

## FONKSİYONLAR

### FONKSİYON MANTIĞI VE TANIMLAMASI

Biz fonksiyonları matematikten tanıyoruz

Aşağıdaki fonksiyonu inceleyelim:

$$f(x) = 3x + 5$$

bu fonksiyon bir sayı alıyor ve o sayının 3 katının 5 fazlasını döndürüyor yani  $f(5)$  in sonucu 20 oluyor.

programlamada fonksiyonlar isimleri olan, belirli kodlardan oluşan ve belirli işlemleri yapan, tekrar tekrar kullanabilen kod topluluğudur.

bir fonksiyonun nasıl tanımlanacağına bakalım

```
döndürülecek_veri_tipi fonksiyon_ismi(parametre_veri_tipi parametre_ismi){  
    kodlar ( bu kısma aynı zamanda body de denir)  
    return döndürülecek_değer;  
}
```

şimdi  $f(x) = 3x + 5$  fonksiyonunun kodunu yazalım

fonksiyonun sadece tamsayı alacağını varsayalım -> parametre veri tipi: int

parametre ismi: istediğimi verebilirim: sayı

döndüreceği veri tipi: eğer x tamsayı olursa  $3x + 5$  tamsayı olur -> en azından int olarak tanımlarım ama double, int tipini kapsadığı için double da tanımlayabilirim.

fonksiyon ismi: istediğimi verebilirim: foo (foo kelimesi internette öylesine yazılmış fonksiyon isimlendirmek için kullanılan bir kelimedir)

Şimdi nerede tanımladığımıza geleyim. bir fonksiyon başka bir fonksiyon içinde kullanılabilir ama başka bir fonksiyonun içinde tanımlanamaz. Biz de main dışında tercihen main fonksiyonunun üstünde tanımlayacağız. aşağısında da tanımlayabiliriz ama bunu ilerde "prototip" dersinde göreceğiz.

```

C ders1.c > ...
1  #include <stdio.h>
2
3  double foo(int sayi){
4
5      double sonuc = 3 * sayi + 5;
6
7      return sonuc;
8  }
9
10 int main(){
11
12     int bir_sayi;
13     printf("bir sayi girin: ");
14     scanf("%d",&bir_sayi);
15
16     double uc_katinin_5_fazlasi = foo(bir_sayi);
17
18     printf("%f\n",uc_katinin_5_fazlasi);
19
20     return 0;
21 }

```

uc\_katinin\_5\_fazlasi değişkeni double olarak tanımladık çünkü foo fonksiyonunun sonucu bir double.

bir\_sayi değişkeninin tipini tekrar yazmadık

bu programın nasıl çalıştığından bahsedelim:  
ilk önce bir sayı aldık. diyelim ki 5 verdik

16. satırda önce = in sağ tarafı çalışıyor ve o kısımda bir fonksiyon var.

foo fonksiyonu, içine 5 sayısını alarak çalışmaya başlar.

sonuc değişkeni  $3 * 5 + 5$  ile 20.000000 değerini alır

return sonuc ifadesi return 20.000000 olarak çalışır

en sonunda foo(5) ifadesinin değeri 20.000000 olur.

20.000000 değeri uc\_katinin\_5\_fazlasi değişkenine atanıyor

uc\_katinin\_5\_fazlasi değişkeninin değeri bastırılıyor

foo fonksiyonunu sonuc isimli değişken tanımlamadan aşağıdaki gibi de yazabiliriz ama ben tarz olarak bu şekilde pek kod yazmıyorum.

```
double foo(int sayi){  
  
return 3 * sayi + 5;  
}
```

sonraki derste iki tane örnek soru çözeceğiz

SORU1:Parametre olarak aldığı yarıçap değeri ile dairenin alanını bulan alan\_bul adlı fonksiyon yazın ve kullanıcıdan alınan değerle dairenin alanını hesaplayın

```
C ders2.1.c > ...  
1  #include <stdio.h>  
2  
3  double alan_bul(double r){  
4  
5      const double pi = 3.14159;  
6  
7      double alan = pi * r * r;  
8  
9      return alan;  
10 }  
11  
12 int main(){  
13  
14     double sayi;  
15     printf("bir yarıcap girin: ");  
16     scanf("%lf",&sayi);  
17  
18     double alan = alan_bul(sayi);  
19  
20     printf("alan: %f\n",alan);  
21  
22     return 0;  
23 }
```

