Fortran Programlama
Diline Giriş

Ercan ERSOY **Erdem ERSOY**

Fortran Programlama Diline Giriş

Birinci Sürüm

Yayın Tarihi: Ekim 2018

Ercan ERSOY
E-Posta: ercanersoy@ercanersoy.net
Web Sitesi: http://ercanersoy.net

Erdem ERSOY
E-Posta: erdemersoy@erdemersoy.net
Web Sitesi: http://erdemersoy.net

ISBN: 978-605-245-343-8

Ön Söz

Fortran programlama dili, elli yıldan fazla sürede kullanılarak günümüze kadar ulaşmıştır. Bu dil, ülkemizde uzun bir süre eğitim ve hesaplama alanlarında kullanılmıştır. Fortran programlama dilinin son yıllarda kullanımı azalmasına karşın Fortran programlama dili, eğitimde, akademide ve bazı hesaplama uygulamalarını geliştirmede kullanılmaktadır.

Türkçe Fortran kaynaklarının çoğu eski sürümleri anlatmakta ve bu konuda pek kitap yayınlanmamaktadır. Bundan dolayı Fortran'ın yeni sürümlerine göre yeni bir kitap yazılmasının gerekli olduğu görülmüştür. Bu yüzden bu kitapta, Fortran programlama dilinin, eski sürümleriyle birlikte yeni sürümlerine de yer verilmektedir.

Bu kitapta, Fortran programlama dilinin anlatılmasında Fortran'la ilgili akademik kaynakların da olduğu birçok kaynak araştırılmıştır. Bu kitap, eğitim amaçlı, akademik amaçlı veya başvuru kaynağı olarak kullanılabilir.

Bu kitapta on konu ve on beş ek bulunmaktadır. Konularda basit örnekler de yer almaktadır. Konuların sonunda özet verilmiştir ve örnek sorular da bulunmaktadır. Bu kitapta, kitapta geçen bazı Türkçe kavramların İngilizce karşılıkları verilmiştir. Ayrıca kitabın sonunda kitabın yazımında yararlanılan kaynaklar verilmiştir.

İyi çalışmalar.

Kısaltmalar

ASCII American Standard Code for Information Interchange (Bilgi Değişimi İçin Amerikan Standart Kodlama Sistemi)

CPU Central Processing Unit (Merkezi İşlemci Birimi)

Fortran Formula Translation (Formül çevirisi)

IBM International Business Machines (Uluslararası İş Makineleri)

RAM Random Access Memory (Rastgele Erişimli Bellek)

İçindekiler

1. Bölüm: Fortran Diline Giriş

Fortran, sayısal ve bilimsel hesaplamalar için tasarlanmış yüksek seviyeli bir programlama dilidir. Fortran, yarım yüzyıldan fazla bir zaman diliminde yoğun bilimsel hesaplamalı alanlarda kullanılmaktadır. Fortran, dünyanın en hızlı süper bilgisayarları kıyaslama ve onları derecelendirme gibi yüksek performans gerektiren hesaplamalar için de kullanılır.

1.1. Fortran’ın Tarihçesi

Fortran, ilk olarak 1954 yılında John Warner Backus ile ekibi tarafından IBM bünyesinde geliştirilmiştir. İlk Fortran kılavuzu 1956 yılında ortaya çıkmış, 1957 yılında ilk Fortran derleyicisi dağıtımına başlanmıştır. Bu dilin ortaya çıkmasından önce yüksek seviyeli derlenebilir diller yoktu.

Aşağıda, Fortran'ın günümüze kadar yayınlanmış olan sürümleri kronolojik sırayla listelenmiştir:

- **FORTRAN I:** 1957 yılında derleyici olarak piyasaya sürülmüştür. Bir süre sonra birçok firma çeşitli Fortran derleyicileri geliştirmiştir.
- **FORTRAN II:** FORTRAN I’deki bazı sorunları çözmek için çıkarılmıştır.
- **FORTRAN III:** FORTRAN II’ye ek olarak bazı özellikler içeriyordu.
- **FORTRAN IV:** Bazı geliştirmelerle birlikte yaygın olarak kullanılan bir standart olmuştur.
- **FORTRAN 66:** Bazı geliştirmelerle birlikte taşınabilir olarak kullanılan ilk Fortran standardı olmuştur.
- **FORTRAN 77:** Birçok kolaylaştırıcı yeniliklerle gelmiş bir standarttır.
- **Fortran 90:** serbest kod biçimi, kayıtlar, yeni denetim birimleri gibi özelliklerle gelmiştir. Ayrıca FORTRAN ismi Fortran 90 ile birlikte Fortran olarak anılmaktadır.
- **Fortran 95:** Fortran 90’a ek olarak bazı özellikler içeriyordu.

- **Fortran 2003:** Nesneye yönelik programlama, C programlarıyla birlikte çalışma gibi özelliklerle gelmiştir.
- **Fortran 2008:** Fortran 2003'e ek olarak bazı geliştirmelerle birlikte gelmiştir. Bunlardan en önemlisi Coarray Fortran ile paralel programlamadır.
- **Fortran 2018:** Gelecek Fortran sürümüdür.

Günümüzde genel olarak FORTAN 77 ve onun sonrasındaki sürümler kullanılmaktadır. Fortran 90 ve sonrasındaki sürümlere çağdaş Fortran sürümleri de denmektedir.

Fortran, C ve BASIC gibi dillerinin oluşmasında da etkili olmuştur.

Bu kitabın yazılmasında Fortran 95, Fortran 2003 ve Fortran 2008 sürümlerini temel alan kaynaklardan yararlanılmıştır.

1.2. Fortran'ın Genel Özellikleri

- Fortran dili, İngilizce sözcüklerden oluşan bir programlama dilidir.
- Fortran, bir zorunlu programlama dilidir.
- Fortran; sayısal analiz ve bilimsel hesaplama, yapısal programlama, dizisel programlama, modüler programlama, genel programlama, süper bilgisayarlar için yüksek performanslı programlama, nesne tabanlı programlama ve eş zamanlı programlama için uygun bir dildir.
- Fortran, programlamaya giriş için uygun bir dildir.
- Fortran, bilimsel hesaplamalarda C, C++ gibi dillerden daha güçlü ve daha hızlıdır.
- Fortran, taşınabilirliği yüksek bir dildir, birçok platform için derleyicileri vardır.
- Fortran'ın standart olarak grafik yetenekleri yoktur, Fortran diliyle çoğunlukla metin tabanlı programlar yazılır.

Fortran, eskiden yaygın olarak kullanılsa da günümüzde kullanımı azalmıştır. Fortran'ın yakın zaman öncesinde yeni sürümlerinin ortaya çıkması Fortan dilinin günümüzde de kullanıldığını göstermektedir.

Fortran’da sabit biçimlendirmeli yazım ve serbest biçimlendirmeli yazım olmak üzere iki tür yazım biçimi vardır. Sabit biçimlendirmeli yazımda Fortran kodları belirli biçimlendirme kurallarına göre yazılmalıdır. Sabit biçimlendirmede büyük harf kullanımı zorunluluğu, açıklama bölümlerinin yalnızca satır başından başlaması, belirli sayılarda ve belirli yerlerde boşluk kullanımı gibi kaynak kodu biçimlendirmesiyle ilgili serbest biçimlendirmede olmayan katı kurallar vardır. Serbest biçimlendirmede sözceler hâricinde büyük küçük harf ayrımı yoktur. Fortran 90 öncesindeki sürümler yalnızca sabit biçimlendirmeli yazıma izin vermekte, Fortran 90 ve sonrasındaki sürümler serbest biçimlendirmeye de izin vermektedir. Sabit biçimlendirmeli yazım, güç olduğu, artık gerekli olmadığı ve pek kullanılmadığı için bu kitapta yer almamıştır.

1.3. Fortran'la Yazılmış Bir Program Örneği

Aşağıda yarıçap uzunluğu kullanıcı tarafından girilen bir çemberin çevresiyle alanını hesaplayan bir Fortran örneğinin kaynak kodu verilmiştir:

```
! Bu örnek, yarıçap uzunluğu kullanıcı tarafından girilen bir
çemberin çevresiyle alanını hesaplar.

program cember_cevre_alan

   implicit none
   real :: yaricap
   real, parameter :: pi = 3.141593

   write (*,"(1x, a)", advance="no") "Çemberin yarıçapını giriniz: "
   read (*,*) yaricap
```

```
   write (*,*) "Çemberin çevresi: ", yaricap * 2 * pi
   write (*,*) "Çemberin alanı: ", yaricap ** 2 * pi

end program cember_cevre_alan
```

1.4. Fortran ile Programlama İçin Gereken Yazılımlar

Fortran kodlarını derlemek için **GNU Fortran** (diğer adıyla **Gfortran**), **G95**, **Open64**, **Open Watcom** gibi özgür derleyiciler kullanılabilir. Bunların yanında **Absoft Pro Fortran**, **Intel Fortran Compiler**, **NAG Fortran Compiler** gibi özgür olmayan derleyiciler de vardır. Üzerinde Fortran kodlarıyla çalışabilecek tümleşik geliştirme ortamları için **Geany**, **Code::Blocks**, **Photran** gibi özgür tümleşik geliştirme ortamlarının yanında **Microsoft Visual Studio**, **Oracle Developer Studio**, gibi özgür olmayan tümleşik geliştirme ortamları örnek olarak verilebilir. Bu kitapta, Fortran konuları GNU Fortran derleyicisine göre anlatılmıştır, program örnekleri GNU Fortran derleyicisiyle derlenmiştir ve örnek programların ekran çıktıları GNU/Linux ortamında alınmıştır.

Fortran ile programlama yapmak için, bir tümleşik geliştirme ortamının kullanılması zorunlu değildir. Fortran kodlarını yazmak için **Gedit**, **Kate**, **KWrite**, **Notepad++**, **xed**, **GNU Nano**, **Vi**, **GNU Emacs**, **Microsoft Notepad** gibi basit bir metin düzenleyicisi bile kullanabilir, Fortran kodlarını derlemek için ise tercih ettiğiniz bir derleyiciyi işletim sisteminizin komut satırından kullanabilirsiniz.

Tam kullanıcı özgürlüğü sağlamadığı için her ne kadar çevrim içi yazılımlar önerilmese de bilgisayarınızda Fortran için bir yazılım barındırmak istemezseniz **TutorialsPoint** sitesinde GNU Fortran derleyicisi bulunan çevrim içi bir tümleşik geliştirme ortamı bulunmaktadır. Buna **https://www.tutorialspoint.com/compile_fortran_online.php** bağlantısıyla erişebilirsiniz.

1.5. GNU Fortran'ın Komut Satırından Kullanımı

Eğer tümleşik geliştirme ortamı kullanacaksanız komut satırı kullanımına gereksinim duymayacaksınız. Tümleşik geliştirme ortamı kullanmazsanız muhtemelen komut satırını kullanmak zorunda kalacaksınız. GNU Fortran'ı komut satırından kullanmak için GNU Fortran'ı sisteminize kurduktan sonra bir Fortran kaynak kodunu derlemek için komut satırından,

```
gfortran <Fortran Kod Dosyası Adı> -o <Çalıştırılabilir Dosya Adı>
```

komutunu girmelisiniz. Burada, <Fortran Kod Dosyası Adı>, Fortran kaynak kodu dosyasının adı, <Çalıştırılabilir Dosya Adı> ise bu kaynak kodundan üretilecek çalıştırılabilir dosyanın adını işaret eder. Ayrıntılı bilgi için GNU Fortran derleyicisinin başvuru kaynaklarına bakabilirsiniz.

Fortran kaynak kodu dosya adlarında, .f90, .f95, .f03, .f08, .F90, .F95, .F03 ve .F08 uzantıları serbest biçimli Fortran kaynak kodu dosyalarını, .f, .for, .fpp, .ftn, .F, .FOR, .FPP, .FTN uzantıları ise sabit biçimli Fortran kaynak kodu dosyalarını belirtir.

Örnek olarak, yukarıdaki programı derlemek için ilk önce bu program bir dosyaya kaydedilmelidir. Programı kaydedeceğimiz dosya, serbest biçimli Fortran kaynak kodu dosyası olduğu için dosya adını belirlerken uzantıyı ona göre seçmek gerekir. Dosya adına örnek olarak cember.f95 verelim. Şimdi yukarıdaki kodu derlemek için komut satırına GNU/Linux için,

```
gfortran cember.f95 -o cember
```

ya da Microsoft Windows için,

```
gfortran cember.f95 -o cember.exe
```

komutunu giriniz. Program, dosyadan derleyici tarafından derlenecektir, eğer derleyici, programda hata tespit ettiğinde derlemeyi durduracaktır. Program derlendikten sonra GNU/Linux için,

```
./cember
```

ya da Microsoft Windows için,

```
cember.exe
```

komutu girildiğinde program çalışacaktır. Programı çalıştırıp yarıçap değerine örnek olarak 5 girildiğinde oluşan ekran çıktısı aşağıdaki gibi olacaktır:

```
 Çemberin yarıçapını giriniz: 5
 Çemberin çevresi:    31.4159298
 Çemberin alanı:    78.5398254
```

Microsoft Windows işletim sistemlerinde Türkçe karakter sorunu olabilir, çünkü Microsoft Windows'ta grafik programlarının kullandığı karakter kümesi ve komut satırı programlarının kullandığı karakter kümesi birbirlerinden farklıdır.

1.6. Özet

Bu bölümde, Fortran dilini öğrenmeye başlamadan önce okuyucuya gerekli ön bilgiler verilmiştir. Fortran, büyük çoğunlukla bilimsel hesaplamalarda kullanılan, taşınabilir, yüksek seviyeli bir dildir. Fortran, eski bir dil olmasına karşın günümüzde de kullanılmaktadır ve bu dilin yeni standartları geliştirilmektedir. Fortran’ın sabit biçim ile serbest biçim olarak iki tür yazım biçimi vardır. Fortran kodlarını, çeşitli düzenleyicilerle yazabilir, çeşitli derleyicilerle derleyebilir, hattâ bu işlemler için tümleşik geliştirme ortamları bile kullanabilirsiniz. Bu kitabın kod örnekleri GNU Fortran derleyicisiyle GNU/Linux sisteminde derlenmiş, derlenen programların ekran çıktıları GNU/Linux sisteminden alınmıştır.

1.7. Sorular

1.1) Fortran nedir?

1.2) Fortran programlama dili nerelerde kullanılır?

1.3) Fortran'ın tarihçesini sürümlerini de belirterek özetleyiniz.

1.4) Fortran programlama dilinin serbest kod biçimine izin verilen ilk standardı nedir?

1.5) Fortran programlama dilinin genel özelliklerinden üçünü yazınız.

1.6) Fortran kodlarının kaç tür yazım biçimi vardır?

1.7) Fortran derleyicilerinden üçünü yazınız.

1.8) Fortran için kullanılan tümleşik geliştirme ortamlarından üçünü yazınız.

1.9) GNU Fortran derleyicisinin bir Fortran kaynak kodu dosyasını derlemesi için GNU/Linux komut satırına girilmesi gereken komut nedir?

2. Bölüm: Fortran Dilinin Temelleri

Bu bölümde Fortran dilinin temel konuları anlatılacaktır.

2.1. Açıklama Bölümleri

Fortran'da açıklama bölümleri, ünlem işareti ile başlayan ve yeni satır ile biten bölümlerdir. Bu bölümleri derleyici önemsemeden geçer. Bu bölümlerle kaynak kodlarınızın çeşitli yerlerine açıklamalar ekleyebilirsiniz.

```
!Açıklama bölümü

Program kodu parçası !Açıklama bölümü
```

Örnek program aşağıda verilmiştir:

```
! Bu örnek, kullanıcı tarafından girilen iki tamsayıyı toplayarak
! toplamı çıktılar.

program toplama

   ! Değişken bildirimleri
   implicit none
   integer :: a, b ! Girilecek sayılar

   ! Yürütülebilir deyimlerin bulunduğu kısım
   write(*,*) "İki tamsayı giriniz:"
   read(*,*) a, b
   write(*,*) "Toplam: ", a + b ! Toplamı görüntüler.

end program toplama
```

Fortran 90 öncesindeki sürümlerde açıklama bölümleri ! yerine C harfiyle belirtilir. Bu açıklama bölümleri satırın en başından başlamalıdır.

2.2. Tanımlayıcılar

Fortran'da tanımlayıcılar; programları, alt programları, değişkenleri, sabitleri, türetilen yapıları ve kullanıcı tanımlı başka ögeleri tanımlamaya yarayan kullanıcı tanımlı isimlerdir. Tanımlayıcılar oluştururken anlamlı isimler kullanmak programlamayı kolaylaştırır.

Tanımlayıcılar; Fortran'da bulunan, anahtar sözcüklerin ve dahili alt programların isimleri dışında olanlardan seçilmelidir (Anahtar sözcükler ve dahili alt programlar kitabın ileriki bölümlerinde belirtilecektir). Fortran'da tanımlayıcı yalnız tek bir şeyi belirtir, başka bir öge için aynı tanımlayıcı kullanılamaz.

Tanımlayıcılarda İngiliz abecesinde bulunan harfler (ASCII tablosunda bulunan harfler), rakamlar ya da alt çizgi işareti olabilir. Tanımlayıcılar harf ile başlamak zorundadır. Tanımlayıcılarda büyük küçük harf ayrımı yoktur. Tanımlayıcılar, en çok, Fortran 90 ve Fortran 95'te 31, Fortran 2003 ve Fortran 2008'de 63 karakterden oluşabilir.

Tanımlayıcılara örnekler aşağıda verilmiştir:

```
x

y

abc123

alanHesapla

secim_girisi
```

```
belirtilen_adresteki_degiskenin_bit_degeri
```

2.3. Basit Veri Türleri

Veri türleri, sabiterin, bellekte ne kadar yer kaplayacağını ve programda nasıl kullanılacağını belirtir. Veri türleri, var olmak için başka bir veri türüne, gereksinim duymuyorsa **basit veri türü**, gereksinim duyuyorsa **türetilmiş veri türü** denir.

Fortran'da tanımlanmış basit veri türleri aşağıda verilmiştir:

- Tamsayı
- Tek Duyarlıklı Kayan Noktalı Sayı
- Çift Duyarlıklı Kayan Noktalı Sayı
- Karmaşık Sayı
- Mantıksal
- Karakter

2.3.1. Tamsayı Veri Türü

Bu veri türü tamsayı belirtir. Bu veri türünü tanımlamak için **integer** deyimi kullanılır. Bu veri türünde verileri belirtmek için rakamlar kullanılır.

Tamsayılara örnekler aşağıda verilmiştir:

```
5
```

```
12
```

```
8192
```

Bir tamsayının başına bir **eksi işareti** ile tamsayının sıfırdan küçük olduğu belirtilebilir.

Sıfırdan küçük tamsayılara örnekler aşağıda verilmiştir:

```
-1
```

```
-74
```

```
-965
```

Aynı zamanda sıfırdan büyük tamsayı belirtiminde isteğe bağlı olarak **artı işareti** kullanılabilir.

Artı işaretiyle belirtilmiş sıfırdan büyük tamsayılara örnekler aşağıda verilmiştir:

```
+2
```

```
+515
```

```
+77
```

2.3.2. Tek Duyarlıklı Gerçek Sayı Veri Türü

Bu veri türü tek duyarlıklı gerçek sayı belirtir. Bu veri türünde tamsayıların yanında aynı zamanda ondalıklı sayıları belirtebilirsiniz. Bu veri türünü tanımlamak için **real** deyimi kullanılır. Bu veri türünde sabiti belirtmek için rakamlarla nokta işareti kullanılır. Nokta işareti sayının ondalık ayıracıdır.

