
}
```

→ buraya dikkat.

&& (ve) || (ya da) operatörlerini if else sorguları içinde kullanabiliriz.

Böylelikle daha spesifik sorgular elde edebilir fe soru mekanizmasını genişletebilir, özelleştirebilirz.

ÇOK KRITIK

(eğitimde atlamışım galiba bu kısmı)

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

int main () {
    char deger;
    printf("lutfen büyük harf tipinde bir karakter giriniz::: \n");

    scanf("%c", &deger);

    if(deger>= 'A' && deger<='Z'){

        printf("\ngirdiginiz buyuk harfin kucuk hali ::: %c", tolower(deger));
    }
    else{

        printf("lutfen kabul edilebilir ifadeler giriniz...");
    }

    return 0;
}
```

```

}

// tolower yapısı benim derleyicide çalışmadı ama örnek birebir.

```

→ burada şu durum dikkatimi çekti::;

char ile kıyaslama yaptığım zaman ASCII karşılığına bakıp ona göre işlem yapıyor. (hatırlayalım.)

yukarıdaki if sorusunun içine dikkat. gerçekten değerli bir bakış açısı katıyor...

```

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

int main () {
    int i=1, n;
    float sonuc=0;
    printf("n kac olsun? ::: ");
    scanf("%d", &n);

    do{
        sonuc = sonuc + (float) 1/i;
        printf("1/%d + ", i);
        i++;
    }while(i<=n);

    printf(" = %.2f", sonuc);

    return 0;
}

```

AMAN DİKKAT!!

float olarak belirtmemiz gereken yerdeki syntaxe dikkat hocam.

belirtme şekli bi tık önemli.

yanlış syntaxte belirttiğim için 15 dk dir bakıyorum koyun gibi.

Şuan yazacağım örnekte aynı şekilde sürekli değer isteyen ve bu durumdan çıkarmak için olan senaryolara gayet güzel bir örnek.

```

#include<stdio.h>
#include<stdlib.h>
#include <math.h>
// bu örnek önemli sürekli tekrar eddcek şekilde karşidan deger istemek ve sonuc doldurmek gerekiyor. Bun uyaparken if yapısı ile c

int main () {
    float i=1, n;

    do{
        printf("\nlutfen bir pozitif deger giriniz :::   ");
        scanf ("%f", &n);
        if(n>0){

            printf("\n\ngirmis oldugunuz sayinin karekoku = %.3f", sqrt(n));

            break ;
        }

        else if (n<0){
            printf("\n\nlutfen yeniden deger giriniz... \n\n");
            continue ;
        }
        else{
            break;
        }
    }
}
```

```

        }
    }while(1);

    return 0;
}

```

→ bu örnek birçok yönden kıymetli,,,

klasik bir pozitif ve negatif örneğide koymak istedim.

```

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

int main(){
    int a,b;
    printf("Carpimlerinin pozitif veya negatif oldugunu soleyecegimiz 2 deger giriniz:::\n\n");
    scanf("%d%d", &a,&b);

    if(a>0 && b>0 || a<0 && b<0){
        printf("2 sayinin carpiminin degeri POZITIFTIR!");
    }
    else if(a<0 && b>0 || a>0 && b<0){
        printf("2 sayinin carpiminin degeri NEGATIFTIR!");
    }
    else{
        printf("2 sayinin carpiminin degeri SIFIRDIR!");
    }

    return 0;
}

```

```

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

int main () {
    float a,b,c;
    do{

        printf("lutfen 3 genin acilarini giriniz::: \n\n");
        scanf("%f%f%f", &a,&b,&c);

        if(a+b+c == 180){

            if(a==b && b!=c || a==c && c!=b || c == b && b!= a ){
                printf("girdiginiz degerlerdeki ucgen ikizkenardir");
            }

            else if(a==b && b==c){
                printf("girdiginiz degerlerdeki ucgen esitkenardir");
            }

            else{
                printf("girdiginiz degerlerdeki ucgen cesitkenardir");
            }

            break;
        }
    }

```

```

    else{
        printf("girdiginiz degerlerde ucgen olusmamaktadir.\n\n yeniden deneyiniz; \n\n");
        continue;
    }
}while(1);

return 0;
}

```

→ Üçgenin iç açılmasına göre gruplandırma yapan güzel bir senaryo. Burda bulunmasını istedim.

```

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

int main(){
    int a,b,d;
    char c;
    float sonuc;
    printf("islemnizi seciniz:: (+ , -)");
    scanf("%c", &c);
    printf("lutfen 2 sayi degeri giriniz;; ");
    scanf("%d%d", &a, &b);

    if(a==0 || b==0){
        printf("lutfen gecerli bi yapi giriniz:::");

    }
    // kesirle toplamanın matematiksel bir yaklasimini switch case yapısı ile kullanabiliriz.
    // ve özellikle operatörlere göre değişiklik göstereceği zaman gerçekten değerli olan bir yapı.
    else{
        switch(c){
            case '+': {
                printf("\n\n 1/%d + 1/%d = %d/%d", a,b, (a+b), (a*b));
                break;
            }
            case '-': {
                printf("\n\n 1/%d - 1/%d = %d/%d", a,b, abs(a-b), (a*b));
                break;
            }
            // matematiksel şekilde yöntem kullanarak bu şekilde bir yazım elde ettik.
        }
    }

    return 0;
}

```

→ switch case yapısına ait olan çok güzel bir örnek bence bu.

ve sonuç olarak basit kesir olarak direkt görüntü döndürmeside gerçekten işe yarar bir özelliği var.

NOTE:: default yapısını da eklemeyi unutmamak lazım.

aslı örnekte kullandım .

```

default : {

printf("yanlis operatör girisi:::");

break;
}

```

şeklinde switch case yapısına ekledim.

Sıradaki örnek ascii anlama anlamında değerli lütfen odaklanalım.

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

int main(){
    char kr;
    int sira;
    printf("lutfen karakterinizi giriniz:::");
    scanf("%c", &kr);

    if ((kr >= 'A') && (kr<='Z')){

        sira= (int) kr- (int) 'A' +1;
        printf("girmis oldugunuz karakter ing alfabede %d.", sira);
    }
    // burdaki olaya dikkat! sıra buudrmak için ascii numaralarının farkından yararlanıp burada bir yapı kurduk ve 1 ekleyerekte sıray:
    // eklemeseydik 0 dan başlardı buda istedigimiz sıra olayını vermezdi.
    // aynı olayı küçük tipteki harfler içinde geçerlidir.
    // buradaki ascii kullanımı gerçekten değerli ve kullanılımaya müsait.

    else if((kr >= 'a') && (kr<='z')){

        sira= (int) kr- (int) 'a' +1;
        printf("girmis oldugunuz karakter ing alfabede %d.", sira);
    }

    else{
        printf("girmis oldugunuz karakter bu program için uygun değil.");
    }

    return 0;
}
```

ASCII kodları kullanarak yaptığımız senaryoya bakalım.

gerçekten ascii kullanarak durumu işletmetize etmek güzel bir mantık!

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

int main(){
    int n,i;
    float x, sonuc=0;
    printf("lutfen bir pozitif tam sayı giriniz;; \n\n");
    scanf("%d", &n);
    printf("\n\nbir reel sayı giriniz;; \n\n");
    scanf("%f", &x);
    float k=1;
    do
        if(n>0){
            while(n>0){
                i=1;
                k=1;
                for(i=1;i<=n;i++){
                    k=k*x*x;

                }
                sonuc += (float)(2*n-1)/k;
                n--;
            }
            printf("seri işlemi sonucu => %.2f", sonuc);
        }
}
```

```

    }
else{
    printf("yanlis girdi tekraer deneyinizz");
}
return 0;
}

```

→ bu pow kullanmadan yapılan versiyon::

pow kullanarak while döngüsüne sokmadan da yapılabilirdi.

ÇOK KRİTİK

bazen scanf i do while de vs ifadenin sonunda kullandığımız zaman görmüyor.

Bu direkt scanf in problemi ile ilgilidir.

alternatif bazı şeyleri kullanabiliriz. bunu ilerde bahsedeceğim.

Bu örnekteki ana mevzu do while in kendi koşulunu bir if gibi yapılandırdığımızı unutmamaktı!!
seçimi kullanıcıya bırakarakta döngüyü sürekli hale getirebildiğimizi öğrendik.

```

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

// önceden bulduğumuz min max yapılarına benzer bir durum. kendi limit değerimizi olduğunda aşağı bir değere while + if kombinasyonunu kullanırız.

int main(){
    int no, boy, fark, min;
    printf("ogrenci NO::: \n");
    scanf("%d", &no);
    printf("ogrenci boy::: \n");
    scanf("%d", &boy);
    int a,b;

    min = abs(150-boy);

    while(no>0){
        printf("ogrenci NO::: \n");
        scanf("%d", &no);
        printf("ogrenci boy::: \n");
        scanf("%d", &boy);

        fark = abs(150-boy);

        if(min>fark){

            min = abs(150-boy);

            a=no;
            b=boy;
        }

        else{
            continue;
        }
    }

    printf(" %d no' lu ogrencinin boyu 150 ye en yakın olup %d kadardır", a,b);
}

```

```
// çekinme lo yaz ver değişken fazla fazla kısıtlayan mı var!!!!  
return 0;  
}
```

```
#include<stdlib.h>  
#include<stdio.h>  
#define tavhiz 0.38  
#define kushiz 0.12  
int main(){  
    int yil=0;  
  
    float kus, tav;  
    kus= 2272;  
    tav= 1042;  
  
    while(tav <kus){  
        kus += 2272* kushiz;  
        tav += 1042* tavhiz;  
        yil += 1;  
  
        printf( "\n\n %d. yilda tavsan sayisi::: %d  kus sayisi ::: %d", yil, (int) tav, (int) kus );  
    }  
  
    printf("\n\n\n%d. yilda tavsan sayisi daha fazla", yil);  
  
    return 0;  
}
```

→ define ve while kullanımı anlamında gerçekten net bir örnek!!!

Lecture 8 (General exercises)

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

→ kelime sayısını hesaplanmasıın ayarayan gerçekten harika bir örnek geliyor..

,

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

int main(){

    char cumle1[100];
    int i=0, sayi=1;

    printf("şutfen bir cümle grinizz");
    //fgets(cumle1,sizeof(cumle1),stdin);
    gets(cumle1);

    while(cumle1[i]){
        // space in boşluk asci kodu 32 dir.
        // boşluk sayısının bir fazlası kadar kelime var. bundan yararlanarak kelime sayısını bulabilriz
        i++;
        if(cumle1[i] == 32){

            sayi++;

        }
        else {
            continue;
        }
    }

    printf("cumlenizin toplam kelime sayısı %d kadardır.", sayi);

    return 0;
}
```

→ while kullanımına dikkat!!! cümle bitene kadar döndürme işlemi sağlayacaktır.

gets veya fgets kullanımında sağlayabiliriz göründüğü şekilde.

VE ELSE KULLANMAK ZORUNDA DEĞİLİM. SİSTEMDEN ÇIKARDIĞIMDA Bİ ŞEY KAYBETMEM. sadece bir soru belirtirim if ile aksi halinde bir şey yapmasamda

olacaktır.

ama continue kullanımı arkasında geriye kalan ne avrsa görmeden direkt bir sonraki harekete devam edcektir.

hep kullanımı tehlikeli olabilir. Dikkatli kullanılmalıdır.

başka bir örnek üzerinde while in işleyişindeki bir noktaya dikkat çekmek istiyorum

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

int main(){
    char cumle2[100];
    int i=0;

    printf("lutfen bir cümle girinizzz \n\n");
    fgets(cumle2,sizeof(cumle2),stdin);
    printf("\n");
    while(cumle2[i]){
        printf("\n %c ", cumle2[i]);
        i++;
    }

    printf("\n\n ::: verdiginiz cumlenin karakter karakter alt alta yazilmis hali :::");
    return 0;
}
```

\0 durumu vardı hatırlıyorsak char char string ifadeler sistemde tutulduğu zaman sonuna \0 gibi bir ifade ekleniyordu.

aslında buradaki döngülerdeki olayda bu.

bu ifade görüldüğünde sistem durdurulmuş oluyor.

karakterlerden piramit oluşturma şeklinde yazım yapılabilir.

güzel bir örneği var

İç içe farklı döngüler için güzel bir örnek:::

```

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

int main(){
    char cumle2[100];
    int i=0,n;

    printf("lutfen bir cumle girinizzz  \n\n");
    fgets(cumle2,sizeof(cumle2),stdin);
    printf("\n");
    while(cumle2[i]){

        for(n=0;n<i;n++){

            printf("%c ", cumle2[n]);
        }

        i++;
        printf("\n");
    }

    printf("\n\n :::: verdiginiz cumlenin  birer karakter artirarak piramit olusturulmus hali ::::");

    return 0;
}

```

sistemdeki max karakterin ne kadar sayıda olduğunu döndüren bir yapı yazınız;;;

```

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

int main(){
    char cumle[100];
    int i=0,k,n, sayac, max=0;
    printf("lutfen bir cumle girinizz\n\n");
    fgets(cumle,sizeof(cumle),stdin);

    max=sayac;

    while(cumle[i]){
        sayac=0;
        for(k=0;cumle[k];k++){

            if(cumle[i] == cumle[k]){

                sayac++;

            }
        }
        if(sayac>max){

            max = sayac;
        }
    }
}

```

```
    i++;
}

printf("cumledeki en cok tekrar eden karakter %d kadar tekrar etti.", max);

return 0;
}
```

Lecture 9 (Function exercises)

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

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

/*void f1(int x, int y){
    int sonuc;
    printf("lutfen bir deger girinizz:::");
    scanf("%d", &x);
    printf("lutfen bir deger girinizz:::");
    scanf("%d", &y);

    printf("\n\nsayilarin toplami --> %d ", x+y );
}
*/
int f1(int x, int y){
    int k;

    k=x+y;

    return k;
}

// bu şekilde iki farklı fonksiyon kullanımını göstermiş oldum.

int main(){
    int a,b,sonuc;
    printf("lutfen bir deger girinizz:::");
    scanf("%d", &a);
    printf("lutfen bir deger girinizz:::");
    scanf("%d", &b);
    sonuc=f1(a,b);

    printf("%d", sonuc);

    return 0;
}
```

iki farklı örneğide vermek istedim.

yne hatırlayacak olursak,,,

döndürmeli fonksiyonlarda direkt olarak fonksiyonu belirttiğim işleme döndürürüm binevi int veya float gibi bir değere döner.

bunu printf içinde veya başka yerlerde direkt bunun bilincinde döndürebilirim.

Can BOZ un yaptığı örnekteki gibi.

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

void f1(int x, int y, char c){
    switch(c){
        case '+': {
            printf("\n\nsayilarin toplami --> %d ", x+y );
            break;
        }
    }
}
```

```

        case'*' : {
            printf("\n\nsayilarin carpimi --> %d ", x*y );
            break;
        }
        case'/' : {
            printf("\n\nsayilarin bolumu --> %f ", (float )x/y );
            break;
        }
        case'-' : {
            printf("\n\nsayilarin cikarimi --> %d ", x-y );
            break;
        }

    }

// bu şekilde iki farklı fonksiyon kullanımını göstermiş oldum.

int main(){
    int a,b;
    char c;
    printf("lutfen bir operatör giriniz::: \n\n");
    scanf("%c", &c);
    printf("lutfen bir değer giriniz:::");
    scanf("%d", &a);
    printf("lutfen bir değer giriniz:::");
    scanf("%d", &b);

    f1(a,b,c);

    return 0;
}

```

!!! önceden olan bir hadise yeniden kendini gösterdi.

char bir ifadeyi diğerlerinden daha sonra scanf ilr almak problem oluşturdu.

bunun çözümü için daha öncede alınabilir veya boşluk da bırakılabilir

(scanfteki karakter tanımlamasının içine) HATIRLATMA

```

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

int f1(int x){
    int r=1;
    while(x>0){

        r *= x;

        x--;
    }

    return r;
}

// bu şekilde iki farklı fonksiyon kullanımını göstermiş oldum.

int main(){
    int a,b;
    char c;
    printf("lutfen bir değer giriniz:::");
    scanf("%d", &b);

    printf("girmis oldugunuz sayini faktoriyel karsiliği ==> %d", f1(b));
}

```

```
    return 0;  
}
```

faktoriyel karsiligi;::

bir üs alma örrneği burda bulunsun

```
#include<stdlib.h>  
#include<stdio.h>  
  
int f1(int x, int y){  
    int r=1;  
    while(y>0){  
        r*= x;  
        y--;  
    }  
    return r;  
}  
  
// bu şekilde iki farklı fonksiyon kullanımını göstermiş oldum.  
  
int main(){  
    int a,b;  
  
    printf("lutfen bir taban degeri ve buna bir üst degeri girinizzz::");  
    scanf("%d%d", &a,&b);  
  
    printf("girmis oldugunuz tabainin üst ile islem sonu => [%d ^ %d = %d] ", a,b, f1(a,b));  
    return 0;  
}
```

1 den 10 na kadar olan tüm sayıların faktoriyellerinin bulunumunu sağlayan bir fonksiyon:::

```
#include<stdlib.h>  
#include<stdio.h>  
  
int f1(int x=1){  
    int i, r=1;  
  
    for(i=1;i<=10;i++){  
        while(r<=i){  
            x *=r;  
            r++;  
        }  
        printf("\n\n%d. sayinin faktoriyeli --> %d", i,x);  
        x=1;  
        r=1;  
    }  
}
```

```
// bu şekilde iki farklı fonksiyon kullanımını göstermiş oldum.

int main(){
    int a,b;

    printf(":::birden ona kadar olan sayıların her birinin faktoriyelleri:::");
    f1(a);

    return 0;
}
```

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

int f1(int x){
    int i, r=1;

    for(i=1;i>0;i++){
        printf("lütfen bir değer giriniz:::");
        scanf("%d", &x);
        if(x<0){
            printf("negatif bir değer girdiniz bu problem oluşturacak!");
            break;
        }
    }
}

int main(){
    int a,b;

    f1(a);

    return 0;
}
```

!!! SCNF KULLANIRKEN ADRELSME YAPMAYI UNUTMA !!!

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

void f1(int m){
    int i=0,a2=1,a1=0;
    int son=0;

    for(i=0;i<=m;i++){
        if(m<=1){
            son=i;

            printf(" %d. sayı %d",i+1,son);
        }

        else{
            son= a1+a2;
            a1 = a2;
            a2 = son;
            // mantık bier sağa kayma olarak ilerliyor. (ben aynı mantığı dizi indexlerini birer birer artırmak olarak düşünmüştüm ama bu d
            printf(" %d. sayı %d",i+1,son);
        }
    }
}
```

```
}

int main(){
    int x;
    printf("kac adet fibonacci sayisi bastirilacagini yaziniz::::");
    scanf("%d", &x);
    f1(x);

    return 0;
}
```

→ fibonacci sayı dizisi senaryosu budur....

bir floyd üçgeni senaryosu denemek istedim.

