

6- Turbo Pascal Programlamada Fonksiyonlar

6.1 Yapısal Fonksiyonlar

Yapısal bilgisayar programlama dillerinde fonksiyon ve alt programlar (Turbo Pascalda PROCEDURE) olmadan program yazmak oldukça zordur. Karşılaşılan problemler çoğu zaman alt program veya fonksiyonlara ayrılmadan yapılması sonuçları etkileyeceği gibi , problemlerin uygulanabileceği alanları da daraltmaktadır. Örneğin yazılan bir program kodu genelleştirilerek daha sonradan yazılacak başka programlarda kullanılabilir. Programlama işlemlerini modüler hale getirmek yapısal programlamada oldukça kolaydır. Modülerite yazılan program kodlarını başka bir program içerisinden çağrılmasına olanak vermektedir.

Matematikte kullanılan fonksiyonlar örneğin trigonometrik fonksiyonlar sin, cos, tan v.s. daha önceden formu bilinen ve hesaplanacak değerlerin fonksiyonun karakteristiğine göre olan fonksiyonlar olarak tanımlayabiliriz. Turbo Pascal bilgisayar programlama dilinde fonksiyonları yazım kurallarına göre yazarak ana program bloğu içerisinde kullanırız. Fonksiyonun yazımı için önce tanımlanması ve kullanılacak değişkenlerinin tanımlanması ve fonksiyonun geri dönüş değerinin belirlenmesi gereklidir. Turbo pascal da fonksiyon önce program içerisinde tanımlanmalıdır. Dışarıdan çağrılacak fonksiyonlar veya alt programlar için operatör (*\$I DIS_DOSYA.PAS*) komutları kullanılır. Bu komutların kullanılma amacı önceki Turbo Pascal bilgisayar programlama dillerinde editörler bilgisayarın hafızasında 62KB (KiloByte) fazlasını tutamamaktaydı. Program derlendiği anda çalışma zamanı hatası (Runtime Error) vermekte ve programın çalışmamaktaydı. Dışarıdan çağırma (Include) işlemi programınıza aynı zamanda daha önceden yazmış olduğunuz kodların eklenmesini sağlar. Örneğin daha önceden yazdığınız bir fonksiyon veya alt programı (Procedure) yazdığınız program kodları içerisine gömebilirsiniz.

```
Function beta(VAR z,w: real): real;  
BEGIN
```

```
beta := exp(gammln(z)+gammln(w)-gammln(z+w))
END;
```

Bet dağılımını gösteren Turbo Pascal fonksiyon kodları olsun. Programlama kodları içerisinde bunu çağırmak için programı yazdığımız dizine bu fonksiyonu BETA.PAS adı ile saklayalım. Daha sonra program kodlarımızı aşağıdaki gibi yazalım

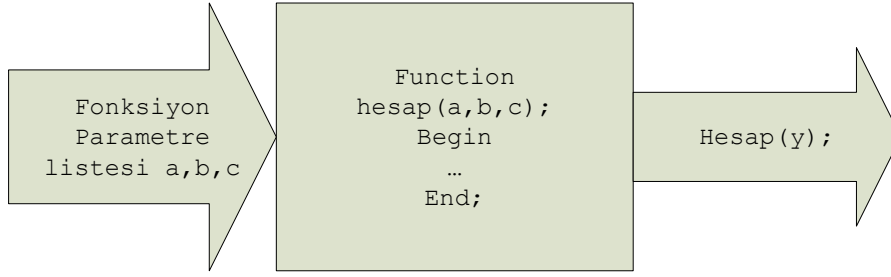
```
Program beta_dag;
VAR
  a1,a2: real;

(*$I BETA.PAS *)

BEGIN
  writeln('Beta dağılım için alfa 1 ve alfa 2
  değerlerini giriniz -> ');
  readln(a1,a2);
  writeln(a1:6:2,a2:6:2,'Değeri için Beta dağılımı
  değeri ':5 ,beta(a1,a2):13)
END.
```

Klavyeden girdiğimiz beta dağılımına ait parametrelere göre dağılımın mod ve medyanı ve alacağı değer bulunabilir. Bu örneği biraz daha genişletebiliriz. Belirli bir aralık değeri olarak verilen aralık değerlerini hesaplayabilir veya bir dosyadan verilen aralık değerlerini hesaplayabiliriz. Fonksiyonlarda parametrelerin kullanılması önemli bir konudur. Parametreler fonksiyonun ana programa veya alt programlara geçireceği/aktaracağı değişken değerlerini taşımaktadır. Parametre olarak bahsedilene örnek olarak verebileceğimiz `function hesap(n);` ifadesinde n hesap fonksiyonunun parametresidir.

Fonksiyonlarda girdi olarak verilen birden fazla parametre değerleri fonksiyondan bir sonuç değeri olarak çıkar. Bir sayının karekökünü almak için kullanılan `SQRT(n)` gibi bir fonksiyonda girilen değer $n=64$ ise çıkan değer sekiz (8) olacaktır. Karekök fonksiyonu görüldüğü gibi tek bir değer üretecek ve sonuç olarak gönderecektir.



Şekil 6.1 Fonksiyon Parametre İşleme Listesi.