SORU2: parametre olarak taban ve yükseklik değeri alıp dikdörtgen alanını bulan fonksiyonunu yazınız

```
C ders2.2.c > alan_bul(double, double)
1  #include <stdio.h>
2
3  double alan_bul(double taban, double yukseklik){
4
5      double alan = taban * yukseklik;
6
7      return alan;
8  }
9
10 int main(){
11
12     double taban,yukseklik;
13     printf("taban ve yukseklik girin");
14     scanf("%lf %lf",&taban, &yukseklik);
15
16     double alan = alan_bul(taban, yukseklik);
17
18     printf("alan: %f\n",alan);
19
20     return 0;
21 }
```

## RETURN

return ifadesine gelindiğinde fonksiyon birer ve bir değer döndürür. fonksiyonlar çoğunlukla bir değer döndürürler. Bu ders değer döndürme işleminden bahsedeceğiz.

return ifadesi fonksiyonun sonunda olmak zorunda değildir. Aşağıdaki kodu inceleyelim

```
double kup_hacim(double kenar){  
    if (kenar < 0) {  
        return 0;  
    }  
    double hacim = kenar * kenar * kenar;  
    return hacim;  
}
```

verilen kenar uzunluğu negatif olduğunda hacim hesaplamadan 0 döndürüyoruz çünkü negatif uzunlukta bir kenar olamaz. negatif bir sonuç çıkmaması için fonksiyona bir nöbetçi koşul ekledik.

bu fonksiyonun başka bir halini yazalım. Aşağıdaki kodu inceleyelim.

```
C a.c > ...  
1  #include <stdio.h>  
2  
3  double kup_hacim(double kenar){  
4  
5      if (kenar >= 0) {  
6          double hacim = kenar * kenar * kenar;  
7  
8          return hacim;  
9      }  
10  
11 }  
12  
13 int main(){  
14  
15     printf("%f\n",kup_hacim(-10));  
16  
17     return 0;  
18 }
```

bu kodda eğer fonksiyon negatif bir değer alırsa if bloğu çalışmıyor ve bir return ifadesiyle karşılaşmıyoruz. yine de bir negatif değer -10 verdik

bu kodu derleyip çalıştırmayı denersek hatalı bir değer veya gönderdiğimiz

sayıyı alırız çünkü fonksiyon -10 aldığında returnla karşılaşmamış yani bir sıkıntı var ve bu sıkıntıyı derleyici şimdilik göstermiyor. Bazılarınızın derleyicisinde kod uyarı vermiş olabilir ama benim derleyicimde vermedi. bu uyarıyı almak için olabilecek tüm uyarı ve hataları gösteren bir derleyici komutu olan "-Wall" kullanacağım

```
mehmet@mehmet-R580-R590:~/Masaüstü/EGITIM/bolum4$ gcc a.c -Wall -o a
a.c: In function 'kup_hacim':
a.c:11:1: warning: control reaches end of non-void function [-Wreturn-type]
}
```

Diyor ki bazı durumlarda return ile karşılaşmıyorum. Bunu aşmak için ilk fonksiyonu baştaki gibi tanımlamalıyız ya da if bloğunun içinde bir adet return ve fonksiyonun sonunda problemliler için ayrı bir return ifadesi kullanabiliriz.

```
double kup_hacim(double kenar){
    if (kenar >= 0) {
        double hacim = kenar * kenar * kenar;
        return hacim;
    }

    return 0;
}
```

## VOID FONKSİYONLAR

Önceki derste fonksiyonların "çoğunlukla" bir değer döndürdüğünü söylemiştik demek ki her zaman döndürmüyor.

Bir değer döndürmeyen fonksiyonlara void fonksiyonlar denir ve birtakım işlemler için kullanılır.

Mesela `dunyayi_selamla` isminde `int` tipinden parametre alan ve o parametre kadar ekrana "hello world" yazdıran bir fonksiyon tanımlayalım.