Ondalık sayılara örnekler aşağıda verilmiştir:

```
0.5
```

```
3.0
```

```
95.5
```

Bir tek duyarlıklı gerçek sayının başına bir eksi işaretiyle sayının sıfırdan küçük olduğu belirtilebilir.

Sıfırdan küçük ondalık sayılara örnekler aşağıda verilmiştir:

```
-1.5
```

```
-56.0
```

```
-273.15
```

Aynı zamanda sıfırdan büyük sayı belirtiminde isteğe bağlı olarak artı işareti kullanılabilir.

Artı işaretiyle belirtilmiş sıfırdan büyük ondalık sayılara örnekler aşağıda verilmiştir:

```
+5.5
```

```
+47.0
```

```
+5943.54
```

Tek duyarlıklı gerçek sayı veri türünde noktadan sonra bir sayı yazılmazsa sayının ondalık bölümü sıfır olur.

Ondalığı belirtilmemiş ondalık sayılara örnekler aşağıda verilmiştir:

```
4.
```

```
-170.
```

```
+1000.
```

Bu veri türünde sayıyı bilimsel bir biçimde gösterme olanağı vardır. Böyle bir gösterme, bir sayının 10'un herhangi bir kuvvetiyle çarpılacağını kısa yoldan belirtmeye yarar. Bu göstermede, **e** harfi kullanılır, bu harfin soluna yazılan sayı çarpılan sayıdır, sağına yazılan sayı 10'un kuvvetini belirtir.

Bilimsel gösterimde olan sayılara örnekler aşağıda verilmiştir:

```
2.4e10
```

```
-1.6e8
```

```
+3.7e-5
```

2.3.3. Çift Duyarlıklı Gerçek Sayı Veri Türü

Bu veri türü de gerçek sayı belirtir. Bu veri türünde bu gerçek sayı çift duyarlıklı olmayan gerçek sayı veri türünden olan sayılardan daha büyük ve daha küçük olabilir. Bu veri türünü tanımlamak için **double precision** deyimi kullanılır. Bu tür gerçek sayıların gösterimi, tek duyarlıklı gerçek sayı sabitlerin bilimsel gösterilmesi gibidir ancak bu gösterimde e harfi yerine **d** harfi kullanılır.

Bilimsel gösterimde olan çift duyarlıklı ondalıklı sayılara örnekler aşağıda verilmiştir:

```
2.5d9
```

```
1.2345d0
```

Bu veri türü eskisi kadar kullanılmamaktadır. Bu veri türünü kullanmak yerine gerçek sayı türünü **kind** belirteciyle kullanabilirsiniz. Bu belirtecin kullanılması ilerleyen sayfalarda işlenecektir.

2.3.4. Karmaşık Sayı Veri Türü

Bu veri türü karmaşık sayı belirtir. Bu veri türünü tanımlamak için **complex** deyimi kullanılır. Bu veri türünde, karmaşık sayı ayraç içinde belirtilir, ayraç içinde, soldaki sayı karmaşık sayının gerçek sayı bölümünü, sağdaki sayı karmaşık sayının sanal bölümünü belirtir. Ayraç içindeki bu sayıların türü gerçek sayıdır. Bu iki sayıyı ayırmak için bir virgül kullanılır.

Karmaşık veri türünden olan sayılara örnekler aşağıda verilmiştir:

```
(4.0 , 3.0)

(-5. , 6.)

(80.5 , 10.6)
```

Karmaşık sayı belirtiminde sayıların arasına, ayraçların arasına ve virgülün yanlarına boşluklar eklenebilir. Boşluk eklemek program kodlarının okunurluğu arttırabilir.

2.3.5. Mantıksal Veri Türü

Bu veri türünde olan bir sabit, mantıksal olarak doğru ya da yanlış ifadesidir. Bu veri türünü tanımlamak için **logical** deyimi kullanılır. Bu veri türünde sabit, doğruyu belirtiyorsa,

```
.true.
```

yanlışı belirtiyorsa,

```
.false.
```

olarak gösterilir.

2.3.6. Karakter Veri Türü

Bu veri türü bir ya da daha çok karakterden oluşan bir sözce belirtir. Bu veri türünü tanımlamak için **character** deyimi kullanılır. Sözce, tek tırnak ya da çift tırnak içinde yazılır. Burada sabitin ne kadar karakter içereceği belirtilebilir. Sabitin uzunluğu belirtilmemişse sabit ancak tek bir karakter saklayabilir.

Karakter türündeki sabitlere örnekler aşağıda verilmiştir:

```
'Deneme 12345'
```

```
"ABCÇDEFG"
```

```
"Merhaba!"
```

İçinde çift tırnak olan bir sabit belirtirken eğer sabit çift tırnak içinde yazılmışsa bu sabitin içindeki çift tırnak çift olarak yazılır. İçinde tek tırnak olan bir sabit belirtirken eğer sabit tek tırnak içinde yazılmışsa bu sabitin içindeki tek tırnak çift olarak yazılır.

İçinde tek tırnak ya da çift tırnakla belirtilmiş karakter veri türünden sabitler aşağıda verilmiştir:

```
'Edirne''deki camiler...'
```

```
"""Su 100 derecede kaynar."""
```

```
"a'nın ve b'nin değeri 10'dur."
```

```
'Fortran''da "if" yapısı'
```

2.4. Değişkenler

Değişkenler, program çalışırken, bilgisayarın belleğinde bulunan, istenildiğinde değiştirilebilen veriyi saklayan veri bölgeleridir. Programda bir değişkeni kullanmak için önce o değişkeni bildirmek gerekir. Değişkenler, kolaylık için bellek adresleriyle bildirmek yerine tanımlayıcılarla bildirilir. Değişkenleri bildirme şekli aşağıda belirtilmiştir:

```
Veri Türü :: Tanımlayıcı
```

Değişken bildirmelere örnekler aşağıda verilmiştir:

```
integer :: x

real :: a

character :: komut
```

Aynı veri türünde olan birden çok değişkeni tek bir satırda bildirmek için değişken tanımlayıcılarının arasına virgül konulmalıdır.

```
Veri Türü :: 1. Tanımlayıcı, 2. Tanımlayıcı, ...
```

Aynı veri türünde olan birden çok değişkeni tek satırda bildirmeye örnekler aşağıda verilmiştir:

```
integer :: x, y, z

real :: a, b, c

character :: harf, komut, ad, soyad
```

2.4.1. Değişkenlere Başlangıç Değeri Atamak

Değişkenler tanımlanırken kendilerine ilk değer ataması yapılabilir. Derlenmiş programda, o bölgede o değer bulunur. Değişkenlere ilk değer atamasında eşittir işareti kullanılır.

```
Veri Türü :: Tanımlayıcı = Değer
```

Başlangıç değeri atanmış değişken tanımlamalarına örnekler aşağıda verilmiştir:

```
complex :: i = (4.0 , 3.0)

integer :: sayi_1 = 150, sayi_2 = 300, sayi_3 = 600

character :: sinamadizgisi = "12345"
```

Fortran 2003 ile birlikte değişkenlere başlangıç değeri olarak bir fonksiyonun döndüreceği değer de atanabilir. Fonksiyon kavramı 8. Bölüm'de işlenecektir. Örneğin:

```
real :: a = sin(1.0), b = cos(1.0)
```

Eşittir işaretinin yanlarına boşluklar koyulmasa de program çalışır. Eşittir işaretinin yanlarına boşluklar koymak program kodlarının okunurluğunu arttırır.

2.4.2. Bir Değişkene Erişmek

Tanımlanmış bir değişkene erişmek için o değişkenin tanımlayıcısını kullanmak gerekir.

```
Tanımlayıcı
```

2.4.3. Değişkenlerin Bellekte Kapladığı Alanı Bayt Cinsinden Belirlemek

Değişkenlerin bellekte kapladığı alanların derleyici tarafından önceden tanımlandığı bayt cinsinden büyüklükleri vardır. Bu ön tanımlı değeri değiştirmek için **kind** belirteci kullanılabilir. Bu belirtecin kullanımı aşağıda verilmiştir:

```
Veri Türü(kind = Sayı) :: Tanımlayıcı
```

Kullanımda eşittir işaretinin yanlarına boşluklar koyulmasa da program kodu sorunsuz şekilde derlenir. Ancak, eşittir işaretinin yanlarına boşluklar koymak program kodlarının okunurluğunu arttırır.

Tamsayı değişkenler, gerçek sayı değişkenler veya karmaşık sayı değişkenler tanımlanırken bunların bayt cinsinden büyüklükleri de tanımlandığında, parantez içinde yalnızca sayı da yazılabilir ya da parantezler olmadan sayı ve veri türü deyimi arasına yıldız işareti konulabilir.

GNU Fortran'ın tanımladığı, veri türlerine göre bayt büyüklükleri aşağıda bulunan tabloda verilmiştir:

Tablo 2.1 – Veri Türlerinin Olabileceği Büyüklükler

Veri Türü	Büyüklükler	Ön Tanımlı
Tamsayı	1, 2, 4, 8*, 16*	4
Tek Duyarlıklı Gerçek Sayı	4, 8, 10*, 16*	4
Çift Duyarlıklı Gerçek Sayı	4, 8, 10*, 16*	8
Karmaşık Sayı	4, 8, 10*, 16*	4
Mantıksal	1, 2, 4, 8*, 16*	4
Karakter	1, 4	1

* Tüm sistemlerde kullanılmayabilir.

kind belirteciyle veri türü büyüklüğü belirtimine örnekler aşağıda verilmiştir:

```
integer(kind = 8) :: sayi_1, sayi_2

character(kind = 4) :: isim

real(kind = 16) :: uzunluk

real(8) :: en, boy

real*8 :: toplam
```

2.4.4. Karakter Türü Değişkenlerin Karakter Cinsinden Uzunluklarını Belirlemek

Karakter türünde değişkenler, tanımlanırken karakter cinsinden uzunlukları belirtilebilir. Bu tür bir değişkeni tanımlarken bu cinsten uzunluk belirtilmediyse ve bu değişkene başlangıç değeri ataması yapılmadıysa bu değişken tek karakterlik olarak varsayılır. Bu cinsten uzunluğu belirtmek için character deyiminden sonra parantez içinde uzunluk belirtilir.

```
character(Uzunluk) :: Tanımlayıcı
```

Karakter türünde olan değişkenlerin karakter cinsinden uzunluklarının belirtilmesine örnekler aşağıda verilmiştir:

```
character(30) :: tam_ad

character(6) :: uyari = "UYARI!"

character(8) :: numara = "123456"
```

Karakter veri türünde ilk değer ataması yapıldığında eğer uzunluk, belirtilmemişse atanan değerin karakter sayısına göre yer ayrılır, belirtilmişse atanan değerin karakter sayısı belirtilen uzunluktan büyük olmamalıdır, fazla gelen karakterler sayılmaz.

Karakter türünde olan değişkenlerin karakter cinsinden uzunluklarını parantezler olmadan sayı ile veri türü deyimi arasına yıldız işareti konularak da belirtilebilir. Karakter türünde olan değişkenlerin karakter cinsinden uzunluklarının böyle belirtilmesine örnekler aşağıda verilmiştir:

```
character*4 :: ogrenci_no

character*2 :: il_kodu = “22”
```

Karakter veri türünden değişkenlerin içereceği karakter sayısı **len** belirteciyle de belirtilebilir. len belirteciyle belirtmeyle bir önceki belirtme arasında bir fark yoktur.

```
character(len=Uzunluk) :: Tanımlayıcı
```

len belirteciyle belirtmeye örnekler aşağıda verilmiştir:

```
character(len=80) :: adres

character(len=200) :: ileti

character(kind=1, len=11) :: kimlik_numarasi = "12345678901"
```

2.4.5. Alt Sözceler

Alt sözce, bir sözcenin bir kısmının belirtimesiyle elde edilen sözcedir. Alt sözce belirtimi aşağıdaki gibidir:

Karakter Türünde Değişken(**Kaçıncı Karakterden Beri İfadesi**:**Kaçıncı Karaktere Kadar İfadesi**)

Alt sözce belirtimine örnekler aşağıda verilmiştir:

```
il_kodu = plaka(1:2)

yeni_dizgi = eski_dizgi(baslangic:bitis)

dizgi = alinan_1(x:y) // alinan_2(a:b)
```

2.4.6. Kapalı Bildirme ve Açık Bildirme

Kapalı bildirme; bir programın, bir alt programın ya da bir modülün başında değişken bildirimlerinden önce hangi harfle başlayan değişkenin hangi türde olacağının belirlenmesidir. Bunun için **implicit** deyimi kullanılır. Örnek olarak ilk harfi, a'dan c'ye ve e'den h'ye olan harflerden olan değişkenlerin tamsayı, d olan değişkenlerin çift duyarlıklı gerçek sayı türünde olması için aşağıdaki satırlar yazılır:

```
implicit integer (a-c,e-h), double precision (d)
```

ya da

```
implicit integer (a-c,e-h)
implicit double precision (d)
```

FORTRAN 77'den beri kapalı bildirmede varsayılan olarak adları **I**, **J**, **K**, **L**, **M**, **N** veya **i**, **j**, **k**, **l**, **m**, **n** harflerinden biriyle başlayan değişkenler tamsayı, diğerleri gerçek sayı olarak bildirilmektedir.

Açık bildirme, kapalı bildirmenin olmadığı bildirimdir. Kapalı bildirmeyi önlemek ve değişkenleri tek tek açık olarak bildirmek için değişken bildirimlerinden önce Fortran 90 ile birlikte gelen **implicit none** deyimi kullanılır. Bu deyimin kullanılması, değişkenleri bildirme konusunda kafa karışıklığını önlemek ve değişken isimlerinde yazım yanlışlarını saptamak açısından önerilir.

Bu kitaptaki tüm örnek programlarda değişkenler açık olarak bildirilmiştir.

2.5. Sabitler

Sabitler, aslında değişken olmayan verilerdir. İsimli sabit tanımlaması **parameter** niteleyicisiyle yapılır, böylece belirtilen isme atanan belirtilen değer program çalışırken değiştirilemez. İsimli sabit tanımlamasında ilk değer mutlaka verilmelidir.

```
Veri Türü, parameter :: Tanımlayıcı = Değer
```

İsimli sabit tanımlamasına örnekler aşağıda verilmiştir:

```
real, parameter :: pi = 3.14159, e = 2.71828
```

```
character, parameter :: sinamailetisi = "Bu bir sınama iletisidir."
```

```
complex, parameter :: i = ( 0.0 , 1.0 )
```

Aslında verilerin doğrudan belirtimi de sabitlerdir.

Doğrudan belirtilen sabitlere örnekler aşağıda verilmiştir:

```
5
```

```
-43.65
```

```
-8e-10

(3.0, 4.0)

“Sonuç:”
```

İsimli sabit tanımlamasında, değişken tanımlamada olduğu gibi, bayt türünden olan büyüklük belirtimi ya da eğer sabit karakter türündeyse karakter uzunluğu belirtimi yapılabilir. İsimli bir sabite erişmek, bir değişkene erişmek gibidir.

Sabit kullanmak, programın çalışma hızını arttırır ama böyle program bellekte biraz daha yer kaplar. Hız arttırımıyla yer kaplama çok küçük olsa da çok büyük programlar için önemli olabilir.

Sabit isimlendirme kuralları, değişken isimlendirme kurallarıyla aynıdır.

2.6. Diziler

Dizi, boyutu veya boyutları olan, aynı veri türünde olan verilerin oluşturduğu bir kümedir. Dizinin istenilen bir ögesine istenildiği zaman erişilebilir. Fortran'da dizi tanımlaması değişken tanımlaması gibidir, yalnız bu tanımlamada öge sayısı belirtimi de yapılır.

2.6.1. Tek Boyutlu Diziler

Tek boyutlu bir dizi bildirimi aşağıdaki iki şekilde olabilir:

```
Veri Türü :: Tanımlayıcı(Öge Sayısı)

Veri Türü, dimension(Öge Sayısı) :: Tanımlayıcı
```

Tek boyutlu dizi bildirimlerine örnekler aşağıda verilmiştir:

```
integer :: sayi_dizisi(1000)

real, dimension(3) :: a, b, c

character, dimension(20) :: isim
```

Dizinin bir ögesine erişmek için dizi tanımlayıcısının yanında o ögenin sıra numarasını da belirtmek gerekir.

Tanımlayıcı(**Sıra Numarası**)

Dizinin belirli bir kısmındaki ögelerine erişmek içinse başlangıç ve bitiş sıra numaralarını belirtmek gerekir. Burada, başlangıç sıra numarası yazılmazsa en baştaki sıra numarasından başlanır, bitiş sıra numarası yazılmazsa en sondaki sıra numarasında sonlanılır, atlama sayısı ve önündeki iki nokta yazılmazsa 1 olarak kabul edilir.

Tanımlayıcı(**Başlangıç Sıra Numarası**:**Bitiş Sıra Numarası**:**Atlama Sayısı**)

Dizinin ögelerine erişmeye örnekler aşağıda verilmiştir:

```
x(5)

notlar(3)

kume(200)

a(1:10:2)

b(10:1:-2)
```

Bir çok programlama dilinde dizilerde sıra numaraları sıfırdan başlar, öge sayısının bir eksiğine kadar gider. Ancak Fortran'da dizilerde sıra numaraları ön tanımlı olarak birden başlar, dizinin öge sayısına kadar gider.

Fortran'da dizilerde farklı sıra numaralarıyla bildirme olanaklıdır. Bunun için dizi bildiriminde ilk ögenin sıra numarasıyla sonuncu ögenin sıra numarası belirtilir, diğer ögelerin sıra numaraları bu numaraların arasındadır.

```
Veri Türü :: Tanımlayıcı(İlk Sıra Numarası:Son Sıra Numarası)

Veri Türü, dimension(İlk Sıra Numarası:Son Sıra Numarası) ::
Tanımlayıcı
```

Dizileri, farklı sıra numaralarıyla bildirmeye örnekler aşağıda verilmiştir:

```
integer :: a(0:9), b(-5:5), c(10:49)

complex, dimension(-5:5) :: x, y

logical, dimension(100:999) :: numaralar
```

Fortran'da dizi tanımlaması yapıldığında başlangıç değeri belirtilebilir. Dizi tanımlamasında iki tür başlangıç belirtimi vardır: Birincisi, bütün ögelerine tek bir değer atamak; ikincisi, dizinin ögelerinin hepsini ayrı ayrı belirtmektir. Bunlardan birincisi için tanımlamalar iki şekilde olabilir:

```
Veri Türü :: Tanımlayıcı(Öge Sayısı) = Değer

Veri Türü, dimension(Öge Sayısı) :: Tanımlayıcı = Değer
```

Bunlardan ikincisi için tanımlamalar da iki şekilde olabilir:

```
Veri Türü :: Tanımlayıcı(Öge Sayısı) = (/ 1. Değer, 2.Değer, ... /)

Veri Türü, dimension(Öge Sayısı) :: Tanımlayıcı = (/ 1. Değer,
2.Değer, ... /)
```

Bunlardan ikincisi için tanımlamalarda, içinde atanacak değerlerin bulunduğu, parantezlerle ve eğik çizgilerle kapalı sıraya **dizi yapıcısı** denir.

Dizileri ilk değer atamaları yapılarak tanımlamaya örnekler aşağıda verilmiştir:

```
real :: x(100) = 0, y(100) = 0

logical, dimension(4) :: durumlar = .false.

character, dimension(500) :: dizgiler = "DENEME!"

real :: x(5) = (/ 1.0, 2.0, 3.0, 4.0, 5.0 /)

integer, dimension(4) :: a = (/ 5, 10, 20, 50 /), b = (/ 85, 41, 33,
96 /)

character(len=10), dimension(6) :: isimler = (/ "Ercan", "Erdem",
"Ersoy", "Hasan", "Ali  ", "Ayşe " /)
```

Karakter veri türünde tek tek atama yapıldığında dizinin her bir ögesi için aynı karakter uzunluğu ayrılır ve atanan değerler aynı karakter uzunluğunda olmalıdır. Bir öge için, belirtilen karakter sayısı, ayrılan karakter uzunluğundan azsa boşta kalan kısım (sol taraf) boşluklarla doldurulur, ayrılan karakter uzunluğundan fazlaysa fazla gelen karakterler (en sağdaki karakterler) sayılmaz.