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

void f1(int m){
    int i=1;
    int j,a;
    a=1;

    while(i<=m){ // ilk while döngüde alt satırı atlamak için alındı.

        j=i;
        while(j<=i){

            printf("%d\t", a);
            a++;
            j++;

        }

        printf("\n\n");
        i++;
    }
}

int main(){
    int n;
    printf("lutfen sayılardan oluşan üçgen için satır sayısını giriniz :::");
    scanf("%d", &n);
    f1(n);

    return 0;
}

// benze örnekler yapmış olsakda daha karşılaştırma olmadan küçüklük ve eşitleme kullanarak ifade şekli bu fonksiyonda değerli.
```

time.h kütüphanesi ve random değer alma işlemlerinde içeren gayet güzel bir min max senaryosu.

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

void f1(int diq[], int k){
    int m=0;
    int max=0;
    while(m<=k){
        if(diq[m]>max){
            max=d iq[m];
        }
    }
}
```

```

        m++;
    }
    printf("random secilen sayilardaki max deger :::: %d", max);

}

void f2(int dif[], int s){
    int m=0;
    int min=1000;
    while(m<s){
        if(dif[m]<min){
            min=dif[m];
        }

        m++;
    }
    printf("\n\nrandom secilen sayilardaki min deger :::: %d", min);
}

int main(){
    int x;
    int i;
    printf("lutfen dizinizdeki sayı sayısını giriniz ::: ");
    scanf("%d", &x);
    int dizi1[x];
    srand(time(0));
    for(i=0;i<x;i++){
        dizi1[i]=rand()%100;
    }

    f1(dizi1, x);
    f2(dizi1, x);

    return 0;
}

```

→ burad aasıl önemli nokta dizili ifadeleri nasıl fonksiyon ile birlikte gönderme - atama yapıyoruz dikkat.

fonksiyonun ilgili alanında köşeli parantez ile belirtilirken, fonksiyonla çağrıma kısmında köşeli parantez ile bekirtme kullanılmaz!!!!

anasını yaaaa

örnekler baya matrak olmaya başladı bu durum hoşuma gidiyorrr

dizideki belirtilen bölgmedeki sayıyı silmeye yarayan bir senaryo geliştirelim.

```

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
// gösterilecek silme fonksiyonu aslında diziyi birer birer kapatmasını sağlamakla yapıldı.

void f1(int A[], int k){
    int i;
    int s;
    printf("\n\nlutfen silinecek sayının sırasunu girinizz :::");
    scanf("%d", &s);

    for(i=0;i<k;i++){
        A[s-1]=A[s];
        s++;
    }
    for(i=1;i<k;i++){

        printf("%6d", A[i-1]);
    }
}

```

```

int main(){
    int x;
    int i;
    printf("lutfen dizinizdeki sayı sayisini giriniz ::: ");
    scanf("%d" ,&x);
    int dizi1[x];
    srand(time(0));
    for(i=0;i<x;i++){
        dizi1[i]=rand()%1000;
    }
    for(i=0;i<x;i++){
        printf("%d \n", dizi1[i]);
    }
    f1(dizi1, x);

    return 0;
}

```

belirtilen yerdeki ifadenin birer kaydırarak işlem yaptırtıp son terimdeki kayma 0 olacak ve onuda printfte görünmeyecek şekilde yazdırırmak.

Vay Can BOZ deli adamsın ya!

bu örnek üzerinde biraz duracaağımız:::

```

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

void f1( int k){
    int n=1;
    int i,a;
    int j=k-1;
    int bosluk=1;
    for(i=0;i<=k-1;i++){

        bosluk=1;
        while(bosluk<=j){
            printf(" ");
            bosluk++;
        }
        j --;

        for(a=1;a<=2*n-1;a++){
            printf("c");
        }
        n++;
        printf("\n");
    }
}

void f2(int m){
    int i,j;
    int k=m-1;
    int bosluk=1;

    for (i=1;i<=m-1;i++){

        for(j=1;j<=bosluk;j++){
            printf(" ");
        }
        bosluk++;
        for(j=1;j<=2*k-1;j++){
            printf("x");
        }
    }
}

```

```

        k--;
        printf("\n");
    }

// birer arttırılacak veya bu şekildeki adımlı looplarda for kullanılması çok avantajlı.
// miin bir fonksyon gibi davranış gereksiz yere çok fazla değişkene gerek kalıyor


int main(){
    int x;
    int i;
    printf("lutfen satır sayısını giriniz :::");
    scanf("%d" ,&x);
    f1(x);
    f2(x);

    return 0;
}

```

- buradaki for looplarına dikkat etmek istiyorum. Çok değerli ;
 - fazladan değişken kullanmamızı ve extradan da yapmak istediğimiz yapının anlaşılmasında kilit rol oynadı.
 - yerel değişkenle for döngüsünü bizden istenen (bir baklava deseni çizilmesi idi.) ifadeyi satır sayısını kontrol ederek bağladık.
 - burdak konan boşluk sayısı vs. gayet güzel ve iç içe bağlanan for yapılarında
 - ilk forla sınır da (narrow) çizildiği için tam olarak istediğimiz duruma ulaştık diyebilirim.
- tekrar tekrar incelenmesinde gerçekten yararlı bir program diyebiliriz.**
-

Lecture 10 -P-

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

Pointer (İşaretçiler)

aslında C nin temelinnde olan bir yapıdır.

bildiğimiz gibi hafızada belli adresleme ile birlikte scanf,, veya fonksiyonlarda adreslemeyle istediğimiz yere atama veya çağrıma işlemleri gerçekleştiriyorduk.

işte burada adresi tutan değişkenlere **pointer** denir.

Pointer'ın türü işaret ettiği değişkenin türünü anlamamızı sağlar.

- int* türündeki bir pointer int türüne işaret eder.

variable pointed === pointer'ın adresinde tuttuğu değişkeni işaret eder.

öncekte göstermesi daha kolay:::

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

int main(){
    int *sayi;
    int n=10;

    printf("%d\n\n", &n);
    // ampersan ile direkt adresi yazdırılmış oldum. DEMEKKİ BENİM 10 İLE ATADIĞIM DEĞER BU ADRESTE KAYITLI.
    sayi= &n;

    printf("%d", *sayi);

    // çıktıda direkt n e atadığım değeri gösterdi.

    return 0;
}

çıkıcı>>>
6487580
10
```

→ burada şunu öğrenmiş oluyoruz.

adreslemeyi atadığımızı değeri yıldızla adresten çekme işlemi yaptırdığımızda o adresteki değeri döndürmüştür.

ve yıldızla tanımlama yaptığımızda direkt değerin kendisi olmasada türüne göre bir yer ayrılmış oluyor.

bu mantığın en yatkın tarafı direkt adresleme ile olan fonksiyonlardadır.
direkt pointee mantığı buna dayanır.

Pointerlarda İşlemler

adresin değişmeyip içerisindeki değerin değiştiğinin üzerine yeniden bir senaryo geliştirelim.

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

int main(){
    int *sayi, *sayi2;
    int n=10, j=20;

    sayi=&n;
    sayi2=&j;

    *sayi = *sayi+*sayi2;

    // *sayi yazan adresdeki değer değişmiş olacak ama adres aynı kalmış olacaktır.
    printf("\n\n%d", *sayi);

    return 0;
}

çıkıcı>>
30
```

Adres değişimiyle global değişken değeri değişimi (fonksiyonlar)

→ aslında herhangi bir şekilde bir kargaşa filan yok.

Hafıza içerisinde zaten her değişkenin değeri tutulmakta (ister global ister yerel)

ve biz adresleme ile belirtili yere gönderme yaparsak eğer... (adresleme metodlarında bunu yapıyorduk.) otomatikman oadrsteki değeri değiştirmiştir oluruz.

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

void f1(int *de1, int *de2){
    *de1 = 789;
    *de2 = 566;
}
```

```

int main(){
    int n=10, j=20;
    printf("fonksiyondan önceki belirtilen adreslerdeki degiskenlerin degerleri;;;; %d %d", n,j);
    f1(&n, &j);

    printf("\n\nfonksiyondan sonraki belirtilen adreslerdeki degiskenlerin degerleri;;;; %d %d", n,j);
    return 0;
}

çikti>>
fonksiyondan önceki belirtilen adreslerdeki degiskenlerin degerleri;;;; 10 20
fonksiyondan sonraki belirtilen adreslerdeki degiskenlerin degerleri;;;; 789 566

```

klasik bir adreslemeyle değişim örneği .

Kısaca adresleme ile direkt global değişkenlere müdahale etmiş olduk.

→ adresleme olmadan normal bir fonksiyon olarak yazsaydık eğer bu sadece yerel bir değişkenlikte kalmış olurdu.

ÖRNEK

Aşağıdaki programın çıktısını gösteriniz.

1. short x,y,*p1,*p2;
2. x = 4;
3. p1 = &x;
4. y = *p1;
5. y = 8;
6. p2 = &y;
7. (*p1)++;
8. y = p1;
9. y++;
10. p2 = 12;
11. p1 = y;

SATIR	x	y	p1	*p1	p2	*p2
1	-	-	NULL	-	NULL	-
2	4	-	NULL	-	NULL	-
3	4	-	4	4	NULL	-
4	4	8	4	8	NULL	-
5	4	8	4	8	24	8
6	4	8	4	8	24	8
7	5	8	5	8	24	8
8	5	8	5	8	24	8
9	5	8	5	8	24	8
10	5	8	5	8	24	8
11	5	8	5	8	12	5
12	5	8	5	8	12	5
13	5	8	5	8	12	5
14	5	8	5	8	12	5
15	5	8	5	8	12	5
16	5	8	5	8	12	5
17	5	8	5	8	12	5
18	5	8	5	8	12	5
19	5	8	5	8	12	5
20	5	8	5	8	12	5
21	5	8	5	8	12	5
22	5	8	5	8	12	5
23	5	8	5	8	12	5
24	5	8	5	8	12	5
25	5	8	5	8	12	5
26	5	8	5	8	12	5
27	5	8	5	8	12	5
28	5	8	5	8	12	5
29	5	8	5	8	12	5
30	5	8	5	8	12	5
31	5	8	5	8	12	5
32	5	8	5	8	12	5
33	5	8	5	8	12	5
34	5	8	5	8	12	5
35	5	8	5	8	12	5
36	5	8	5	8	12	5
37	5	8	5	8	12	5
38	5	8	5	8	12	5
39	5	8	5	8	12	5
40	5	8	5	8	12	5
41	5	8	5	8	12	5
42	5	8	5	8	12	5
43	5	8	5	8	12	5
44	5	8	5	8	12	5
45	5	8	5	8	12	5
46	5	8	5	8	12	5
47	5	8	5	8	12	5
48	5	8	5	8	12	5
49	5	8	5	8	12	5
50	5	8	5	8	12	5
51	5	8	5	8	12	5
52	5	8	5	8	12	5
53	5	8	5	8	12	5
54	5	8	5	8	12	5
55	5	8	5	8	12	5
56	5	8	5	8	12	5
57	5	8	5	8	12	5
58	5	8	5	8	12	5
59	5	8	5	8	12	5
60	5	8	5	8	12	5
61	5	8	5	8	12	5
62	5	8	5	8	12	5
63	5	8	5	8	12	5
64	5	8	5	8	12	5
65	5	8	5	8	12	5
66	5	8	5	8	12	5
67	5	8	5	8	12	5
68	5	8	5	8	12	5
69	5	8	5	8	12	5
70	5	8	5	8	12	5
71	5	8	5	8	12	5
72	5	8	5	8	12	5
73	5	8	5	8	12	5
74	5	8	5	8	12	5
75	5	8	5	8	12	5
76	5	8	5	8	12	5
77	5	8	5	8	12	5
78	5	8	5	8	12	5
79	5	8	5	8	12	5
80	5	8	5	8	12	5
81	5	8	5	8	12	5
82	5	8	5	8	12	5
83	5	8	5	8	12	5
84	5	8	5	8	12	5
85	5	8	5	8	12	5
86	5	8	5	8	12	5
87	5	8	5	8	12	5
88	5	8	5	8	12	5
89	5	8	5	8	12	5
90	5	8	5	8	12	5
91	5	8	5	8	12	5
92	5	8	5	8	12	5
93	5	8	5	8	12	5
94	5	8	5	8	12	5
95	5	8	5	8	12	5
96	5	8	5	8	12	5
97	5	8	5	8	12	5
98	5	8	5	8	12	5
99	5	8	5	8	12	5
100	5	8	5	8	12	5
101	5	8	5	8	12	5
102	5	8	5	8	12	5
103	5	8	5	8	12	5
104	5	8	5	8	12	5
105	5	8	5	8	12	5
106	5	8	5	8	12	5
107	5	8	5	8	12	5
108	5	8	5	8	12	5
109	5	8	5	8	12	5
110	5	8	5	8	12	5
111	5	8	5	8	12	5
112	5	8	5	8	12	5
113	5	8	5	8	12	5
114	5	8	5	8	12	5
115	5	8	5	8	12	5
116	5	8	5	8	12	5
117	5	8	5	8	12	5
118	5	8	5	8	12	5
119	5	8	5	8	12	5
120	5	8	5	8	12	5
121	5	8	5	8	12	5
122	5	8	5	8	12	5
123	5	8	5	8	12	5
124	5	8	5	8	12	5
125	5	8	5	8	12	5
126	5	8	5	8	12	5
127	5	8	5	8	12	5
128	5	8	5	8	12	5
129	5	8	5	8	12	5
130	5	8	5	8	12	5
131	5	8	5	8	12	5
132	5	8	5	8	12	5
133	5	8	5	8	12	5
134	5	8	5	8	12	5
135	5	8	5	8	12	5
136	5	8	5	8	12	5
137	5	8	5	8	12	5
138	5	8	5	8	12	5
139	5	8	5	8	12	5
140	5	8	5	8	12	5
141	5	8	5	8	12	5
142	5	8	5	8	12	5
143	5	8	5	8	12	5
144	5	8	5	8	12	5
145	5	8	5	8	12	5
146	5	8	5	8	12	5
147	5	8	5	8	12	5
148	5	8	5	8	12	5
149	5	8	5	8	12	5
150	5	8	5	8	12	5
151	5	8	5	8	12	5
152	5	8	5	8	12	5
153	5	8	5	8	12	5
154	5	8	5	8	12	5
155	5	8	5	8	12	5
156	5	8	5	8	12	5
157	5	8	5	8	12	5
158	5	8	5	8	12	5
159	5	8	5	8	12	5
160	5	8	5	8	12	5
161	5	8	5	8	12	5
162	5	8	5	8	12	5
163	5	8	5	8	12	5
164	5	8	5	8	12	5
165	5	8	5	8	12	5
166	5	8	5	8	12	5
167	5	8	5	8	12	5
168	5	8	5	8	12	5
169	5	8	5	8	12	5
170	5	8	5	8	12	5
171	5	8	5	8	12	5
172	5	8	5	8	12	5
173	5	8	5	8	12	5
174	5	8	5	8	12	5
175	5	8	5	8	12	5
176	5	8	5	8	12	5
177	5	8	5	8	12	5
178	5	8	5	8	12	5
179	5	8	5	8	12	5
180	5	8	5	8	12	5
181	5	8	5	8	12	5
182	5	8	5	8	12	5
183	5	8	5	8	12	5
184	5	8	5	8	12	5
185	5	8	5	8	12	5
186	5	8	5	8	12	5
187	5	8	5	8	12	5
188	5	8	5	8	12	5
189	5	8	5	8	12	5
190	5	8	5	8	12	5
191	5	8	5	8	12	5
192	5	8	5	8	12	5
193	5	8	5	8	12	5
194	5	8	5	8	12	5
195	5	8	5	8	12	5
196	5	8	5	8	12	5
197	5	8	5	8	12	5
198	5	8	5	8	12	5
199	5	8	5	8	12	5
200	5	8	5	8	12	5
201	5	8	5	8	12	5
202	5	8	5	8	12	5
203	5	8	5	8	12	5
204	5	8	5	8	12	5
205	5	8	5	8	12	5
206	5	8	5	8	12	5
207	5	8	5	8	12	5
208	5	8	5	8	12	5
209	5	8	5	8	12	5
210	5	8	5	8	12	5
211	5	8	5	8	12	5
212						

11. koşulda biz adresindeki değişkeni sabit tutup adresin değerinde bir değişiklik yapabiliriz. ve değer aynı kalır.

10. koşulda ise direkt adrese bir atama yapılıyor. buradaki adreste değişkende bilinmez.

diziler ve pointer da anlayacağizzzz

Kritik noktalardan bir tanesi !!!!

::: Pointer ve Diziler :::

asında her dizi bir pointerdir ve her pointirda bir dizidir !!!

düşündüğümüz gibi!!!!

can bozun şu ekran alıntısı her şeyi anlatıyor...

The image shows a screenshot of a C code editor with handwritten annotations. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char can[100] = "Canan";
7     char *p1;
8
9     p1 = can;
10
11
12     printf("5. karakterim %c dir", can[4]);
13
14
15
16
17
18     return 0;
19 }
20
```

Handwritten annotations include:

- A red box highlights the string "Canan" in memory, with addresses 1005, 1006, 1007, 1008, 1009, and 1010 written below it.
- An arrow points from the variable declaration "char can[100]" to the start of the string in memory.
- A red box highlights the assignment "p1 = can;" with the address 1005 written above it.
- An arrow points from the pointer variable "p1" to the start of the string in memory.
- A red box highlights the expression "*(p1+4)" with the address 1009 written above it.
- A red box highlights the character 'n' in the string, with the address 1008 written above it.

p1=can; ataması direkt p1 i can[0] adresine eşitler. ve her char 1 byte kadardır.

eğer bizzz bu adres değerini bir arttırırsak...??!?

o zaman diğer indexlerdeki değerlerin bilgisini çekebiliriz anlamına gelir.

ve atanmış değerde pointer mantığını kullanarak mesela,,,

*(p1+4) dersek eğer direkt 1006 adresi ile başlayan dizide 1010. adres değerindeki indexe gidip o değeri çekmiş olacaktır.
ve ekrana n değerini bastırmış olacak.

işte bizim başından beri kullandığımız dizi özelliklerini farkında olmadan nasıl çalıştığını anlatan olay budur!!!!

sabit bir hafızadaki byte ayarlamasına gidip ilgili değere müdahale etmek.....

hackerin bakiş açısı için bile gerçekten çok değerli.
developerin düşünücülerini geçtim bile.

senaryoyu birde biz deneyelim.

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

int main(){
    char t[100]="anan";
    char *p;

    p=t; // p değeri direkt t'nin 0. indexinin adresine eşitlenmiş oldu. adresi 3 arttırıp * ifadesiyle n değerine ulaşmaya çalışalım.

    printf("3. indexin degeri %c dir::", *(p+3));

    return 0;
}

çıkıcı>>
3. indexin degeri n dir::
```

burada dikkat edilmesi gereken bir husus var.

;;;; p değerinin char tanımlaması ;;;;

çünkü hatırlatmak gerekirse biz diğer pointer yazılarda *p değerini sisteme tanıtmıştık (BU ÖNEMLİ YOKSSA HATA VERİR.)

örneği iyi incelemek lazım gerçekten pointer ile dizi arasındaki çalışma prensibini anlamak için birebir bir örnek.

Pointer aritmetiği !!!

baya değerli bir ders

dizilerde bir arttırarak pointerlarda diğer indexleri bulmuştuk.

burada şunu söylemek lazım. eğer 1 indexte 1 den büyük byteda olsa(int, float, double vs gibi)..

pointerda bir artırma direkt diğer indexin adresine gidecektir.

byte miktarı fark etmeksızın pointerda 1 artırma direkt diğer adrese gider.

bu pointer aritmetığının kendi içindeki çalışma şeklidir. ki (pratiksel anlamda mantıklı bir atlama (çünkü her byte da dikkat etsek bu sefer büyük yapılı uygulamalardır baya index atlarken zaman kaybı yaşayabiliridir))

bir senaryo üzerinde direkt görelim.

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

```

int main(){
    int dizi2[]={1,2,3,4,5,345,34,543,534};

    int *k;
    k=dizi2;

    // k = &dizi2[0]

    -> bu şekilde adresleme yapabilirdim(bana sorulacak olursa ilki daha kolay.)

    printf("%d 7.karakter ", *(k+6));


    return 0;
}
çıktı>>>
34 7.karakter

```

görüldüğü gibi .. integer formatı bir byttan yüksek bir alan alsada pointer içinde bir arttırmı direkt bir index atlatmış oldu .

.....

pointer kendi içinde zaten o byte farkını çarparak diğer indexe geçiyor.

bide biz bunu kendimizde eklersek sonuç istediğöiz gibi gelmeyecektir.

hatta bu durumu görelim.