Fonksiyonları değerleri herhangi bir değişkene atanabilir. Ön tanımlı olarak kullanılan fonksiyonlar da dahil olmak üzere herhangi bir fonksiyonun değerini bir değişkene atayabilir veya bir başka değişken ile toplayıp tekrar başka bir değişkene atanabilir.

$t = \frac{1}{2z} * \ln k$ gibi bir t eşitliğimiz (6.1) olsun. Bu eşitlik daha önceden tanımlı olan doğal logaritma fonksiyonunu kullanarak Pascal bilgisayar programlama kodlarını yazalım.

$$t := (\text{LN}(k) / (2 * z)); \quad (6.1)$$

Eşitlik hesaplanmaya başlarken önce 2 ile z değeri çarpılacak daha sonra k değişkenini doğal logaritması alınacak ve doğal logaritması alınan k değeri 2 ile çarpılan z değerine bölünecektir. Görüldüğü gibi ön tanımlı olan fonksiyon $\text{LN}(k)$ bir eşitlik içerisinde de kullanılmakta ve değerleri diğer değişkenlere aktarılabilmektedir.

Sıralı iki değer veya toplam değişken değerlerini fonksiyonlara aktararak fonksiyonun alacağı değerlerde kullanabiliriz.

```

k:= SQRT(sayi_1);
k:= SQRT(sayi_2);
k:= SQRT(sayi_1+ sayi_2);

```

Yazım kurallarına uygun fonksiyon kullanımına örnek olarak verilebilir. Hazır fonksiyon olan karekök fonksiyonuna normal değer `sayi_1` veya `sayi_2` ile aktarımlar yapılır. Değişken değer aktarımı iki sayının toplamından da oluşabilir. Sonuç değeri `sayi_1` ve `sayi_2` toplamından oluşur.

6.2 Öntanımlı (Hazır) Fonksiyonlar

Öntanımlı fonksiyonlar derleyici üreten firmaların hazır olarak programcının kullanımına sunduğu fonksiyon kütüphanesinde yer alan fonksiyonlardır. Bu fonksiyonlar üretici firmanın yanı sıra üçüncü parti yazılımcıları tarafından da yazılarak hazır kodlar haline getirilebilir. Hazır hale getirilen kodlar bazı özel durumlar için üretilmiş olabilir. Özel durumlar için yazılmış olan hazır kod veya kütüphaneler ücretli olarak satılmakta veya araştırmacıların kullanımına sunulmaktadır.

Turbo Pascal bilgisayar programlama dilinde hazır matematik kütüphanesi oldukça kullanışlı ve yararlıdır. Bu kütüphanelerden yararlanılarak kendi kütüphanemizi oluşturabilir veya özel durumlar için kendi kütüphanemizi yazabiliriz. Turbo pascal bilgisayar programlama dilinde hazır bazı matematik fonksiyonları **Tablo 6.1**'de verilmiştir. **Tablo 6.1**'de verilen hazır matematik kütüphanesi dışında bir çok hazır fonksiyon Turbo Pascal derleyicisinin içerisinde yer almaktadır.

Hazır fonksiyonların bazıları tamsayı (integer) veya gerçel (real) tipte değişken veri tipine sahip olsalar da fonksiyonun alacağı geri dönüş değeri bazı durumlarda sadece tamsayı veya gerçel sayı olabilmektedir. Örneğin `ABS(x)`, `ROUND(x)`, `SQR(x)` veya `TRUNC(x)` fonksiyonları gerçel tipte (real) veri tipi alsalar bile fonksiyonun geri dönüş tipi tam sayı (integer) olma zorunluluğu vardır. yine aynı şekilde bazı fonksiyonlara tamsayı (integer) veri tipi değer alsalar fonksiyonun geri dönüş veri tipi gerçel (real) olma zorunluluğu vardır.

Programlama sırasında veya program tasarlanırken yazılım geliştiricileri veri tiplerine dikkat etmelidir. Özellikle fonksiyon veya alt program

(procedure) tarafından üretilen ve geri dönüşleri belirli olmayan veri ve geri dönüş tipleri için yaklaşık veya tahmini tipler göz önüne alınmalıdır.

Tablo 6.1 Turbo Pascalda Hazır Matematik Kütüphanesi

Fonksiyon	Açıklama	Veri Tipi	Sonuç
ABS(x)	x değişkenin mutlak değerini alır	real/integer	real/integer
COS(x)	x açısının kosinüs değerini radyan cinsinden verir	real/integer	real
EXP(x)	E=2.71828..., e^x değerini hesaplar	real/integer	real
LN(x)	x > 0.0 değeri için doğal logaritmasını verir	real/integer	real
ROUND(x)	x sayısını en yakın tam sayıya yuvarlar	real	integer
Sin(x)	x açısının sinüs değerini radyan cinsinden verir	real/integer	real
SQR(x)	x değerinin kare değerini geri gönderir	real/integer	real/integer
SQRT(x)	x > 0.0 değeri için x değerinin karekök değerini geri gönderir	real/integer	real
TRUNC(x)	x değerinin tam sayı kısmını alır	real	integer
FRAC(x)	x değerinin kesirli kısmını geri gönderir	real	real
INT(x)	x değerinin tam sayı kısmını verir	real/integer	integer
RANDOM(x)	tamsayı türü bir rasgele değer üretir	integer	integer
RANDOM	0 < x < 1 arasında rasgele sayı üretir	real	real