Tek boyutlu bir diziye değerlerin döngü (Döngüler, “Döngü Yapıları” bölümünde anlatılacaktır) kullanılarak atanmasını sağlanabilir. Başlangıç ve bitiş değerleri tamsayı olmak zorundadır, atlama sayısı yazılmazsa 1 olarak kabul edilir. Bu özellik yalnızca sayı türlerinden olan dizilerde geçerlidir.

```
Veri Türü :: Tanımlayıcı(Öge Sayısı) = (/ (İfade, Değişken = Başlangıç, Bitiş, Atlama Sayısı) /)
```

```
Veri Türü, dimension(Öge Sayısı) :: Tanımlayıcı = (/ (İfade, Değişken = Başlangıç, Bitiş, Atlama Sayısı) /)
```

Döngü kullanılarak değer atamaya örnekler aşağıda verilmiştir:

```
integer :: a(100) = (/ (j, j = 0, 99) /)
```

```
real, dimension(8) :: x = (/ (i*0.1, i = 1, 5), 1, 2, 3 /)
```

```
complex, dimension(5) :: sayilar = (/ (sayac, sayac = 1, 10, 2) /)
```

2.6.2. Çok Boyutlu Diziler

Fortran 95 ve Fortran 2003 sürümlerinde en çok 7, Fortran 2008'de en çok 15 boyutlu diziler oluşturulabilir. Çok boyutlu bir dizideki toplam öge sayısı dizinin tüm boyutlarının öge sayılarının çarpımıdır. Çok boyutlu bir dizi belirtimi iki şekilde olabilir:

```
Veri Türü :: Tanımlayıcı(1. Boyuttaki Öge Sayısı, 2. Boyuttaki Öge Sayısı, ...)
```

```
Veri Türü, dimension(1. Boyuttaki Öge Sayısı, 2. Boyuttaki Öge Sayısı, ...) :: Tanımlayıcı
```

Çok boyutlu dizilere örnekler aşağıda verilmiştir:

```
integer :: a(3, 3), b(4, 5)

real, dimension(6, 6, 6) :: koordinatlar

character, dimension(100, 2) :: adlar, soyadlar
```

Tek değerli değişkene ya da sabite **sayıl**, tek boyutlu dizilere **vektör**, iki boyutlu dizilere **matris** de denir. Buradaki vektör ve matris kavramları matematikteki vektör ve matris kavramları gibidir.

m × n türünde bir matrisi belirtmek için dizi belirtiminde boyutların belirtimi (m, n) olmalıdır.

Dizilerde; boyut sayısına **derece**, bir boyutun öge sayısına **kapsam**, tüm boyutların toplam öge sayısına **büyüklük**, dizilerin tüm boyutlarının kapsamlarına **biçim** denir.

Çok boyutlu dizilerde her boyutun sıra numaraları bağımsızdır. Çok boyutlu dizilerde sıra numaralarının belirtimi aşağıdaki iki şekilde olabilir:

```
Veri Türü :: Tanımlayıcı(1. İlk Sıra Numarası:1. Son Sıra Numarası,
2. İlk Sıra Numarası:2. Son Sıra Numarası, ...)

Veri Türü, dimension(1. İlk Sıra Numarası:1. Son Sıra Numarası, 2.
İlk Sıra Numarası:2. Son Sıra Numarası, ...) :: Tanımlayıcı
```

Çok boyutlu dizilerde sıra numaralarının belirtimine örnekler aşağıda verilmiştir:

```
integer :: a(1:20, 21:40, 41:60)

character, dimension(0:99, 0:99) :: veriler, veirler_2
```

```
integer, dimension(-5:5, -5:5) :: x, y, z

real, dimension(-5:5, 3:7) :: i, j, k
```

Çok boyutlu dizilerde başlangıçta öge ataması yalnızca tek bir değer atamayla yapılır.

Veri Türü :: **Tanımlayıcı**(**1. Boyuttaki Öge Sayısı**, **2. Boyuttaki Öge Sayısı**, **...**) = **Değer**

Veri Türü, dimension(**1. Boyuttaki Öge Sayısı**, **2. Boyuttaki Öge Sayısı**, **...**) :: **Tanımlayıcı** = **Değer**

Çok boyutlu dizilerde başlangıçta öge atamasına örnekler aşağıda verilmiştir:

```
integer :: veriler(100,100) = 0

logical, dimension(2, 2) :: a = .true.

real, dimension(10, 5) :: i = 3.0, j = 4.0
```

Çok boyutlu dizilerde bulunan bir ögeye erişmek için tanımlayıcının yanında o ögenin sıra numaraları da belirtilir.

Tanımlayıcı(**1. Boyuttaki Sıra Numarası**, **2. Boyuttaki Sıra Numarası**, **...**)

Çok boyutlu bir dizinin belirli bir kısmındaki ögelerine erişmek içinse başlangıç sıra numaralarını ve bitiş sıra numaralarını belirtmek gerekir. Burada bir boyutta, başlangıç sıra numarası yazılmazsa o boyutta en baştaki sıra numarasından başlanır, bitiş sıra numarası yazılmazsa o boyutta en sondaki sıra numarasında sonlanılır, atlama sayısı ve önündeki iki

nokta yazılmazsa 1 olarak kabul edilir, yalnızca tek bir sayı belirtilirse o boyuttaki tek sıra numarası kabul edilir, atlama sayısı negatif olabilir fakat başlangıç sıra numaralarına ve bitiş sıra numaralarına dikkat etmek gerekir.

Tanımlayıcı(**1. Boyuttaki Başlangıç Sıra Numarası**:**1. Boyuttaki Bitiş Sıra Numarası**:**Atlama Sayısı**, **2. Boyuttaki Başlangıç Sıra Numarası**:**2. Boyuttaki Bitiş Sıra Numarası**:**Atlama Sayısı**, ...)

Çok boyutlu bir dizinin belirli bir kısmındaki ögelerine erişmek için başlangıç ve bitiş sıra numaralarını belirtmeye örnekler aşağıda verilmiştir:

```
matris(3, 2)

a(1:10:2, 5)

b(1, 10:1:-2)
```

Sabit dizi tanımlaması sayıl sabitte olduğu gibi, **parameter** niteleyicisiyle yapılır. Bu niteleyici, dimension niteleyicisinin sonrasında yer alır. Burada da ilk değer ataması yapılmalıdır.

Veri Türü, parameter :: **Tanımlayıcı**(...) = **Değer ya da Dizi Yapıcısı**

Veri Türü, dimension(**...**), parameter :: **Tanımlayıcı** = **Değer ya da Dizi Yapıcısı**

Sabit dizi tanımlamaya örnekler aşağıda verilmiştir:

```
integer, parameter :: degerler(5, 5, 5) = 100

real, dimension(3), parameter :: carpanlar = (/ 1.0, 2.0, 5.0 /)

character, dimension(2), parameter :: isimler = (/"Ercan", "Erdem"/)
```

2.6.3. Dizi Ögelerinin Bellekte Saklanması

Fortran'da, dizilerde bulunan ögeler bellekte art arda saklanır. Tek boyutlu dizilerde sıralama ilk sıra numarasından son sıra numarasına doğru gider. İki boyutlu dizilerde (matrislerde), sıralama sütuna göredir, yani ögeler sütun sütun sıralanır. C ve C++ dillerinde bu sıralama satıra göredir.

2.6.4. Alt Diziler

Fortran'da bir dizi içinden bir öge yerine bir ögeler dizisi de belirtilebilir, böyle diziye alt dizi denir. Bir dizinin n. boyutunda belirli sıra numarasından belirli başka bir sıra numarasına kadar olan ögelerden oluşan bir alt dizi belirtimi aşağıda verildiği gibidir:

```
Tanımlayıcı(..., n. Boyuttaki Sıra Numarası (Başlangıç):n. Boyuttaki Sıra Numarası (Bitiş), ...)
```

Örneğin, a × b × c biçimindeki bir dizide (1, 2:5, 4) olarak bir alt dizi belirtildiğinde tanımlanan alt dizi, 1. boyutta sıra numarası 1 ve 3. boyutta sıra numarası 4 olan, 2. boyutta sıra numarası 2'den 5'e kadar olan ögelerden oluşur. Eğer bir boyuttaki tüm ögelerden alt dizi belirtilecekse, örnek olarak a × b × c biçimindeki bir dizinin, 1. boyutta sıra numarası 1 ve 3. boyutta sıra numarası 4 olan, 2. boyutundaki tüm ögelerinden bir alt dizi belirtilecekse (1, :, 4) olarak belirtme yapılmalıdır.

2.6.5. Dinamik Diziler

Dinamik diziler, boyutları program içinde değişebilen dizilerdir. Dinamik dizilerle ana belleği daha etkin biçimde kullanan programlar yazılabilir.

Dinamik diziler, programın derlenmesi sırasında dizinin boyutu bilgisi derlenmiş kodda bulunmaz.

2.6.5.1. Dinamik Dizilerin Tanımlanması

Dinamik dizilerin tanımlanması diğer dizilerinkinden farkları, **allocatable** niteleyicisinin kullanılması ve sahip olacağı boyut sayısı kadar virgüllerle ayrılan iki nokta ile belirtilmesidir.

```
Veri Türü, dimension(:), allocatable :: Tanımlayıcı
```

Örnekler aşağıda verilmiştir:

```
integer, dimension(:), allocatable :: a, b, c

real, dimension(:,:), allocatable :: alan

real, dimension(:,:,:), allocatable :: ortam_1, ortam_2
```

2.6.5.2. Dinamik Dizilere Bellekte Yer Ayrılması

Dinamik dizilerde bellekte yer ayrılması **allocate** deyimiyle sağlanır.

```
allocate(Dizi Adı(Kullanıcı Tanımlı Boyut Belirtimleri)[, stat = Durum Değişkeni])
```

Durum değişkenine, diziye bellekte yer ayrılmışsa sıfır değeri, yer ayırma esnasında bir sorun olduysa sıfırdan büyük bir değer atanır.

allocate deyiminin kullanımına örnekler aşağıda verilmiştir:

```
allocate(dizgiler(1000), stat = durum)

allocate(harita(50,50))
```

2.6.5.3. Dinamik Dizilere Bellekte Ayrılan Yerin Serbest Bırakılması

Dinamik dizilere bellekte ayrılan yer **deallocate** deyimiyle serbest bırakılır. Program sonlandırılmadan önce böyle dizilere ayrılan yerlerin serbest bırakılması gerekir.

```
deallocate(Dizi Adı)
```

2.7. Türetilmiş Veri Türleri

Türetilmiş veri türü, farklı veri türlerinden olan değişkenlerin oluşturduğu kullanıcı tanımlı bir veri türüdür. Bu değişkenlerin veri türleri basit veri türlerinden ve başka türetilmiş veri türlerinden olabilir. Veri türü bildirimi aşağıdaki gibi yapılır:

```
type Tür İsmi
   Bildirimler
end type Tür İsmi
```

Türetilmiş veri türü bildirilmesine örnekler aşağıda verilmiştir:

```
type ogrenci
   character(25) :: ad
   character(25) :: soyad
   integer(kind=4) :: numara
   real(kind=4) :: not_ortalamasi
end type ogrenci

type urun
   character(6) :: urun_kodu
   character(50) :: urun_ismi
   real :: urun_fiyati
```

```
end type urun
```

Yukarıdaki örneklerde olduğu gibi bildirim satırlarının başlarına birkaç boşluk bırakmak program kodlarının okunurluğunu arttırır.

Türetilmiş bir veri türünün bir bileşeni isimli sabit olamaz.

Tanımlanan bir türetilmiş veri türünden olan bir değişken bildirimi normal değişken bildirimi gibidir ancak burada **type** anahtar sözcüğü de kullanılır ve veri türü adı parantez içinde yazılır.

```
type(Tür İsmi), Niteleyiciler :: Tanımlayıcı
```

Tanımlanan bir türetilmiş veri türünden olan bir değişken bildirimine örnekler aşağıda verilmiştir:

```
type(urun) :: elma, armut, muz

type(kayit), parameter :: kayit_1, kayit_2

type(ogrenci), dimension(1000) :: ogrenciler
```

Türetilmiş bir veri türünde olan bir değişkenin ya da bir sabitin bir bileşenine erişimek için tanımlayıcı adı ve bileşen adı arasına yüzde işareti konur.

```
Tanımlayıcı%Bileşen
```

Türetilmiş bir veri türünde olan bir değişkenin ya da bir sabitin bir bileşenine erişimek için tanımlayıcı ve bileşen adlarıyla yüzde işareti kullanılmasına örnekler aşağıda verilmiştir:

```
elma%urun_fiyati
```

```
kayit_1%kayit_adi

ogrenciler(5)%numara

zaman%dakika
```

Bir türetilmiş veri türü bildirmek için başka bir türetilmiş veri türünü temel alabiliriz. Veri türü temel alma için **extends** deyimi kullanılır. Bu özellik, Fortran 2003 ve sonrası içindir.

```
type, extends(Temel Alınacak Türün İsmi) :: Yeni Veri Türünün İsmi
   Bildirimler
end type Yeni Veri Türünün İsmi
```

Bir türetilmiş veri türünü temel alma örnekleri ağaşğıda verilmiştir:

```
type, extends(sekil) :: kare
   integer(kind=8) :: kenar_uzunlugu
end type kare

type, extends(tasit) :: otomobil
   integer(kind=1) :: kapi_sayisi
   integer(kind=2) :: motor_gucu
end type tasit
```

Başka bir türetilmiş veri türünü temel almış türetilmiş veri türünün bir bileşenine erişmek için türetilmiş veri türlerinin isimleri temel alma sırasına göre tersten yazılır ve aralarına yüzde işareti konulur.

```
Tanımlayıcı%Türetilen Veri Türleri%Bileşen
```

Başka bir türetilmiş veri türünü temel almış türetilmiş veri türünün bir bileşenine erişmeye örnekler aşağıda verilmiştir:

```
birinci_otomobil%tasit%agirlik

birinci%otomobil%yaris_arabasi%tasit%en_yuksek_hiz
```

2.8. Program Ana Birimini Belirtme

Fortran'da program ana birimini belirtmek için **program** deyimi ve **end program** deyimi kullanılır. Program ana biriminin başında program sözcüğü ve programın ismi yazılır. Program ana biriminin sonunda end program anahtar sözcüğü ile programın ismi yazılır.

```
program Program İsmi
   ...
end program Program İsmi
```

Sonda program sözcüğü ile programın ismi yazılmayabilir.

```
program Program İsmi
   ...
end
```

Başta ve sonda program sözcükleri ile programın ismi de yazılmayabilir.

```
...
end
```

Program sözcüğünü belirtmek, büyük programlarda program ana biriminin alt programlardan kolayca ayırt edilmesini sağlar.

program ve end program arasındaki satırların başlarında bir kaç boşluk bırakmak program kodlarının okunurluğunu arttırır.

2.9. İstenilen Yerde Programdan Çıkma

Fortran'da **stop** deyimi ile istenilen yerde programdan çıkılabilir.

```
stop
```

2.10. Kod Yazımında Satırların Ayrılması ya da Birleştirilmesi

Fortran'da bir satırı bölüp birden çok satıra dağıtmak için ve (&) işareti kullanılır. Örneğin:

```
integer :: x = 5, y = 6, z = 34, t = 25, a = 4, b = 634, c= 45, &
sayi_1 = 10, sayi_2 = 65
```

Fortran'da birden çok satırı tek bir satırda yazmak için noktalı virgül kullanılır. Örneğin:

```
real, parameter :: x = 5.7, y = 7.3, z = 3.2 ; integer :: a = 5, b =
34, c = 6
```

GNU Fortran'da serbest biçimde kod yazmada bir satırın ön tanımlı azami uzunluğu 132 karakterdir. Bu tür uzunluk kısıtlamalarını aşmak için satırları bölüp daha çok satıra dağıtabilirsiniz. Ayrıntılı bilgi için GNU Fortran'ın belgelerine başvurabilirsiniz.

2.11. Özet

Fortran dilinde açıklamalar ünlem işaretiyle belirtilir. Fortran’da değişkenler ve sabitler tanımlanır. Veri türleri tamsayı, tek duyarlıklı gerçek sayı, çift duyarlıklı gerçek sayı, karmaşık sayı ve karakterdir. Art arda veriler tanımlamak için dizi tanımlaması yapılabilir ve diziler birden çok boyutlu olarak da tanımlanabilir. Ayrıca diziler, dinamik olarak bildirilebilir. Fortran programlarında ana program **program** deyimiyle ve **end program** deyimiyle belirtilir. Alt

programların belirtimine daha sonra değinilecektir. Fortran programlama dilinde, bir satır birden çok satıra dağıtılabilir veya birden çok satır tek bir satırda yazılabilir.

2.12. Sorular

2.1) Fortran programlama dilinde Fortran 90 ve sonrası için açıklama bölümleri nasıl oluşturulur?

2.2) Fortran programlama dilinde tanımlayıcılar hakkında bilgi veriniz.

2.3) Fortran programlama dilinde hangi basit veri türleri vardır?

2.4) Fortran programlama dilinde her veri türünden olan ifadelerden ikişer örnek veriniz.

2.5) Fortran programlama dilinde değişken tanımlaması ilk değer atamasıyla birlikte nasıldır?

2.6) Fortran programlama dilinde tek boyutlu dizi tanımlaması ilk değer atamasıyla birlikte nasıldır?

2.7) Fortran programlama dilinde iki boyutlu dizi tanımalaması ilk değer atamasıyla birlikte nasıldır?

2.8) Fortran programlama dilinde üç boyutlu dizi tanımlaması ilk değer atamasıyla birlikte nasıldır?

2.9) Fortran programlama dilinde dinamik dizi bildirimi nasıl yapılır?

2.10) Fortran programlama dilinde türetilmiş veri türü oluşturmak için hangi deyimler kullanılır?

2.11) Fortran programlama dili kullanılarak türetilen bir türetilmiş veri örneği veriniz.

2.12) Fortran programlama dilinde ana program nasıl belirtilir?

2.13) Fortran programlama dilinde istenilen yerde programdan çıkmak için hangi deyim kullanılır?

2.14) Fortran programlama dilinde kod satırları hangi işaretle ayrılabilir?

2.15) Fortran programlama dilinde kod satırları tek satırda nasıl yazılabilir?

3. Bölüm: İfadeler, Deyimler ve İşleçler

Bu bölümde temel programlama ögelerinin anlatılmasına devam edilecektir.

3.1. İfadeler

İfadeler, hesaplama ve değerlendirme için geçerli bir birim oluşturan sembollerin birleşimidir. İfadeler, bir değer ya da bir işlem olabilir. Her bir ifade bir değer belirtir, bu değerin veri türü, belirtilen ifadeye göre değişebilir.