```

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

int main(){
    int dizi2[]={1,2,3,4,5,345,34,543,534};

    int *k;
    k=dizi2;

    printf("%d\n\n", k+1);
    printf("%d", k);

    çıktı>>>
6487540
6487536

// gayet kolay şekilde kendisi işlem yaptı ve 4 byte kadar otomatik kendisi yaptı!

```

for döngüsüne direkt pointer kullanarak atama gösterelim.

```

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

int main(){
    int k[1000];
    int *pt;

```

```
pt = k;// bunu direkt for dongüsünün içinde de söyleyebildik
srand(time(NULL));
for(pt;pt<k+100;pt++){
    *pt=rand()%100;
    printf("%d \n\n ", *pt);
}
return 0;
}
```

bu sayede k içinde 100 tane random sayı girilmiş oldu.

Lecture 11 (Pointer exercise)

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

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

int main(){
    char k[1000];
    printf("lutfen bir cümle giriniz:::");
    gets(k);
    char *p1;
    char *p2;
    int kontrol=1;
    for(p2=k;p2;p2++);
        // buradaki yazım koşuluna dikkat. direkt dizinin elemanını verip işlem yapmak onnu zaten son değere kadar götürecekтир.
        // buda bize son elemanı bulmak konusunda çok işimize yarar.
        // eğer karakter dizisi ise /0 i alacaktır o yüzden bir azaltmak lazım.
        p2--;
    for(p1=k;kontrol && p2>p1;p1++, p2--){
        if(*p1 != *p2){
            kontrol =0;
            printf("\n\nnifade polindrom falan degildir.");
            break;
        }
    }
    if(kontrol==1){
        printf("\n\n%s ;; aldiginiz cümle polindrom bir ifadedir.", k);
    }
    return 0;
}
```

Burada polindrom mu değil mi bir cümle onu test edip öğrendik.

ama buradaki syntax ve mnatık örgüsü aşşırı derecede önemli.

lutfen forları çalıştırırken eşitleme olmadan ve direkt rahatlıkla yapılan yazımlara dikkat.

dizinin avnatajlarını ve dizi içine birden fazla pointerla nasıl değerlendirilmiş bu örneği tekrar incelemek gerekir.

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

```
int main(){
    printf("lutfen 2 tane sayı girinizzz;;");
    int s1,s2;
    int *a,*b;
    // scanfte biz zaten sayıyı isterken aslında belli bir adresde değer verilmesini sağlıyoruz. bunun yerine direkt pointer kullanarak
    a=&s1;
    b=&s2;

    scanf("%d%d", a, b );
    printf("---- %d \n\n2-- %d", *a, *b);
    int *gecici;
    gecici=b;
    b=a;
    a=gecici;
    printf("\n\n\n");
```

```

printf("1-- %d \n\n2-- %d", *a, *b);

return 0;
}

```

→ Buradaki geçici eşitlemesine dikkat. (klasik bir sayı değiştirme örneği.)

pointer olmadan ikincil eşitlemelerde hata aldığımızı unutmuyalım.

bu kritik bir konu diye düşünüyorum

.....

bunu adres ile fonksiyon değerlendirmesiyle göstereлим bide.

```

#include<stdlib.h>
#include<stdio.h>
void f1(int *a, int *b){
    int *gecici;

    gecici=b;
    b=a;
    a=gecici;
    printf("\n\n1-- %d \n\n2-- %d", *a, *b);
}

/// adresleme le yaptığım işlemde nedense bunu kalıcı görmüyor. heralde void f1(.....) de belirttiğim değerler direkt * pointer ifadel
/// adres olarak işlem yaptığımda tanımlı olmadığı için belkide yerelde değişiklik kalıyor olabilir.

int main(){
    printf("lutfen 2 tane sayı girinizzz;;");
    int s1,s2;

    // scanfte biz zaten sayıyı isterken aslında bir asderece değer verilmesini sağlıyor. bunun yerine direkt pointer kullanarak
    scanf("%d%d", &s1, &s2);

    printf("1-- %d \n\n2-- %d", s1, s2);

    f1(&s1, &s2);

    //printf("\n\n1-- %d \n\n2-- %d", s1, s2);
    return 0;
}

```

... gerçekten güzel bir örnek.

pointer kullanarak bence çok güzel olan bir kullanıcıdan dizi alma ve yazdırma örneği;

```

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

int main(){
    int x;
    int i=1;
    int a;
    printf("lutfen dizinizin eleman sayısını giriniz girinizzz;;");
    scanf("%d", &x);
    int dizi[x];
    int *pt;

    for(pt=dizi;pt<dizi+x;pt++){
        printf("lutfen %d. karakteri giriniz::: ", i);
        scanf("%d", pt);
    }
}

```

```

        i++;
    }

    printf("\n\n\n");
    for(a=0;a<x;a++){

        printf("%d\t", dizi[a]);
    }

    return 0;
}

```

Elemanları rastgele seçilip kullanıcı tarafından toplam karakter sayısı belirlenen dizinin başka bir diziye kopyalandığı yapıyı pointer kullanarak bir senaryodur.

pointer kullanımın diziye kattığı esneklik muazzam bir şey!!!!

```

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

int main(){
    int x;
    int i;
    int a;
    printf("lutfen dizinizin eleman sayisini girinizzz;;");
    scanf("%d", &x);
    int dizi[x];
    int dizi2[x];
    int *pt;
    srand(time(0));
    printf("1.dizinin elemanları:::  ");
    for(i=0;i<x;i++){
        dizi[i]=rand()%1000;
        printf("%d\t", dizi[i]);
    }

    pt=dizi;
    for(i=0;i<x;i++,pt++){
        dizi2[i]=*pt;

    }
    printf("\n\n\n");
    printf("2.dizinin elemanları:::  ");
    for(i=0;i<x;i++){
        printf(" %d\t ",dizi2[i] );
    }

    return 0;
}

```

kullanıcının aradığı değeri kendi girdiği dizi içinde bulmasını sağlayan bi senaryodur

```

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

```

```

int main(){
    int n;
    printf("girmek istediginiz dizinin eleman sayisini giriniz:::");
    scanf("%d", &n);
    int dizi[n];
    int i;
    int *pt;

    pt=dizi;

    for(i=0;i<n;i++){
        printf("\nLutfen %d. karakteri giriniz ::: ",i+1);
        scanf("%d", &dizi[i]);
    }

    int k;
    int kontrol=0;
    printf("\n\nNaradiginiz elemanın değerini giriniz:: ");
    scanf("%d", &k);

    for(i=0;i<n;i++){
        if(k==*pt){
            printf("\nNaradiginiz değer dizinin %d. elemanidir.", i+1);
            kontrol=1;
        }
        pt++;
    }

    if(kontrol ==0){
        printf("\nNaradiginiz eleman bu dizi içinde bulunmamaktadir.");
    }

    return 0;
}

```

bizim daha çok önceki örneklerdeki fonksiyon ile adresleme örnekleri üzerinde durmamız önemli olacaktır.

pointer kullanarak bir tane faktoriyel bulma senaryosu bulunmasını istedim.

```

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

int f1(int *sayi){
    int i=*sayi;
    int h=1;
    for(i;i>0;i--){
        h *= i;
    }
    return h;
}

int main(){
    int a;
    int k;
    printf("Faktoriyelin alınmasını istediğiniz değeri giriniz:: ");
    scanf("%d",&a);

    k=f1(&a);

    printf("%d! => %d", a, k);
}

```

```
    return 0;
}
```

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

void yerdegistirme(int *arr, int *u){
    int s=0;
    int *k;
    int gecici;
    k=arr+*u-1;
    for(arr;arr<k;arr++, k--){
        gecici= *k;
        *k=arr;
        arr=gecici;
    }
}

int main(){
    int a,i;
    int *pr;
    int *endpt;
    printf("dizinizin eleman sayiini giriniz:: ");
    scanf("%d",&a);
    int dizi[a];

    pr=dizi;
    for(i=0;i<a;pr++, i++){
        printf("\n\nlutfen %d. pozisyonu giriniz::: ",i);
        scanf("%d", pr);

    }
    for(i=0;i<a;i++){
        printf("\n%d",dizi[i]);
    }

    yerdegistirme(dizi, &a);
    printf("\n\n\n");
    for(i=0;i<a;i++){
        printf("%d\t",dizi[i]);
    }

}
return 0;
}
```

bu pointer kullanarak güzel bir dizi içinde yer değiştirme senaryosu örneği efnm;;;;

hem pointerli hemde pointersız dizi uzunluğu gösteren bir örnek girelim.

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

```

int main(){
    int x=0,i=0;
    char dizi1[100];
    char *pt;
    printf("lutfen bir karakter dizisi giriniz::::");
    gets(dizi1);
    pt=dizi1;

    while(dizi1[i]){
        i++;
    }
    printf("girmis oldugunuz dizi uzunluğu %d kadardir.", i);

    // deemek kiiii while dongusunde direkt degerin kendisi dönüyor pointer ilede bir ifade deneyebilirizzzz
    printf("\n\n\n");
    printf("\n\n\n");
    printf("\n\n\n");

    while(*pt){
        pt++;
        x++;
    }

    printf("%d kadar eleman vardir dizide", x);

    //buda pointer kullanarak bir örrrrnektir.

    return 0;
}

```

burda eleman varlığı ve yokluğu mantığını kullanarak bir fikir geliştirip bunun çerçevesinde ilerledik.
hem normal hem de pointer kullanarak gösterimde bulundum.

<string.h> kütüphanesi kullanarak yazdığımız gerçekten güzel bir silme örneği

```

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

void ksilme(char *arr, char *k){
    char *ptr;

    for(ptr=arr; *ptr; ptr++){
        if(*ptr==*k){

            strcpy(ptr,ptr+1);
            // BURDAKİ SILME İŞLEMİNE DİKKAT!!!!
            //çokça incelenmesi gereken bir örnektir.
        }
    }

    int main(){
        char a;
        char dizi[200];
        char *pt;

        printf("lutfen bir cümle giriniz::: ");

```

```

gets(dizi);
printf("\n\n");
printf("cumlede kaldırmak istediğiniz karakteri giriniz::: ");
scanf("%c", &a );
ksilme(dizi, &a);

printf("\n\n%s", dizi);

return 0;
}

```

→ burada önceki bir yer değiştirme kullanarak yaptığımiz silme işlemindeki problem burada ortadan kalktı . strcpy yapısını kullanarak direkt geçerli belirttiğimiz adresi [adresle işlem önemli!!!! çünkü var olmayan adresi direkt kopyalamadı buda bize son değerde avantaj sağladı.] pointer özelliğiyle çekti ve silinmesi istenen değerleri geçerli yerden kaldırılmış olduk.

→ for() döngüsüne elzem bir dikkat gereklidir.

aynı while da olduğu gibi dizideki geçerli yere kadar ilerler ve \0 i gördükten sonra döngü durur.

→ diğer sayı dizilerinde de benzer şekilde pointerden yararlanılarak diiz boyunca for döngüs işlemi uygulanabilir.
(kütüphane kullanmaktan çekinmemek lazımdır. amaaa bu bizim düşünmemiz önünde engel olmamalıdır.)

tek ve çift sayılar üzerine br pointer senaryosu;;

```

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

int main(){
    int dizi[21]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

    int i;
    int *p;
    printf("normal array :::: ");
    for(i=0;i<21;i++){
        printf("%4d", dizi[i]);
    }
    printf("\n\n\n");
    printf("array with it's dual numbers :::: ");
    for(p=dizi;p<dizi+21;p+=2){
        printf("%4d", *p);
        // Öğrenmiş olduk ... >>> for(ptr=dizi;*ptr;ptr++) <<< yapısı char tipi diziler için geçerli \0 dan kaynaklanan bir özellik çünkü
        //Buda sadece karakter dizileri için geçerli bir özellik
    }

    printf("\n\n\n");
    printf("array with it's single numbers :::: ");
    for(p=dizi+1;p<dizi+21;p+=2){
        printf("%4d", *p);
    }
}

return 0;
}

```

→ pointerin davranışlarını anlamak için birebir örnek.

for döngüsü kuralları pointer ve adresleri içinde geçerli. bunu çok pis kullanıriz😊.

→ kod içindeki maddeye dikkat!!!

dizi türlerine göre kullanım standartlarını gösteriyor.

```

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

int main(){

    int a,b;
    int *pt;
    int *pr;
    int i=0;
    printf("lutfen 1. dizinin eleman sayisini girinizz;;; ");
    scanf("%d", &a);

    printf("\n\nlutfen 2. dizinin eleman sayisini girinizz;;; ");
    scanf("%d", &b);
    int A[a];
    int B[a+b];

    printf("\n\n");

    for(pt=A;pt<A+a;pt++){

        printf("\nlutfen %d. elemani giriniz:: ", i+1);
        scanf("%d", pt);
        i++;
    }

    printf(":::: 2. dizinin elemanlarinin alınması :::: ");

    printf("\n\n");
    i=0;
    for(pr=B;pr<B+b;pr++){

        printf("\nlutfen %d. elemani giriniz:: ", i+1 );
        scanf("%d", pr);
        i++;
    }
    i=0;
    pt=A;
    //
    for(pr=B+b;pr<B+b+a;pr++){

        *pr=*pt;
        pt++;

    }
    //
    for(pr=B;pr<B+b+a;pr++){
        printf("%d", *pr);
    }

    return 0;
}

```

dizi birleştirilmiş pointerli çoksel bir örnek diye düşünüyorum üzerinde değişiklik yapıp yeniden bunu ele alacağım.

pointerlarda iki boyutlu dizi kullanarak bir tersine çevirme senryosu.
bayadır baktık 2 boutlu dizilere gerçekten iyi oldu .

→ Hatırlatma::: char ikinci dereceden dizilerinde ilk köşeli parantez kaç tane içinde ayrı ayrı sell olduğunu, ikincisi ise total karakter sayısının bilgisidir.

```

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

int main(){
    char dizi[5][60];
    int i;
    char *ptr1;
    char *ptr2;
    char gecici;
    for(i=0;i<5;i++){
        printf("\n%d. kelime ::: ", i+1);
        scanf("%s", &dizi[i]);
    }

    for(i=0;i<5;i++){

        ptr1=ptr2=dizi[i];
        printf("\n\n");
        while(*ptr2){
            *ptr2++;
        }
        *ptr2--;
        while(ptr1<ptr2){
            gecici=*ptr1;
            *ptr1=*ptr2;
            *ptr2=gecici;
            ptr1++;
            ptr2--;
        }

    }

    for(i=0;i<5;i++){

        printf("\n%d. kelime ::: ", i+1);
        printf("%s", *(dizi+i));
    }

    return 0;
}

```

→ bu örnekler kısmının belkide en değerli örneklerinden biridir.

2li dizilerde pointer yapısının nasıl çalıştığını çok güzel bir kanıtı.

ve şunu gördüm::

char dizisinde her alana ayırdığımız 60 karakterlik 5 alan direkt adreslemedede görülüyor.

ve diziyi pointera eşitlerkenki duruma dikkat!

pointer dizinin belirli bölümü boyunca çalışmakta.

bu sayede zaten bir kelime boyunca yer değiştirmeyi gerçekleştirebilmiş olduk .

bunu belirterek pointer yapılarını kullanmamız gereklidir.

ki bunuda zaten for döngüsü ile her bir dizi içi dizilere (burada adlandırdığımız kelimelere) zaten direkt girmiş oluyor.

lütfen bu örneği tekrar inceleyelim çok mühimmm!

haftanın günlerini pointer yardımı ile buldurur ve değer girdiren genel bir senaryo::: 1 ile 7 arasında değer girilmediği zamanda yeniden istemelidir.

```

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

int main(){


```

```
char dizi[7][20]={"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};
int n;
char *pt;
do{
    printf("lutfen 1 ile 7 arasında bir sayı giriniz ::: ");
    scanf("%d", &n);

}while(n<1 || n>7);
printf("\n\n\n");

pt=dizi[n-1];
printf("-> ");
for(pt;*pt ;pt++){
    printf("%c", *pt);

}
printf(" <-");

return 0;
}
```

gerçekten öz, iki boutlu dizi bulunduran ve net bir senaryo.

:: pointer kapsamındaki örnekleri bitirmiş bulunmaktayız :::
struct ... seni dört gözle bekliyorum!

Lecture 12

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

Struct Yapısı Üzerine

struct adından mütevelliit yapı demektir.

birden fazla değişkeni içinde bulundurabilen ve başka yapılar içerisinde dahil olabilen bir yapıdır.

bu yönyle aslında class özellikleri yazılım dillerine benzerde diyebiliriz.

biner bir değişkenler topluluğunu içeren genel toparlayıcılar, alanlar belkide topluluklarda diyebiliriz.

syntaxını görelim...

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

int main(){

    struct tarih{
        int gun;
        char ay[12][10];
        int yil;

    };
    struct tarih evlilik_yildonumu;

    return 0;
}

-> buradada görüldüğü gibi bu struct yapısının adını belirtip buna göre hareket ettik.
```

struct yapısı sadece bir tane tanımlanmayabilir.

bir dizi içinde tutulabilirde.

görelim.

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

int main(){

    struct sehir{
        char sehir_adi;
        int plaka;
        int sicaklik;

    };
}
```

```
    struct sehir turkiye[81];

    return 0;
}
```

bu sayede dizini her indexinde struct yapısı olmuş olacaktır.

BİLGİLENDİRME ==> typedef komutu belirli bir tipi kendi tipimize dönüştürmeye yarar.
binevi isim etiketi gibi bir şey yani.

struct yapısı ile birlikte görebiliriz.

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

int main(){

    typedef float float2;

    struct prizma{

        float2 en;
        float2 boy;
        float2 yukseklik;
        float2 hacim;

    };

    struct prizma prizma_orn;

    // typedef bleirtip struct değerinde değiştirebilirdik.
    // typedef struct prizma prizma_genel_hesap;
    //prizma_genel_hesap prizma_1;
    bu şekilde de sadece alışık olduğumuz veri tipleri değil struct veri tipi üzerinde oynamaya yapabiliyoruz.

    return 0;
}
```

bahsettiğimde yine de bir tanımlama oluşturalım.

→ struct:: bir tanımlayıcı altında birbirinden farklı tiplerde veri saklamamızı sağlayan veri tipidir.

bir struct ::;

- ismi
- içindeki veri listesi (türleriyle birlikte)

isim tanımlanmayan direkt anonim struct yapıalarında vardır ama çok kullanılmaz dikkate gerek yok .

<<<<<<<<<<<<<<<<<<<<<<<<<

struktur yapılarına atama “.” kullanarak yapılır:

harcı olarak char ifadelerde strcpy kullanarak yapabiliyoruz.