Temel matematik bilgisi olarak verilen bir bilgiye göre hazır fonksiyonları kullanarak bir program yazalım. İkinci dereceden bir bilinmeyenli fonksiyonun köklerini bulalım. Genel formu $ax^2 + bx + c = 0$ olan bu fonksiyonun köklerinden birisini bulmak için aşağıdaki formüllerden yararlanıyoruz.

$$\text{Köklerden birincisi için } k_1 = \left(\frac{-b - \sqrt{b^2 - 4ac}}{2a} \right) \quad (6.2)$$

$$\text{Köklerden diğeri olan ikincisi için } k_2 = \left(\frac{-b + \sqrt{b^2 - 4ac}}{2a} \right) \quad (6.3)$$

formüllerinden yararlanıyoruz.

Bu formülde kısaca yazacağımız diskriminant ise $b^2 - 4ac$ 'dir (6.4). Kökleri hesaplamak (6.2 ve 6.3) için aşağıdaki kod parçacıklarını program içerisine yerleştirerek bu hesaplamaları yapabiliriz.

$$\text{disc} := \text{SQR}(b) - 4 * a * c; \quad (6.4)$$

$$k1 := (-b + \text{SQRT}(\text{disc})) / (2 * a); \quad (6.2)$$

$$k2 := (-b - \text{SQRT}(\text{disc})) / (2 * a); \quad (6.3)$$

Görüldüğü gibi birkaç kod parçacığı ve hazır olan fonksiyonlar yardımı ile ikinci dereceden bir bilinmeyenli fonksiyonun köklerini bulan kod parçacıklarını yazdık. Burada öncelikle kısaca formüle edilecek olanlar üzerinde durulmalıdır. Daha sonra bu hazırlanan kısa yazımlara göre formülasyon genişletilir ve kodlar modüler olarak yazılır. Kısa yazım kullanımının yararlarından birisi herhangi bir değişken veya sabit üzerinde veya kod üzerinde yapılacak olan değişiklikler anında program içerisinde kullanılır.

Sayısal analiz (Numerical Recipes in Pascal, Cambridge University Press, 1986) kitabında yer alan Gauss quadrature yöntemi programlamada yararlanabileceğimiz önemli bir kaynaktır. Aşağıdaki programda (GAUSSQuad) 5x5 büyüklüğündeki değerleri daha önceden belirenmiş olan bir matrisin gauss yöntemi ile gerçek değerini (eigenvalue) bulmaya çalışılıyor.

```
PROGRAM GAUSSQuad(input,output);
  TYPE double = real; char12 = string[12];
  FUNCTION sngl(x:real):real; BEGIN sngl := x END;
  PROCEDURE glopen(VAR infile:text; filename:char12);
  BEGIN assign(infile,filename); reset(infile) END;
CONST
  x1=0.0;
  x2=5.0;
  nval=10;
VAR
  dx,ss,x : real;
  i : integer;
PROCEDURE qgaus(a,b: real; VAR ss: real);
VAR
  j: integer;
  xr,xm,dx: real;
  w,x: ARRAY[1..5] OF real;
BEGIN
  x[1] := 0.1488743389;
  x[2] := 0.4333953941;
  x[3] := 0.6794095682;
  x[4] := 0.8650633666;
  x[5] := 0.97390652;
  w[1] := 0.2955242247;
  w[2] := 0.2692667193;
  w[3] := 0.2190863625;
  w[4] := 0.1494513491;
  w[5] := 0.06667134;
  xm := 0.5*(b+a);
  xr := 0.5*(b-a);
  ss := 0;
  FOR j := 1 TO 5 DO BEGIN
    dx := xr*x[j];
    ss := ss+w[j]*(func(xm+dx)+func(xm-dx))
```

```
END;
  ss := xr*ss
END;
FUNCTION func(x: real): real;
BEGIN
  func := x*exp(-x)
END;
BEGIN
  dx := (x2-x1)/nval;
  writeln;
  writeln('0.0 to', 'qgaus':10, 'expected':13);
  writeln;
  FOR i := 1 to nval DO BEGIN
    x := x1+i*dx;
    qgaus(x1, x, ss);
    writeln(x:5:2, ss:12:6, (- (1.0+x) *exp(-x)
      + (1.0+x1) *exp(-x1)) :12:6)
  END
END.
```