İfadelere örnekler aşağıda verilmiştir:

```
"Merhaba Dünya!"
```

```
-1738.52
```

```
.true.
```

```
4.3d4
```

```
yaricap ** 2. * pi
```

```
a * x ** 2 + b * x + c
```

```
(x – a) + (y – b)
```

```
"Fortran" // " " // "2008"
```

```
2 * sin(a) * cos(a)
```

3.2. Deyimler

Deyimler, her biri bir satırı oluşturan tam yönergelerdir. Deyimler, yürütülebilen ve yürütülmeyen olmak üzere ikiye ayrılır. Yürütülmeyen deyimler modül kullanma (use) deyimleri, değişken bildirimleri, fonksiyon ara birimleri (fonksiyon ilk örnekleri), derleme sürecinde yüklenen veriler gibi deyimlerdir. Yürütülebilen deyimler çalışma zamanında yürütülen deyimlerdir. Biçimlendirme (format) deyimleri gibi deyimler dışındaki tüm yürütülmeyen deyimler yürütülebilen deyimlerden önce gelmelidir. Her deyim bir satır sonu karakteriyle sonlanır.

Yürütülmeyen deyimlere örnekler aşağıda verilmiştir:

```
use islemler

integer(kind=4) :: x = 10, y = -200, z = 5

interface kurenin_alani

end type ogrenci

implicit none
```

Yürütülebilen deyimlere örnekler aşağıda verilmiştir:

```
x = x + 1

z = sin(y)

print *, "Merhaba!"
```

3.2.1. Atama Deyimleri

Bir önceki bölümde bir değişkenin tanımlandığı sırada değişkene başlangıç değeri ataması vardı. Başlangıç değeri ataması yapılan değişkenleri tanımlamadan sonra bu değişkenlerin değerlerini değiştirmek için ve başlangıç değeri ataması yapılmayan değişkenleri bildirmeden sonra bu değişkenlere değer atamak için atama deyimleri kullanılır.

```
Tanımlayıcı = İfade
```

Atama deyimlerine örnekler aşağıda verilmiştir:

```
x = x + 1

y = a * x ** 2 + b * x + c

z = "Erdem " // "Ersoy"
```

3.3. İşleçler

İşleçler, Fortran'da belirli ifadeleri belirtmek, birtakım deyimleri oluşturmak ve ifadelerle belirli işlemleri yapmak için kullanılan sembollerdir. Fortran'da tanımlanan işleç türleri aşağıda verilmiştir:

- Atama işleci
- Aritmetik işleçleri
- İlişki işleçleri
- Mantıksal işleçler
- Sözceleri birleştirme işleçleri

3.3.1. Atama İşleci

Fortran'da atama deyimlerini oluşturmak için atama işleci kullanılır. Atama işleci eşittir işaretidir. Atama işlecinin nasıl kullanıldığı "Atama Deyimleri" konusunda gösterilmiştir.

3.3.2. Aritmetik İşleçler

Fortran'da aritmetiksel ifadeler oluşturmak için aritmetik işleçleri kullanılır. Aşağıdaki tabloda bu işleçler verilmiştir:

Tablo 3.1 – Aritmetik İşleçler

İşleç	Açıklama
+	Aritmetiksel olarak toplama işlecidir.
-	Aritmetiksel olarak çıkarma işlecidir.
*	Aritmetiksel olarak çarpma işlecidir.
/	Aritmetiksel olarak bölme işlecidir.
**	Aritmetiksel olarak kuvvet alma işlecidir.

Aritmetik işleçleriyle aritmetiksel bir ifade oluşturma aşağıdaki gibidir:

```
Birinci İfade  Aritmetik İşleci  İkinci İfade
```

Yukarıdaki gösterimde, birinci ifade, aritmetikte birinci terimi; ikinci ifade, aritmetikte ikinci terimi ifade eder. Aritmetik işleçlerde işleçlerin kullanım önceliği aritmetikteki öncelik gibidir: Önce üs alma, sonra çarpma ve bölme, en sonda toplama ve çıkarma.

Aritmetik işleçleriyle oluşturulan ifadelere örnekler aşağıda verilmiştir:

```
x + 1
```

```
5.4 + -6.9 * 3
```

```
a + b – c * d / e ** kuvvet

sayi_1 – sayi_2

r ** 2 – 2 * r + 1

a ** 3 – 1
```

Fortran'da aritmetiksel ifadeler kullanırken aritmetikte olduğu gibi parantezler de kullanılabilir.

Parantezli aritmetik ifadelere örnekler aşağıda verilmiştir:

```
(3.5 – 2.0) / 0.3

(a + 1) * (a ** 2 – a + 1)

(k * (k + 1)) / ((k + 2) * (k + 3))
```

Yukarıdaki örneklerde olduğu gibi ifadelerin ve işleçlerin aralarına birer boşluk bırakmak program kodlarının okunurluğunu arttırır.

3.3.3. Karşılaştırma İşleçleri

Fortran'da karşılaştırma ifadeleri oluşturmak için karşılaştırma işleçleri kullanılır. Karşılaştırma ifadeleri, iki ifadenin belirtilen koşulu sağlayıp sağlamadığına göre değer döndüren ifadelerdir. Bu ifadelerde koşul sağlanıyorsa **.true.**, sağlanmıyorsa **.false.** değeri döndürülür. Bu ifadeler koşullar oluşturmak için kullanılır. Karşılaştırma işleçleriyle bir karşılaştırma ifadesi oluşturma aşağıdaki gibidir:

```
Birinci İfade  Karşılaştırma İşleci  İkinci İfade
```

Aşağıdaki tabloda karşılaştırma işleçleri verilmiştir:

Tablo 3.2 – Karşılaştırma İşleçleri

İşleç	İşlece Karşılık Fortran 90 Öncesi Kullanım	Açıklama
==	.eq.	İki ifade birbirine eşittir işlecidir.
/=	.ne.	İki ifade birbirine eşit değildir işlecidir.
<	.gt.	Birinci ifade ikinci ifadeden küçüktür işlecidir.
>	.lt.	Birinci ifade ikinci ifadeden büyüktür işlecidir.
<=	.ge.	Birinci ifade, ikinci ifadeden küçüktür veya ikinci ifadeye eşittir işlecidir.
>=	.le.	Birinci ifade, ikinci ifadeden büyüktür veya ikinci ifadeye eşittir işlecidir.

Tamsayı, gerçek sayı ve çift duyarlıklı gerçek sayı türlerindeki ifadeler karşılaştırıldığında, ifadelerin sayı olarak büyüklükleri; karakter türündeki ifadeler karşılaştırıldığında sözcelerin işlemcinin karakter sıralaması (Bu sıralama bazı işlemci mimarilerinde ASCII olabilir.) dikkate alınır. Karmaşık sayı türündeki ifadeler yalnızca == ve /= işleçleriyle karşılaştırılabilir.

Karşılaştırma işleçlerine örnekler aşağıda verilmiştir:

```
x == 0

x .eq. y

a /= b

b ** 2 – 4 * a * c < 0
```

```
sicaklik >= -273.15

a < (b + c)

a .gt. x * y

"Ercan" >= "Ersoy"

ileti == "Hata!"
```

Aritmetik işleçlerinin karşılaştırma işleçlerine göre kullanım önceliği vardır. Karşılaştırma işleçlerinin kendi aralarında bir kullanım önceliği yoktur.

Yukarıdaki örneklerde olduğu gibi ifadelerin ve işleçlerin aralarına birer boşluk bırakmak program kodlarının okunurluğunu arttırır.

3.3.4. Mantıksal İşleçler

Fortran'da mantıksal ifadeler oluşturmak için mantıksal işleçler kullanılır. Mantıksal ifadeler, iki ifadenin mantıksal işlemlerden geçirildikten sonra **.true.** ya da **.false.** gibi mantıksal değerler döndüren ifadelerdir. Mantıksal işleçlerle bir mantıksal ifade oluşturma aşağıdaki gibidir:

```
Birinci İfade  Mantıksal İşleç  İkinci İfade
```

Aşağıdaki tabloda mantıksal işleçler verilmiştir:

Tablo 3.3 – Mantıksal İşleçler

İşleç	Açıklama
.and.	İki ifadeyi mantıksal ve işlemine geçirir.
.or.	İki ifadeyi mantıksal veya işlemine geçirir.
.eqv.	İki ifadeyi mantıksal ancak ve ancak işlemine geçirir.
.neqv.	İki ifadeyi mantıksal özel veya işlemine geçirir.
.not.	Belirtilen ifadenin mantıksal olarak değili sonucunu döndürür.

Bu işleçler, ancak mantıksal değer döndüren ifadeleri işlemlerden geçirebilir; tamsayı, tek duyarlıklı gerçek sayı, çift duyarlıklı gerçek sayı, karmaşık sayı ve karakter değerler döndüren ifadeleri geçirmez.

Mantıksal işleçlere örnekler aşağıda veilmiştir:

```
x > 0 .and. y < 0

x1 + x2 > 0 .or. x3 + x4 > 0

sayi1 .eq. 0 .and. sayi2 .eq. 50 .and. sayi3 .eq. 100

a .or. b

.not. x

.not. (a + b == 0 .and. c + d /= 0)

ileti == "Hata!" .neqv. ileti == "İstisna!"

(a .or. b) .eqv. (c .or. d)
```

```
a .neqv. (b .or. c)
```

Karşılaştırma işleçlerinin mantıksal işleçlere göre kullanım önceliği vardır. Mantıksal işleçlerde, önce .and., .or. ve .not. işleçleri, sonra .eqv. ve .neqv. işleçleri kullanılır, parantezler de kullanılabilir.

Yukarıdaki örneklerde olduğu gibi ifadelerin ve işleçlerin aralarına birer boşluk bırakmak program kodlarının okunurluğunu arttırır.

3.3.5. Sözceleri Birleştirme İşleci

Sözceleri birleştirmek için art arda gelen iki eğik çizgi kullanılır.

```
1. Sözce // 2. Sözce
```

Sözceleri birleşitrme işlecinin kullanımıyla ilgili örnekler aşağıda verilmiştir:

```
"Ercan " // "Ersoy"
```

```
"Sayın " // ad // soyad // ", işleminiz gerçekleştirildi."
```

3.4. İfadelerde Fonksiyon Değerlerini Kullanma

İfadelerde fonksiyonların döndürdükleri değerler de kullanılabilir. Fonksiyonlar ilerleyen konularda işlenecektir.

İfadelerde fonksiyon değerini kullanımına örnekler aşağıda verilmiştir:

```
sayi * log(a)
```

```
2.0 * cos(x) * sin(x)
```

```
10 – mod(a, b)
```

Yuarıdaki örneklerde verilen fonksiyonlar hakkında bilgi almak için kitabın "Dâhilî Alt Programlar" bölümüne bakınız.

3.5. Farklı Veri Türünde Olan İfadelerin İşleme Girmesi

Fortran'da, işlemler yapılırken, aynı tür veri yapısında ifadeler kullanıldığında sonuç yine o tür ifadeden olur, farklı tür veri yapısında ifadeler kullanıldığında derleyici hata verir ya da aşağıdaki durumlar gerçekleşir:

- Eğer bir tamsayı ifadeyle bir gerçek sayı ifade işleme girerse sonuç gerçek sayı olur. Tamsayı, gerçek sayıya dönüştürülür.

- Eğer bir tamsayı ifadeyle bir karmaşık sayı ifade işleme girerse sonuç karmaşık sayı olur. Tamsayı, gerçek sayıya dönüştürülerek karmaşık sayı ile işleme alınır.

- Eğer bir gerçek sayı ifadesiyle bir karmaşık sayı ifadesiyle işleme girerse sonuç karmaşık sayı olur.

Fortran'da veri türleri arasındaki dönüştürülme işlemleri şöyledir:

- Bir tamsayının bir gerçek sayıya dönüşümünde sonucun ondalıklı bölümü sıfır olur. Örnek: 4 → 4.0

- Bir gerçek sayının bir tamsayıya dönüşümünde gerçek sayının ondalıklı bölümü atılır. Örnek: 15.5 → 15

- Bir tamsayının ya da bir gerçek sayının bir karmaşık sayıya dönüşümünde sonucun sanal bölümü sıfır olur. Örnek: 2 → (2.0 , 0.0) ya da 2.0 → (2.0 , 0.0)

- Bir karmaşık sayının bir gerçek sayıya dönüşümünde sonuç karmaşık sayının gerçek bölümüdür. Örnek: (4.7 , 14.4) → 4.7

- Bir karmaşık sayının bir tamsayıya dönüşümünde sonuç karmaşık sayının gerçek bölümünün tam bölümüdür. Örnek: (3.5 , 5.4) → 3

3.6. DATA Deyimiyle Değişken Tanımlama

Fortran'da data deyimiyle de değişkenlere takım halinde değerler atanabilir. Bu deyim, yürütülemeyen deyimdir.

```
data Değişken Listesi /Değer Listesi/
```

Yukarıdaki değişken listesinde n'ninci sırada olan değişken yukarıdaki değer listesinde n'ninci değerle eşleştirilir ve değişkenlerin aynı türde olması zorunlu değildir. Çünkü her bir değişkene bağımsız olarak değer atanır.

data deyimiyle değişken tanımlamaya örnekler aşağıda verilmiştir:

```
data x, y, z / -5, 20, 34 /
```

```
data sayi, sayi_ciftduyarlikli, karakter / 3.4, 5.21d-2, "Selam!" /
```

Yukarıdaki örneklerde olduğu gibi değer listesinin başına ve sonuna birer boşluk bırakmak program kodlarının okunurluğunu arttırır.

data deyimiyle bir dizi de tanımlanabilir, ancak böyle tanımlamada bir diziye bir değer atanacaksa değer listesinde diziye atanacak değerin önünde dizinin öge sayısı ve yıldız işareti de olmalıdır. Eğer bir diziye ögeler teker teker atanacaksa o ögeler sırayla değişken listesine yazılır.

data deyiminin diziyle kullanımına örnekler aşağıda verilmiştir:

```
integer, dimension(100) :: dizi_1, dizi_2, dizi_3
data dizi_1, dizi_2, dizi_3 / 100*4, 100*-68, 100*0 /

real, dimension(2) :: a, b, c
data a, b, c / 1., 2., 3., 4., 5., 6. /

character, dimension(5, 5) :: x, y
data x, y / 25*"Erdem", 25*"Ercan" /

integer, dimension(3) :: i
integer :: j, k
data i, j, k / 10, 20, 30, 40, 50 /
```

data deyimiyle döngü kullanarak da bir diziye değer atanabilir.

data (**İfade**, **Değişken** = **Başlangıç**, **Bitiş, Atlama Sayısı**) /**Değer Listesi**/

data deyimiyle döngü kullanarak da bir diziye değer atanabilmesinin örneği aşağıda verilmiştir:

```
data (c(i), i = 2, 10, 2) / 5 * 2 /
```

data deyimiyle isimli değişmezlere değer atanamaz.

3.7. Etiketler

Fortran'da herhangi bir deyim etiketlendirilebilir. Etiketler, programda belirli bir deyimi ifade eden, deyimlerin başına koyulan, 1'den 99999'a kadar olan sayılardır.

`Etiket Deyim`

Sayıların başlarına sıfırlar eklenebilir fakat bu sıfırlar derleyici tarafından dikkate alınmaz. Etiketler en çok beş basamaklı olmalıdır.

Etiket belirtimine örnekler aşağıda veirlmiştir:

```
100 continue

6000 format
```

3.8. Özet

Fortran'da ifadeler, bir değer belirten ögelerdir. Bu ögeler değer, işlem, değişken vb. türlerde olabilir. Deyimler, programın çalışmasına yön verirler. İşleçler, Fortran'da işlem yapılması için kullanılan işaretlerdir. Farklı veri türlerinden olan ifadelerde sonuç uygun bir veri türüne dönüştürülür. data deyimiyle atama işlemi de yapılabilir. Fortran programlama dilinde etiket belirtimi de yapılabilir.

3.9. Sorular

3.1) Fortran programlama dilinde ifadelerin tanımını yapınız.

3.2) Fortran programlama dilinde üç ifade örneği veriniz.

3.3) Fortran programlama dilinde deyimlerin tanımını yapınız.

3.4) Fortran programlama dilinde işleçlerin tanımını yapınız.

3.5) Fortran programlama dilinde atama işleci olarak ne kullanılır?

3.6) Fortran programlama dilinde aritmetik işleçlerin öncelik sıralarını belirtiniz.

3.7) Fortran programlama dilinde karşılaştırma işleçlerini belirtiniz.

3.8) Fortran programlama dilinde mantıksal işleçleri belirtiniz.

3.9) Fortran programlama dilinde sözceleri birleştirme işleci olarak ne kullanılır?

3.10) Fortran programlama dilinde bir tamsayıdan bir ondalıklı sayının çıkarılması işleminden olan sonuç ne olur?

3.11) Fortran programlama dilinde DATA deyimi nerelerde kullanılır?

3.12) Fortran programlama dilinde alt dizgiler nasıl oluşturulur?

3.13) Fortran programlama dilinde etiketler hakkında bilgi veriniz.

4. Bölüm: Standart Girişi ve Standart Çıkışı Kullanma

Bilgisayarlarda bildiğiniz üzere giriş birimleri, çıkış birimleri ve hem giriş birimleri hem çıkış birimleri vardır. Giriş birimleri, klavye, fare gibi birimlerdir. Çıkış birimleri ekran, ses birimleri gibi birimlerdir. Hem giriş birimleri hem çıkış birimleriyse dokunmatik ekran, sabit disk gibi birimlerdir.

Giriş birimine veri girilmesi giriş işlemidir. Örneğin, bir klavyeden bir tuşa basıldığında bir giriş işlemi yapılmış olur. Aynı şekilde çıkış birimine veri çıktılanması bir çıkış işlemidir. Örneğin, ses ünitesinden ses gelmesi bir çıkış işlemidir.

Programalma dilleriyle giriş birimleri ve çıkış birimleri kullanılarak programın kullanıcılar tarafından algılanması sağlanır.

Giriş birimlerinde standart giriş, çıkış birimlerinde standart çıkış kavramları vardır. Bu kavramlar, en sık kullanılan giriş ve çıkışı belirtir. Standart giriş genellikle klavyeden yapılır, standart çıkış ortamı genellikle ekrandır.

4.1. Standart Girişi Kullanma

Fortran'da standart girişten veri alma **read** deyimiyle yapılır. Bu deyim kullanılarak standart girişten veriler alınıp belirtilen değişkenlere yazdırılır. Program çalışırken istemde veriler klavyeden girildikten sonra yeni satır girilerek girilen verilerin belirtilen değişkenlere atanması sağlanır. Veri girilirken, veriyi türüne göre doğru ve geçerli aralıklarda yazmak ve verilerin arasına en az bir boşluk veya bir virgül koymak gerekir. Programın çalışma sırasında yanlış veri girilirse program hata vererek sonlandırılır. Eğer belirtilen değişkenlerin tümüne veri girilmemişse yeni satır yapıldığında bile istem değişkenlerin tümüne veri girilene kadar devam eder.

```
read *, Değişkenler
```

Yukarıdaki kullanımda virgülün solunda olan belirtim verilerin giriliş biçimini belirtir. Bu belirtimde yıldız kullanılarak giriliş biçiminin serbest olacağı belirtilir. İlerleyen sayfalarda yıldız yerine başka belirtimler de kullanılacaktır.

`read(*,*)` **`Değişkenler`**

Yukarıdaki kullanımda parantez içine yazılan ve virgülün solunda olan belirtim, girişin nereden olacağını belirtir. Yukarıdaki kullanımda parantez içine yazılan ve virgülün sağında olan belirtim verilerin giriliş biçimini belitrir. Birinci belirtimde yıldız kullanılarak girişin standart girişten olacağı, ikinci belirtimde yıldız kullanılarak giriliş biçiminin serbest olacağı belirtilir. İlerleyen sayfalarda yıldız yerine başka belirtimler de kullanılacaktır.

Örnekler:

```
read *, secim

read *, ad, soyad, yas, kimlik_numarasi

read(*,*) sayi_1, sayi_2, sayi_3

read(*,*) x, y, z, t
```

Program çalışması sırasında mantıksal doğru değeri **T** harfi, mantıksal yanlış değeri ise **F** harfi olarak geçerlidir. Standart girişe mantıksal bir veri yalnızca T ya da F harfiyle belirtilmelidir.