.....

görelim

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

struct time{
    int day;
    char month[100];
    int year;
};

int main(){

    struct time year1;
    year1.day=19;
    strcpy(year1.month, "June");
    year1.year=2004;

    printf("%d -- %s -- %d", year1.day, year1.month, year1.year);

    return 0;
}
```

böyle olduğu gibi int main içindeki strcuk çağrıma sırasında struct değişkenlerinin sırası ilede tanımlama yapılabilir.

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

struct time{
    int day;
    char month[100];
    int year;
};

int main(){

    struct time year1={19, "June", 2005};

    printf("%d -- %s -- %d", year1.day, year1.month, year1.year);

    return 0;
}

// açıkçası bu kullanım diğerinden daha pratik olduğunu söylemeliyim.
```

→ bunlar struct yapısındaki genel atama şekilleri (.) yapısı ile çağrıınca otomatik işaretleme yapıyor zaten.
olay bu.

Öğrenci bilgileri atanmasına dair güzel bir örnek .

```

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

struct ogrenci_bilgi{
    char adi[50];
    char soyadi[50];
    int yasi;
};

int main(){

    struct ogrenci_bilgi ogrenci1={"hamza", "Aydin", 2013};
    // ogrenci.ad="sadasfsfasfsafs", ogrenci.soyad="ksamanscjnajsnc" şeklinde de atama yapabiliyoruz.
    // struct yapısı biraarada tutma ve ilişkilendirme konusunda bir yardımcı vardır.
    // bu sayede önceki öğrendiklerimizi direkt kullanabiliriz

    struct ogrenci_bilgi ogrenci2;
    ogrenci2= ogrenci1;

    printf("%s -- %s -- %d", ogrenci2.adi, ogrenci2.soyadi, ogrenci2.yasi);

    return 0;
}

```

>>> strcpy yapısı struct içerisinde karakter atamalarında baya bi kullanılan bir özelliktir. Nedense Can Boz bunu diğer atama şekilde daha çok kullanıyor.

koordinat boyunca nokta ataması yapan genel bir senaryodur.

```

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

typedef struct{
    float x;
    float y;

}point;

int main(){

    point nokta_1;
    printf("lutfen alınan noktanın koordinatlarını giriniz:: ");
    printf("\n\n");
    printf("x ekseninde >> ");
    scanf("%f", &nokta_1.x);
    printf("y ekseninde >> ");
    scanf("%f", &nokta_1.y);

    int i=nokta_1.x;
    int k=nokta_1.y;

    for(k;k>0;k--){
        printf("\n");
    };

    for(i;i>0;i--){
        printf("   ");
    }
}

```

```
    printf(".");
    return 0;
}
```

- typedef kullanımına dikkat. direkt fazladan struc kullanılmasının önüne geçmek için direkt adlandırma yoluna gidiliyor.
Başlarda karşılaşılan bir durum değildir ama zaman içinde gelişikçe genel büyük programlarda kullanılan bir yapı oluyor.
-

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

struct noktalar{
    char name[20];
    float x;
    float y;
};

int main(){
    int i;
    struct noktalar all[5]={
        {"birinci", 1.23232, 3.4323}, {"ikinci", 6.2312, 2.32}, {"ucuncu", 5.5435, 7.3563}, {"dorduncu", 8.4342, 1.99}, {"besinci", 3.7878,
    };

    for(i=0;i<5;i++){
        printf("%s ==> (%.3f) (%.3f)\n", all[i].name, all[i].x, all[i].y);
    }

    return 0;
}

çıkıcı>

birinci ==> (1.232) (3.432)
ikinci ==> (6.231) (2.320)
ucuncu ==> (5.543) (7.356)
dorduncu ==> (8.434) (1.990)
besinci ==> (3.788) (9.889)
```

- bir dizide atanan struct yapısında, atama ve çağrıma işlemlerinin nasıl yapıldığının çok güzel bir örneği.
. kullanarak nasıl çağrıma yapıyorsak dizi boyunca struc ifadelerini tutarakta benzer bir hareketi yapabiliyoruz.
-

Enumeration

- biraz daha özelleşmiş ve farklı işlemlere açık bir dizi yapısı diyebiliriz.
diziler gibi çalışır ve indexlere aynen 0 ile başlar.

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

int main(){

enum bolumler{

    pc,
    elektrik,
    biyoloji,
    fizik,
```

```

};

enum bolumler a=fizik;

printf("%d", a);

return 0;
}

çıktı>
3

```

→ Göründüğü gibi enumeration yapısı böyle işliyor.,

ama typedef kullanımı fazla fazla yazmaktan ziyade direkt hızlıca yazmka için bence daha ideal.
aynı yapıyı typedef kullanarak göstrereyim

```

int main(){

typedef enum bolumler{

    pc,
    elektrik,
    biyoloji,
    fizik,

}all;
all a=fizik;

printf("%d", a);

return 0;
}

```

bu sayede yeniden yeniden strcut enum yazmaktansa typedef ile isimlendirme yapıp program boyunca onu kullanmak daha işe yarar olabilir.

hem pratiklik hemde düzenlilik sağlayacaktır.

.....

enum içindeki belirttiğimiz değerlerde bir tanesini farklı bir değere eşitlersek ondan sonraki değerler ardışık şekilde o değerden iti barekn ilerler.

görelim.

```

int main(){

typedef enum bolumler{

    pc,
    elektrik=90,
    biyoloji,
    fizik,

}all;
all a=fizik;

printf("%d", a);

return 0;
}

çıktı>>
92

```

ve gösterdiğimiz gibi enumeration yapısını direkt değişken kullanarak çağrıabiliyoruz (zaten yazdığımız örneklerde de gördük) .

→ enum da dikkat edilmesi gereken yapı olmasıdır.

birkaç enumeration senaryosu görelim.

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

typedef enum aylar{

    ocak=1,
    subat,
    mart,
    nisan,
    mayis,
    haziran,
    temmuz,
    agustos,
    eylul,
    ekim,
    kasim,
    aralik,


}all;

void ay_goster(all x){

    switch(x){
        case ocak:
            printf("ocak");
            break;
        case subat:
            printf("subat");
            break;
        case mart:
            printf("mart");
            break;
        case nisan:
            printf("nisan");
            break;
        case haziran:
            printf("haziran");
            break;
        case temmuz:
            printf("temmuz");
            break;
        case agustos:
            printf("agustos");
            break;
        case eylul:
            printf("eylul");
            break;
        case ekim:
            printf("ekim");
            break;
        case kasim:
            printf("kasim");
            break;
        case aralik:
            printf("aralik");
            break;
        default:
            printf("problemli girisss");
            break;
    }
}

int main(){

    all k=4;
    ;
```

```

ay_goster(k);

return 0;
}

```

→ Can BOZ'un synatxinde int değişkeni yerleştirse bile çalışıyordu derleyiciler arasındaki farktan kaynaklanıyor diye düşünüyorum.

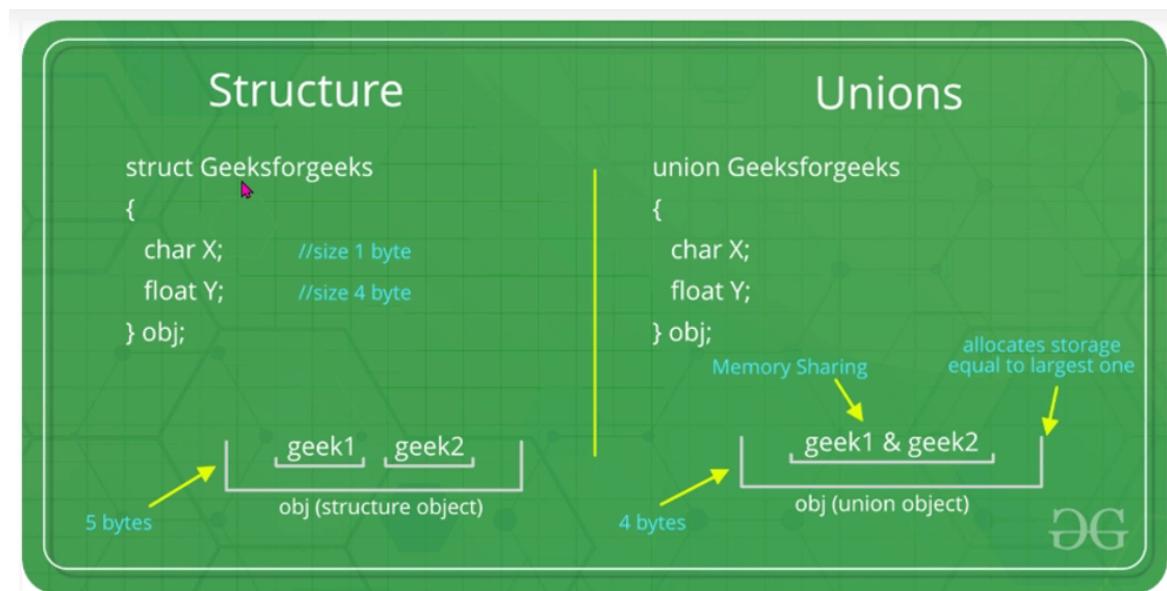
bzide direkt enum bir değişken olarak ifade ettiğimiz için heralde int ifadeye eşitlik istemiyor.

göründüğü şekilde çalışan bir dizi, int, struct gibi bir yapıya bakıyoruz.

UNION

union direkt olarak struct yapısı ile aynıdır.

sadece depolama yani alan konusunda farklı bir durum bulunmaktadır.



struct yapısında olmayan bir alan paylaşımı union içinde bulunmaktadır.

aslında addaki birleştirme hitabında zaten buradan gelmektedir. (alanı paylaşma, birleştirme gibi)

yazıldığı gibi max alan union içindeki max byte alan yapıya eşittir.

float yerine double kullanacak olsaydı 4 değil 8 byte gösterecekti.

union ismnini sonda vererek çalıştırabiliyoruz.

ve union dediğimiz bu data type kullanarak bir strcut yapısı oluşturursak;;;;

```

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

```

```

typedef struct arabalar{
    int durum;

    union{
        char marka[50];
        int fiyat;
    }info;
}araba_b;

int main(){
    araba_b a;
    a.info.fiyat=200000;
    printf("%d", a.info.fiyat);
    return 0;
}

```

yine . ifadesi kullanarak daha fazla ayrıntıdaki noktalar üzerine odaklanabiliyoruz.

→ strcuk içindeki ifadeleri biraz daha spesifikeştirip gruplandırmak istersem union bu konuda gayet iyi iş görüyor.
Bunu özellikle geniş örneklerde vs de görüyoruz.

struct kullanarak öğrenim bölümünün son örneğini biraz daha modifiye edip daha komplike ve eğlenceli hale getirdim.

```

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

typedef struct sayilar{

    float a;
    float b;

}sayilar;

void islemler(float m, float n, float k, float l){
    char f;
    printf("\nhangi işlem (+/-)::: ");
    scanf(" %c",&f);

    if(f=='+'|| f=='-'){
        switch(f){
            case '+':{
                if(n+l>0){
                    printf("sonuc >> %.1f+%.1fi", m+k, n+l);
                }
                else if(n+l==0){
                    printf("sonuc >> %.1f", m+k);
                }
                break;
            }
            case '-':{
                if(n-l>0){
                    printf("sonuc >> %.1f+%.1fi", m-k, n-l);
                }
                else if(n-l==0){
                    printf("sonuc >> %.1f", m-k);
                }
            }
        }
    }
}

```

```

    }
    else{
        printf("sonuc >> %.1f%.1fi", m-k, n-l);
    }

    break;
}

}
else{
    printf("yanlış operatör girdisi!!!!");
}

}

int main(){

sayilar sayi1;
sayilar sayi2;
printf("1. sayının lütfen gerçek kısmını giriniz::: ");
scanf("%f", &sayi1.a);
printf("\n1. sayının lütfen sanal kısmını giriniz::: ");
scanf("%f", &sayi1.b);
printf("\n2. sayının lütfen gerçek kısmını giriniz::: ");
scanf("%f", &sayi2.a);
printf("\n2. sayının lütfen gerçek kısmını giriniz::: ");
scanf("%f", &sayi2.b);

islemeler(sayi1.a,sayi1.b,sayi2.a,sayi2.b);

return 0;
}

```

güzelid...

→ union lar üzerindeki durumları daha iyi anlamak gerekiyor ilerideki süreçte yapacağımızı düşünüyorum.
struct egzersizleri sonrası artık daha dosya işlemleri ve bnezeri noktaya gelecek. hadi bakalım.

struck yapılarını kullanırken ard arda gelen karakter scanflerinde bir hata ile karşılaşılabilir.
(başımıza geldi)

Bunun için ayrı bir scanf tanımlayıp direkt araya eklemek çalışmıyorsa hatalı yere de eklenebilir.

bu sayede hata giderilebilir.

Lecture 13 (valuable exercise of structs)

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

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

union paylassdas{
    int deg1;
    int deg2;

}paylas;

int main(){
    int *i;
    int *k;

    paylas.deg1=2312;
    paylas.deg2=2321312;

    i=&paylas.deg1;
    k=&paylas.deg2;

    printf("bellek konumu :::: %d\n\n ", i);
    printf("bellek konumu :::: %d", k);

    return 0;
}

çıktı>>>
bellek konumu :::: 4223024
bellek konumu :::: 4223024
```

→ union yapılarında aynı bellek kullanıldığından pointer destekli güzel bir hatırlatıcısı.

şimdi bahsedeceğim örnek birçok noktadan birçok ders içermektedir.

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

typedef struct personel{
    int sifre;
    char ad[20];
    char soyad[20];
    int yas;
    float maas;
    char cinsiyet;

}kisiler;

void bilgi(kisiler liste[], int n){
```

```

int i;
char satir;
printf("      personellerin bilgi girişi\n\n\n");

for(i=0;i<n;i++){

    printf("[%d.]personelin adı ::: ",i+1);
    scanf("%c", &satir);
    scanf("%s", &liste[i].ad);
    printf("\n\n");
    printf("[%d.]personelin soyadı ::: ",i+1);
    scanf("%s", &liste[i].soyad);
    //gets ve fgets zillet şeylerdir. uzak durmakta fayda var. ilk harfleri tarayamıyor liste içlerinde haberimiz ola.

    /*printf("\n\n");
    printf("[%d.]personelin şifresi ::: ",i+1);
    scanf("%d", &liste[i].sifre);
    printf("\n\n");
    printf("[%d.]personelin yaşı ::: ",i+1);
    scanf("%d", &liste[i].yas);
    printf("\n\n");*/
}

void sirala(kisiler liste1[], int a){

    int h;
    int k;
    int i;
    kisiler mgecici;

    // STRING DEĞİŞKENLER İLE UĞRAŞIRKEN KÜTÜPHANELER DEĞERLİ OLUYOR BE!!!(burada yeniden görmüş olduk.)

    for(i=0;i<a-1;i++){
        for(h=i+1;h<a;h++){
            if(strcmp(liste1[i].ad,liste1[h].ad)> 0){
                mgecici=liste1[i];
                liste1[i]=liste1[h];
                liste1[h]=mgecici;
            }
        }
    }

    printf("\n\n");

    for(i=0;i<a;i++){
        printf(" -> %s\n",liste1[i].ad);
    }
}

int main(){

    int m;
    int n;

    printf("kac personel var? == ");
    scanf("%d", &m);
    kisiler genel_liste[m];
    kisiler gecici;

    bilgi(genel_liste, m);
    sirala(genel_liste, m);

    return 0;
}

```

!!!

→ fgets ve gets metodları tehlikeli mettolardır.

döngü içinde liste elemanlarına bir string ataması yapılacağı zaman mantıksal problemler oluşturuyor. buda program için problem oluyor.

bunun yerine scanf kullanarak daha rahat hareket edilebiliyor.

→ string problemlerinde olduğunda string.h kütüphanesinden yararlanmak gereklidir.

işler komplike bir hale geldikçe, makul en az sayıda kodu daha rahat ve sağlam çalışıtmak gereklidir.

Ve bir şeyleri yazdırırmak söz konusu olduğunda ve helede stringse..

bunun gibi komplike problemlerde kütüphane kullanımı gerçekten değerlidirç.

→ gets konusunda hata aldığımız noktada fflush(stdin) yapısını kullanarak hareket ederek hayat kurtarıcı olabilir.

bir tane math.h kütüphanesini kullanarak güzel bir koordinat örneği.

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

typedef struct kk{

    float x;
    float y;
}koordinat;

int main(){
    float t;

    koordinat top;
    printf("\n\n    koordinatlar");
    for(t=0;t<=10;t+=0.1){
        top.x=20-6*cos(t);
        top.y=15+2*sin(t);
        printf("\n\n");
        printf("%.2f. zamanda (%.2f,.2f) ",t,top.x,top.y);

    }
    return 0;
}
```

→ bellekte kapladığı alanı buldurmayla yönelik özellik sizeof() idi hatırlamamız gerekirse.

direkt struck ve union yapılarının kapladığı alanın bir ifadesini yapalım.

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

struct kontrol1{
    int a;
    float b;
};

union kontrol2{
    int c;
    float d;
};

int main(){
    printf("strcut --> %d /n/n union --> %d", sizeof(struct kontrol1), sizeof(union kontrol2));
    return 0;
}

çıktı>>
struct --> 8 /n/n union --> 4
```

→ bu yapıyı uzun tutmadım. Bilgisayar içinde daha uzun çeşitli örnekler var.

Bu noktada Can BOZ'un udemy kursunun örnekleri gayet güzel yollardır.

yararlanılabilir...

Lecture 14

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

!!! Çok değerli bir konu

Recursive Functions

recursive fonksiyonlar aynı fonksiyonun kendi içinde çağrılmıştır.

bu sayede alınan değer yerine geçip kendine sürekli tekrar edecektir.

(fonksiyonlardan bir loop mekanizması gibidir.)

ama bi tık farklı anlatacağım.

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

int f1(int n){

    if(n>0){
        f1(n-2);
        printf("%4d", n);
    }
    else{

    }

}

int main(){
    int h=20;

    f1(h);

    çıktı>>>

2    4    6    8    10   12   14   16   18   20
```

recursive fonksiyonda burada belli bir zamanlama olayı var, adım adım işliyor.

n=20 değerindeyken görev f(18) fonksiyonunu çağırırmak ve printf(20) yapmak.

n=18 değerindeyken görev f(16) fonksiyonunu çağırırmak ve printf(18) yapmak.

...

bu işlem sona kadar geliyor ve çıktı alınacağı zaman sondan başlayarak ilerlemeye başlıyor.

fonksiyon bilgileri alındıktan sonra zaten otomatik olarak görev yerine getiriliyor.

böyle düşünülebilir.

- bunun mantığını anlamak için —> <https://youtu.be/cv7CY8UmFL0> bu videoya bakılmalıdır.

Şadi hoca anlatmış zaten genel olarak işlerinin nasıl çalıştığını.

bir faktoriyel örneği ile bi sıralı fonksiyon işlemsine bir örnek daha verelim.

```

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

int f1(int m){
    int sonuc;

    if(m==1){
        return 1;
    }

    else{
        sonuc = m*f1(m-1);

        return sonuc;
    }
}

int main(){
    int n;
    int faktoriyel;

    printf("faktoriyel sayisini girinizz;; ");
    scanf("%d", &n);

    faktoriyel =f1(n);

    printf("%d", faktoriyel);

    return 0;
}

```

Matematikte gördüğümüz iç içe fonksiyon yapısına benziyor.

ve sondan (bitiş koşulunu tanıtıldığı elemandan) yukarı doğru yavaş yavaş çıkarıyor ve en son duruma geliyor.