`dunyayi_selamla` fonksiyonu sadece mesaj yazacak o yüzden `sayi` cinsinden bir değer döndürmesine gerek yok

void fonksiyonu tanımlarken normal fonksiyonlardan farklı olarak döndürülecek\_değer\_veri\_tipi kısmına void yazıyoruz ve `return` ifadesini kullanmıyoruz.

ayrıca void fonksiyonlar bir değer döndürmediği için `dunyayi_selamla` fonksiyonunu bir değişkene atamak zorunda değiliz.

```
C ders4.c > ...
1  #include <stdio.h>
2
3  void dunyayi_selamla(int sayi){
4      |
5      |     for(int i = 1; i <= sayi; i++)
6      |         printf("hello world\n");
7      |
8      |
9      |
10     |
11     |
12     |
13     |
14     |
15     |
16     |
17     |
18     |
19 }
```

Void fonksiyon içinde return kullanmamız gerekebilir.  
Diyelim ki `dunyayi_selamla` fonksiyonuna negatif değer verildiği zaman "imkansiz" diyip fonksiyonu sonlandırmak istiyoruz. bunun için "return;" ifadesini kullanacağız.

```
C ders4.2.c > ...
1  #include <stdio.h>
2
3  void dunyayi_selamla(int sayi){
4
5      if(sayi < 0){
6          printf("imkansiz\n");
7          return;
8      }
9
10     for(int i = 1; i <= sayi; i++)
11         printf("hello world\n");
12 }
13
14 int main(){
15
16     dunyayi_selamla(-5);
17
18     return 0;
19 }
```



## FONKSİYON PROTOTİPİ

Şu zamana kadar fonksiyonları hep main in üstünde tanımladık bu ders mainin altında tanımlayacağız.

C kodlarının yukarıdan aşağıya doğru çalıştığını biliyoruz. eğer fonksiyonu altta tanımlarsak main çalışırken henüz fonksiyonlarımız tanımlı olmadığı için main fonksiyonu çalışmaz. bu problemi çözmek için fonksiyonları aşağıya yazar, main üzerine ise fonksiyon prototiplerini ekleriz

bir prototip aşağıdaki yapıda tanımlanır

dönecek\_veri\_tipi fonksiyon\_ismi(param\_tipi param\_ismi) ;

yani yapacağımız şey sadece fonksiyonun ilk satırını yazıp ; koymak

bir tane örnek yapalım:

soru: içine parametre almayan void bir "foo" fonksiyonu yazın bu fonksiyon çalıştığında kullanıcıdan bir sayı alıp iki katını bastırsın:

```
C ders5.1.c > ...
1  #include <stdio.h>
2
3  void foo();
4
5  int main(){
6
7      foo();
8
9      return 0;
10 }
11
12 void foo(){
13     int sayi;
14     printf("foo fonksiyonu calisiyor\n");
15     printf("bir sayi girin: ");
16     scanf("%d",&sayi);
17
18     printf("%d\n",sayi * 2);
19 }
```

prototipler çok önemlidir. #include <stdio.h> satırı ile stdio.h kütüphanesindeki fonksiyonların prototiplerini kodumuza dahil ederiz yani .h uzantılı dosyalar fonksiyonların prototiplerini içerir. prototipler var ama fonksiyon body'leri nerede diye soracak olursanız derleyicinizin olduğu dosyaların içinde. onları programa dahil etmek ise linker-bağlayıcının görevi

Son olarak prototipler fonksiyonları iç içe kullanmakta bize çok fayda sağlar. iki tane fonksiyonu mainin üstünde tanımlayalım ve birini diğerinin içinde kullanalım.

```
C ders5.2.c > ...
1  #include <stdio.h>
2
3  void goo(){
4      printf("goo calisti\n");
5      foo();
6  }
7
8  void foo(){
9      printf("foo calisti\n");
10 }
11
12 int main(){
13
14
15     return 0;
16 }
```

yukarıdan aşağı doğru çalışıyoruz ilk önce goo okunur ve içerdeki foo henüz tanımlı olmadığı için programı derlemeye çalıştığımızda

```
mehmet@mehmet-R580-R590:~/Masaüstü/EGITIM/bolum4$ gcc ders5.2.c -o ders5.2
ders5.2.c: In function 'goo':
ders5.2.c:5:5: warning: implicit declaration of function 'foo'; did you mean 'goo'? [-Wimplicit-function-declaration]
    foo();
    ^~~~
    goo
ders5.2.c: At top level:
ders5.2.c:8:6: warning: conflicting types for 'foo'
    void foo(){
    ^~~~
ders5.2.c:5:5: note: previous implicit declaration of 'foo' was here
    foo();
    ^~~~
```