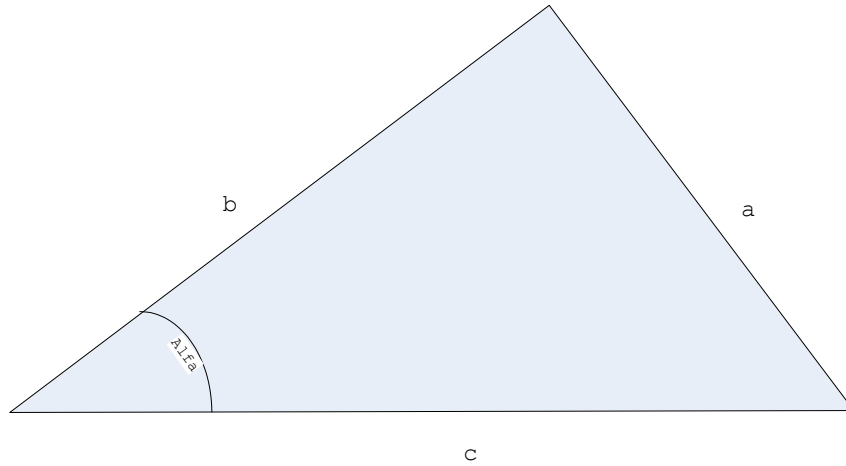
Sayısal analiz kitaplarında yer alan bir çok yöntemi Pascal bilgisayar programı ile programlama mümkündür. Özellikle yöntemleri belirli olan problemler üzerinde uygulama yapmak programcılığın yanında mühendislik becerilerini de geliştirecektir. Normal dağılım istatistikte kullanıldığı kadar mühendislikte de çok kullanılan bir dağılımdır. Genel adı Gauss dağılımı olarak adlandırılan dağılımın parametreleri ortalama ve standart sapmadır. Ortalama dağılımın ortalamasını gösteren, standart sapma ise dağılımın ölçülebilir yığılımını gösteren parametrelerdir. Hazır Pascal kütüphanelerinde bulunabilen bu dağılım fonksiyonu (Dr J.R. Stockton, Turbo Pascal Mathematics With Trigonometry kitabından alınmıştır) aşağıdaki gibi yazılır.

```
function RandomGaussianSD1 ;
var X : DatT ;
begin
repeat
X := (Random-0.5)*8.0 until Exp(-Sqr(X)*0.5)>Random ;
RandomGaussianSD1 := X ;
end;
procedure GaussTest ;
var j : byte ; s, t : DatT ;
const k = 80 ;
begin t := 0.0 ;
  for j := 1 to k do begin
    s := RandomGaussianSD1 ;
    t := t+Sqr(s) ;
    Write(s:9:3) ;
    if (j and 7)=0 then
      Writeln ;
  end ;
Writeln('Karekök: ', Sqrt(t/k):10:3, ' '^G) ;
Delay(500) ;
Writeln('Normal dağılım, mean 0, RMS 1 ?'^G) ;
end ;
```

Turbo Pascal bilgisayar programlama dilinde hazır fonksiyonları kullanarak bir çok problem kısa sürede çözülebilir. Trigonometrik fonksiyonlardan yararlanarak başka bir problemi çözmek için gerekli olan pascal program kodlarını yazalım. Bilindiği gibi bir üçgende iki kenar uzunluğu ve karşı açı biliniyor ise üçgenin diğer kenarının uzunluğunu bulmak için

$$a^2 = b^2 + c^2 - 2bc \cos(\text{alfa}) \quad (6.5)$$

eşitliğinden yararlanıyoruz. Üçgenin aşağıdaki **Şekil 6.2** de görülmektedir.



Şekil 6.2 Uzunluğu Hesaplanacak Üçgen

```
Program ucgen;  
USES CRT;  
Const  
pi = 3.1416;  
var  
a,b,c,alfa :real;  
begin  
writeln('Üçgenin c ve b uzunluklarını girin -> ');  
readln(a,b);  
writeln('Üçgenin alfa açısını girin -> ');  
readln(alfa);  
a:= SQRT(SQR(b)) + SQR(c) + (2 * b * c * COS( alfa *  
(pi / 180.0)));  
writeln('Üçgenin a uzunluğu ',a,' dir');  
end.
```

Bu program kodunda π sayısı 180.0 gerçel sayısına bölünmüştür. Dikkat edilecek olursa π sayısı gerçel bir sayıya (real) bölünmüştür. Eğer bu sayıyı tam sayı olarak almış olsaydık genel programlama hatası verecektir. Turbo Pascal bilgisayar programlama dilinde üstel fonksiyonlarda (u^v) kullanılabilir hazır hali bulunmamaktadır. Bunun yerine pseudo (yerine)

fonksiyon denilen bir fonksiyon ile bu işlemi yapabiliriz. Üstel fonksiyonlarda teoriden bilindiği gibi her iki tarafında doğal logaritmasını alacak olursan $\ln(u^v) = v \times \ln(u)$ ile denkliği sağlanır. Bir başka eşitlik şeklinde yazılır ise $z = e^{\ln(z)}$ (e'nin tanımlı değeri 2.71828...). Eğer biz z yerine u^v yazarsak, $u^v = e^{\ln(u^v)} = e^{(v \times \ln(u))}$ elde ederiz. Bu formülü Turbo Pascal bilgisayar programlama dili ile aşağıdaki gibi kodlarız.

$$u_ussu_v := \text{EXP}(v * \text{LN}(u)); \quad (6.6)$$

Kartezyen koordinat sisteminde iki nokta arasındaki uzaklığı veren formül

$$l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (6.7)$$

şeklindedir. Pascal program kodları ise aşağıdaki gibi yazılarak hesaplanır.

$$\text{uzaklık} := \text{SQRT}(\text{SQR}(x1-x2) + \text{SQR}(y1-y2)); \quad (6.8)$$

Yukarıda yazılan kodlardaki değişkenler gerçel tipte olmalıdır. Tam sayı tipinde belirlenen değişkenler kesirli sonuçlarda doğru uzaklığı vermeyecektir.