Tablette notlarda bulunmakta. buraya da koyabilirim

""

ileriki örneklerde return tarzındaki fonksiyonların bu kadar çok kullanılmasının nedeni bu aslında.

bağımsız şekilde fonksiyonlar kendine kendilerine yazdırırmak yerine işlemin içerisinde bir tür dahilinde dahil edilmek gerçekten aşırı mantıklı bir hareket.

bu yüzden çarpma vs işlerinde bunu göstermiş olduk.

```

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

void listele(int m){

```

```

if(m>=0){
    printf("%d", m);
    printf("\n");
}

int main(){
    int n;

    printf("sayiyi giriniz :: ");
    scanf("%d", &n);

    liste(n);
    return 0;
}
çıktı>>
6
5
4
3
2
1
0

```

→ fonksiyon kendi içinde ilk aldığı değer üzerinden işlem yapıyor. Ki zaten yerel değişkenlik üzerinden hareket ettiği için bize recursive gibi bir mantık kazandırmış oluyor.

AMA BU HATALI BİR YAZIMDIR.

RECURSİVELER HERAHNGİ BİR ŞART İÇİNDE KOŞULMADIĞI SÜRECE SÜREKLİ ÇALIŞACAKTIR.

bu yüzden bunu bir if else kondisyonuna bitirme şartını koşarak katmak gereklidir.

o yüzden >>>

```

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

void liste(int m){

    if(m==0){
        printf("%d", m);
    }
    else{
        printf("%d", m);
        liste(m-1);
        printf(".");
    }

}

int main(){

    int n;

    printf("sayiyi giriniz :: ");
    scanf("%d", &n);

    liste(n);

    return 0;
}

```

→ bu çok daha efektif ve temiz bir yazım.

Recursive valuable examples.

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

int toplama(int m){

    if(m==1){
        return 0;
    }

    else{
        return m+toplama(m-1);
    }
}

int main(){

    int n;
    int toplam;

    printf("sayiyi giriniz :: ");
    scanf("%d", &n);

    toplam=toplama(n);

    printf("toplam sonucu == %d", toplam);
    return 0;
}
```

→ BU ÖRNEK İNCELENCEK

klasik bir çarpma işlemi sorusu.

ama recursive fonksiyonların nasıl çalıştığını güzel bir örnek.

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

int carpma(int m, int n){
    int sonuc;

    if(n==1){
        return 0;
    }

    else{
        sonuc=m+carpma(m, n-1);

        return sonuc;
    }
}
```

```

int main(){
    int a,b;
    int sonuc1;

    printf("sayiları giriniz :: ");
    scanf("%d%d", &a,&b);

    sonuc1 = carpma(a,b);

    printf("%d", sonuc1);

    return 0;
}

```

recursive in mantığını anlamak için bunu kağıda dökebiliriz.

tek ve çift sayıları yazdırma ile ilgili bi recursive fonksiyon yapısı

```

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

void f1(int m){
    int k;
    if(m%2==0){
        if(m==2){
            printf("%d", m-1);

        }
        else{
            printf("%d", m-1);
            printf("\n");
            f1(m-2);
        }
    }
    else{
        if(m==1){

        }
        else{
            printf("%d", m-2);
            printf("\n");
            f1(m-2);
        }
    }
}

int main(){
    int n;
    int faktoriyel;

    printf("bir sayı girinizz;; ");
    scanf("%d", &n);

    f1(n);

    return 0;
}

--> bu fonksiyonlarda işlemi normal bir printf döngüleri ile yapılması normal bi döngü mantığı katıyor.
ama döndürmeli recursive fonksyonlarda uç baya açılıyor.

```

!!! bu örnekleri olduğunca tekrar etmek gerçekten yararlı olacaktır.

→ asal sayıları hesap eden bir recursive fonksiyon?!!! aşırı güzel bir şey bu.

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

void f1(int m){
    int i;
    int sayac=0;
    if(m==2){
        printf("2\t");
    }

    else{
        for(i=2;i<m;i++){
            if(m%i==0){
                sayac=1;
            }
        }
        if(sayac==0){
            printf("%d\t",m);
        }
        f1(m-1);
    }
}

int main(){
    int n;
    int faktoriyel;

    printf("bir sayı girinizzz;; ");
    scanf("%d", &n);

    f1(n);

    return 0;
}

çıktı>>

bir sayı girinizzz;; 100
97      89      83      79      73      71      67      61      59      53      47      43      41      37      31
      29      23      19      17      13      11       7       5       3       2       1
```

fibonacci sayı dizisini üreten aşırı pratik ve dinamik (bu çok önemli bu ders buradan alınmalı .)

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

int f1(int m){

    if(m==0){
        return 0;
    }
    else if(m==1){
        return 1;
    }
    else {
        return(f1(m-1)+f1(m-2));
    }
    // direkt olarak iki değerin toplamını 3.indexe atar gibi döndürmek ?! çok mantıklı. recursive fonksiyonların işleme biçimini hakkında
    // karmaşık bir denklemde gerek filan yok. Daha sade şekilde dinamik bir fonksiyon örgüsü kurabiliriz.
    // zaten bu dinamiklik işini recursive çok büyük katkı sağlıyor.
```

```

int main(){
    int n;
    int i;

    printf("kac tane fibonacci ayisi uretilecektir? == ");
    scanf("%d",&n);

    for(i=0;i<n;i++){
        printf("\n%d\n",f1(i));
    }

    return 0;
}

```

→ fonksiyonun kullanılma dinamiği çok iyi.

2'nin katlarını almaya sağlayan temiz ve aşırı derece etkili dönüşlü bir recursive yapısı.

basit ama yöntem olarak recursive fonksiyonları çok etkili

```

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

int f1(int m){
    int sonuc;

    if(m==0){
        return 1;
    }

    else{
        sonuc=2*f1(m-1);
        return sonuc;
    }
}

int main(){
    int n,i;

    printf("2 nin katsayiları için max degeri giriniz:: ");
    scanf("%d", &n);

    for(i=0;i<n+1;i++){
        printf("\n%d",f1(i));
    }

    return 0;
}

çikti>>
2 nin katsayiları için max degeri giriniz:: 6

1
2
4
8
16
32
64

```

recursive fonksiyon kullanarak üs bulma.

```

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

int f1(int m,int n){
    int sonuc;

    if(n==0){
        return 1;
    }

    else{
        sonuc=m*f1(m,n-1);

        return sonuc;
    }
}

int main(){
    int a,b,k;

    printf(":: ussu alinacak sayiyi ve ne kadar üs alinacagini giriniz :: \n\n");
    printf("taban == ");
    scanf("%d",&a);
    printf("\nus == ");
    scanf("%d",&b);

    k=f1(a,b);

    printf("\n\n---> %d", k);

    return 0;
}

çikti>>
:: ussu alinacak sayiyi ve ne kadar üs alinacagini giriniz ::

taban == 3
us == 4
---> 81

```

baslangic ve bitis degerleri alınmış bir seri dizisi ama recursive fonksyonlarla...

```

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

void f1(int t,int artis,int bitis){

    if(t>bitis){
        printf("\n\n:: bitis degerine ulasildi :: ");
    }

    else{
        printf("%d\n",t);
        f1(t+artis,artis,bitis);
    }
}

int main(){
    int b, bt,s;
    int i;
    int sonuc;

    printf("baslangic degeri :: ");
    scanf("%d",&b);

```

```

printf("\nbitis degeri :: ");
scanf("%d",&bt);
printf("\nseri degeri :: ");
scanf("%d",&s);

f1(b,s,bt);

return 0;
}
çikti>>

baslangic degeri :: 10
bitis degeri :: 56

seri degeri :: 5
10
15
20
25
30
35
40
45
50
55

```

falan filan.

dizi elemanlarının küçütekn büyüğe sıralaması recursive fonksiyonlarından yardım alarak.

```

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

void f1(int dizi[],int m){
    int gecici,s;
    if(m==0){
        }
    else{
        f1(dizi,m-1);

        for(s=0;s<m-1;s++){
            if(dizi[m-1]<dizi[s]){
                gecici=dizi[m-1];
                dizi[m-1]=dizi[s];
                dizi[s]=gecici;
                }
            }
        }
    }

int main(){
    int n;
    int i;
    printf("Lutfen dizinin eleman sayisini giriniz :: ");
    scanf("%d",&n);
    int dizi[n];
    srand(time(0));

    for(i=0;i<n;i++){
        dizi[i]=rand()%1000;
    }

    f1(dizi,n);

    printf("\n::kucukten buyuge dogru dizi elemanlarinin siralaması::\n");
    for(i=0;i<n;i++){
        printf("\n%d",dizi[i]);
    }
}

```

```

    }

    return 0;
}

çıktı>>

Lutfen dizinin eleman sayisini girinizzz :: 5

::kucukten buyuge dogru dizi elemanlarinin siralamasi::

1
30
382
832
916

```

```

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

void f1(char can[],int n,int m){
    char gecici;

    if(n==m-n || n==m-n-1){

    }

    else{
        gecici=can[(m-1)-(n-1)];
        can[(m-1)-(n-1)]=can[n-1];
        can[n-1]=gecici;
        f1(can,n-1,m);

    }
}

int main(){
    char dizi[100];
    int i=0;
    int a;

    printf("lutfen bir cümle giriniz ::: ");
    gets(dizi);

    while(dizi[i]){
        i++;
    }
    printf("%d",i);
    a=i;

    f1(dizi,i,a);
    printf("\n\n");
    printf("%s",dizi);

    return 0;
}

```

bi tık zorlama ile yapılan cümle harf değiştirme sorusu.

Recursive kullanacağız diye bi tık zorlama yaptı ama değil .

recursivin nasıl çalıştığı konusunu iyice pekiştirmemize yardım ediyor.

yıldız bastırma***

ama recursive extra for döngüsü kullanarak.

```
#include<stdlib.h>
#include<stdio.h>
void f1(int m){
    int k;

    if(m==0){
    }

    else{
        k=m;
        for(k;k>0;k--){
            printf("*");
        }
        printf("\n");
        f1(m-1);
    }
}

int main(){
    int n;
    printf("lutfen bir deger giriniz:: ");
    scanf("%d",&n);

    f1(n);

    return 0;
}

çıktı>
lutfen bir deger giriniz:: 8
*****
*****
****
***
**
*
```

recursive fonksiyon kısmının son alıştırmasına geldim be.

→ klasik bir karakter dizisinin eleman sayısının bulunması

```
#include<stdlib.h>
#include<stdio.h>
void f1(char dizi1[],int i){
    int k;

    if(dizi1[i]=='\0'){
        printf("\n\n%d",i);
    }

    else{
        f1(dizi1,i+1);
    }
}

int main(){
    int n=0;
    char dizi[100];
```

```
printf("lutfen bir cumle giriniz:: ");
gets(dizi);

f1(dizi,n);

return 0;
}
```

Lecture 15 - File Dynamics

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

→ sona doğru yaklaşıyoruz ve dosya işlemleriyle ilgili sağlam bilgi ediniminin zamanı geldi.

Linux subsystem içindede işimize yarayacak bir bakış açısı edinebiliriz.

FILE yapısı pointer yapısında açılır.

syntaxı dikre tcan bozun görselinden bakalım.

```
int main()
{
    FILE Yapısı file pointer olarak adlandırılır

    FILE *dosya;      şeklinde tanımlanır.

    Dosyayı açmak için fopen(),
    Dosyayı kapatmak için fclose(), kullanılır

    FILE *dosya;
    dosya=fopen(const char dosya_adi,const char mod);
    ....
    dosyaislemleri    [
    ....
    fclose(dosya);
```

→ dosya açılırken hem bir iism değeri ve mod değeri istemekte.

bunlar ise :::

DosyaacmaModlari

r	ReadOnly	Sadece okuma için. Dosyanın açılabilmesi için önceden oluşturulması gereklidir.
w	WriteOnly	Yanlızca yazma için. Dosya kayıtlı olsun veya olmasın yeniden oluşturulur.
a	append	Ekleme. Kayıtlı bir dosyanın sonuna veri eklemek için kullanılır.
r+	Okuma ve Yazma	Bu modda açılmış olan bir dosyanın daha önceden var olması gereklidir.
w+	Okuma ve Yazma	Bu modda açılmış olan bir dosya var olsun veya olmasın yeniden oluşturulur.
a+	Okuma ve Yazma	Kayıtlı bir dosyanın sonuna veri eklemek için açılır.

Fonksiyon

fgetc()	Dosyaya bir karakter veri okur
fgets()	Dosyaya bir karakter dizi okur
fread()	Dosyaya bir kayıt dizi veya karakteri ikili olarak okur
fscanf()	Dosvadaki verileri biçimlendirerek okur

şeklidneki fonksiyonlardır.

-feof-

→ tanımlanmış dosyanın direkt sonunu denetlemek için kullanılır.

[dosya içinde sonlu loop işlemlerinde bu özelliğin önemi olacağı aşikar.]

DOSYANIN SONUNUN BELİRENMESİ feof()

Dosyadan okuma işlemleri yapılırken, çoğu kez dosyanın sonunu denetlemek gerekecektir. Dosya göstergesinin dosyanın sonuna işaret edip etmediğini anlamak için feof fonksiyonu kullanılır

```
FILE *dosya;
while (!feof(dosya))
{
    fgetc(dosya);           Dosyanın sonuna gelinmediği sürece dosyadan karakter okur
}
```

dosalardan bilgiler çekip bunun ne kadarıyla işlem yapacağımızı vs bilgilerini çekmede çok gereklidir.

hatta string lerle çok işimiz olacağrı için string.h kütüphanesinin bu civarda çok işe yarayacağını düşünüyorum.

bunlar okuma üzerine idi, şimdi veri kaydetme işine bakalım.

Dosyaya veri kaydetme

fputs() → Dosyaya bir karakter dizisi kaydeder.

fprintf() → dosyaya biçimlendirilmiş veri kaydetmek için kullanılır. (printfin aynısı ya sadece dosyalara özel)

bir tane dosya açalım o halde.

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

int main(){
    FILE *dosya;

    dosya=fopen("anana sor.txt","w"); // dikkat !!! moddlardan bahsetmiştık ve synatxide normal bu şekilde....
    fclose(dosya);
    return 0;
}
```

göründüğü gibi

dosyayı oluşturmak için kullandığımız mod “w” idir.

o yüzden dosyanın uzantısı ile adını verip bu modda çalıştık.

→ ananasor.txt dosyasını direkt c dökümanını kaydettiği yere kaydetti.

uzantılar dikkatimi çekti. Direkt uzantı adıyla dosyayı belirttik. BU AŞŞIRI İYİ!

fprintf kullanımını yakından bir görelim.

bu direkt dosya içine bir şeyi yazdırma(hatırlayalım hep direkt olarak cmd ye yazdırıyorduk şimdide dosyaya yazdıracağız.);

```

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

int main(){

    char dizi1[50]="anana sor";

    FILE *dosya;

    dosya=fopen("metin_d.txt", "w");

    fprintf(dosya,"olsturdugumuz dosyada %s ifadesini yazdırma pratigi",dizi1);

    fclose(dosya);

    return 0;
}

ehhehehe
oluyor.
metin_d ifadesindeki yazıyı göstereyim.

-> olsturdugumuz dosyada anana sor ifadesini yazdırma pratigi

// direkt karakter dizisini bastırma şeklinde vs dikkat. Çok değerli.

```

evet direkt c ifadelerinin hepsini sanki terminalde sonucu döndürür gibi döndürebiliyoruz.
fprintf yapısı bunu sağlıyor işte.

```

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

// bi çarpım tablosu egzersizi... ama biraz kendim uğraşayım diyorum.

int main(){
    int i;
    int j;

    char dizi1[50]="anana sor";

    FILE *dosya;

    dosya=fopen("metin_d.txt", "w");

    for(i=1;i<=10;i++){
        for(j=1;j<=10;j++){
            fprintf(dosya,"%d%d = %d ",i,j,i*j);

        }
        fprintf(dosya,"\n");
    }
}

```

```
fclose(dosya);

return 0;
}
```

güzel bir çarpım tablosu egzersiziyydi.

buna şuna dikkat çekmek isterim hem derleyip çalıştırırmak gerekiyor ve w modunu iyi bilmek lazım
çünkü kod doğrultusunda verilen dosyayı sıfırdan yeniden oluşturuyor.

!!!

→ ÇOK KRİTİK!

eğer biz bi önceki örnekte loop ifadelerini filan sağladıysak

scanf kullanıp bilgileri kullanıcıdan aldığı dosya içine direkt kaydınıda sağlayabiliriz.

Direkt olarakda kayıtlı olmuş olur.

vayyy anasını... görelim.

(öğrencilerin adını, soyadını okul numarasını tutan bir senaryo)

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

int main(){

FILE *dosya;

dosya=fopen("ogrenci_bilgi.txt","w");

char isim[20],soyad[20];
int no;
printf("lutfen ogrencinin adini giriniz ::: ");
gets(isim);
printf("lutfen ogrencinin soyadini giriniz ::: ");
gets(soyad);
printf("lutfen ogrencinin numarasini giriniz ::: ");
scanf("%d", &no);

fprintf(dosya,"ogrenci adi: %s\nogrenci soyadi: %s\n ogrenci numarasi: %d ",isim,soyad,no);

fclose(dosya);

return 0;
}

dosya çıktısı ogrenci_bilgi.txt
ogrenci adi: mahmut
```

```
ogrenci soyadi: tuncer  
ogrenci numarasi: 1331
```

```
//-> harika bir şey lan bunu yapmak.
```

→ bu bize yapacağımız onca psikopoat şeyin önünü açıyor.

bunu struct yapısıyla birlikte yaptığımızı üstüne otomasyona bindirdiğimizi düşünsene be!

direkt kayıtlı bir şekilde eline döküman geçiyor.

dosya işlemleri gerçekten güzel.

Kritik

→ şu ana kadarki dosya işlemlerinde kullanmış olduğumuz mod "w" yani yeniden oluşturma idi.

ama biz var olan dosyaya yeni bilgiler girmek istersek hangi modda çalıştırımmamız lazım?

append , "a" :: var olan bir dosyaya bilgi eklemek için kullanılmış bir moddur.

ve özellikle birden fazla kullanıcı bilgilerin girişi için bu özellik önemlidir diyebiliriz.

→ fputs komutunu burada hatırlatalım.

fprintf gibi dosya içine yazdırınma kullanılıyor ama sadece karakter dizilerine özgü bir yapı.

```
fputs("anana sor",dosya);  
-> syntaxı bu şekildedir ve direkt dosya içerisinde bu yazdığımız karakter dizisini yazdıracaktır.
```

Dosyadan veri okuma ve ekrana yazdırma

- fgetc() → dosyadan bir karakter okur.
- fscanf() → dosyadan biçimlendirilmiş karakter dizisi okur.

tam tersi olarak dosyadaki veriyi console çekmeye yarayan bir kısımdır bu.

!!! burada okuma modlarını kullanacağız.

yani sıra bir if else soru mekanizması ile birlikte kullanıyoruz.

aynı recursive fonksiyonlarda kendi kendine loopun sınırlarını çizdiğimiz gibi.

Buradada dosyanın içinin boş olup olmamasına göre bir soru eklememiz gereklidir.

fgetc ile bir karakter çekelim.