Program çalışması sırasında gerçek sayı türünde bir değişkene bir tamsayı belitilebilir. Bu durumda tamsayı, gerçek sayı türüne dönüştürülecektir.

4.2. Standart Çıkışı Kullanma

Fortran'da standart çıkışa veri yazma **write** veya **print** deyimleriyle yapılır.

4.2.1. WRITE Deyimiyle Standart Çıkışı Kullanma

Bu deyimle standart çıkışa belirtilen ifadelerin döndürdükleri değerler yazdırılabilir.

```
write(*,*) İfadeler
```

Yukarıdaki kullanımda parantez içine yazılan ve virgülün solunda olan belirtim, çıkışın nereden olacağını belirtir.Yukarıdaki kullanımda parantez içine yazılan ve virgülün sağında olan belirtim ise verilerin yazılış biçimini belirtir. Birinci belirtimde yıldız kullanılarak çıkışın standart çıkışa olacağı, ikinci belirtimde yıldız kullanılarak verilerin yazılış biçiminin serbest olacağı belirtilir. İlerleyen bölümlerde yıldız yerine başka belirtimler de kullanılacaktır.

Bu deyim, belirtilen verileri yazdırdıktan sonra yeni satır yazdırılarak, yeni satıra geçilmesi sağlanır. Yeni satıra birşey yazılmadan atlanması için ifade olarak birşey belirtilmemesi gereklidir.

Örnek program aşağıda verilmiştir:

```
program standart_cikis

implicit none
real :: yaricap = 5.0, pi = 3.14159

write(*,*) "Merhaba Dünya."
write(*,*) 7
write(*,*) 5.4
write(*,*) 3.12e5
```

```
write(*,*) 9.8d-2
write(*,*) (43.6, 2.78)
write(*,*) .true.
write(*,*) "Merhaba Dünya.", 7, 5.4, 3.12e5, 9.8d-2, .true.
write(*,*)
write(*,*) 3 * 5
write(*,*) yaricap ** 2 * pi

end program standart_cikis
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 Merhaba Dünya.
           7
   5.40000010
   312000.000
   9.8000000000000004E-002
            (43.5999985,2.77999997)
 T
 Merhaba Dünya.           7   5.40000010       312000.000
9.8000000000000004E-002 T

          15
   78.5397491
```

4.2.2. PRINT Deyimini Kullanma

print deyimiyle write deyimi gibi standart çıkışa veri yazdırılabilir. Bu deyimi kullanmakla write deyimini kullanmak arasında çıkışın nereden olacağını belirtme farkı dışında bir fark yoktur.

```
print *, İfadeler
```

Yukarıdaki kullanımda virgülün solunda olan belirtim verilerin yazılış biçimini belirtir. Bu belirtimde yıldız kullanılarak verilerin yazılış biçiminin serbest olacağı belirtilir. İlerleyen bölümlerde yıldız yerine başka belirtimler de kullanılacaktır.

Yeni satıra birşey yazılmadan atlanması için ifadelerin ve virgülün belirtilmemesi gereklidir.

4.3. Standart Girişte ve Standart Çıkışta Biçimlendirme

Standart girişte veya standart çıkışta biçimlendirme için standart giriş veya standart çıkış deyimlerinde biçim belirtimleri yapılır. Biçim belirtimi karakter veri türünde olan bir değişmezdir. Biçim belirtiminde başta ve sonda ayraçlar kullanılır.

Biçim belirtiminde **düzenleme tanımlayıcıları** kullanılır. Bir düzenleme tanımlayıcısıyla işlenecek olan verilerin basamak sayısı, veri yazılırken en az kaç rakam kullanılacağı gibi özellikler tanımlanır. Düzenleme tanımlayıcıları veri türüne göre değişir. Bir düzenleme tanımlayıcısıyla birlikte tanımlanan özelliklerle başka bir düzenleme tanımlayıcısıyla birlikte tanımlanan özellikler arasında virgül kullanılır. Düzenleme tanımlayıcıları ve bu tanımlayıcılarla birlikte tanımlanan özellikler aşağıdaki tabloda gösterilmiştir:

Tablo 4.1 - Düzenleme Tanımlayıcıları ve Bu Tanımlayıcılarla Birlikte Tanımlanan Özellikler ve Açıklamaları

Düzenleme Tanımlayıcısı ve Onunla Tanımlanan Özellikler	Açıklama
[Bir Satırda Art Arda Yazılacak Veri Sayısı]i[Ayrılan Karakter Sayısı][.][Rakam Yazılacak Asgari Basamak Sayısı]	Tamsayı verinin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]b[Ayrılan Karakter Sayısı][.][Yazılacak Basamak Sayısı]	İkili tabanda olan sayının ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]o[Ayrılan Karakter Sayısı][.][Yazılacak Basamak Sayısı]	Sekizli tabanda olan sayının ne biçimde olacağını belirtir.

[Bir Satırda Art Arda Yazılacak Veri Sayısı]z[Ayrılan Karakter Sayısı][.][Yazılacak Basamak Sayısı]	On altılı tabanda olan sayının ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]f[Ayrılan Karakter Sayısı].[Ondalıklı Bölümün Basamak Sayısı]	Gerçek sayı türünde olan verinin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]e[Ayrılan Karakter Sayısı].[Ondalıklı Bölümün Basamak Sayısı]	Gerçek sayı türünde olan verinin üstel gösteriminin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]es[Ayrılan Karakter Sayısı].[Ondalıklı Bölümün Basamak Sayısı]	Gerçek sayı türünde olan verinin bilimsel gösteriminin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]l[Ayrılan Karakter Sayısı]	Mantıksal verinin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]a[Ayrılan Karakter Sayısı]	Sözcenin ne biçimde olacağını belirtir.
[Bir Satırda Art Arda Yazılacak Veri Sayısı]g[Ayrılan Karakter Sayısı].[Ondalıklı Bölümün Basamak Sayısı]	Niteliği çok iyi bilinmeyen verinin ne biçimde olacağını belirtir. Gerçek sayı türünde olmayan verilerde ondalıklı bölümün basamak sayısı yok sayılmasına karşın o kısma bir şey yazılmazsa derleyici hata verir.
[Sayı]x	Belirtilen sayıda boşluk belirtir.
sp	Artı işaretinin yazılacağını belirtir.
t[Uzunluk]	Belirtilen özellikte sekme belirtir.
/	Dikey aralık belirtir.

Karmaşık verilerde biçimlendirme, yukarıdaki kayan nokta biçimleme tanımlayıcılarıyla, yani f, e veya es tanımlayıcılarıyla belirtilir.

Bir düzenleme tanımlayıcısında bir satırda art arda yazılacak veri sayısı yazılmasa 1 olarak kabul edilir.

Sayısal bir veri biçimlendirmede ayrılan karakter sayısı yetersiz kalırsa programda veri yerine yıldızlar çıktılanır. Karakter bir veri biçimlendirmede, veri belirtilen uzunluktan fazla olursa sözcenin belirtilen uzunluk kadarı dikkate alınır, uzunluk belirtilmezse sözcenin tümü dikkate alınır.

Basamak sayısına sıfırdan küçük sayıların işaretiyle eğer varsa ondalık ayıracı da dahil edilir.

Biçim belirtiminde belirtimin bir parçası olarak bir sözce de tanımlanabilir.

Örnek program aşağıda verilmiştir:

```
program bicimli_cikis

write(*,"(i3)") 20
write(*,"(spi3)") 20
write(*,"(2i4)") -654, 56, 0, -78, 54
write(*,"(f12.5)") 100.5
write(*,"(e12.5)") 100.5
write(*,"(es12.5)") 100.5
write(*,"(b12.5)") 100
write(*,"(o12.5)") 100
write(*,"(z12.5)") 100
write(*,"(l5)") .true.
write(*,"(a10)") "Deneme", "Deneme_2"
write(*,"(a10,a4)") "Deneme", "Deneme_2"
write(*,"(a10,1x,i3)") "Sonuç =", 6
write(*,"(/,1x,a,/)") "Hoşgeldiniz!"
write(*,"(t5,a)") "Erdem Ersoy"
write(*,"(g4.2)") "Deneme"
write(*,"(g8.3,g5.2)") 34.56, -239
write(*,"(1x,'Erdem',1x,a)") "Ersoy"
write(*,"(1x,""Erdem"",1x,a)") "Ersoy"
```

```
end program bicimli_cikis
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 20
+20
-654  56
   0 -78
  54
   100.50000
 0.10050E+03
 1.00500E+02
     1100100
       00144
       00064
    T
    Deneme
  Deneme_2
    DenemeDene
  Sonuç =   6

 Hoşgeldiniz!

    Erdem Ersoy
Dene
34.6     -239
 Erdem Ersoy
 Erdem Ersoy
```

Girişte f düzenleme tanımlayıcısıyla biçim belirtimi çıkıştakinden farklıdır. f ile biçimli girişte tamsayı sayı türünde olan veriler ve gerçek sayı türünde olan veriler bitişik ve ondalık

ayıraçsız olmadan yazılmalıdır. Böyle yazılan verilerin kaç karakterlik olduğu ve ondalık kısmında kaç karakterin olduğu giriş için belirtilen biçimlendirmeye göre program tarafından kabul edilir ve çıkışta bu sayılar ona göre yazılır. Bunun nedeni delikli kartlar zamanında kağıttan ve zamandan tasarruf etmektir ancak günümüzde bu duruma gerek kalmamıştır. Örnek program:

```
program f_belirtimi_ozel_durum

implicit none
real :: x, y, z

read(*,"(3f8.4)") x, y, z
print "(3f10.4)", x, y, z
print *, x, y, z

end program f_belirtimi_ozel_durum
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
123456781234567812345678
 1234.5677 1234.5677 1234.5677
   1234.56775       1234.56775       1234.56775
```

4.4. Biçimlendirmede FORMAT Deyimini Kullanma

Eğer birçok satırda aynı biçimlendirme belirtimi kullanılıyorsa böyle her satırda aynı belirtimin tek tek yazılması yerine **format** deyimiyle belirtimi yalnızca bir kere yazarak istenilen çıktı elde edilebilir. Format deyimiyle biçim belirtimi programın herhangi bir yerinde olabilir ve format deyimleri her zaman etiketli olmalıdır. Bu deyimle bir biçim belirtimi yapıldığında bir standart giriş ya da çıkış deyiminde o biçim belirtimini kullanmak için biçim belirtimi kısmına o biçim belirtiminin bulunduğu format deyiminin etiketini yazmak gerekir.

Örnek program:

```
program format_deyimi

write(*,300) "1. satır"
write(*,300) "2. satır"
write(*,300) "3. satır"
write(*,300) "4. satır"
write(*,300) "5. satır"
write(*,400)

300 format (1x,a)
400 format (1x,"Ersoy")

end program format_deyimi
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 1. satır
 2. satır
 3. satır
 4. satır
 5. satır
 Ersoy
```

4.5. ADVANCE Değiştirgeni

write deyiminin belirtilen ifadelerin sonuçlarını standart çıktıya yazdırdıktan sonra yeni satır yapıp yapmayacağı write deyiminde **advance** değiştirgeniyle belirtilir.

advance değiştirgeni kullanıldığı zaman biçim belirtimi de yapılmalıdır.

```
write(*,Biçim Belirtimi, advance=Değer)
```

advance argümanı için değerler yalnızca karakter türünde **yes** ya da **no** olabilir. write deyimi, yes değeri kullanıldığında yeni satır yapar, no değeri kullanıldığında yeni satır yapmaz. Örnek program aşağıda verilmiştir:

```
program advance_degistirgeni

implicit none
integer :: x

write(*,"(a,1x)", advance="no") "Bir tamsayı giriniz:"
read(*,*) x

print *, x

end program advance_degistirgeni
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Bir tamsayı giriniz: 130965
      130965
```

4.6. Özet

Bilgisayarlarda giriş birimlerini ve çıkış birimlerini kullanma bütün programlama dillerinde uygulanır. Bu, kullanıcının programı algılaması sağlanır. Fortran programalam dilinde giriş deyimi **read**, çıkış deyimleri **write** ve **print**'tir. Bu deyimleri kullanılmasında biçim belirtimi de yapılabilir. Ayrıca **format** deyimiyle varsayılan biçimler belirtilebilir. write deyimini kullanıldığı zaman çıktıda yeni satır olup olmaması, **advance** değiştirgeniyle belirlenir.

4.7. Sorular

4.1) Fortran programlama dilinde giriş deyimi nedir?

4.2) Fortran programlama dilinde çıkış deyimleri nelerdir?

4.3) Fortran programlama dilinde giriş ve çıkış kullanıldığı zaman mantıksal türü verileri nasıl belirtilir?

4.4) Fortran programlama dilini kullanarak 1000 sayısını ikili tabanda yazdırınız.

4.5) Fortran programalama dilini kullanarak 1000 sayısını sekizli tabanda yazdırınız.

4.6) Fortran programalama dilini kullanarak 1000 sayısını on altılı tabanda yazdırınız.

4.7) Fortran programalama dilini kullanarak 123.5 sayısını artı işaretiyle on tabanında yazdırınız.

4.8) Fortran programalama dilini kullanarak 7.4 üzeri 10 sayısını yazdırınız.

4.9) Fortran programalama dilini kullanarak "Fortran" sözcüğünü solda iki boşluk olacak şekilde yazdırınız.

4.10) Fortran programalama dilinde biçimli yazdırmada sekme için hangi işaret kullanılır? Fortran programalama dilinde biçimli yazdırmada sekme için kullanılan işaretin nasıl kullanıldığını da yazınız.

4.11) Fortran programalama dilinde format deyimi ne için kullanılır?

4.12) Fortran programalama dilinde WRITE deyimi kullanıldığı zaman yeni satır çıktılanmaması için ne kullanılır? Bu kullanıma bir örnek veriniz.

5. Bölüm: Karşılaştırma Yapıları

Karşılaştırma yapıları, mantıksal ifadeler kullanılarak bu ifadelerin sonucuna göre deyimler çalıştırmayı sağlayan yapılardır. Mantıksal ifadeler değer döndürür. Karşılaştırma ifadeleri, bu değerlere göre dallanma işlemlerini gerçekleştirirler. Dallanma işlemleri, programın içindeki komutların çalışma sırasını değiştirir.

Karşılaştırma yapıları, programların en önemli parçalarındandır. Bütün programlama dillerinde karşılaştırmalar mevcuttur.

Fortran dilinde karşılaştırma yapıları üçe ayrılır. Bunlar **if** yapısı, **case** yapısı ve **where** yapısıdır.

5.1. IF Yapısı

if yapısı Fortran'da çok sık kullanılan yapıdır. if deyimi, belirtilen mantıksal ifadenin doğru olup olmadığını, yani koşulun sağlanıp sağlanmadığını denetler. Bu ifade doğruysa, yani koşul sağlanmışsa belirli deyimleri çalıştırır; ifade yanlışsa, yani koşul sağlanmamışsa bir şey çalıştırmaz. Koşul yanlışsa belirli bir kod parçasının çalışması için **else** deyimi de kullanılmalıdır.

if ve else deyimlerinin kullanımları aşağıda verilmiştir:

```
if (Koşul) Koşul Sağlanıyorsa Çalıştırılacak Deyim
```

```
if (Koşul) then
   Koşul Sağlanıyorsa Çalıştırılacak Deyim ya da Deyimler
end if
```

```
if (Koşul) then
   Koşul Sağlanıyorsa Çalıştırılacak Deyim ya da Deyimler
```

```
else
   Koşul Sağlanmıyorsa Çalıştırılacak Deyim ya da Deyimler
end if

if (1. Koşul) then
   1. Koşul Sağlanıyorsa Çalıştırılacak Deyimler
else if (2. Koşul) then
1. Koşul Sağlanmıyorsa ve 2. Koşul Sağlanıyorsa Çalıştırılacak
Deyimler
.
.
.
else
   Hiçbir Koşul Sağlanmıyorsa Çalıştırılacak Deyimler
end if
```

En baştaki hariç yukarıdaki gösterimler birer öbek ifade eder. Bunlar gibi belli bir amaç için belirli kurallar altında kümelenen deyimlere **blok** denir. Yukarıdaki bloklar if blokudur.

Aşağıda if ve else kullanımına ilişkin örnekler verilmiştir:

```
if (x < 5.) write(*,*) "Değer düşüktür."

if (x >= 0 .or. x <= 100) write(*,*) "Notunuz: ", x

if (basari_durumu) write(*,*) "Başarılı."

if (x > 0) then
   y = x ** 2
end if

if (not >= 0 .or. not <= 100) then
```

```
   write(*,"(a,i3)") "Notunuz: ", not
end if

if (a + b > c .and. a + c > b .and. b + c > a) then
   write(*,"(a,f6.3)") "Üçgenin çevresi = ", a + b + c
   write(*,"(a)") "Üçgenin çevresi hesaplandı."
end if

if (durum) then
   write (*,*) "Başarılı."
else
   write (*,*) "Başarısız."
end if

if (sure /= 0) then
   sure = sure - 1
else
   write(*,*) "Süre bitti."
end if

if (oran >= 0.5) then
   write(*,*) "Verim yüksek."
else if (oran >= 0.25) then
   write(*,*) "Verim biraz düşük"
else if (oran >= 0.1) then
   write(*,*) "Verim düşük."
else
   write(*,*) "Verim çok düşük."
end if

if (sayi == 100) then
   write(*,*) "Mükemmel."
```

```
else if (sayi >= 50) then
   write(*,*) "İyi."
else
   write(*,*) "Kötü."
end if
```

İç içe if yapıları belirtilebilir. Aşağıda, bu duruma bir örnek verilmiştir:

```
if (a == 10) then
   if (b == 10) then
     print *, "a'nın ve b'nin değeri 10'dur."
   end if
end if
```

Örneğin, girilen öğrenci notuna göre 100 üzerinden 60 ve yukarısı olan notları geçer not, 60'tan daha düşük notları geçmez not olarak belirleyen bir program aşağıda verilmiştir:

```
program ogrenci

integer :: ogrenci_notu = 0

write (*,"(a,1x)", advance="no") "Öğrenci Notunu Giriniz:"
read (*,*) ogrenci_notu

if ( ogrenci_notu >= 60) then
   write (*,*) "Geçer not."
else
   write (*,*) "Geçmez not."
end if

end program
```

Başka bir örnek olarak kenarları girilen üçgenin ne tür bir üçgen olduğunu hesaplayan program aşağıda verilmiştir:

```
program ucgen

integer :: a = 0
integer :: b = 0
integer :: c = 0

write (*,"(a,1x)", advance="no") "Birinci Kenar:"
read (*,*) a

write (*,"(a,1x)", advance="no") "İkinci Kenar:"
read (*,*) b

write (*,"(a,1x)", advance="no") "Üçüncü Kenar:"
read (*,*) c

if (a == b .and. a == c) then
   write (*,*) "Eşkenar üçgen."
else if ((a == b .and. a /= c) .or. (a == c .and. a /= b) .or. (b ==
c .and. a /= b)) then
   write (*,*) "İkizkenar üçgen."
else
   write (*,*) "Çeşitkenar üçgen."
end if

end program ucgen
```

Bir başka örnekse kullanıcı tarafından girilen üç kenar uzunluğunun bir üçgenin kenarları olabilip olamayacağını bulan bir program verilmiştir:

```
program ucgen_cizilebilirligi

   implicit none
   real :: a, b, c

   write(*,"(a)", advance="no") "Birinci kenar uzunluğunu giriniz: "
   read(*,*) a
   write(*,"(a)", advance="no") "İkinci kenar uzunluğunu giriniz: "
   read(*,*) b
   write(*,"(a)", advance="no") "Üçüncü kenar uzunluğunu giriniz: "
   read(*,*) c

   if (a + b > c .and. a + c > b .and. b + c > a) then
      write(*,"(a)") "Girilen kenar uzunluklarıyla bir üçgen
çizilebilir."
   else
      write(*,"(a)") "Girilen kenar uzunluklarıyla bir üçgen
çizilemez."
   end if

end program ucgen_cizilebilirligi
```

5.2. CASE Yapısı

case yapısı, bir değişkenin aldığı değere göre deyimler çalıştıran yapılardır. select case yapısının yaptığı işi aslında if yapısı de yapabilir ancak if yapısı yerine select case yapısı kullanmak programın çalışma hızını arttırabilir, bu durumda biraz daha çok bellek harcanır.

case yapısının kullanımı aşağıda belirtilmiştir:

```
select case (Değişken)
```

```
   case (Değer)
      Değişkenin Değeri Bu Değerdeyse Çalıştırılacak Deyim ya da Deyimler
   case (Başka Bir Değer)
      Değişkenin Değeri Bu Değerdeyse Çalıştırılacak Deyim ya da Deyimler
   .
   .
   .
   case default
      Değişkenin Değeri Bu Yapıda Belirtilen Hiçbir Değerde Değilse Çalıştırılacak Deyim ya da Deyimler
end select
```

case yapısına girecek olan değişken yalnızca tamsayı, karakter ya da mantıksal olmalıdır. Bu değişken eğer karakter türündeyse yapıda belirtilecek değerler tek karakterli olmalıdır.

case yapısında case default kısmının kullanılması zorunlu değildir.

case yapısında bir case deyiminde birden fazla değer belirtilebilir, böyle durumda bu deyimde belirtilen değerler virgülle ayrılır.

case yapısında bir case deyiminde bir değer aralığı belirtilebilir, böyle durumda deyimdeki tek bir değerin yerine aralıktaki en küçük değerle en büyük değer yazılır, bu değerlerin arasına iki nokta koyulur. İki noktanın solundaki değer aralıktaki en küçük değer, sağındaki değer aralıktaki en büyük değerdir. Eğer en küçük değer belirtilmezse en büyük değere eşit ve en büyük değerden küçük olan değerler, en büyük değer belirtilmezse en küçük değere eşit ve en küçük değerden büyük olan değerler belirtilir. Değerin küçüklüğü ya da değerin büyüklüğü, tamsayı değerlerde sayının büyüklüğüne, karakter değerlerde işlemcinin karakter sıralamasına göredir.