iki adet uyarı alıyoruz. bu hatayı düzeltmek için foo yu yukarı goo yu aşağıya tanımlamak gayet makul bir çözüm fakat çok fazla iç içe fonksiyon yazılacağı zaman prototip kullanarak bu sıralama kaygısından tamamen kurtulabiliriz.

```
C ders5.2.c > ...
1  #include <stdio.h>
2
3  void goo();
4  void foo();
5
6  //////////////////////////////////////
7  int main(){
8
9      foo();
10     goo();
11     return 0;
12 }
13 //////////////////////////////////////
14
15 void goo(){
16     printf("goo calisti\n");
17     foo();
18 }
19
20 void foo(){
21     printf("foo calisti\n");
22 }
```

foo yu aşağıda tanımlamamıza rağmen sıkıntı yok

## VARIABLE SCOPE - ÖMÜR KATEGORİSİ

Burası benim ilk okulum olan Uluborludaki Atatürk ilköğretim okulu



Burası da Ispartadaki Atatürk İlköğretim Okulu





burası da Antalyadaki Atatürk İlköğretim Okulu




Bunun konumuzla ne alakası var diye soracak olursanız: farklı şehirlerde aynı isimlerde okullar var ve bu okulların farklı sayıda öğrencisi var. Programlamada ise farklı fonksiyonlar içinde aynı isimde değişkenler tanımlayabiliriz.

```
C ders6.c > ...
1  #include <stdio.h>
2
3  void uluborlu(){
4      int ataturk_100 = 100;
5  }
6
7  void isparta(){
8      int ataturk_100 = 250;
9  }
10
11 void antalya(){
12     int ataturk_100 = 350;
13 }
14
15 int main(){
16     uluborlu();
17     isparta();
18     antalya();
19
20     return 0;
21 }
22 }
```


Peki bu nasıl oluyor hemen c tutora geçelim:  
gri ok o anda çalışan satırı  
kırmızı ok bir sonra çalışacak satırı gösteriyor

```
C (gcc 4.8, C11)
EXPERIMENTAL! known limitations
1
2
3 void uluborlu(){
4     int ataturk_ioo = 100;
5 }
6
7 void isparta(){
8     int ataturk_ioo = 250;
9 }
10
11 void antalya(){
12     int ataturk_ioo = 350;
13 }
14
15 int main(){
16
17 →   uluborlu();
18     isparta();
19     antalya();
20
21     return 0;
22 }
```



henüz başlamadı uluborlu fonksiyonundan başlayacak ve main susturulacak

```
C (gcc 4.8, C11)
EXPERIMENTAL! known limitations
1
2
3 void uluborlu(){
4 →   int ataturk_ioo = 100;
5 → }
6
7 void isparta(){
8     int ataturk_ioo = 250;
9 }
10
11 void antalya(){
12     int ataturk_ioo = 350;
13 }
14
15 int main(){
16
17     uluborlu();
18     isparta();
19     antalya();
20
21     return 0;
22 }
```



uluborlu fonksiyonuna girdi ve ataturk\_ioo değişkenini 100'e eşitledi.bir sonraki adımda fonksiyondan çıkılacak ve maine geri döndürülecek

```
C (gcc 4.8, C11)
EXPERIMENTAL! known limitations
2
3 void uluborlu(){
4     int ataturk_iao = 100;
5 }
6
7 void isparta(){
8     int ataturk_iao = 250;
9 }
10
11 void antalya(){
12     int ataturk_iao = 350;
13 }
14
15 int main(){
16
17     uluborlu();
18     isparta();
19     antalya();
20
21     return 0;
22 }
```