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

int main(){

    FILE *dosya;
    char k;
    dosya=fopen("anana sor.txt","r");

    if(dosya!=NULL){
        k=fgetc(dosya);
        printf("%c",k);
    }

    else{
        printf("dosya bulunamadı");
    }

    fclose(dosya);

    return 0;
}

çıkıcı>
n
-> evet çalışıyor ama şunu eklemek isterim//
// buradaki dosyayı 0 a eşit olup olmaması içinden boşluğu değil dosyanın var olup olmaması ile ilgiliidir.!!!
```

fscanf;; ile ilgili bir durum var.

fscanf boşluğa kadar olan kadar kısmı alır. ikinci sefer çalıştığında boşluktan sonraki ikinci kısmı alır ve böyle devam eder.

bir örnek üzerinde görelim.

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

int main(){

    FILE *dosya;
    char k1[50], k2[50];
    int sayı;

    dosya=fopen("anana sor.txt","r");
```

```

if(dosya!=NULL){
    fscanf(dosya,"%s",&k1);
    fscanf(dosya,"%s",&k2);
    fscanf(dosya,"%d",&sayi);
    printf("%s %s %d", k1,k2,sayi);
}

else{
    printf("dosya bulunamadi");
}

fclose(dosya);

return 0;
}

çıktı>>>

anan sor 13331
// boşluk mevzusu önemli DİKKAT!

```

daha uzun boşluklu cümle yazıları için bizim fscanf yapısı loop olarak alınmalıdır.
bunun için “DİKKAT ÇEKTİĞİMİZ”feof fonksiyonu kullanırız.

feof dosyanın sonunun geldiğini denetlemek için kullanılır.

ve biz bunu while döngüsü ile birlikte kullanırken “dosyanın sonuna gelmeyene kadar” gibi bir anlam isteriz.

bunun içinde

```

while(!feof(dosya)){
    --islemler--
}

```

şeklinde bsr yapıyla sağlarız uzun bir cümleyi fscanf ile çekip böyle bir şekilde console içine bastıralım.

```

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

int main(){

FILE *dosya;
char k1[20][50];
int i=0,j;

dosya=fopen( "anana sor.txt","r");

if(dosya!=NULL){
    while(!feof(dosya)){

```

```

        fscanf(dosya,"%s",&k1[i]);
        i++;
    }
    for(j=0;j<i;j++){
        printf("%s ",k1[j]);
    }
}

else{
    printf("dosya bulunamadi");
}

fclose(dosya);

return 0;
}

çikti>>>
ne bekliyodun askimdan askim askindan vazgeceli iki yasindaaaaaa

```

bu şekilde iki boyutlu bir karakter dizisinin içini doldurabiliyoruz.

burada iki tane döngüye de gerek yok.

her aldığı indexlerle direkt olarak while içindedede bastırabilirdik.

yinede uzun yoldan çalıştırmanın yolunu çözdük.

örnekteki çözümden farklı olarak struct kullanarak çözdüğüm bir örnekti.

belirli yerdeki txt dosyasındaki öğrenci bilgilerini console da yazdırmaak için genel bir örnek.

```

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

typedef struct ogrenci_b{
    char isim[20];
    int no;
    int sonuc;
}bilgi;

int main(){

FILE *dosya;
char k1[20][50];
int i=0;
bilgi k2[10];

dosya=fopen("ogr.txt","r");

if(dosya!=NULL){
    while(!feof(dosya)){

```

```

fscanf(dosya,"%d",&k2[i].no);
printf("%d\t",k2[i].no);
fscanf(dosya,"%s",&k2[i].isim);
printf("%s\t",k2[i].isim);
fscanf(dosya,"%d",&k2[i].sonuc);
printf("%d\t\n",k2[i].sonuc);

i++;
// direkt olarak fscanf'te 3 değişenide alabiliriz.
// :: fscanf(dosya,"%d", "%s" "%d",&....,&....,&....);

}

else{
    printf("dosya bulunamadi");
}

fclose(dosya);

return 0;
}

çıktı>>>
txt dosyasındaki bilgiler;
1515 Osman 50
9872 Ayse 64
4321 Tuncay 78
7656 Veli 96
-----
console:::

1515 Osman 50
9872 Ayse 64
4321 Tuncay 78
7656 Veli 96

```

Lorem ipsum kullanarak rastgele bir paragraf generatordan bir paragraf çekip direkt olarak bunu consoleda yazdırmasını istedik.

```

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

int main(){
    int i;
    char deg[30];
    FILE *dosya;
    dosya=fopen("lorem.txt","r");

    if(dosya!=NULL){

        while(!feof(dosya)){
            fscanf(dosya,"%s",&deg);
            printf("%s ",deg);
        }
    }
    else{

```

```
    printf("dosya bulunamadi");
}

fclose(dosya);

return 0;
}
-> direkt olarak deg karakter dizisinin kendisine kelime kelime atamada bulunuyor.
-> Can Boz abimiz fgetc kullanarak her bir karakteri alarak yazdırma işlemi yaptı.(aynı şey.)
DİKKAT EOF kullanımını var.
```

!!!

EOF kullanımı ;;;

direkt olarak imleci sonuna götürmesi olarakda düşünülebilir.

yani hangi değişkene veya bir ifadeye işaretleme yapılmışsa onun sonunu ifade eder.

biraz mantıksal olarak kafada şekli oluşmuyor feof tamamen aynı sadece dosyanın sonunu ifade ediyor.

örnekler üzerinden işledikçe kafada oluşacaktır.

Lecture 16 (File exercise)

Author/Publisher

C derlemesi - Talha AYDIN

UDEMY Source

Can BOZ

kullanıcıdan bir cümle alınmasını isteyen ve bunu n sefer dosya içine yazdırmasını isteyen bir senaryo::

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

int main(){
    int i,n;
    char cumle[50];
    FILE *dosya;
    dosya=fopen("ornek.txt","w");
    printf("lutfen bir cümle giriniz:::");
    gets(cumle);
    printf("kaç defa yazdırma yapacaksınız? :: ");
    scanf("%d",&n);

    for(i=0;i<n;i++){
        fprintf(dosya,"%s\n",cumle);
    }

    fclose(dosya);

    return 0;
}
```

→ 0 dan veya yeniden oluşturma için direkt 'w' fonksiyonunu kullandık.

belirli bir txt içindeki bilgilern karakter karakter okunmasını sağlayan bir senaryo...

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

int main(){

    char karakter;
    FILE *dosya;
    dosya=fopen("ornek.txt","r");

    if(dosya!=NULL){
        while(!feof(dosya)){
            karakter=fgetc(dosya);
            printf("%c",karakter);
        }
    }

    else{
        printf("dosya bulunamadi");
    }
    fclose(dosya);

    return 0;
}
```

!!!

fgets in dosya içinde satır okuma yapmasını sağlayan bir senaryo

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

// satır satır txt dosyasını okumak için fgets kullanacağımızda belirtilen max karakteri entera kadar alıyor. fscanfın enter kadarki önceden kullandığımızda ama dosya versionunu görmemiştik.

int main(){

    char *karakter,satir[20];
    FILE *dosya;
    dosya=fopen("ornek.txt","r");

    if(dosya!=NULL){
        do{
            karakter=fgets(satir,20,dosya);
            printf("%s",satir);

        }while(karakter!=NULL);
        printf("\n\nislem başarılı şekilde gerçekleştirildi!");
    }
    else{
        printf("dosya bulunamadi");
    }

    return 0;
}
```

direkt olarak belirtilen dosyadaki satırlardaki bilgileri satır satır okumasını yaptı.

fgets pointer türünde ifade eşitler. ve direkt olarak içine girdiğim bir dizi(burada satır okumak için karakter dizisi kullandık.) direkt olarak pointer cinsinde tanımladığımız değişkene eşitleyince bu değişkenin adresine gitmektedir. Bu şekilde direkt olarak satır boyunca aldığımız bilgiyi printf kullanıp direkt console yazdırıyoruz. Vay anasını....

→ eğer biz sonu gelemeyecek şekilde while koşuluna vs sokarsak (feof kullanımı) dosya boyunca tüm bilgileri satır satır alabiliyoruz ve adım adım başka dökümana yedekleyebiliyoruz demek oluyor.

>>FGETS COK ÖNEMLİ . ÜZERİNDE İYİCE DURMAK GERÇEKTEM ÖNEMLİ<<

ama şu dikkatimi çekti.

satır karakter dizisinin ben boyutunu farklı versemde işlemi sürdürdü ve doğru şekilde gerçekleşti.

büyük ihtimalle scanf gibi davranış var bundada.

farkı scanfteki boşluğu tanılama varken bu ifade \n tanıyana kadar işlem yapıyor.

(detaylıca fikrimde gelişme olursa buraya yazacağım.)

⋮⋮⋮

append(ekleme) modu ile karakter karakter girdiyi yazdırma senaryosu

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

int main(){
    char cumle[20];
    int i;

    FILE *dosya;
    dosya=fopen("ornek.txt","a");

    if(dosya!=NULL){
```

```

printf("lutfen bir cümle giriniz:: ");
gets(cumle);
for(i=0;cumle[i];i++){ //hatırlayalım. char dizileri için \o özelliği ile for döngüsü oluşturabiliyoruz.

    putc(cumle[i],dosya);

}
putc(' ', dosya);
printf("\n\nislem basarili");
}

else{
    printf("dosya bulunamadi");
}

return 0;
}

```

→ putc kullanımına dikkat "" şeklindeki kullanımlara izin vermiyor. switch case'deki gibi case atarken " şeklinde bir kullanım yapmamız gereklidir.

direkt karakter ataması yapıyor. Belirgin başka bir şey yok. syntaxine biraz dikkat sadece.

Öğrencilerin girdiği 3 sınav totalinde ortalmalarına göre geçiklerini veya kaldıklarını bir txt dosyası içerisinde ekleyen bir senaryo

```

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

typedef struct ogrenci_b{
    char isim[20];
    int no;
    float vize1,vize2,final;
    float ort;
}ogr;
int main(){
    int n,i;
    char satir;

FILE *dosya;

dosya=fopen("ornek.txt","a");
printf("lutfen ogrenci sayisini giriniz:: ");
scanf("%d", &n);
ogr ogr_a[n];
if(dosya!=NULL){
    for(i=0;i<n;i++){
        printf("lutfen %d.ogrencinin ismini giriniz::",i+1);
        scanf("%s",&ogr_a[i].isim);
        scanf("%c",&satir);
        printf("\nlutfen ogrencinin sinav sonuclarini giriniz \n");
        printf("1.vize :: ");
        scanf("%f",&ogr_a[i].vize1);
        printf("2.vize :: ");
        scanf("%f",&ogr_a[i].vize2);
        printf("final :: ");
        scanf("%f",&ogr_a[i].final);
        ogr_a[i].ort=((float)ogr_a[i].vize1*0.2+(float)ogr_a[i].vize2*0.2+(float)ogr_a[i].final*0.6);
        printf("%.2f",ogr_a[i].ort);
    }
    fprintf(dosya,"isim\tbasari durumu\n-----\n");
    for(i=0;i<n;i++){
        if(ogr_a[i].ort>50)
            fprintf(dosya,"%s\tgecti\n",ogr_a[i].isim);
        else{
            fprintf(dosya,"%s\tkaldi\n",ogr_a[i].isim);
        }
    }
    else{
        printf("dosya bulunamadi");
    }
}

örnek.txt içi>>
isim basari durumu

```

```
-----  
mahmut gecti  
tuncay kaldi
```

→ gerçektenn güzel bir örnekti.

0 dan 360 a kadar olan tüm derecelerin sin ve cos değerlerini açılan txt dosyasına yazdırın bir senaryo.

```
#include<stdlib.h>  
#include<stdio.h>  
#include<math.h>  
#define pi 3.14  
  
int main(){  
    int i;  
  
    double sinus,kosinus;  
    /*BURAYA DİKKAT sinüs ve kosinüsün özellikle double tipinde alınması,  
    bu fonksiyonları çalıştırıldığımızda double tipinde basım yapacağındandır.*/  
  
    FILE *dosya;  
  
    dosya=fopen("ornek2.txt","w");  
  
    if(dosya!=0){  
        fprintf(dosya,"DERECE\tsin\tcos\n");  
        fprintf(dosya,"=====t==\t==\n");  
  
        for(i=0;i<361;i++){  
            sinus=sin(i*pi/180);  
            kosinus=cos(i*pi/180);  
            fprintf(dosya,"%d\t%.3lf\t%.3lf\n",i,sinus,kosinus);  
        }  
        printf("\n\nbilgiler dosyaya yazıldı :");  
    }  
  
    else{  
        printf("dosya bulunamadi");  
    }  
  
    fclose(dosya);  
    return 0;  
}  
// direkt olarak sin ve cos fonksiyonlarını kullanması bi tık beni üzmedi değil.
```

→ Belirttiğim gibi tahminimce çoğu math.h kütüphanesinden kullanılan fonksiyonlar double tipde dönüştürülüyor. Bu yüzden bizimde double tipinde bir değişken kullanmamız gerekiyor.

AŞIRI DERECE ÖNEMLİ Bİ SENARYO.

```
#include<stdlib.h>  
#include<stdio.h>  
  
int main(){  
    char satir[200],dosyaismi[45];  
    char *ptr;  
  
    printf("lutfen console a yazdirilacak dosyanin adini ve yolunu giriniz:: ");  
    gets(dosyaismi);  
    FILE *dosya;  
    dosya=fopen(dosyaismi,"r");  
  
    printf("\n\n");  
  
    if(dosya!=0){  
        do{
```

```

ptr=fgets(satir,200,dosya);

if(*ptr!=NULL){
    printf("%s",satir);
}

}while(!feof(dosya));
// -> EOF(END OF FILE) ///
fclose(dosya);
}

else{
    printf("dosya bulunamadi;; tekrar deneyiniz\n\n");
}
return 0;
}

```

- burada fgets kullanırken kullandığımız pointer *dosya şeklinde atadığımız pointer yapısı gibi.
- atadığımız if sorgusuna baktığımızda aynı dosyanın var olduğuna dair veya olmadığına dair bir bilgi veriyor bize.
- ve eğer satır yoksa (boş olması demek satır yok demek değildir.) enter \n olduğunun bilgisini verir.) direkt olarak atlıyor. direkt dosya sonu ise de döngünün kendisinden çıkış oluyor.
- !!! fgets kullanımı satır işlem yaptığı için çok önemli**

kullanıcı tarafından dosyayolu, dosyaadi ve uzantısı girilen br text dosyasının belirtilen adrese yedeklenmesini sağlayan bir senaryo

iki farklı şekilde yaptım bunu .

bir karakter karakter

birde satır satır.

```

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

int main(){

FILE *dosya,*yedek;
char dosyaadi[60],yedekadi[60];
char *karakter,satir[400];
printf("lutfen icerigin alanagini dosya adini ve uzantisini giriniz:: ");
gets(dosyaadi);
printf("lutfen icerigin akatarilacagi dosya adini ve uzantisini giriniz:: ");
gets(yedekadi);

dosya=fopen(dosyaadi,"r");
yedek=fopen(yedekadi,"w");
if(dosya!=0){

    while(!feof(dosya)){
        karakter=fgets(satir,400,dosya);
        if(*karakter!=NULL){
            if(yedek!=0){
                fprintf(yedek,"%s",satir);
            }
            else{
                printf("\nyedeklenecek dosya ulunamadi");
            }
        }
    }

    printf("\nislem basarili sekilde tamamnlandi");
}

else{
    printf("dosya bulunamadi");
}
fclose(dosya);
fclose(yedek);

return 0;
}

```

→ bu klasik bir fgets kullanarak satır satır alıp direkt fprintf ile diğer dosyaya bastırdım.

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

int main(){

    FILE *dosya,*yedek;
    char dosyaadi[60],yedekadi[60];
    char karakter,satir[400];
    printf("lutfen icerigin alanacagi dosya adini ve uzantisini giriniz:: ");
    gets(dosyaadi);
    printf("lutfen icerigin akatarilacagi dosya adini ve uzantisini giriniz:: ");
    gets(yedekadi);

    dosya=fopen(dosyaadi,"r");
    yedek=fopen(yedekadi,"w");
    if(dosya!=NULL){
        if(yedek!=NULL){
            for(karakter=getc(dosya);karakter!=EOF;karakter=getc(dosya)){
                putc(karakter,yedek);
            }
        }
        else{
            printf("yedeklenecek konum bulunamadi");
        }
        printf("\nislem basarili sekilde tamamnlandi");
    }
    else{
        printf("dosya bulunamadi");
    }
    fclose(dosya);
    fclose(yedek);

    return 0;
}
```

→ Buda Karakter meterolojisi içeren Can BOZ edition

bu arada karakter karakter işlem yapma bunu for döngüsünde sağlama aşırı mantıklı imiş.
direkt dosyada kullanılan karakter sayısını görmek için vs de kullanılabilir.

O girene kadar kullanıcıdan aldığı kelimeleri belirtilen bir dosya içine yazdırın bir senaryo

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

int main(){

    FILE *dosya;
    char kelime[100],*pt;
    int control=0;
    dosya=fopen("ornek3.txt","w");

    if(dosya!=NULL){
        do{
            printf("lutfen bir kelime giriniz :: ");
            gets(kelime);
            for(pt=kelime,*pt;pt++){
                if(*pt!='\0'){
                    fprintf(dosya,"%c",*pt);
                }
            }
            else{
                control=1;
            }
        }while(control==0);
    }
}
```

```

        }
        fprintf(dosya, "\t");

    }while(control!=1);

    printf("\nislem basarili sekilde tamamnlandi");
}

else{
    printf("dosya bulunamadi");
}
fclose(dosya);

return 0;
}

```

putc burada çalışmadı sebebini anlamadım.

| sonlanma fonksiyonu → **exit(1)**bu fonksiyon direkt tüm programı sonlandırıyor. !!!

file examplesların son örneği >>>

strupr ile küçükharflerden oluşan bir text dosyasını büyük harflere çevirme. ve bunu yeni oluşturulmuş bir txt dosyası içerisinde veme.

```

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

int main(){
    char kelime[50];
    FILE *dosya, *dosya1;

    dosya=fopen("kucukh.txt", "r");
    dosya1=fopen("buyukh.txt", "w");

    if(dosya!=NULL){
        while(!feof(dosya)){
            fgets(kelime, 50, dosya);
            if(dosya1!= NULL){
                fprintf(dosya1, "%s", strupr(kelime));
            }
        }
    }
    else{
        printf("dosya bulunamadi");
        exit(1);
    }
}

return 0;
}

```

→strupr kullanmak için string bir ifadeye ihtiyaç var bende satır satır işlem yaparak satır satır upr içine soksutum.

Öyle bir örnekti

ara sıra bu örnekleri tekrar etmekte yarar var.

Lecture 17 - Dinamik Bellek yönetimi

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

Bellek fonksiyonları

Malloc fonksiyonu



-Malloc hafızadan belirli bir yer ayırmamıza yarar. Geriye void tipinde Bir pointer döndürür. Void in tipi olmadığından isteğimiz değişkene Cast işlemi yapabiliriz.
Eğer yeterli alan ayrılmadıysa geriye NULL pointer döndürür.

→ dendiği gibi bellilediğimiz bir miktarda hafıza içeriisnde yer açma işlemi yapmaktadır.
syntaxını ve pointer kullanımını görelim.