Örnekler aşağıda verilmiştir:

```
select case (secenek)
   case (1)
      write(*,*) "Birinci seçenek seçildi."
   case (2)
      write(*,*) "İkinci seçenek seçildi."
   case (3)
      write(*,*) "Üçüncü seçenek seçildi."
   case (4)
      write(*,*) "Dördüncü seçenek seçildi."
   case default
      write(*,*) "Belirtilmeyen bir seçenek değeri girildi."
end select

select case (ay_numarasi)
   case (1,3,5,7,8,10,12)
      ayda_kac_gun_var = 31
   case (2)
      ayda_kac_gun_var = 29
   case (4,6,9,11)
      ayda_kac_gun_var = 30
end select

select case (sayi)
   case (:-1)
      print *, "Sayı 0'dan küçüktür."
   case (0)
      print *, "Sayı 0'dır."
   case (1:)
      print *, "Sayı 0'dan büyüktür."
end select
```

Örnek program olarak bir sınav notu hesaplama programı verilmiştir. Bu not hesaplama programında 0 ile 100 arasındaki not aralıklarıyla bu aralıklara verilen nitelendirmeler aşağıdaki gibidir:

85 – 100: Pekiyi
70 – 84: İyi
55 – 69: Orta
45 – 54: Geçer
25 – 44: Geçmez
0 – 24: Kötü

Program kodları aşağıda verilmiştir:

```
program not_hesaplama

integer(kind=1) :: sinav_notu
character(7) :: durum

write (*,"(a,1x)", advance="no") "Öğrenci Notunu Giriniz:"
read (*,*) sinav_notu

select case (sinav_notu)
   case (85 : 100)
      durum = "Pekiyi"
   case (70 : 84)
      durum = "İyi"
   case (55 : 69)
      durum = "Orta"
   case (45: 54)
      durum = "Geçer"
   case (25 : 44)
      durum = "Geçmez"
```

```
   case (0 : 24)
      durum = "Kötü"
end select

write (*,"(a,1x,a)", advance="no") "Not:", durum

end program not_hesaplama
```

5.3. WHERE Yapısı

where yapısı, belirtilen bir mantıksal ifadeye göre bir dizinin belirli ögelerine erişim sağlayan deyimlerdir. where deyimi where bloku içinde kullanılır.

```
where(Koşul)
   Koşulu Karşılayan Ögelere Atama Deyimi
end where

where(Koşul)
   Koşulu Karşılayan Ögelere Atama Deyimi
else where
   Koşulu Karşılamayan Ögelere Atama Deyimi
end where
```

Örnekler:

```
where(a >= 0 .and. a <= 100)
   a = 0
end where

where(a .eqv. .true.)
   a = .false.
else where
```

```
   a = .true.
end where
```

5.4. Karşılaştırma Yapılarını Adlandırma

Fortran'da if, select case ve where yapıları adlandırılabilir. Bu adlar tanımlayıcılardır. Etiketlendirme, end if gibi end ile biten yapıların ayırt edilmesini sağlar.

```
Etiket: if (Koşul) then
   ...
end if Etiket

Etiket: select case (Değişken)
   ...
end select Etiket

Etiket: where (Koşul)
   ...
end where Etiket
```

Örnekler:

```
degerlendirme: if (sayi == 100) then
   write(*,*) "Mükemmel."
else if (sayi >= 50) then
   write(*,*) "İyi."
else
   write(*,*) "Kötü."
end if degerlendirme

menu_secim: select case (secenek)
   case (1)
      write(*,*) "Birinci seçenek seçildi."
   case (2)
      write(*,*) "İkinci seçenek seçildi."
```

```
   case (3)
      write(*,*) "Üçüncü seçenek seçildi."
   case (4)
      write(*,*) "Dördüncü seçenek seçildi."
   case default
      write(*,*) "Belirtilmeyen bir seçenek değeri girildi."
end select menu_secim

degisim: where(a .eqv. .true.)
   a = .false.
elsewhere
   a = .true.
end where degisim
```

5.5. Özet

Bir programlama dilinde en önemli yapılardan biri karşılaştırma yapılarıdır. Fortran'da **if yapısı**, **case yapısı** ve **where yapısı** vardır. if yapısı karşılaştırma sonucuna göre dallanma sağlayan yapıdır. case yapısı, if yapısından farklı olarak birçok değeri ana belleğe yükler. Bu, çalışma zamanını arttırır ancak daha çok ana bellek sığası harcanır. where yapısı, belirtilen bir mantıksal ifadeye göre bir dizinin belirli ögelerine erişim sağlar. Bu üç yapı etiketlendirilebilir.

5.6. Sorular

5.1) Fortran'da kullanılan karşılaştırma yapılarını yazınız ve bunları açıklayınız.

5.2) Fortran'da karşılaştırma yapıları neden kullanılır?

5.3) Fortran programlama dilinde if yapısı nasıl kullanılır?

5.4) Fortran programlama dilinde case yapısı nasıl kullanılır?

5.5) Fortran programlama dilinde where yapısı nasıl kullanılır?

5.6) Fortran programlama dilinde neden case yapısı if yapısından daha hızlı olarak çalışır?

5.7) Fortan programlama dilinde case yapılarında hiçbir koşul sağlanmadığı takdirde dallanılan kod öbeği hangi deyimle sağlanır?

5.8) Fortran programalama dilinde karşılaştırma yapılarında etiketlendirme nasıl yapılır?

5.9) Fortran programalama dilinde karşılaştırma yapılarında etiketlendirme neden kullanılır?

5.10) Fortran programlama dilinde örnek bir menü sistemini case yapısı kullanarak oluşturunuz.

5.11) Fortran programlama dilinde mantıksal türde olan bir dizinin mantıksal olarak tersini alan programı where yapısı kullanarak oluşturunuz.

5.12) Fortran programlama diliyle toplama, çıkarma, çarpma ve bölme işlemi yapan basit bir hesap makinesi yazınız.

6. Bölüm: Döngü Yapıları

Döngü yapıları, her zaman ya da belirli bir koşula göre sürekli olarak belirli deyimleri çalıştıran yapılardır. Bir döngüde bir kod parçası yeniden çalıştırılacağında o kod parçasının başına dallanılır. Döngü yapıları, programın içindeki komutların çalışma sırasını değiştirir.

Döngü yapıları, karşılaştırma yapıları gibi programların en önemli parçalarıdır. Fortran dilinde iki tür döngü yapısı vardır. Bu yapılar **do** yapısı ve **do while** yapısıdır.

6.1. DO Yapısı

do yapısındaki döngü, belli bir koşula bağlı olmadan sonsuza kadar sürekli belirli deyimleri çalıştırır. Bu yapı **do** deyimiyle başlar, **end do** deyimiyle biter.

```
do
   Sürekli Çalıştırılacak Deyimler
end do
```

Örnek:

```
do
   write (*,*) "Erdem"
   write (*,*) "Ercan"
end do
```

Sonsuza kadar belirli deyimleri çalıştırmak Fortran ile yazılan programlar için kullanışlı değildir.

Döngünün sonsuza kadar çalışmasını engellemek için döngüyü belirli bir koşula bağlayıp döngüden çıkma deyimi olan **exit** belirtilerek döngüden çıkma olanağı sağlanabilir. Sonsuz döngü oluşturmak istemiyorsak döngüyü bir koşula bağlamalıyız. Döngü sonsuza kadar

çalışmayacağına göre döngü içindeki deyimlerin kaç kez çalıştırılacağı da belirtilebilir. Örnek program aşağıda verilmiştir:

```
program sonlu_do_dongusu

   implicit none
   integer :: sayac = 1



   do
      if (sayac == 5) exit
      write(*,*) sayac
      sayac = sayac + 1
   end do

end program sonlu_do_dongusu
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
           1
           2
           3
           4
```

Yukarıda örnek programdaki gibi bir işlev do yapısının kendisinde de vardır, böylece do yapısını yukarıdaki örnek programdaki gibi bir koşula bağlamış oluruz ve bu yapı belirli sayıda çalışır.

```
do Sayaç Değişkeni = İlk Değer, Son Değer, Arttırım Değeri
   Program Parçası
end do
```

Sayaç değişkeni tamsayı türünden olmak zorundadır. Gerçek sayı ve çift duyarlıklı gerçek sayı sayaç değişkenlerinin kullanılması Fortran 90 ile birlikte terk edilmiş, Fortran 95 ile birlikte bu tamamen kaldırılmıştır.

Arttırım değeri negatif tamsayı olabilir. Arttırım değeri belirtilmezse arttırım değeri bir olarak kabul edilir.

Örnek program aşağıda verilmiştir:

```
program sayacli_do_yapisi

   implicit none
   integer :: i

   do i = 1, 10, 2
      write(*,*) i
   end do

      write(*,*)

   do i = 10, 1, -2
      write(*,*) i
   end do

         write(*,*)

   do i = 1, 10
      write(*,*) i
   end do

end program sayacli_do_yapisi
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 1
 3
 5
 7
 9

10
 8
 6
 4
 2

 1
 2
 3
 4
 5
 6
 7
 8
 9
10
```

Döngü çalışırken döngünün sonuna gelmeden döngünün bir sonraki yinelemesine geçilebilir. Bu, **cycle** deyimiyle sağlanır. Örnek program aşağıda verilmiştir:

```
program do_yapisi_cycle

   implicit none
   integer :: i
```

```
   do i = 1, 10
      if (i == 5) cycle
      write(*,*) i
   end do

end program do_yapisi_cycle
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
           1
           2
           3
           4
           6
           7
           8
           9
          10
```

Tek boyutlu bir dizinin ögelerinin döngü kullanılarak yazdırılması sağlanabilir. Aşağıda bulunan birinci örnekle ikinci örnek hemen hemen aynı işlevi görür:

```
program diziyi_donguyle_yazdirma

   implicit none
   integer :: i
   integer, dimension(10) :: sayilar

   do i = 1, 10, 2
      sayilar(i) = i
      write(*,*) sayilar(i)
```

```
   end do

end program diziyi_donguyle_yazdirma

program diziyi_donguyle_yazdirma_2

   implicit none
   integer :: i
   integer, dimension(10) :: sayilar = (/ (i, i = 1, 10, 1) /)

   write(*,*) (sayilar(i), i = 1, 10, 2)

end program diziyi_donguyle_yazdirma_2
```

Tek boyutlu bir diziye değerlerin döngü kullanılarak tek satırda atanması kitabın 2. Bölüm'ünde "Diziler" konusunda vardır.

Bir başka örnek olarak kullanıcı tarafından belirtilen sayının pozitif çarpanlarını yazdıran bir program aşağıda verilmiştir:

```
program pozitif_carpanlar

   implicit none
   integer :: sayi, carpan

   write(*,"(a)", advance="no") "Sayıyı giriniz: "
   read(*,*) sayi

   do carpan = 1, sayi / 2

      if (mod(sayi, carpan) == 0) write (*,*) sayi / carpan
```

```
   end do

   write(*,*) 1

end program pozitif_carpanlar
```

6.2. DO WHILE Yapısı

Yalnızca belirli bir koşul sağlandığında döngünün çalıştırılması için do while döngüsü kullanılır. do while döngüsünde her bir döngü yinelenmesinden önce koşul sınanır. Koşul yanlış olduğunda döngü biter.

```
do while (Koşul)
   Koşul Sağlandığında Çalıştırılacak Deyim ya da Deyimler
end do
```

Örnek program:

```
program do_while_yapisi

   implicit none
   integer :: a, b

   write(*,"(a)", advance="no") "Küçük sayıyı giriniz: "
   read(*,*) a

   write(*,"(a)", advance="no") "Büyük sayıyı giriniz: "
   read(*,*) b

   do while (a <= b)
      write(*,*) a
      a = a + 1
```

```
   end do

end program do_while_yapisi
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Küçük sayıyı giriniz: 30
Büyük sayıyı giriniz: 45
          30
          31
          32
          33
          34
          35
          36
          37
          38
          39
          40
          41
          42
          43
          44
          45
```

İç içe do veya do while yapıları belirtilebilir. Örneğin, bir matrisin satırlarının ve sütunlarının yazdırılmasında iç içe döngü yapıları belirtilebilir. İç içe döngü belirtimi için örnek program aşağıda verilmiştir:

```
program ic_ice_do_yapilari

   implicit none
```

```
   integer :: matris(3, 3), i, j

   do i = 1, 3
      do j = 1, 3
         matris(i, j) = i + j - 1
      end do
   end do

   do i = 1, 3
      do j = 1, 3
         write(*,"(i2)", advance="no") matris(i, j)
      end do
      write(*,*)
   end do

end program ic_ice_do_yapilari
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 1 2 3
 2 3 4
 3 4 5
```

Bir başka örnekse ilk hızı sıfır olan bir cismin azami düşme hızı kavramının göz ardı edildiği ortamdaki serbest düşmesini hesaplayan bir programdır.

```
program serbest_dusme

   implicit none
   real :: yukseklik, konum, yercekimi_ivmesi, zaman = 0.0

   write(*,"(a)", advance="no") "Yüksekliği giriniz (m): "
```

```
   read(*,*) yukseklik
   konum = yukseklik

   write(*,"(a)", advance="no") "Yerçekimi ivmesini giriniz (m/s**2):
"
   read(*,*) yercekimi_ivmesi

   write(*,*)

   do while (konum <= yukseklik)

      konum = 0.5 * yercekimi_ivmesi * zaman ** 2.0

      if (konum > yukseklik) exit

      write(*,"(f6.1,a)") zaman, " saniye sonra:"
      write(*,"(a,f9.2)") "Hız (m/s): ", yercekimi_ivmesi * zaman
      write(*,"(a,f9.2)") "Konum (m): ", yukseklik - konum
      write(*,*)

      zaman = zaman + 0.1

   end do

end program serbest_dusme
```

6.3. Döngü Yapılarını Adlandırma

Döngü yapıları, karşılaştırma yapılarında olduğu gibi adlandırılabilir. Etiket kullanmanın amacı, iç içe olan döngülerde end do deyimlerinin hangi döngü olduğunun bilinmesinin kolay olmasını sağlamaktır.

```
Ad: do
   ...
end do Ad

Ad: do while (Koşul)
   ...
end do Ad
```

6.4. Özet

Bir programda döngü yapıları programın en önemli yapılarındandır. Fortran'da **do yapısı** ve **do while yapısı** olmak üzere iki tür döngü yapısı kullanılır. do yapısı, bir sayaç kullanılmadığında sonsuz, sayaç kullanıldığında sayaca göre çevirim yapan bir yapıdır. Sonsuz döngüler, bir koşula bağlı çıkış deyimi belirtilmedikçe Fortran dilinde yazılan programlarda kullanışsızdır. do while yapısında koşula göre döngü işlemi yapılır. Fortran'da iç içe döngü yapıları belirtilebilir. Fortran'da döngüler, etiketli olarak da belirtilebilir.

6.5. Sorular

6.1) Fortran programlama dilinde kullanılan döngü yapılarını, yazınız ve açıklayınız.

6.2) Fortran programlama dilinde döngü yapıları neden kullanılır?

6.3) Fortran programlama dilinde do yapısının kullanımlarını açıklayınız.

6.4) Fortran programlama dilinde do while yapısının kullanımlarını açıklayınız.

6.5) Fortran programalama dilinde döngü yapılarında etiketlendirme nasıl yapılır?

6.6) Fortran programalama dilinde döngü yapılarında etiketlendirme neden kullanılır?

6.7) Girilen sayının asal sayı olup olmadığını hesaplayan bir programı Fortran programlama dilinde yazınız.

6.8) 2'den 100000'e kadar olan asal sayıları hesaplayan bir programı Fortran programlama dilinde yazınız.

6.9) Girilen sayının mükemmel sayı olup olmadığını hesaplayan bir programı Fortran programlama dilinde yazınız.

6.10) 2'den 100000'e kadar olan mükemmel sayıları hesaplayan bir programı Fortran programlama dilinde yazınız.

6.11) Girilen sayının palindormik sayı olup olmadığını hesaplayan bir programı Fortran programlama dilinde yazınız.

6.12) 2'den 100000'e kadar olan palindormik sayıları hesaplayan bir programı Fortran programlama dilinde yazınız.

6.13) Kullanıcı tarafından belirtilen sayının asal çarpanlarını yazdıran bir programı Fortran dilinde yazınız.

6.14) Kullanıcı tarafından belirtilen iki sayının dost sayılar olup olmadığını yazdıran bir programı Fortran dilinde yazınız.

6.15) Kullanıcı tarafından belirlenen sayıda olmak üzere kullanıcı tarafından girilen sayıların harmonik ortalamasını hesaplayan bir programı Fortran dilinde yazınız.

7. Bölüm: Dosya İşlemleri

Dosyalar, bir saklama arabirimindeki verilerin tutulduğu yapılardır. Dosyalarla bir saklama birimindeki veriler daha iyi yönetilir.

Dosyalar, **dizin** denilen ayırıcı ögelerle sınıflandırılabilirler.

Bir dosya içinde bulunan alt birimlere **kayıt** denilir.

Bir kayıt içinde bulunan birimlere **alan** denilir.

Dosyalara erişim eylemi iki türde tanımlanmıştır. Bunlar **sıralı erişim** ve **doğrudan erişim**dir. Sıralı erişimde, bir kayıtla işlem yapmak için önceki kayıtların taranması gerekir. Doğrudan erişimde istenilen kayıta istenilen sırada erişilebilir.