6.3. Programcı Tanımlı Fonksiyonlar

Programcı tanımlı fonksiyonlar ön tanımlı fonksiyonlar ile aynı amaca yönelik olarak yazılır. Hazır fonksiyonlardan farkı ise programcı kullanıcının amacına yönelik olarak yazılmasıdır. Kullanıcı bazı problemlerde hazır fonksiyonlardan yeterince yararlanamayabilir veya hazır fonksiyonlar kullanıcının ihtiyacını tam olarak karşılamayabilir. Bu gibi durumlarda programcı tanımlı olarak adlandırılan ve programcının kendisinin algoritmasını karşılaştığı probleme göre geliştirdiği fonksiyonlar yazılır.

Önceki bölümlerde gördüğümüz rasgele sayı üreticisi program kodlarına örnek olarak aşağıdaki örnek ile devam edelim.

```
Program rasgele;
USES CRT;
var
i,n: integer;
r: real;
Begin
writeln(' Rasgele sayı aralığını girin ->');
readln(r);
writeln(' Rasgele sayılardan kaç adet istediğinizi
girin ->');
readln(n);
for i:= 1 to n do
randomize;
writeln(' Rasgele sayılar' ,Random(r):3);
end;
End.
```

Programda (`rasgele`) rasgele sayı üretimi bilgisayarın saatine bağlı olduğundan `random(r)`; fonksiyonu her seferinde aynı sayıyı üretir. Bu durumu engellemek için döngünün başlangıcına `randomize`; fonksiyonu ekleyerek kaldırabiliriz.

Programcının geliştireceği fonksiyonlara örnek olarak girilen sayının karesini alan bir fonksiyon yazarak devam edelim.

```
Program kare;
USES CRT;
var
sayi, sonuc: integer;
function kare_al(sayi);
begin
    kare_al:= (sayi * sayi);
end;
```

```
begin
writeln('Karesi alınacak olan sayıyı girin -> ');
readln(sayi);
writeln('Karesi alınan sayı ',kare_al(sayi),' dir');
end.
```

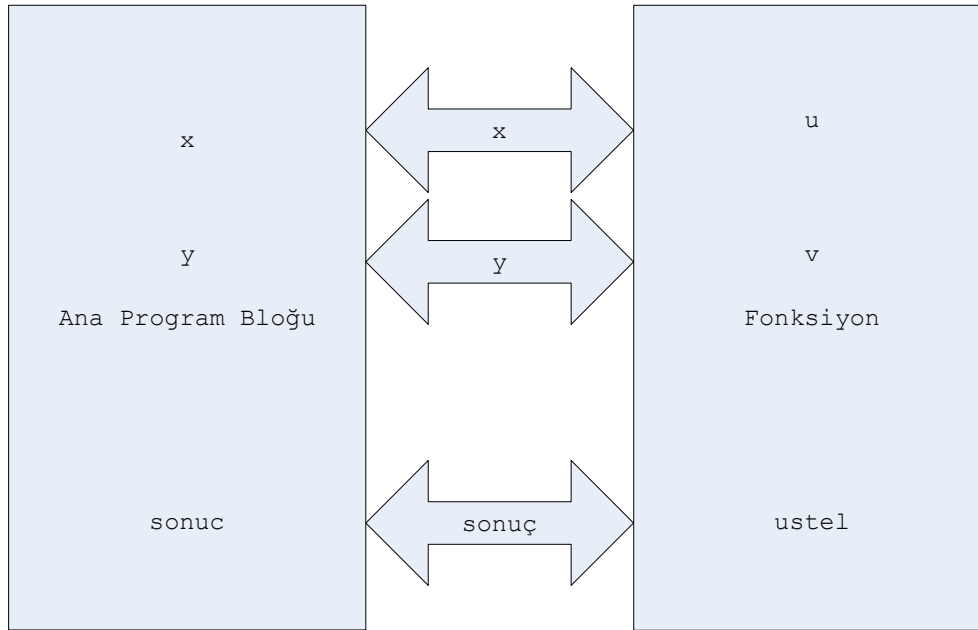
Karesi alınacak olan sayıyı klavyeden girerek hesaplayana ve sonuç olarak bunu geri gönderen basit kare alma ($SQR(x)$) fonksiyonunu yazdık. Bu tip fonksiyonlar daha önceden tanımlı olduğunu biliyoruz. Fonksiyonların yazılması sırasında fonksiyon ile göndereceğimiz değer görüldüğü gibi fonksiyon adı ile gönderilir. Bu örnekte fonksiyon adı `kare_al (sayi);` döndüreceği değer `kare_al` değişkenine aktarılan `kare_al:= (sayi * sayi);` değerdir.