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

int main(){

    int *ptr;
    int n,i;
    n=5;

    ptr=(int*)malloc(n*sizeof(int)); // integer tipinde ayırdık. çünkü almak istediğimiz değerde pointerda inetger tipinde.

    if(ptr==NULL){
        printf("hafıza bolunmedi");

    }

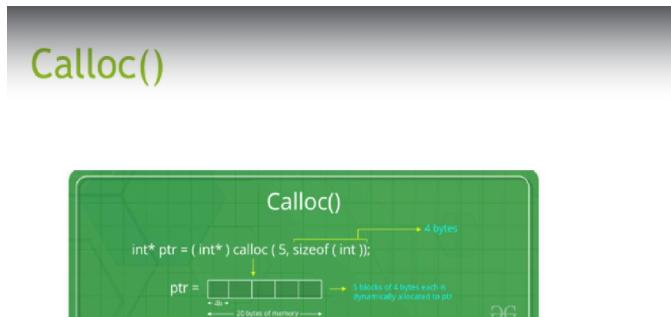
    else{
        for(i=0;i<n;i++){
            ptr[i]=i+1;
            printf("%d -",ptr[i]);
        }
    }
    return 0;
}
```

- birkaç noktaya dikkat.
- tanımladığımız pointer üzerinde adresler sanki dizinin elemanları gibidir. evet genel pointer yapısına benzıyor ama adlandırma yaparkenki şekilde dikkat.(direkt olarak `ptr[i]` değişkenine atama yapmış oluyoruz.)

- fonksiyonu çağrıırken fonksiyonun türünü belirtmeyi unutmamak çok önemlidir.(int*)
-

Calloc fonksiyonu

→ mantık malloc ile tamamen aynı ama sadece belirtme şeklinde küçük bir farklılık var.



görüldüğü gibi .

Malloc içinde n defa ifadesini çarpım ile verirken, burada ise n sefer ifadesini „,” ile ayırarak belirtiyoruz. onun dışında mantık ve çalışma şekli komple aynı.

extra bir örneğe gerek duymuyorum.

!!!

eğer aldığımız değer ayırdığımız alandan daha fazla olursa fazlalar +garbage value+ olarak nitelendirilir ve istek dışı sonuçlar gözlenir.

FREE fonksiyonu

→ ÇOK DEĞERLİ BİR FONKSİYONDUR.

malloc ve calloc ile hafızadan yer ayrıldıktan sonra işimiz bittiğinde free fonksiyonu ile bu alanı boşaltmamız gereklidir.

aksi halde memory leak (bellek sizıntısı) denen bir problem oluşur. ileride ana makinenin hafızası yetmeyebilir. çökmeler yaşanabilir vs.

bu yüzden bu free fonksiyonu gerçekten önemli.



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

int main(){
    float *ptr;
    int n,i;
    n=5;

    ptr=(float*)calloc(n,sizeof(float)); // integer tipinde ayırdık. çünkü almak istediğimiz değerde pointerda integer tipinde.

    if(ptr==NULL){
        printf("hafiza bolunmedi");
        exit(1);
    }

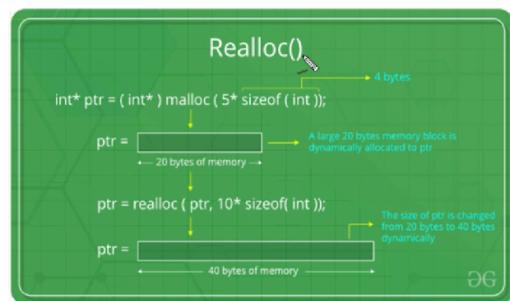
    else{
        for(i=0;i<n;i++){
            ptr[i]=i+1;
            printf("%f ",ptr[i]);
        }
    }
    <<<<free(ptr);>>>
    return 0;
}
```

- bi fark yok görünüşte ama gerçeken önemlidir.
- bir alan açtıysak proje sonu bunu kapatmak gereklidir.
- bu arada direkt görsellerdeki gibi tek satırda tanımlama yapabiliriz.
- hatta tek satırda tanımlama ilerideki pointerların pointerlarında anlamak için önemli bir basamak oluyor.

HATIRLATMA → direkt orgramdan çıkmak için **exit()** fonksiyonunu kullanıyoruz.(bazı istenmeyen koşullardan sonra direkt çıkışını istersek kullanıyoruz.)

Realloc fonksiyonu

Realloc()



Realloc fonksiyonu daha öncesinde bellekten ayrılmış olan alanımızda değişiklik yapmamızı sağlar. Diğer bir deyişle Malloc veya Calloc kullanarak bellekten ayırdığımız alan yetersizdir. Realloc dinamik olarak ayırdığımız alanı tekrardan güncellememizi sağlar.

06 Eğer bellekten yeterli alan ayrılmadıysa işlem başarısız olur ve geriye NULL

→ biraz önce açıkladığımız yetersizlik durumunu realloc fonksiyonu sayesinde çözüyoruz.

bu bize açmış olduğumuz alanları güncellememizi sağlayan br fonksiyondur..

zaten syntaxinde görüldüğü gibi ptr pointerini direkt işleme alıyor.

ve yanındaki alanı malloc tarzında yeniden biçimlendiriyoruz.

```
ptr=realloc(ptr,10*sizeof(int));  
--> önceden açtığımız alanlı ifadede tanımlamasını yaptıgımız için bunda tanımlama yapmaya gerek kalmadı.
```

(Syntaxı verdim extra örnekle uğraşmayacağım.)

HATIRLATMA → malloc içine belirttiğimiz direkt size (ayırılan alan) olduğu için direkt kaç byte lik yer ayırmak istiyorsak direktde yazabiliriz.

aha .

geldik dizinin gerçek yüzüne...

Pointerin Dizi gibi kullanımı

Göstericinin Dizi Olarak Kullanımı

Dizilerde aslında bir göstericidir. Bir diziye gösterici gibi erişmek veya bir göstericiye bir dizi gibi erişmek mümkündür. Diziler, özel bir söz dizimi ve erişim yöntemleri bulunan birer göstericidir.

```
int a[10];
int *p=(int *)calloc(10,sizeof(int));

a[3]=5;
p[3]=5;
```

Burada a ile p eşdeğerdir.

bildiğimiz ve bu sayfa başında yaptığımız gibi.

zaten pointer egzersizlerindede belirtmiştim.

dizi mekanizması bir pointer mantığında çalışır.

uzatmaya gerek yok çokça tekrar ettik.

manyak bir şey geliyor..

Pointerların pointerlarında dinamik bellek yönetimi

çok fazla yorum eklemeyeceği.

bir tık karışık gelebilir, çok bir şey yok sadece çok boyutlu ifadeler için hangi türdeyse onun pointerini ayırip sonradan işleme geçiriyoruz.

Göstericinin Göstericisi

```
int m[3][4];

int **m=(int **)malloc(3*sizeof(int *));
m[i]=(int *)malloc(4*sizeof(int));
```

-İkinci tamsayı gösterici göstergesi için yer ayırmaktadır.
-Burada ayrılan veri tipi int değil int* biçiminde matrisin her satırını temsil eden bir tamsayı göstergesidir.

-İkincisi
-Her satırın kendi elemanları için yer ayırma int tipi için olur.

Baska bir deyişle önce 3 adet tamsayı göstergesi için yer ayrılr. Sonra Bu göstergisinin 3 elemani olan göstergiciler için de 4 adet tamsayı yeri ayrılr.

bir kere **ptr nin cinsini belirttikten sonra bir daha belirtmeye gerek yok.

!!!!

bunun daha net anlaşılması için aynı sonucu bastırın iki boyutlu bir dizi üzerinde anlayalım.

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

int main(){
    int m[3][4];
    int i,j;
    for(i=0;i<3;i++){
        for(j=0;j<4;j++){

            m[i][j]=i*10+j+1;

            printf("%3d",m[i][j]);
        }
        printf("\n");
    }
    free(**k);
    return 0;
}

çıktı>>>
 1  2  3  4
11 12 13 14
21 22 23 24
```

→ klasik bir 2 boyutlu bir dizi

ve bunu alan ayırarak

ve pointerlarla ifade edelim.

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

int main(){
    int i,j;

    int **k=(int**)malloc(3*sizeof(int*));

    for(i=0;i<3;i++){
        for(j=0;j<4;j++){
            k[i]=(int *)malloc(4*sizeof(int));
            k[i][j]=i*10+j+1;

            printf("%3d",k[i][j]);
        }
        printf("\n");
    }

    return 0;
}

çıktı>>>
 1  2  3  4
11 12 13 14
21 22 23 24
```

→ 2 kere incelemeliyiz.

tanımlamadaki 2 noktaya dikkat.

alan fonksiyonlarıyla pointer tanımlaması yaparken ilk tür tanımlaması yaptıktan ve bunun 2 boyutlu bir pointer olduğunu(**) şeklinde belirttiğinden sonra ikinci tanımlama yapmıyoruz.

sonrasında bir alt boyutun ne kadar alan istedğini yukarıda göründüğü mantıkta alıyoruz.
dediğim gibi iki boyutlu dizi mekanizmasına çok yakın.
sadece tanımlamaları daha nesnel.

Hatırlatma pointer ve sicim dizileri

→ bunu direkt olarak görmedik ama mantığı çok hızı çözebileceğimiz bir yapısı var.

mekanizma belli bir dizi ve pointer belirtip bunu iki boyutlu bir dizi mekanizmasında çalışırmak.
görelim.

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

int main()
{
    int i,j;
    char *dizi[10]={"anana", "sordun", "mu?"};

    for(i=0;i<3;i++){
        printf("--> %s\n", dizi[i]);
    }

    return 0;
}
çıktı>>
--> anana
--> sordun
--> mu?

-> bu yapı codeblocksta çalışıyor ama devc de çalışmıyor hata verdiriyor.
  hata vermesinde gerekçeli bir duurum olsada bilgilendirmek istedim.
```

→ derleyiciler arasında farklılıklar var.
bir tanesinde çalışan birinde çalışmıyor.
iki derleyici kullanmanın avantajı bu işte 😊

Lecture-18 (DBY examples)

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

hafif takılacak başta. ama yinede elimin ve kafamın alışması için yazacağım.

n kadar int alanı açmak için bir senaryo

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

int main(){
    int n,i;
    int *ptr;

    printf("lutfen bir sayı giriniz :: ");
    scanf("%d",&n);

    ptr=(int *)malloc(n*sizeof(int));

    // bu şekilde girilen değer kadar sistem içerisinde alan açtım. n tane indexten oluşan bir int dizisi gibi
    // ptr ilede bunu yönetebilirim.

    for(i=0;i<n;i++){
        printf("\n%d. indexi giriniz ::",i+1);
        scanf("%d",&ptr[i]);
        !!!! scanf("%d",&(*ptr+i))

    }
    for(i=0;i<n;i++){
        printf("%3d",ptr[i]);
    }

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

bu benim yol ama orada belirttiğim gibi can boz usulü ptr ifadesini farklı şekilde atama yapıyalbiliyor.

→ pek bir anlam farkı yok benimkiyle. sadece adres atanın pointerin değerini * ile belirtip içerisindeki adresi ona göre koydu. (klasik dizi ve pointerlarda yaptığımız egzersizler gibi)

buda çalışıyor

```
scanf("%d",ptr+i);
```

çok extreme düşünmeye falan gerek yok hocam. pointer işte.

yukarıdaki örneği bi tık kendime göre dönüştürüp karelerini aldığım bir işlem.

```

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

int main(){
    int n,i;
    int *ptr;

    printf("lutfen bir sayı giriniz :: ");
    scanf("%d",&n);

    ptr=(int *)malloc(n*sizeof(int));

    // bu şekilde girilen değer kadar sistem içerisinde alan açtım. n tane indexten oluşan bir int dizisi gibi
    // ptr ileden bunu yönetebilirim.

    for(i=0;i<n;i++){
        printf("\n%d. indexi giriniz ::",i+1);
        scanf("%d",ptr+i);

    }
    for(i=0;i<n;i++){
        printf("\n%d. nin karesi = %d",i+1,*((ptr+i)**(ptr+i)));
    }

    free(ptr);

    return 0;
}

```

alanların açılma miktarları önemli baya.

fazla açılmışsa bunu düzenelemek gerekiyor yoka hata geliyorum diyor yani
 iki alanın sayılarının değiştirilmesi üzerine bir senaryo.

```

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

int main(){
    int *p1,*p2;
    int k,n,i;
    int gecici;

    printf("lutfen 1. alanın sayı miktarını giriniz ::");
    scanf("%d",&n);
    printf("lutfen 2. alanın sayı miktarını giriniz ::");
    scanf("%d",&k);
    p1=(int *)malloc((k)*sizeof(int));
    p2=(int *)malloc((n)*sizeof(int));

    for(i=0;i<k;i++){

```

```

printf("\nlutfen %d. indexi giriniz ;; ", i+1);
scanf("%d", p1+i);
}
printf("\n=====\\n");
for(i=0;i<n;i++){
    printf("\nlutfen %d. indexi giriniz ;; ", i+1);
    scanf("%d", p2+i);
}

for(i=0;i<n;i++){
    gecici=p1[i];
    p1[i]=p2[i];
    p2[i]=gecici;
}
for(i=0;i<n;i++){
    printf("\n%d", *(p1+i));
}
free(p1);
free(p2);
return 0;
}

```

n tane rastgele alınmış sayının alan açılıp hem dosyaya hemde consola yazdıracak şekilde bir senaryo geliştirelim.

```

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

int main(){
    int n,i;
    int *ptr;

    printf("lutfen bir sayı giriniz::: ");
    scanf("%d",&n);
    srand(time(0));
    FILE *dosya;
    dosya=fopen("dinamikb.txt","w");

    ptr=(int *)malloc(n*sizeof(int));

    for(i=0;i<n;i++){
        ptr[i]=rand()%1000;
        if(dosya!=0){
            fprintf(dosya,"--> %d\\n",ptr[i]);
        }
        else{
            printf("dosya bulunamadi");
            exit(0);
        }
        printf("--> %d\\n",ptr[i]);
    }

    free(ptr);
    fclose(dosya);
    return 0;
}

```

```
Çıktı>>

--> 87
--> 930
--> 316
--> 531
--> 863
// dosyada aynı görünüyor kontrol ettim.
```

→ sayıları tek ve çift ayıracak şekilde bir iki bellekli senaryo.

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

int main(){
    int n,j,m;
    int *p1,*p2;
    int i;

    printf("lutfen bir sayı giriniz::: ");
    scanf("%d",&n);

    p1=(int *)malloc(n*sizeof(int));
    p2=(int *)malloc(n*sizeof(int));
    j=0;
    m=0;

    for(i=1;i<n;i++){
        if(i%2==0){
            p1[j]=i;
            j++;
        }
        else{
            p2[m]=i;
            m++;
        }
    }
    printf("\nbellek 1\n=====\\n");

    for(i=0;i<n;i++){
        if(p1[i]!=0){
            printf("- %d\\n",p1[i]);
        }
    }

    printf("\nbellek 2\\n=====\\n");
    for(i=0;i<n;i++){
        if(p2[i]!=0){
            printf("- %d\\n",p2[i]);
        }
    }

    free(p1);
    free(p2);
}
```

```
    return 0;
}
```

```
çıktı>>
bellek 1
=====
- 2
- 4
- 6
- 8
```

```
bellek 2
=====
- 1
- 3
- 5
- 7
- 9
```

→ kritik olarak gördüğüm açılan alanın hepsi kullanılmadığı zaman boş yerlere sıfır ataması yapıyor.
bunun bilincide olduktan sonra belleğin neresine kadar durdurmak istediğimizde ayarlayabiliyoruz.

klasik bir sıralama sorusu ama bellek kullanarak.

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

int main(){
    int n,j,i;
    int *ptr;
    int gecici;

    ptr=(int *)malloc(n*sizeof(int));

    printf("lutfen bir sayı giriniz:: ");
    scanf("%d", &n);
    srand(time(0));

    for(i=0;i<n;i++){
        ptr[i]=rand()%1000;

    }

    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(ptr[i]>ptr[j]){
                gecici=ptr[i];
                ptr[i]=ptr[j];
                ptr[j]=gecici;
            }
        }
    }
    printf("=====");
    for(i=0;i<n;i++){
```

```

printf("--> %d\n",ptr[i]);

}

free(ptr);
return 0;
}

çıktı>>
lutfen bir sayı giriniz:: 10
=====> 93
--> 187
--> 427
--> 446
--> 565
--> 631
--> 636
--> 675
--> 772
--> 825
// küçük bir \n problemi yaşanmış sallıyorum şuan.

```

!!! fonksiyonlar ile bellek yönetimi.

önceki örnekten yararlanarak 3 farklı fonksiyonla ifade edimi tasarlayacağız.

- belleyi almak ve değerlerin girişi
- değerlerin sıralanması
- değerlerin bastırılması

!!! fonksiyona pointeri tanıtırken dikkat!!!

⇒ direkt pointer tipinde alıyoruz farklı bir şey yok.

```

void anana_sor(int *ptr1,int m);

int main(){
    int n;

    int *klm;

    anana_sor(klm,n);
}

```

dizayn bu şekilde!!!

senaryoyu kuralım.

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

void sayiları_al(int *p2,int m){
    int i;
    srand(time(0));

    for(i=0;i<m;i++){
        p2[i]=rand()%1000;

    }
}

void sayıları_sırala(int *p3,int k){
    int i,j,gecici;

    for(i=0;i<k-1;i++){
        for(j=i+1;j<k;j++){
            if(p3[i]>p3[j]){
                gecici=p3[i];
                p3[i]=p3[j];
                p3[j]=gecici;
            }
        }
    }
}

void sayıları_yazdır(int *p4,int l){
    int i;
    printf("=====");
    for(i=0;i<l;i++){
        printf("\n--> %d",p4[i]);

    }
}

int main(){
    int n,j;
    int *ptr;

    ptr=(int *)malloc(n*sizeof(int));

    printf("lutfen bir sayı giriniz:: ");
    scanf("%d", &n);

    sayıları_al(ptr,n);
    sayıları_sırala(ptr,n);
    sayıları_yazdır(ptr,n);

    free(ptr);
    return 0;
}

çıktı>>
lutfen bir sayı giriniz:: 10
=====
--> 134
--> 141
--> 177
--> 197
--> 256
--> 280
```

```
--> 601  
--> 714  
--> 724  
--> 914
```

→ Baya beğendim bunu .

Pointer olarak fonksiyonlara alabildiğimiz için işlem yapması direkt aynındı.

⇒ bu zamana kadar int türü değişkenlerle baya işlem yaptık.

simdi char ve string üzerine bi tık odaklanacağız hadi bakalım

yne hafiften alacağım.

BU ÇOK ÖNEMLİ

```
#include<stdlib.h>  
#include<stdio.h>  
  
int main(){  
    char *ptr;  
    // direkt olarak alınan pointer ifdadeye ki  
    // (zaten aldığımız bir dizi ifadesi biz neden direkt bunu kullanıcıya eşitlemeyeceğim ki)  
    ptr=(char *)malloc(100*sizeof(ptr));  
    --> char yapısı ptr yerine yazıpta yapabilirdik.  
  
    printf("lütfen bir cümle giriniz :: ");  
    gets(ptr);  
  
    printf("%s",ptr);  
  
    free(ptr);  
    return 0;  
}
```

→ kullanılan şeke lütfen dikkat.

zaten aldığımız bir pointer var ve 100 kere tekrar etmiş. kısaca kullanıma hazır bir 100 lük bir karakter dizisi

direkt olarak almak istediğimiz yere eşitleyebiliriz.

bundan öte kullanım tarzı benim çok dikkatimi çekti.

belirttiğim üzere sizeof içine char diye de belirttiğimizde aynı işlemi gerçekleştiriyor.

nasıl çalıştığını gerçekten güzel bir kanıtı.

→ bir toplama örneği bulunsun diyorum burada. Hazır yapmışken.