Birçok programlama dilinde olduğu gibi Fortran'da da dosya işlemleri vardır. Fortran'da bir dosya oluşturmak veya var olan bir dosyayı kullanmak için o dosya tanıtılmalıdır. Dosya tanıtımında dosyanın ismi ve bir numara belirtilir. Dosyalar, standart giriş ve standart çıkıştaki gibi okunur ve yazılır. Fortran standartlarında dizin işlemleriyle ilgili bir şey tanımlanmamıştır.

Fortran'da iki tür dosya tanımlanmıştır. Bu türler **dış dosyalar** ve **iç dosyalar**dır. Dış dosya, gerçek dosya ya da bir giriş ve çıkış aygıtıdır. İç dosya, ana bellekteki bir karakter değişkenidir.

7.1. Dosya Açılması

Fortran'da, dosya açma, yani dosya tanıtma **open** deyimi ile yapılır. Bu deyimle dosyanın, özellikleri, ne amaçla kullanılacağı gibi belirteçler belirtilir. Bu deyimle yanlızca dış dosya belirtilir.

```
open(Özellikler)
```

Bu deyimde, yalnızca "unit" belirtecinin belirtimiyle "action" belirteci "scratch" olarak belirtilmediğinde "file" belirtecinin belirtimi zorunludur. open deyiminin belirteçleri ve bunların açıklamaları aşağıdaki tabloda verilmiştir:

Tablo 7.1. open Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Dosya numarası. Numara, 10'dan 99'a kadar olan bir tamsayı olabilir. UNIX benzeri sistemlerde dosya numarası olarak 5 ise standart giriş dosyasını, 6 ise standart çıkış dosyasını, 2 ise standart hata dosyasını ifade eder. Numara, programda daha önceden tanımlanmamış bir dosyanın numarası olmalıdır.
`file`	Dosya adı. Karakter türünde değer girilmelidir. Dosya adı, dosya sisteminin ve işletim sisteminin kısıtlamalarına uymalıdır.
`status`	Eğer dosya, önceden oluşturulmuşsa **"old"** değeri, yeni oluşturulacaksa **"new"** değeri, kendisinin önceden oluşturulmuşu silinip yenisi oluşturulacaksa **"replace"** değeri belirtilir. Belirtilmezse "old" değeri varsayılır.
`action`	Dosya, yalnızca okuma içinse **"read"** değeri, yalnızca yazma içinse **"write"** değeri, hem okuma hem yazma içinse **"readwrite"** değeri, kendisinin önceden var olup olmadığı bilinmiyorsa **"unkown"** değeri, yalnızca programın çalıştığı sırada olacaksa **"scratch"** değeri belirtilir. Belirtilmezse "unknown" değeri varsayılır. "unkown" değeri belirtildiğinde dosya, önceden yoksa yenisi oluşturulur, önceden varsa var olanı tanımlanır.
`position`	Dosya işlemi, dosyanın başından başlayacaksa **"rewind"** değeri, dosyanın sonundan başlayacaksa **"append"** değeri belirtilir. Belirtilmezse "append" değeri varsayılır.
`iostat`	Değeri tamsayıdır. Kendisine, dosya açma başarılıysa sıfır değeri, dosya açma başarılı değilse sıfır olmayan bir sayı atanır.

`err`	Bir yanlışlık olduğunda programın çalışmasının handi etiketteki deyime atlanacağını belirtir.
`access`	Dosya, eğer sıralıysa **"sequential"**, ikili veri olarak dizinlenmişse ve kendisine erişmek için tüm dosyayı okumak gerekmiyorsa **"direct"**, akış erişimi içinse **"stream"** değeri belirtilir. Belirtilmezse "sequential" değeri varsayılır.
`form`	Dosyaya yazılacak veya dosyadan okunacak veri, biçimliyse **"formatted"**, ikili biçimdeyse **"unformatted"** değeri belirtilir. Ön tanımlı değer, dosyaya erişim kipi "sequential" ise "formatted", "direct" ya da "stream" ise "unformatted"dır.
`recl`	Dosyaya erişim kipi "direct" ise belirtilir. Her bir veri kaydının uzunluğunu belirtir.

Örnekler aşağıda verilmiştir:

```
open(unit=10, file="kayıtlar.dat", status="old", action="read",
position="rewind", iostat=durum)

open(20, file="sinama.txt", status="new", action="write")

open(55, file="hastalar.dat", status="old", action="readwrite",
position="append")
```

7.2. Dosya Durumu Sorgulama

Dosyanın durumu istenilen yerde sorgulanabilir. Bunun için **inquire** deyimi kullanılır. inquire deyiminde yanlızca dış dosya belirtilir.

```
inquire(Özellikler)
```

Bu deyimde "file" belirtecinin ya da "unit" belirtecinin belirtilmesi ve bunların dışında en az bir başka belirtecin daha belirtilmesi zorunludur. inquire deyiminin belirteçleri ve bunların açıklamaları aşağıdaki tabloda verilmiştir:

Tablo 7.2. inquire Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Dosya numarası. Numara, 10'dan 99'a kadar olan bir tamsayı olabilir. UNIX benzeri sistemlerde dosya numarası olarak 5 ise standart giriş dosyasını, 6 ise standart çıkış dosyasını, 2 ise standart hata dosyasını ifade eder.
`file`	Dosya adı. Karakter türünde değer girilmelidir.
`iostat`	Değeri tamsayıdır. Sorgulama, başarılıysa sıfır değeri, başarılı değilse sıfır olmayan bir sayı atanır.
`exist`	Belirtilen dosyanın olup olmadığı belirtilen mantıksal değişkene atanır.
`opened`	Belirtilen dosyanın açılıp açılmadığı belirtilen mantıksal değişkene atanır.
`number`	Belirtilen dosyanın numarası belirtilen tamsayı değişkene atanır. Eğer dosya tanımlanmamışsa -1 değeri atanır.
`named`	Belirtilen dosyanın bir adı olup olmadığı belirtilen mantıksal değişkene atanır.
`name`	Belirtilen dosyanın adı belirtilen karakter değişkene atanır.
`read`	Belirtilen dosyanın okunabilir olup olmadığı belirtilen karakter değişkene atanır. Dosya okunabilirse **"yes"**, dosya okunabilir değilse **"no"**, dosyanın okunabilir olup olmadığı saptanamamışsa **"unknown"** değeri atanır.
`write`	Belirtilen dosyanın yazılabilir olup olmadığı belirtilen karakter değişkene atanır. Dosya yazılabilirse **"yes"**, dosya yazılabilir değilse **"no"**, dosyanın yazılabilir olup olmadığı saptanamamışsa **"unknown"** değeri atanır.
`readwrite`	Belirtilen dosyanın okunabilir ve yazılabilir olup olmadığı belirtilen karakter değişkene atanır. Dosya okunabilir ve yazılabilirse **"yes"**, dosya okunabilir ve yazılabilir değilse **"no"**, dosyanın okunabilir ve yazılabilir olup olmadığı saptanamamışsa **"unknown"** değeri atanır.

Örnekler aşağıda verilmiştir:

```
inquire(unit=11, iostat=durum)

inquire(file="kayitlar.txt", iostat=islem_durumu)

inquire(45, iostat=islem_durumu, opened=dosya_acik_mi,
readwrite=dosya_okuma_ve_yazma)
```

7.3. Dosya Kapatılması

Fortran'da, açılan dosyalar program sonlandığında otomatik kapatılır. Bazı programlarda dosyayı programı sonlandırmadan kapatmak gerekebilir. Bunun için **close** deyimi kullanılır. close deyiminde yanlızca dış dosya belirtilir.

```
close(Özellikler)
```

Bu deyimde yalnızca dosya numarasının belirtilmesi zorunludur. Bu deyimin belirteçleri aşağıdaki tabloda verilmiştir:

Tablo 7.3. close Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Dosya numarası. Numara, programda daha önceden tanımlanmış bir dosyanın numarası olmalıdır.
`iostat`	Değeri tamsayıdır. Kapatma, başarılıysa sıfır değeri, başarılı değilse sıfır olmayan bir sayı atanır.
`err`	Bir yanlışlık olduğunda programın çalışmasının handi etiketteki deyime atlanacağını belirtir.
`status`	Dosya, kapatıldıktan sonra silinmeyecekse **"keep"** değeri, kapatıldıktan sonra silinecekse **"delete"** değeri belirtilir. Belirtilmezse dosya, "scratch" olarak açılmamışsa "keep" değeri varsayılır, "scratch" olarak açılmışsa "delete" değeri varsayılır ve yalnızca bu değere izin verilir.

Bir dosya kapatıldıktan sonra silinmemişse kapatılan dosya için atanan numarayla başka bir dosya açılabilir ve aynı dosya farklı bir numarayla açılabilir.

Örnekler aşağıda verilmiştir:

```
close(unit=15)
```

```
close(20, iostat=islem_durumu)
```

7.4. Dosyadan Veri Okunması

Dosyadan okuma işlemi için standart girdideki gibi **read** deyimi kullanılır. read deyimiyle dış dosyalarla çalışılabileceği gibi iç dosyalarla da çalışılabilir.

```
read(Özellikler) Okunacak Verinin Atanacağı Değişkenler
```

Bu deyimde tüm koşullarda yalnızca "unit" belirtecinin belirtimi zorunludur, okunacak dosya biçimli olarak tanımlanmamışsa "fmt" belirteci belirtilmez. Bu deyimin belirteçleri aşağıdaki tabloda verilmiştir:

Tablo 7.4. read Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Numara, programda daha önceden tanımlanmış ve boş olmayan bir dosyanın numarası olmalıdır. İç dosya olarak bir karakter değişkeni tanımlanabilir.
`fmt`	Okunacak verinin hangi biçimde olacağını belirtir. Düzenleme tanımlayıcıları için 4. Bölüm'e bakınız.
`rec`	Dosyadaki kaçıncı kayıttan başlanacağını belirtir. Tamsayı değer belirtilmelidir. Kayıt numarası **1**'den başlar. Bir kayıt, bir seferde okunan ya da yazılan içeriktir.
`advance`	Okuma işleminin sonunda dosyada bir sonraki satıra geçilip geçilmeyeceği belirtir. Geçilecekse **"yes"**, geçilmeyecekse **"no"** değeri belirtilmelidir. Bu belirteç kullanıldığı zaman biçim belirtimi serbest olmamalı ve dosya biçimsiz olarak tanımlanmış olmamalıdır.
`iostat`	Değeri tamsayıdır. Kendisine, okuma, başarılıysa sıfır değeri, başarılı değilse pozitif bir sayı, dosyanın sonuna gelinmişse negatif bir sayı atanır.
`err`	Bir yanlışlık olduğunda programın çalışmasının handi etiketteki deyime atlanacağını belirtir.
`end`	Dosyanın sonuna gelindiğinde programın çalışmasının hangi etiketteki deyime atlanacağını belirtir.

Örnekler aşağıda verilmiştir:

```
read(unit=20, fmt="(a,3x,i5)") ad, numara

read(21, fmt="(3f8.4)") x, y, z

read(99, "(3f5.2)", rec=5, iostat=durum) a, b, c
```

7.5. Dosyaya Veri Yazılması

Dosyaya yazma işlemi için standart çıktıdaki gibi **write** deyimi kullanılır. write deyimiyle dış dosyalarla çalışılabileceği gibi iç dosyalarla da çalışılabilir.

```
write(Özellikler) İfadeler
```

Bu deyimde, tüm koşullarda yalnızca "unit" belirtecinin belirtimi zorunludur, kendisine yazılacak dosya biçimli olarak tanımlanmamışsa "fmt" belirteci belirtilmez. Bu deyimin belirteçleri aşağıdaki tabloda verilmiştir:

Tablo 7.5. write Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Numara, programda daha önceden tanımlanmış bir dosyanın numarası olmalıdır. İç dosya olarak bir karakter değişkeni tanımlanabilir.
`fmt`	Yazılacak verinin hangi biçimde olacağını belirtir. Düzenleme tanımlayıcıları için 4. Bölüm'e bakınız.
`rec`	Dosyadaki kaçıncı kayıttan başlanacağını belirtir. Tamsayı değer belirtilmelidir. Kayıt numarası **1**'den başlar.
`advance`	Yazma işleminin sonunda dosyaya satır sonu eklenip eklenmeyeceğini belirtir. Eklenecekse **"yes"**, eklenmeyecekse **"no"** değeri belirtilmelidir. Bu belirteç kullanıldığı zaman biçim belirtimi serbest olmamalı ve dosya biçimsiz olarak tanımlanmış olmamalıdır.
`iostat`	Değeri tamsayıdır. Kendisine, yazma, başarılıysa sıfır değeri, başarılı değilse sıfır olmayan bir sayı atanır.
`err`	Bir yanlışlık olduğunda programın çalışmasının hangi etiketteki deyime atlanacağını belirtir.

Örnekler aşağıda verilmiştir:

```
write(unit=10, fmt="(a, ',')", advance="no", iostat=yazma_durumu) ad

write(unit=11, fmt="(a,x3,i5,x2,i5)") sonuc_1, sonuc_2, sonuc_3

write(unit=15, fmt="(f15.5)", rec=2) sonuc
```

7.6. Dosya İşaretçisinin Okuma ya da Yazma İşlemi Yapmadan Dosyanın Başına Götürülmesi

Dosya işaretçisi, dosya verisi hakkındaki konum bilgisidir. Okuma ve yazma işlemleriyle dosya işaretçisinin konumu değişir. Herhangi bir okuma veya yazma işlemi yapmadan dosya işaretçisini, dosyanın başına konumlandırmak için **rewind** deyimi, bulunduğu satırın en başına konumlandırmak için **backspace** deyimi kullanılır. Bu deyimlerle yalnızca dış dosyalarla çalışılabilir.

`rewind(`**`Özellikler`**`)`

rewind deyiminin belirteçleri aşaıda veirlmiştir:

Tablo 7.6. rewind Deyiminin Belirteçleri

Belirteç	Açıklama
`unit`	Dosya numarası. Numara, programda daha önceden tanımlanmış bir dosyanın numarası olmalıdır.
`iostat`	Değeri tamsayıdır. Kendisine, yazma, başarılıysa sıfır değeri, başarılı değilse sıfır olmayan bir sayı atanır.
`err`	Bir yanlışlık olduğunda programın çalışmasının hangi etiketteki deyime atlanacağını belirtir.

`backspace(`**`Özellikler`**`)`

backspace deyiminin belirteçleri aşağıda veirlmiştir:

Tablo 7.7. backspace Deyiminin Belirteçleri

Argüman	Açıklama
`unit`	Dosya numarası. Numara, programda daha önceden tanımlanmış ve boş olmayan bir dosyanın numarası olmalıdır.
`iostat`	Değeri tamsayıdır. Kendisine, yazma, başarılıysa sıfır değeri, başarılı değilse sıfır olmayan bir sayı atanır.
`err`	Bir yanlışlık olduğunda programın çalışmasının hangi etiketteki deyime atlanacağını belirtir.

Örnekler aşağıda verilmiştir:

```
rewind(10)

backspace(21)
```

7.7. Akış Erişimi

Fortran 2003 ile birlikte Fortran'a dosyalara akış erişimi olanağı gelmiştir. Akış erişimi, C standartına yakın biçimsiz erişimdir. Bu erişimle C ikili dosyalarıyla birlikte kolayca çalışılabilir. Akış erişimiyle erişilen dosyalarda bir verinin konum numarası, kendisinden önce gelen verilerin bayt olarak büyüklüklerinin toplamının bir fazlasına eşittir, konum numarasını ayrı olarak belirtmek için **pos** belirteci kullanılır. Böyle erişimde herhangi bir konum numarasında işlem yapıldıktan sonra bir sonraki konuma geçilir, konum numaraları atlanmaz.

Örnek program aşağıda verilmiştir. Bu örnek program sadece akış erişimini anlamak amacıyla yazılmış olup bu örnek programın çıktıları kaynak koduyla birlikte incelenmelidir.

```
program akis_erisimi

   implicit none
   integer :: deger, konum
```

```
   open(11, file="ornek.dat", status="replace", access="stream")

   inquire(11, pos=konum) ! Dosya işaretçisinin konum bilgisi konum
değişkenine atanır.
   write(*, "(a, 1x, i2)") "Bir veri, konumu belirtilmeden
yazılacaksa şu konumda yazılacak:", konum

   deger = 12345 ; write(11, pos=12) deger

   inquire(11, pos=konum)
   write(*, "(a, 1x, i2)") "Başka bir veri, konumu belirtilmeden
yazılacaksa şu konumda yazılacak:", konum

   deger = 678 ; write(11, pos=3) deger ! pos belirteciyle hangi
konumdan itibaren yazma işleminin yapılacağı belirtilir.

   inquire(11, pos=konum)
   write(*, "(a, 1x, i2)") "Başka bir veri, konumu belirtilmeden
yazılacaksa şu konumda yazılacak:", konum

   read(11, pos=12) deger ! pos belirteciyle hangi konumdan itibaren
okuma işleminin yapılacağı belirtilir.
   print *, deger

   read(11, pos=3) deger
   print *, deger

   close(11)

end program akis_erisimi
```

Örnek programdaki "deger" tamsayısı 4 baytlık olarak varsayılırsa bu örnek program derlenip çalıştırıldığında bu örnek programın standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Bir veri, konumu belirtilmeden yazılacaksa şu konumda yazılacak:  1
Başka bir veri, konumu belirtilmeden yazılacaksa şu konumda
yazılacak: 16
Başka bir veri, konumu belirtilmeden yazılacaksa şu konumda
yazılacak:  7
       12345
         678
```

7.8. NAMELIST Deyimi ile Dosya İşlemleri

Fortran'da, büyük programlarda hangi değişkenin hangi değeri aldığını öğrenmek zor olabilir. Bu zorluğu aşmak için değişkenlerin bir listesi çıkarılabilir, bunun için de namelist deyimi kullanılır. Bu deyim, çalıştırılabilir olmayan bir deyimdir.

```
namelist /Liste Adı/ Değişkenler
```

Bu deyimle oluşturulan liste bir dosyaya yazdırılabilir ve bir dosyadan böyle bir liste okunabilir. Bunun için read ve write deyimlerinde **"nml"** belirteci kullanılır, bu belirteçte listenin adı belirtilir.

Örnek program aşağıda verilmiştir:

```
program namelist_deyimi

   implicit none
   real :: a=15.34, b=-437.28, c=91.55, d=2.1987, e=-6.001

   namelist /degiskenler/ a, b, c, d, e
```

```
   open(10, file="liste.dat")

   write(10, nml=degiskenler)

   close(10)

end program namelist_deyimi
```

Bu örnek program derlenip çalıştırıldığında oluşturulan "liste.dat" dosyasının içeriği şöyle olur:

```
&DEGISKENLER
 A=  15.3400002    ,
 B= -437.279999    ,
 C=  91.5500031    ,
 D=  2.19869995    ,
 E= -6.00099993    ,
 /
```

Bir diğer örnek aşağıda verilmiştir:

```
program namelist_deyimi_2

   implicit none
   real :: a, b, c, d, e

   namelist /degiskenler/ a, b, c, d, e

   open(10, file="liste.dat")

   read(10, nml=degiskenler)
```

```
   print *, a, b, c, d, e

   close(10)

end program namelist_deyimi_2
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olur:

```
   15.3400002      -437.279999       91.5500031       2.19869995
-6.00099993
```

namelist ile oluşturulacak listelere, dinamik olmayan diziler, değişkenler, sabitler de dahil olabilir, dinamik diziler ve göstericiler dahil olamaz.