Önceki bölümde üstel fonksiyon ile ilgili program kodlarını yazmıştık. Bu fonksiyona ait kodları biraz daha geliştirerek aşağıdaki (`function ustel(u,v:real):real`) gibi yazabiliriz. Daha sonra bu kodları aşağıdaki gibi yazarız.

```
function ustel(u,v: real):real;
Begin
if u = 0.0 then
    ustel := 0.0;
else if u > 0.0 then
    ustel:= EXP(v * LN(U));
else
    ustel := 1.0 / EXP ( v * LN(-u));
end;
```

Yazdığımız `ustel` adlı fonksiyon tipi gerçel (`real`) tiptedir. Değişken olarak kullandığımız `u` ve `v` adlı iki değişkenin tipi gerçeldir (`real`). Bu tanımlamam işlemi fonksiyonun parametresinin gönderildiği alanda yapılmış (`function ustel(u,v:real):real`) ve daha sonra fonksiyonun

kendisinin alacağı veri tipi tanımlanmıştır. Şart yapısı kullanarak matematiksel teoriye bağlı kalınarak tanımlama yapılıyor ve görüldüğü gibi değişkenin karşılaştırıldığı değerde gerçel tiptedir. Değişkenin karşılaştırıldığı tipe alt satırlarda da devam edilmekte fonksiyonun geri dönüşünde kullanılacak olan değerlerinde karşılaştırılmasında gerçel tipte (*real*) değerler kullanılmaktadır. Burada `else if şart then` yapısına dikkat edilmesi gerekiyor. Şartın birinci durumda sağlanamaması halinde bu yapı çalıştırılır ve ikinci tür şart yapısı gibi işleme devam edilir. Sonuç fonksiyonun kullanıldığı ana program içerisinde alınır. **Şekil 6.3.**

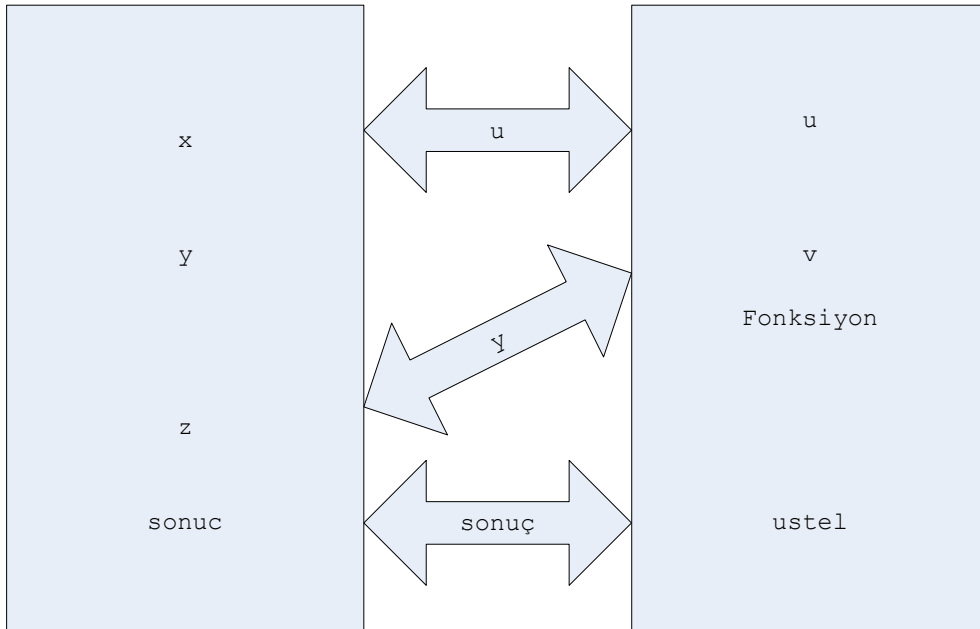


Şekil 6.3 Fonksiyonların Ana Hafızadan Çağırılması

Fonksiyonlar ana program içerisinde gönderilen değişkenlerin değerlerini fonksiyonun ayırdığı hafıza alanına fonksiyon parametreleri ile aktarır. Fonksiyon değeri ana program kullanılana kadar `ustel` fonksiyonun değeri `function ustel (u,v:real):real;` olarak saklı kalır. Fonksiyon ana programdan çağırılıp kullanılmaya başlandığında `sonuc` gibi bir değişkene aktarılabilir. Ana programda parametreler fonksiyon değişkenlerine