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

int main(){
    int i,n;
    int *ptr;
    int toplam=0;

    printf("lutfen gireceginiz sayı miktarini giriniz:: ");
    scanf("%d",&n);

    ptr=(int *)calloc(n,sizeof(int));

    for(i=0;i<n;i++){

        printf("\n%d. sayı -> ",i+1);
        scanf("%d", ptr+i);
        toplam +=ptr[i];
    }

    printf("\n\n %d",toplam);
    free(ptr);
    return 0;
}

çıktı>>

lutfen gireceginiz sayı miktarini giriniz:: 4
1. sayı -> 12
2. sayı -> 432
3. sayı -> 56
4. sayı -> 456

956
```

→ geçen örneğin ortalama işlemini temel alan bir senaryo

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

int main(){
    int i,n;
    int *ptr;
    int toplam=0;
    float ort;
```

```

printf("lutfen gireceginiz sayi miktarini giriniz:: ");
scanf("%d",&n);

ptr=(int *)calloc(n,sizeof(int));

for(i=0;i<n;i++){

    printf("\n%d. sayi -> ",i+1);
    scanf("%d", ptr+i);
    toplam +=ptr[i];
}
ort=(float)(toplam/n);

printf("\n\n %.1f",ort);

free(ptr);
return 0;
}

// ne olur olmaz diye floatla ort aldım.

```

!!!

→ çok boyutlu bir dizi yapısıyla genel bir dinamik bellek hesaplaması.

```

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

int main(){
    int i,j;
    int satir,sutun;

    printf("lutfen matrisin satir sayisini giriniz:: ");
    scanf("%d", &satir);
    printf("lutfen matrisin sutun sayisini giriniz:: ");
    scanf("%d", &sutun);
    int **ptr=(int **)malloc(satir*sizeof(int *));

    for(i=0;i<satir;i++){
        ptr[i]=(int *)calloc(sutun,sizeof(int));

        for(j=0;j<sutun;j++){

            printf("\nmatris - %d. %d. =",i,j);
            scanf("%d",&(ptr[i][j]));
        }
    }
    printf("\n=====\\n");

    for(i=0;i<satir;i++){
        for(j=0;j<sutun;j++){

            printf("%3d",ptr[i][j]);
        }
        printf("\n");
    }

    // -----

```

```

        for(i=0;i<satir;i++){
            for(j=0;j<sutun;j++){

                free(ptr[i]);
            }
            free(ptr);
        }

        return 0;
    }

çıktı>>
3 3 3
3 3 3
3 3 3

```

→ bu örnek iki boyutlu dizi yapılarında nasıl bellek kullanılır gerçekten güzel bir örneği.

tekrar tekrar kritik yerlerine bakmakta yarar vardır.

ama serbest bırakma işlemide matris gibi iki adımlıdır.

.....

gösterelim.

```

// -----
for(i=0;i<satir;i++){
    for(j=0;j<sutun;j++){

        free(ptr[i]);
    }
    free(ptr);
}

--> aynı belleği nasıl iki adımlı şekilde açtıysak bunuda iki adımlı şekilde kapatmamız gerekiyor.

```

→ iki farklı bellekte random üretilen değerleri alıp 3. bellekte birleştirme üzerine bir senaryo

```

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
int main(){
    int k,m,n,i;
    int *p1,*p2,*p3;

    printf("1. bellek icin sayı miktarı :: ");
    scanf("%d", &m);
    printf("2. bellek icin sayı miktarı :: ");
    scanf("%d", &n);

    p1=(int *)malloc(m*sizeof(int));
    p2=(int *)malloc(n*sizeof(int));
    p3=(int *)malloc((m+n)*sizeof(int));
    srand(time(0));

```

```

for(i=0;i<m;i++){
    p1[i]=rand()%1000;
}

for(i=0;i<n;i++){
    p2[i]=rand()%1000;
}

k=0;

for(i=0;i<m+n;i++){

    if(i<m){
        p3[i]=p1[i];
    }
    else{
        p3[i]=p2[k];

        k++;
    }
}

for(i=0;i<m+n;i++){

    printf("%d\t",p3[i]);
}

free(p1);
free(p2);
free(p3);
return 0;
}
çikti>>>

1. bellek icin sayi miktari :: 4
2. bellek icin sayi miktari :: 4
674      211      402      889      528      766      999      80

```

gayet güzel oldu.

oluşturulan bir matrisin köşe değerlerini toplayan genel bir senaryo random fonksiyonu kullanıp bir önceki matris örneğinden yararlanacağım.

```

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
int main(){
    int j,i;
    int satir,sutun;
    int ktoplam=0;

    printf("lutfen matirisin satir sayisini giriniz:: ");
    scanf("%d", &satir);
    printf("lutfen matirisin sutun sayisini giriniz:: ");
    scanf("%d", &sutun);

```

```

srand(time(0));

int **ptr=(int **)calloc(satir,sizeof(int *));

for(i=0;i<satir;i++){

    ptr[i]=(int *)malloc(sutun*sizeof(int));
    for(j=0;j<sutun;j++){

        ptr[i][j]=rand()%1000;
        if(i==0 && j==sutun-1){
            ktoplam +=ptr[i][j];
        }
        else if(i==satir-1 && j==0){
            ktoplam +=ptr[i][j];
        }
        else if(i==0 && j==0){
            ktoplam +=ptr[i][j];
        }
        else if(i==satir-1 && j==sutun-1){
            ktoplam +=ptr[i][j];
        }
        else{

        }
    }
}

//-----

for(i=0;i<satir;i++){

    for(j=0;j<sutun;j++){
        printf("%d\t", ptr[i][j]);
    }
    printf("\n");
}

printf("\n\n--> kosegen degerlerinin toplami => %d",ktoplam);

for(i=0;i<satir;i++){

    for(j=0;j<sutun;j++){
        free(ptr[i]);
    }
}
free(ptr);

return 0;
}

çikti>>>

lutfen matirisin satir sayisini giriniz:: 5
lutfen matirisin sutun sayisini giriniz:: 5
525      656      315      355      79
764      488      758      996      387
501      437      465      530      252
957      275      420      554      700
929      759      847      101      341

--> kosegen degerlerinin toplami => 1874

```

ÇOK ÖNEMLİ!!!

normal bir cümleyi tersine çevirme mevzusu
ama gerçekten char türünde pointer mekanizmasını anlamak iç
n birebir.
tersine çevirmedeki pointer özelliklerini buradada kullanarak işlem yaptım.
birkaç sona getirme tracki kullanamadım değil 😊

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

int main(){
    char gecici;
    int i,sayac=0;

    char *ptr;
    ptr=(char *)malloc(100*sizeof(char));

    printf("lutfen bir cümle giriniz:: ");
    gets(ptr);

    for(i=0;ptr[i];i++){
        sayac++;
    }

    printf("%d", sayac);

    for(i=0;i<sayac;i++,sayac--){
        gecici=ptr[i];
        ptr[i]=ptr[sayac-1];
        ptr[sayac-1]=gecici;
    }

    printf("\n%s", ptr);
    free(ptr);
    return 0;
}

çıktı>>

lutfen bir cümle giriniz:: anana sordun mu
15
um nudros anana
// demekki bizim emektar pointer sona götürme hareketi böylede işe yarıyor :)
```

ve son egzersiz.

istenen kelimededen bir üçgen senaryosu

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

```
int main(){
    int i,j;
    int sayac=1;
    char *ptr;

    ptr=(char *)malloc(100*sizeof(char));
    printf("lutefn bir kelime giriniz:: ");
    gets(ptr);

    for(i=0;ptr[i];i++){
        for(j=0;j<sayac;j++){
            printf("%c", ptr[j]);
        }
        printf("\n");
        sayac++;
    }
    free(ptr);
    return 0;
}

çıktı>>

lutefn bir kelime giriniz:: kuskokulu
k
ku
kus
kusk
kusko
kuskok
kuskoku
kuskokul
kuskokulu
```

Keyifli bir süreçti.

searchlere balıklama zamanı.

Lecture-19 -SEARCH and SORT Algorithms-

Author/Publisher	C derlemesi - Talha AYDIN
UDEMY Source	Can BOZ

[Bubble Sort Algorithms](#)

[Selection Sort Algorithms](#)

[Insertion Sort Algorithms](#)

[Quick Sort Algorithms](#)

[Linear search Algorithms](#)

[Binary Search Algorithms](#)

Bubble Sort Algorithms

buble sort algortiması girilen sayıların sıralanıp ekrana bastırılması işlemini yapan bir algoritma.

çok kez zaten örneklemerini direkt olarak yaptığımız senaryoların direkt bir fonksiyona çevrilmiş halide diyebiliriz.

yav zaten dediğimzi gibiymiş

bunu biz adını bilemeden yapıyormuşuz.

direkt olarak syntaxini bir klasik rand ile alınan değerlerin sıralanmasını sağlayan bir senaryo ile girişelim.

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

int main(){
    int n,i,j;
    int gecici;

    printf(" kac tane sayı olusutrmak istiyorsunuz :: ");
    scanf("%d", &n);
    srand(time(NULL));
    int dizi[n];

    for(i=0;i<n;i++){
        dizi[i]=rand()%1000;
        printf("%d\t",dizi[i]);
    }

    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(dizi[i]>dizi[j]){
                gecici=dizi[i];
                dizi[i]=dizi[j];
                dizi[j]=gecici;
            }
        }
    }
    printf("\n=====\\n");

    for(i=0;i<n;i++){
        printf("%d\\n",dizi[i]);
    }

    return 0;
}

çikti>>>

kac tane sayı olusutrmak istiyorsunuz :: 10
```

```

648     140     450     948     43     370     825     818     224     409
=====
43
140
224
370
409
450
648
818
825
948

```

→ önemli gördüğüm nokta içe for döngüs esnasında ikinci döngünün şartları;;

loop mekanizmasını i den bir büyük şekilde ayarlamalıyız ki karışmasın o yüzden i+1. değerden itibaren başlatmamız gerekiydi.

Selection Sort Algorithms

→ selection sort algoritması belirli olan dizinin elemanını en küçük olanla kıyaslaması ve yer değiştirmesi işlmeine dayanıyor.

bubble mekanizmasına benziyor ama bu direkt olarak en küçük ile birlikte işlem yapıyor.

snearyo içerisinde görelim.

AŞIRI DERECEDE DEĞERLİ BİR SENARYO OLDU.

adım adım inceleyeceğiz.

```

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

int main(){
    int n,i,j;
    int gecici;
    int konum;

    printf(" kac tane sayı olusutrmak istiyorsunuz :: ");
    scanf("%d", &n);
    srand(time(NULL));
    int dizi[n];

    for(i=0;i<n;i++){
        dizi[i]=rand()%1000;
        printf("%d\t",dizi[i]);
    }
    //-----
    for(i=0;i<n-1;i++){
        konum=i;
        for(j=i+1;j<n;j++){
            if(dizi[konum]>dizi[j]){
                konum=j;
                //ÇOK DEĞERLİ BİR MANTIK
                //NOTİON İÇİNDE AYRINTILI ŞEKİLDE AÇIKLANACAKTIR.
            }
        }
        if(konum!=i){
            gecici=dizi[i];
            dizi[i]=dizi[konum];
            dizi[konum]=gecici;
        }
    }
}

```

```

-----
printf("\n=====\\n");

for(i=0;i<n;i++){
    printf("%d\\n",dizi[i]);
}
return 0;
}

```

.....

!!!

→ direkt ilk for döngüsü sonrasında alana odaklanacağım.

biz i değerini n-1 e kadar belirttiğimiz için j+1 yapıp direkt j<n yapabildik

ve sonrasında direkt olarak hangi elemanın min olduğunu tespit etmek için j li for döngüsünden yararlandık.

konum değerini belirtili i değerine eşitleyip j, i+1 den itibaren arttıkça ve eğer dizinin konum. elemanından küçükse artık yeni konum değerine eşitlendi.

bu for döngüsünün sonlanması otomatikman bize en küçük değerli dizi[konum] değerini vermiş oldu.

ve biz aşağıda dizi i en küçük elemana eşit olmadığı müddetçe ki (farklı olmasına ilgili bir if açılmıştır) değerleri yer değiştiriricek.

buradaki kritik nokta belirtilen başlangıç değerinden itibaren min değerini direkt olarak bir dizi elemanını belirtmek oldu.

bundan sonra max ve min sorularının hepsinde bu algoritmayı planlıyorum.

Insertion Sort Algorithms

→ SOKMA algoritması olarakda geçiyor.

buda bir sıralama algoritmasıdır. Farkı iki sayıdan küçük olanına olan bağlılıktır. bu durum sürekli kontrol edilir. ve adım adım kontrol ederek küçükten büyüğe sıralaması yapılmış olur.

yne aynı şekilde bir program kendim oluşturmaya çalışacağım.

ama önceki bölümlerden farklı olarak can bozun çözümlerinde burada bulunduracağım.

```

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

int main(){
    int i,n;
    int gecici;

    printf(" kac tane sayı olusutrmak istiyorsunuz :: ");
    scanf("%d", &n);
    srand(time(NULL));
    int dizi[n];

    for(i=0,i<n,i++){
        dizi[i]=rand()%1000;
        printf("%d\\t",dizi[i]);
    }
}

```

```

}

for(i=0;i<n-1;i++){

    while(dizi[i]>dizi[i+1]){
        gecici=dizi[i];
        dizi[i]=dizi[i+1];
        dizi[i+1]=gecici;

        i--;
    }
    printf("\n=====\\n");
    for(i=0;i<n;i++){
        printf("%d\\n",dizi[i]);
    }
    return 0;
}

çikti>>>

kac tane sayı olusutrmak istiyorsunuz :: 13
734      1      848      474      730      843      7      294      558      602      960      914      776
=====
1
7
294
474
558
602
730
734
776
843
848
914
960

```

Can Bozda while döngüsü kullanarak yaptığıni düşünüyorum ama yinede bulunduracağım.

Can Boz version...

```

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

int main()
{
    int N;
    int i;
    int j;
    int gecici;
    printf("Kac adet sayı uretilecek\\n");
    scanf("%d",&N);
    int dizi[N];

    printf("Dizi elemanlari\\n");

    for(i=0;i<N;i++)
    {
        scanf("%d",&dizi[i]);
    }
    printf("\\nDizi elemanlari\\n");
    for(i=0;i<N;i++)
    {
        printf("%d\\n",dizi[i]);
    }

    for(i=1;i<N;i++)
    {
        j=i;
        while(j>0 && dizi[j]<dizi[j-1])
        {
            gecici=dizi[j];
            dizi[j]=dizi[j-1];
            dizi[j-1]=gecici;
            j--;
        }
    }
}
```

```

        dizi[j]=dizi[j-1];
        dizi[j-1]=gecici;

        j--;
    }
    printf("\n Siralanan Dizi Elemanları\n");
    for(i=0;i<N;i++)
    {
        printf("%d\t",dizi[i]);
    }

```

→ temel farklılıklar::
can Boz farklı bir değişken atayarak for döngüsünde o değişkeni tekrar etti.
yani sıraladığında değerleri bir daha kontrol etmeden direkt olarak i nin devam ettiği yerden işlem yaptı ve eşit bir değerle while döngüsünde küçük olanı başa götürüp götürüp hareket etti.
geri kalanlar değerel farklılıklar.

→ bu bakış açısından gayet önemli incelemekte katı önem var.

Quick Sort Algorithms

→ çeviri yanlışdır diye üzerine gitmedik (ki adam haklı)
biraz uzun bir sıralama şekli ama gerçekten değerli. Direkt olarak bunu iyice öğrenmek için can boz örneğinden ilerleyelim.
⇒
bir tık problem bir algoritma sonrasında yeniden bakacağım.

```

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

int main(){
    int n,i,aranan;

    printf("almak istediginiz deger sayisi:: ");
    scanf("%d",&n);
    int dizi[n];
    for(i=0;i<n;i++){
        printf("\n-> ");
        scanf("%d",&dizi[i]);
    }

    printf("\n\nnaradiginiz sayi:: ");
    scanf("%d", &aranan);

    for(i=0;i<n;i++){
        if(dizi[i]==aranan){
            printf("aradiginiz sayi dizinizin %d. elemanidir.",i+1);
        }
    }

    return 0;
}
-> bu

```

Linear search Algorithms

→ lineer search klasik bir aradığımız sayı kaçinci indexte kullanıcıya bunu göstermeye yarıyor. Bildiğimiz bir şey.
quick sorttan sonra bu pek kesmedi 😊

Binary Search Algorithms

→ bu aradığımız bilgiyi lineer search mekanizmasına göre daha hızlı bulmak için kullandığımız ve olduğunda efektivite amaçlı bir algoritma yapısıdır.

ve bu arama işlemi gerçekten hayat kurtarıcı olabilir.[çoğu genel büyük işlerdede kullanılmaktadır.]

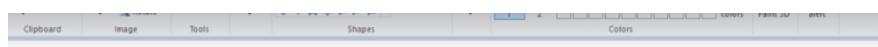
binary search büyükten küçüğe sıralanmış bir dizide orta indexlemesi ile çalışır.

yani sürekli orta elemana karşı bir fantezisi vardır.

bu orta elemandan büyülüklük veya küçüklük çerçevesinde

koşullar dizisi ile çalışır.

buda norma bir dizide istenen elemanı aramaktan çok daha fazla zaman kazandırır.



```
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
-----
void binary_search(int d[], int ilk, int son, int x){
    int i, j;
    int n;
    int ort=(ilk+son)/2;

    while(ilk<=son){
        if(x==d[ort]){
            printf("\nsayiniz dizinin %d. elemanına esittir.",ort+1);
            exit(0);
        }
        else if(x>d[ort]){
            ilk = ort+1;
        }
        else{
            son = ort-1;
        }
    }
}
```

```

    }

    else{
        son=ort-1;
    }

    ort=(ilk+son)/2;
}

if(ilk>son){
    printf("sayi bulunamadi");
}
-----
```

```

int main(){
    int n,i,j;
    int gecici;
    int aranan;

    printf(" kac tane sayı oluşturmak istiyorsunuz :: ");
    scanf("%d", &n);
    srand(time(NULL));
    int dizi[n];

    for(i=0;i<n;i++){
        dizi[i]=rand()%1000;
        printf("%dt",dizi[i]);
    }

    printf("\naranan değeri giriniz:::");
    scanf("%d", &aranan);
    printf("\n\n");
    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(dizi[i]>dizi[j]){
                gecici=dizi[i];
                dizi[i]=dizi[j];
                dizi[j]=gecici;
            }
        }
    }
    //Can boz uğreşaceağımız son ilk ve orta değerleri girdi sayısı etrafında atama yapıp işlem yaptı.
    // uğraşılacak kodun dinamiği ve komplikeliliği içinde önemli aslında böyle yapmak.(sonuçta bu değerler ile işlem yapacağız.)
    int ilk=0;
    int son=n-1;

    binary_search(dizi,ilk,son,aranan);

    return 0;
}
```

⇒ işaretli olan kısma ve yaptığımız genel algoritma mantığına şiddetle dikkat.

ilk son ve ort değerleri ile birlikte yaptığımız genel bir arama şeklidir.

ve buradaki ortalamanın küsratlı gelmesi sistemi ilginç şekilde etkilemedide

binary search ve quick sort algoritmalarını boş bulundukça tekrar edip türevlerini üretmeyi düşünüyorum.

ve bu durum artık C programlamanın öğrenim kısmının sonu olmakta aynı zamanda bu bu çalışmanın sonudur.
bundan sonra master seviye c programlama mayın tarlası ve genel işlemler ve alıştırmalar olacak.