Stack

main

uluborlu bitince maine geri döndük ve farkettilseniz uluborlu adlı kutucuk ve değişken kayboldu. { ve } arasındaki kodlara blok denir ve bloklardaki kodlar bittiğinde blok içindeki değişkenler kaybolur ve bloklara tekrar girildiği zaman o değişken kaybolur( yok edilir ). bunu şöyle düşünebilirsiniz: Ispartadan Antalyaya gittiğimizde Ispartadaki Atatürk İlköğretim Okulu bizim için yok olur ve Antalyadakine gidebiliriz. Birazdan isparta fonksiyonu çalışacak ve farklı bir ataturk\_iao değişkeni yaratılacak

```
C (gcc 4.8, C11)
EXPERIMENTAL! known limitations
2
3 void uluborlu(){
4     int ataturk_iao = 100;
5 }
6
7 void isparta(){
8     int ataturk_iao = 250;
9 }
10
11 void antalya(){
12     int ataturk_iao = 350;
13 }
14
15 int main(){
16
17     uluborlu();
18     isparta();
19     antalya();
20
21     return 0;
22 }
```

Stack

main

isparta

ataturk\_iao 250

isparta fonksiyonundan çıkıldığında o değişken de kaybolacak. main fonksiyonuna geri dönecek. antalya fonksiyonu için farklı bir ataturk\_ıoo değişkeni yaratılacak. antalya fonksiyonu bittiğinde o da yok edilecek.

Bir blokta tanımlanan değişkenlere "yerel değişken" - "local variable" denir ve söylediğimiz gibi blok bitince o değişken de yok olur. Bununla alakalı söyleyebileceklerimiz:

1) bir fonksiyondaki yerel değişkeni main içinde bastıramayız.

```
1  #include <stdio.h>
2
3  void uluborlu(){
4      int ataturk_ıoo = 350;
5  }
6
7  int main(){
8
9      uluborlu();
10
11     printf("%d\n", ataturk_ıoo);
12
13     return 0;
14 }
```

```
mehmet@mehmet-R580-R590:~/Masaüstü/EGITIM/bolum4$ gcc ders6.c -o ders6
ders6.c: In function 'main':
ders6.c:11:19: error: 'ataturk_ıoo' undeclared (first use in this function)
printf("%d\n", ataturk_ıoo);
                  ^~~~~~
ders6.c:11:19: note: each undeclared identifier is reported only once for each function it appears in
```

ataturk\_ıoo değişkeninin tanımlı olmadığını söylüyor çünkü 11. satıra gelinceye kadar ataturk\_ıoo değişkeni yok ediliyor.

2) aynı blokta aynı isimde iki değişken tanımlanamaz yani fonksiyon içinde parametrelerle aynı isimde 2 değişken tanımlanamaz.

```
void foo(int sayi){
    int sayi = 5;
}
```

```
mehmet@mehmet-R580-R590:~/Masaüstü/EGITIM/bolum4$ gcc ders6.c -o ders6
ders6.c: In function 'foo':
ders6.c:4:9: error: 'sayi' redeclared as different kind of symbol
    int sayi = 5;
    ^~~~
ders6.c:3:14: note: previous definition of 'sayi' was here
void foo(int sayi){
           ^~~~
```



blok içine başka bir blok koyarak bu kuralı aşabiliriz

```
void foo(int sayi){  
  
    if(true){  
        |   int sayi = 5;  
    }  
  
}
```

ama okunabilirliği ve anlamayı zorlaştırdığı için ikisini de kesinlikle önermiyorum.

3) bir fonksiyonu birden fazla defa kullanabiliriz

```
1  #include <stdio.h>  
2  
3  
4  int kare_al(int sayi){  
5  
6      int kare = sayi * sayi;  
7  
8      return kare;  
9  }  
10  
11 int main(){  
12  
13     int sayi1 = kare_al(5);  
14     int diger_sayi = kare_al(10);  
15  
16     return 0;  
17 }
```

fonksiyon her çağrıldığında sayi parametresi ve kare değişkeni tekrardan yaratılır.