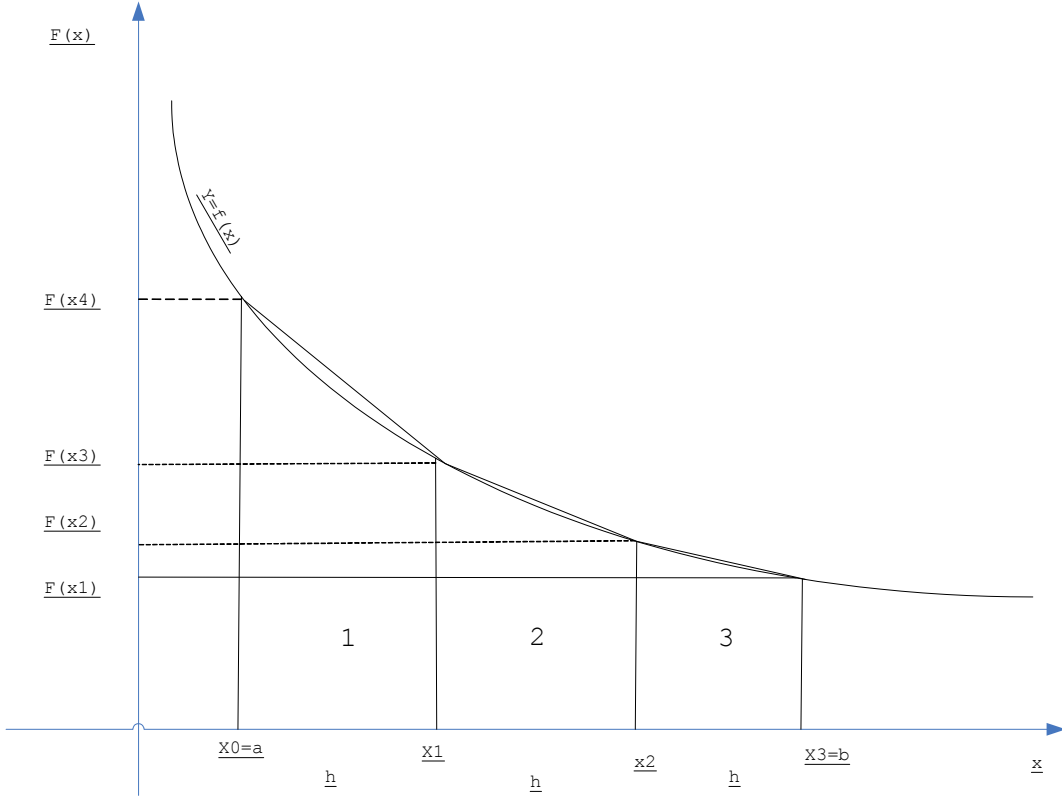
değerlerini aktarsalar bile ana programdan çağrılmadığı sürece fonksiyonun aldığı değer hafıza alanında ayrıldığı yerde saklı kalır.

Fonksiyonlarda parametre olarak gönderilen değerlerin ana program içerisinde **Şekil 6.4**'de görüldüğü gibi başka değerler olarak değişmesi durumunda fonksiyonun geri döndüreceği değer ana program içerisinde aldığı değerleri işleyerek geri göndermesi ile sonuçlanır. Fonksiyon parametreleri ana programdaki farklı bir değişkenden alınan değerler olabilir. Gerçekleşen bu olayın oluşması fonksiyonun etkin kullanımının bir göstergesidir.



Şekil 6.4 Fonksiyonların Değişen Değerlerinin Ana Hafızadan Çağırılması

Belirli integral matematikte sayısal yöntemlerde kullanılan bir eğrinin altında kalan alanı bulmaya yarayan yöntemlerden biridir. Bu yöntemler içerisinde yamuk yöntemi eğrinin altında kalan alanı eşit aralıklı parçalara bölerek her bir parçanın alanını ayrı hesaplayarak hesapladığı alanı daha sonra toplayan algoritma ile gerçekleştiren yöntemdir.



Şekil 6.5 Yamuk Yöntemi İle İntegral

Sürekli bir fonksiyonu inceleyelim, eşit üç aralığa bölünen fonksiyon eğrisinin a ve b aralıklarındaki değerlerini hesaplama isteyelim. Matematiksel yöntemler kullanarak eğrinin hesaplaması aşağıdaki adımlar izlenerek yapılır.

$$h = \frac{b-a}{3} \quad (6.9)$$

ve a ve b noktaları arasındaki yaklaşık integral değeri ise

$$\int_a^b f(x).dx \cong Alan_1 + Alan_2 + Alan_3 \quad (6.10)$$

formülü ile hesaplanır. Burada $Alan_{1,2,3}$ h aralıkları ile bölünen alandır.

$$Alan_1 = \frac{f(x_0) + f(x_1)}{2} h = \frac{f(x_0) + f(x_0 + h)}{2} h = \frac{f(a) + f(a + h)}{2} h \quad (6.11)$$

$$Alan_2 = \frac{f(x_1) + f(x_2)}{2} h = \frac{f(a + h) + f(a + 2.h)}{2} h \quad (6.12)$$

$$Alan_3 = \frac{f(x_2) + f(x_3)}{2} h = \frac{f(a + 2.h) + f(a + 3.h)}{2} h \quad (6.13)$$

Yukarıdaki formülde h aralık sayısı arttırıldıkça a ve b aralığındaki integral değeri gerçeğe yakın değer alacaktır. Aralığı n adet parçaya bölmek istediğimizde genel formülü aşağıdaki gibi olur.

$$\int_a^b f(x).dx = \sum_{k=1}^n \frac{f(a + (k-1).h) + f(a + k.h)}{2} .h \quad (6.14)$$

Bu matematiksel işlemleri Turbo Pascal bilgisayar programlama dili ile programlayalım.

```

Program Integral_yamuk;
USES CRT;
function y(var x:real):real;
Begin
    y:= 2 * x * x +2;{hesaplanacak fonksiyon}
End;
var
a,b,n,k: integer;
z,w,h,t:real;
dosya_: text;
Begin
ASSIGN(dosya,'C:\temp\sonuc.dat')
RESET(dosya);
ClrScr;

```

```

writeln('Hesaplanacak a ve b değer aralığını sırası
ile girin ->');
readln(a,b);
writeln('Aralığı kaç parçaya ayırmak istediğinizi
girin ->');
readln(n);
writeln(dosya, 'Parça numarası   Integral değeri ');
writeln(dosya, '-----   ----- ');
h:= (b-a)/n;
t:= 0;
  for k:=1 to n do
    Begin
      z:= a + (k-1)*h;
      w:= a+k*h;
      t:=t + (y(z) + y(w)) * h/2;
      writeln(dosya,k, '   ' , t:6:3);
    End;
writeln(dosya, '   Hesaplanan   fonksiyonun   integral
aralığı ', n , 'parçaya bölünerek aldığı sayısal
değer ' , t:6:3, '   olarak bulunmuştur ');
CLOSE(dosya);
readln;
End.

```

Belirli integral hesaplayan bu Pascal programında hesaplayacağımız fonksiyonu `function y(var x:real):real;` olarak ve fonksiyonun kendisini altına tanımlıyoruz. Ana program başlangıcında hesaplanacak aralık ve aralığın kaç parçaya bölüneceği klavyeden girilmesi isteniyor. Klavyeden girilen bu değerlerden sonra aralığın değeri h belirleniyor. Formülden bilindiği gibi yamuk halinde hesaplanan fonksiyon z ve w değerleri fonksiyona gönderiliyor ve alınan değerler toplanarak dosyaya (`c:\temp\sonuc.dat`) yazdırılıyor. Elde ettiğimiz bu sonuçları

istediğimizde (c:\temp\sonuc.dat) dosyadan okuyabilir veya sonucun grafiğini buradan aldığımız değerleri grafik olarak çizdirebiliriz.

Cambridge Üniversitesinde (Numerical Recipes in Pascal, Cambridge University Press, 1986) kitabından alınan Chebishev eşitsizliğini hesaplayan program aşağıda görülmektedir. bu programda `function chebev()` fonksiyonu problemi formüle eden kısımdır. Asıl hesaplanacak olan fonksiyon ise `function func ();` olarak tanımlanan yerdedir.

```
PROGRAM CHEBISHEV(input,output);
LABEL 10,99;
CONST
    nval=40;
    pio2=1.5707963;
TYPE
    glcarray=ARRAY [1..nval] OF real;
    double = real; char12 = string[12];
    FUNCTION sngl(x:real):real; BEGIN sngl := x END;
    PROCEDURE glopen(VAR infile:text; filename:char12);
        BEGIN assign(infile,filename); reset(infile) END;

PROCEDURE chebft(a,b: real; VAR c: glcarray; n: integer);
CONST
    pi=3.141592653589793;
VAR
    k,j: integer;
    y,fac,bpa,bma: real;
    sum: double;
    f: glcarray;
BEGIN
    bma := 0.5*(b-a);
    bpa := 0.5*(b+a);
    FOR k := 1 TO n DO BEGIN
        y := cos(pi*(k-0.5)/n);
```

```
        f[k] := func(y*bma+bpa)
END;
fac := 2.0/n;
FOR j := 1 TO n DO BEGIN
    sum := 0.0;
    FOR k := 1 TO n DO BEGIN
        sum := sum+f[k]*cos((pi*(j-1))*((k-0.5)/n))
    END;
    c[j] := sngl(fac*sum)
END
END;
VAR
    a,b,x : real;
    i,mval : integer;
    c : glcarray;

FUNCTION chebev(a,b: real; c: glcarray; m: integer; x:real):
real;

VAR
    d,dd,sv,y,y2: real;
    j: integer;
BEGIN
    IF ((x-a)*(x-b)) > 0.0) THEN BEGIN
        writeln('pause in CHEBEV - x not in range.');
```

```
        readln
    END;
    d := 0.0;
    dd := 0.0;
    y := (2.0*x-a-b)/(b-a);
    y2 := 2.0*y;
    FOR j := m DOWNTO 2 DO BEGIN
        sv := d;
        d := y2*d-dd+c[j];
        dd := sv
```

```
    END;
    chebev := y*d-dd+0.5*c[1]
END;
FUNCTION func(x: real): real;
BEGIN
    func := sqr(x)*(sqr(x)-2.0)*sin(x)
END;

BEGIN
    a := -pio2;
    b := pio2;
    chebft(a,b,c,nval);
10:  writeln;
    writeln('How many terms in Chebyshev evaluation?');
    write('Enter n between 6 and ',nval:2,
        '. (n := 0 to end). ');
    readln(mval);
    IF ((mval <= 0) OR (mval > nval)) THEN GOTO 99;
    writeln;
    writeln('x':9,'actual':14,'chebyshev fit':16);
    FOR i := -8 to 8 DO BEGIN
        x := i*pio2/10.0;
        writeln(x:12:6,func(x):12:6,chebev(a,b,c,mval,x):12:6)
    END;
    GOTO 10;
99: END.
```