## GLOBAL VARIABLES

Önceki derste farklı fonksiyonlarda geçerli aynı isimdeki değişkenlerden bahsettik. Peki tüm fonksiyonların kullanabileceği bir değişken türü var mı? Cevap evet. Böyle değişkenlere "global variable" "global değişken" denir ve yok edilmemesi için main dahil bütün fonksiyonların dışında tanımlanır.

bir tane global değişken tanımlayalım ve onu bastıralım.

```
C ders7.c > ...
1  #include <stdio.h>
2
3  int sayi = 5;
4
5  int main(){
6
7      printf("%d\n",sayi);
8
9      return 0;
10 }
```

Local değişkenlerden farklı olarak global değişkenleri farklı fonksiyonlarda kullanabiliriz bunun dışında diğer değişkenler gibi bir global değişkene atama yapabilir, tuttuğu değeri değiştirebilir ; return değeri olarak kullanabilir ve ona

```
#include <stdio.h>

int hesap = 100;

void para_yatir(int para){
    hesap+=para;
}

int ikiye_katla_aktar(){
    return hesap*2;
}

int main(){
    para_yatir(550);
    int yeni = ikiye_katla_aktar();
    int iki = ikiye_katla_aktar();
    hesap = ikiye_katla_aktar();
    printf("%d %d %d\n",yeni, iki, hesap);

    return 0;
}
```

bir fonksiyon return edebiliriz.

Global değişkenlerin takibi zor olduğundan kullanılması tavsiye edilmez.

## DERS 8

### CALL BY VALUE

Bu derste scope konusunda son bir şey söyleyeceğiz.

```
C ders8.c > ...
1  #include <stdio.h>
2
3  void ikiyle_carp(int sayi);
4
5  int main(){
6
7      int i = 5;
8
9      ikiyle_carp(i);
10
11     printf("%d\n",i);
12
13     return 0;
14 }
15
16 void ikiyle_carp(int sayi){
17
18     sayi *= 2;
19
20 }
```

Kodun çıktısı 5 çünkü 8. satırda aslında `ikiyle_carp(5)` yazıyor. Yani `ikiyle_carp` fonksiyonu değişkenle değil değişkenin tuttuğu değerle ilgileniyor. `i` sayısının değerinin değiştirmek istiyorsak:

1) `int` döndüren fonksiyon fonksiyon yazıp `i` ye atayacağız :

```
C ders8.c > ...
1  #include <stdio.h>
2
3  int ikiyle_carp(int sayi){
4      return sayi * 2;
5  }
6
7  int main(){
8
9      int i = 5;
10
11     i = ikiyle_carp(i);
12
13     printf("%d\n",i);
14
15     return 0;
16 }
```

2) pointer konusunda öğreneceğimiz “call by referance” yöntemini kullanacağız.

## RECURSIVE-ÖZYİNELEMELİ FONKSİYONLAR

Kendi kendini çağıran fonksiyonlara özyineli (recursive) fonksiyon denir. Özyineleme-recursion, problemi, en üstten başlayarak adım adım daha küçük parçalarına ayırmaktır. problemin daha parçalanamayan ya da parçalanmaması gereken adımına taban parçası "base case" denir. Bir recursive fonksiyon yazarken yapmamız gereken şey base case ve daha küçük bir parça bulmaktır.

Faktoriyel bulan recursive fonksiyon yazalım:

faktoriyel o sayıya kadar olan sayıların çarpımıdır 5! sayısını ele alalım

$$5! = 5 * 4 * 3 * 2 * 1$$

daha küçük parçalara nasıl ayırabiliriz? Farkettiyseniz 1 den 4 e kadar olan sayıların çarpımı 4! eder ve 5! in içinde 4! var

$$5! = 5 * 4!$$

4! içinde de 3! var

$$5! = 5 * 4 * 3!$$

bu şekilde f faktöriyel bulan fonksiyon olmak üzere

$f(n) = n * f(n-1)$  formülüyle faktöriyeli daha küçük parçaya ayırabiliriz.

Base case'e gelelim: hangi sayıdan sonra bu işlem devam etmiyor? 1 sayısından sonra. 1! zaten 1 olduğu için daha fazla parçalamaya gerek yok. 0! de bir olduğu için base case'e 0 ekleyebiliriz.

C ders8.c > ...

```
1  #include <stdio.h>
2
3  int fak(int sayi);
4
5  int main(){
6
7      int i;
8      printf("bir sayi girin: ");
9      scanf("%d",&i);
10
11     printf("girdiginiz sayinin faktoriyeli: %d\n",fak(i));
12
13     return 0;
14 }
15
16 int fak(int sayi){
17
18     if(sayi == 1 || sayi == 0){
19         return 1;
20     }
21
22
23     return sayi * fak(sayi - 1);
24 }
```