7.9. Örnek

Bu örnekte, girilen hasta kayıtlarını dosyada saklayan ve dosyada saklanan hasta kayıtlarını okuyup gösterebilen bir program verilmiştir. Programın çalışması için çalıştırılma yolunda **hastalar.dat** isimli bir dosyanın olması gerekir.

```
program hasta_kayit_sistemi

implicit none

character(len=1) :: secenek

character(len=20) :: ad
character(len=25) :: soyad
character(len=20) :: babaadi
integer(kind=2) :: yas
character(len=25) :: tani
```

```
integer :: acmadurumu, yazmadurumu, okumadurumu, satir, kayitnumarasi

write(*,"(/, 1x, a, /, 1x, a, /, 1x, a, /, 1x, a)") "Hasta Kayıt
Sistemi"

do

    write(*,"(/, 1x, a, /, 1x, a, /, 1x, a, /, 1x, a, /)") "Menü:", &
    "1) Hasta Gir", "2) Hasta Kaydına Bak", "3) Çıkış"

    write(*,"(1x, a)", advance="no") "Hangi seçeneği seçeceksiniz: "
    read(*,*) secenek

    select case (secenek)

    case ("1")

        write(*,"(/, 1x, a)", advance="no") "Hastanın adı: "
        read(*,*) ad
        write(*,"(1x, a)", advance="no") "Hastanın soyadı: "
        read(*,*) soyad
        write(*,"(1x, a)", advance="no") "Hastanın baba adı: "
        read(*,*) babaadi
        write(*,"(1x, a)", advance="no") "Hastanın yaşı: "
        read(*,*) yas
        write(*,"(1x, a)", advance="no") "Hastanın tanısı: "
        read(*,*) tani

        open(unit=10, file="hastalar.dat", status="old",
action="readwrite", position="append", iostat=acmadurumu)
```

```
        if (acmadurumu > 0) then
            write(*,"(/, a, /)") "Dosya açılamıyor."
            stop
        end if

        write(10, "(a, 1x, a, 1x, a, 1x, i3, 1x, a)",
iostat=yazmadurumu) ad, soyad, babaadi, yas, tani

        if (yazmadurumu > 0) then
            write(*,"(/, a, /)") "Dosyaya yazdırılmaıyor."
            stop
        end if

        rewind(10)

        satir=1

        do

            read(10, "(a, 1x, a, 1x, a, 1x, i3, 1x, a)",
iostat=okumadurumu) ad, soyad, babaadi, yas, tani

            if (okumadurumu > 0) then
                write(*,"(/, a, /)") "Dosyadan okunamıyor."
                stop
            end if

            if (okumadurumu < 0) exit

            satir = satir + 1

        end do
```

```
        write(*,"(/, a, i4, /)") "Kayıt numarası: ", satir - 1

        close(10)

    case ("2")

        write(*,"(/, 1x, a)", advance="no") "Kayıt numarasını
giriniz: "
        read(*,*) kayitnumarasi

        open(unit=11, file="hastalar.dat", status="old",
action="read", position="rewind", iostat=acmadurumu)

        satir = 1

        if (acmadurumu > 0) then
            write(*,"(/, a, /)") "Dosya açılamıyor."
            stop
        end if

        do satir = 1, kayitnumarasi

            read(11, "(a, 1x, a, 1x, a, 1x, i3, 1x, a)",
iostat=okumadurumu) ad, soyad, babaadi, yas, tani

            if (okumadurumu > 0) then
                write(*,"(/, a, /)") "Dosyadan okunamıyor."
                stop
            end if

        end do
```

```
        write(*,"(/, 1x, a, a, /, 1x, a, a, /, 1x, a, a, /, 1x, a,
i3, /, 1x, a, a, /)") &
        "Hastanın adı: ", ad, "Hastanın soyadı: ", soyad, "Hastanın
baba adı: ", &
        babaadi, "Hastanın yaşı: ", yas, "Hastanın tanısı: ", tani

        close (11)

    case ("3")

        exit

    case default

        cycle

    end select

end do

end program hasta_kayit_sistemi
```

7.10. Özet

Dosyalar, bir saklama arabirimindeki verilerin tutulduğu yapılardır. Fortran'da da dosya yönetimi işlemleri tanımlanmıştır. Fortran'da dış dosya ve iç dosya olarak iki tür dosya tanımlanması vardır. Dış dosyalar, gerçek dosya, giriş aygıtı ya da çıkış aygıtı olabilir. İç dosya, karakter türünde bir değişken olabilir. Dosyayı açmak için **open** deyimi kullanılır. Dosyann durumu **inquire** deyimi ile sorgulananır. Dosyayı kapatmak için **close** deyimi kullanılır. Dosyaya veri okunması için **read** deyimi kullanılır. Dosyaya veri yazılması için **write** deyimi kullanılır. Dosyanın işaretçisini dosya başına getirmek için **rewind** deyimi kullanılır.

Dosyanın işaretçisini bulunduğu satırın başına getirmek için **backspace** deyimi kullanılır. Fortran'da dosyalara akış şeklinde de erişilebilir. **namelist** deyiminde read deyimi ve write deyimiyle Fortran'da bir değişkenin hangi deyimi aldığı öğrenilebilir.

7.11. Sorular

7.1) Fortran programlama dilinde kaç tür dosya tanımlanmıştır?

7.2) Fortran programlama dilinde dosya açılması için hangi deyim kullanılır?

7.3) Fortran programlama dilinde dosya kapatılması için hangi deyim kullanılır?

7.4) Fortran programlama dilinde dosya durumu sorgulanması için hangi deyim kullanılır?

7.5) Fortran programlama dilinde dosyaya veri okunması için hangi deyim kullanılır?

7.6) Fortran programlama dilinde dosyaya veri yazılması için hangi deyim kullanılır?

7.7) Fortran programlama dilinde dosya işaretçisini dosyanın başına konumlandırmak için hangi deyim kullanılır?

7.8) Fortran programlama dilinde akış erişimi için open deyiminde hangi belirteçler nasıl kullanılır?

7.9) Fortran programlama dilinde read deyimi ve write deyimi için değişken listesi hangi deyimle çıkarılır?

7.10) Fortran programlama dilinde dosya durumu için hangi belirteç kullanılır?

7.11) 1'den 100000'a kadar olan sayıları "sıra.dat" isimli bir dosyaya yazan bir programı Fortran programlama dilinde yazınız.

7.12) Bir kayıt dosyasının biçimi "Ad-Soyadı-Yaş" olarak belirtilmiştir. Bu dosyadaki 50. kayıtın belirttiği adı, soyadı ve yaşı okuyup standart çıktıya yazdıran programı Fortran programlama dilinde yazınız.

8. Bölüm: Alt Programlar

Alt programlar, programı yinelenen kod satırlarını bir kere yazılacak şekilde gruplayan yapılardır.

Alt programlar, kaynak kodunda bir kez yazılan program parçasının birden çok yerde çalıştırılmasını sağlar, böylece her çalıştırma için aynı program parçasını tekrar tekrar yazma gereği ortadan kalkar. Bundan dolayı programın boyutu daha az olabilir. Ayrıca alt programlarla program yazmak daha kolaydır ve olabilecek hataları en aza indirir.

Fortran'da alt programlar **fonksiyonlar** ve **alt yordamlar** olarak ikiye ayrılır.

8.1. Fonksiyonlar

Fonksiyon, belirli işlemleri yaparak bir sonuç döndüren program parçasıdır. Fonksiyonlar **kullanıcı tanımlı fonksiyonlar** ve **dâhilî fonksiyonlar (kütüphane fonksiyonları)** olarak ikiye ayrılırlar.

8.1.1. Kullanıcı Tanımlı Fonksiyonlar

Kullanıcı tanımlı fonksiyonlar, Fortran'da ön tanımlı olmayan, yani Fortran standardında ve derleyicide ön tanımlı olmayan fonksiyonlardır. Kullanıcı tanımlı fonksiyonları tanımlamak için **function** ve **end function** deyimleri kullanılır, bu fonksiyonlar, program ana biriminin dışında veya program ana biriminde **contains** deyiminden sonra tanımlanabilir.

Fonksiyon tanımlamasında fonksiyonun döndüreceği sonucun veri türü, function ve end function anahtar sözcükleri, fonksiyonun adı, fonksiyonun parametrelerinin adları, **result** deyimi, sonuç değişkeni ve fonksiyonun içindeki kodlar belirtilir. Fonksiyon parametreleri, parantez içinde belirtilir, virgülle ayrılır. Fonksiyon tanımlama şekilleri aşağıda verilmiştir:

```
Sonuç Verisi Türü function Fonksiyonun Adı(Parametreler) result Sonuç Değişkeni
   Parametreleri Bildirme Deyimleri
   ...
   Sonuç Değişkeni = İfade
end function Fonksiyonun Adı
```

```
Sonuç Verisi Türü function Fonksiyonun Adı(Parametreler)
   Parametreleri Bildirme Deyimleri
   ...
   Fonksiyonun Adı = İfade
end function Fonksiyonun Adı
```

Sonuç değişkeni belirtilmeden de fonksiyon tanımlanabilir.

Fonksiyon tanımlamasının sonunda fonksiyon adı veya bunun yanındaki function anahtar sözcüğü yazılmayabilir, ancak bunları yazmak program kodlarının okunurluğunu arttırır.

Fonksiyon parametreleri, fonksiyona aktarılacak verileri belirtir; fonksiyon, parametrelere göre sonuç verir. Parametrelerin bildirimi değişken tanımlamak gibidir, yalnız bu bildirimde **intent** niteleyicisi de kullanılır. Bu niteleyici yanına parantez içinde parametrenin türü belirtilir. Bu türler aşağıda belirtilmiştir:

"in": Parametrenin fonksiyon içinde kullanılacağını ve fonksiyon içinde programdaki değerinin değiştirilemeyeceğini belirtir. Böylelikle bir fonksiyon içinde bildirilen bir parametre adı program kodunda başka bir yerde başka bir değişken için kullanılabilir.

"out": Parametrenin fonksiyon içinde programdaki değerinin değiştirilebileceğini belirtir.

"inout": Parametrenin fonksiyon içinde kullanılacağı ve fonksiyon içinde programdaki değerinin değiştirilebileceğini belirtir.

Örnek program olarak iki sayının ortalamasını alt program kullanarak alan bir program aşağıda verilmiştir:

```
program fonksiyonlar_1

   implicit none
   real :: x, y

   write (*,"(a)", advance="no") "Birinci sayıyı giriniz: "
   read (*,*) x

   write (*,"(a)", advance="no") "İkinci sayıyı giriniz: "
   read (*,*) y

   write (*,"(a,f15.8)") "Bu iki sayının ortalaması: ", ortalama(x,
y)

   contains

   real function ortalama(a, b)

      implicit none
      real, intent(in) :: a, b
      ortalama = (a + b) / 2.0

   end function ortalama

end program fonksiyonlar_1
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Birinci sayıyı giriniz: 39.7
İkinci sayıyı giriniz: 20.8
Bu iki sayının ortalaması:     30.25000000
```

Bu örnek programdaki ortalama(x, y) ifadesi fonksiyonun çağırılmasıdır, bu ifadedeki x ve y 'ye **değiştirgen** denir. Değiştirgenlerle parametrelerin sırayla eşleşmesi gerekir, bir değiştirgenle eşleşen parametrenin değiştirgenle aynı adda olmasına gerek yoktur.

Değişkenlerin kendiliğinden tanımlanmasını önlemek için implicit none deyimi fonksiyon içinde de kullanılabilir.

Ana program biriminin dışındaki fonksiyonların bildiriminde ana program bölümünde bir arayüz ve fonksiyonlardaki parametreler belirtilir. Arayüz belirtimi için **interface** bloku kullanılır.

```
interface Arayüz Adı
   Sonuç Verisi Türü function Fonksiyon Adı(Parametre)
      Parametre Tanımlama Deyimleri
   end function Fonksiyon Adı
end interface Arayüz Adı
```

Arayüz belirtimi, ana program biriminin başında, implicit none deyiminin ardından yapılır. Değişken tanımlamaları arayüz belirtiminden önce yapılabilir.

Arayüz belirtiminin sonunda arayüz adı veya bunun yanındaki interface anahtar sözcüğü belirtilmeyebilir. Ancak, bunları belirtmek program kodunun okunurluğunu arttırır.

Aşağıda, bir önceki programla aynı işlevi gören; ancak, fonksiyonun ana program biriminin dışında tanımlandığı bir örnek program verilmiştir:

```
program fonksiyonlar_2

   implicit none

   interface arayuz
      real function ortalama(a, b)
         real, intent(in) :: a, b
      end function ortalama
   end interface arayuz

   real :: x, y

   write (*,"(a)", advance="no") "Birinci sayıyı giriniz: "
   read (*,*) x

   write (*,"(a)", advance="no") "İkinci sayıyı giriniz: "
   read (*,*) y

   write (*,"(a,f15.8)") "Bu iki sayının ortalaması: ", ortalama(x,
y)

end program fonksiyonlar_2


real function ortalama(a, b)

   implicit none
   real, intent(in) :: a, b
   ortalama = (a + b) / 2.0

end function ortalama
```

Bir fonksiyon içinde fonksiyonun sonuna gelmeden çıkılmak istendiğinde **return** deyimi kullanılır.

```
return
```

Ana program biriminin dışındaki fonksiyon tanımlamaları ayrı bir dosyada olabilir, böyle fonksiyonların arayüz belirtimleri her zaman ana program biriminde yapılmalıdır. Başka bir dosyada böyle bir fonksiyon tanımlaması yapıldığında programı inşa (derleme ve bağlama) için dosyaların ayrı ayrı belirtilmesi gerekir. Ana program dosyasının ornek.f95, ortalama fonksiyonun tanımlandığı dosyanın ortalama.f95 dosyası olduğu varsayıldığında inşa için komut satırına,

Microsoft Windows sistemleri için,

```
gfortran ornek.f95 ortalama.f95 -o ornek.exe
```

Linux sistemleri için,

```
gfortran ornek.f95 ortalama.f95 -o ornek
```

yazılabilir.

Code::Blocks gibi bir tümleşik geliştirme ortamı kullananların bunları çalıştırmasına gerek yoktur, Code::Blocks karmaşık program inşalarını kolayca gerçekleştirir.

Bir fonksiyonun içinde fonksiyonun kendisine çağrı yapılabilir, böylece fonksiyon kendi kendini çağırmış olur. Böyle fonksiyona **yinelemeli fonksiyon** denir. Yinelemeli fonksiyon tanımlamasında, function deyiminden önce **recursive** sözcüğünün yazılması gerekir ve result deyimi ve sonuç değişkeninin bildirilmesi zorunludur.

```
Veri Türü recursive function Fonksiyonun Adı(Parametreler) result
Sonuç Değişkeni
   Parametreleri Bildirme Deyimleri
   ...
   Sonuç Değişkeni = İfade
end function Fonksiyonun Adı
```

Örnek program olarak bir sayının faktöriyelini alan bir program aşağıda verilmiştir:

```
program yinelemeli_fonksiyonlar

   implicit none
   integer :: x

   write(*,"(a)", advance="no") "Sayıyı giriniz: "
   read(*,*) x

   write(*,"(a,i10)") "Sonuç: ", faktoriyel(x)

   contains

   integer recursive function faktoriyel(sayi) result (sonuc)

      implicit none
      integer, intent(in) :: sayi

      if (sayi == 0 .or. sayi == 1) then
         sonuc = 1
      else
         sonuc = sayi * faktoriyel(sayi - 1)
      end if
```

```
   end function  faktoriyel

end program yinelemeli_fonksiyonlar
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Sayıyı giriniz: 9
Sonuç:     362880
```

Çok fonksiyon çağrısı, yığın taşmasına yol açabilir. Bu yüzden yinelemeli fonksiyonlar kullanırken bunu göz önünde bulundurmak gerekir.

Bazen bir fonksiyon çağrısı için tüm parametrelerin girilmesi her zaman zorunlu olmayabilir. Böyle bir fonksiyon çağrısı yapmak için o fonksiyonun daha bildirilirken girilmesi zorunlu olmayan parametrelerin bildiriminde **optional** niteleyicisi kullanılır.

Veri Türü, intent(**Argüman Türü**), optional :: Değişkenler

Böyle fonksiyonlar bildirilirken function deyiminde girilmesi zorunlu olmayan parametreler girilmesi zorunlu olan parametrelerin sağ yanında olmalıdır. Ayrıca böyle fonksiyonlarda herhangi bir parametrenin kullanılıp kullanılmadığını denetlemek için **present()** dâhilî fonksiyonu kullanılır.

Örnek olarak ikiden daha çok sayıyı toplayabilen fonksiyonu içeren bir program aşağıda verilmiştir:

```
program fonksiyonlar_optional

   write(*,*) toplama(14, 5)
   write(*,*) toplama(27, -6, 4)
   write(*,*) toplama(22, -256, 87, 90)
```

```
   write(*,*) toplama(3, 67, 45)
   write(*,*) toplama(32, 100, -54, 0, 44)
   write(*,*) toplama(54, 23, d=39, e=34)
   write(*,*) toplama(-8, -81, e=22, c=41)

contains

   integer function toplama(a, b, c, d, e)

      implicit none
      integer, intent(in) :: a, b
      integer, intent(in), optional :: c, d, e

      toplama = a + b
      if(present(c)) toplama = toplama + c
      if(present(d)) toplama = toplama + d
      if(present(e)) toplama = toplama + e

   end function toplama

end program fonksiyonlar_optional
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
          19
          25
         -57
         115
         122
         150
         -26
```

8.1.2. Dâhilî Fonksiyonlar

Dâhilî fonksiyonlar, Fortran standardında ve derleyicide ön tanımlı olan fonksiyonlardır, bu fonksiyonlar daha önceden tanımlanmış oldukları için kullanıcı tarafından tanımlanmasına gerek yoktur. Fortran'da birçok dâhilî fonksiyon tanımlanmıştır. Dâhilî fonksiyonlarla ilgili daha çok bilgi bu kitabın eklerinde verilmiştir.

8.2. Alt Yordamlar

Alt yordamlar, istenildiğinde çağırılan program parçalarıdır. Fonksiyonlardan ayrı olarak alt programlar belli bir sonuç döndürmez, sadece içindeki program parçasını yürütür. Bu yüzden alt programlar ifade içinde kullanılmaz. Alt yordamların fonksiyonlarda da olduğu gibi parametreleri vardır, bir alt yordamda hiç parametre bildirilmeyebilir. Alt yordamlar da fonksiyonlarda olduğu gibi **dâhilî alt yordamlar** ve **kullanıcı tanımlı alt yordamlar** olarak ikiye ayrılır.

8.2.1. Kullanıcı Tanımlı Alt Yordamlar

Alt yordamların tanımlanması fonksiyonlardaki gibidir, alt yordamlar ana programın içinde veya ana programın dışında tanımlanabilir. Alt yordamları tanımlamak için **subroutine** ve **end subroutine** deyimleri kullanılır.

```
subroutine Alt Yordamın Adı(Parametreler)
   Parametreleri Tanımlama Deyimleri
   ...
end subroutine Alt Yordamın İsmi
```

Alt yordam tanımlamasının sonunda alt yordamın ismi veya bunun yanındaki subroutine anahtar sözcüğü yazılmayabilir, ancak bunları yazmak program kodlarının okunurluğunu arttırır.

Alt yordamın içindeki parametrelerin tanımlamasında fonksiyonlardaki gibi intent niteleyicisi kullanılır.

Alt yordamlar çağırılmak istenildiği zaman **call** deyimi ile çağırılır.

```
call Alt Yordamın İsmi(Argümanlar)
```

Örnek program olarak toplama, çarpma ve ortalama hesaplama işlemlerini yapan bir alt yordam içeren bir program aşağıda verilmiştir:

```
program alt_yordamlar_1

   implicit none
   real :: x, y, z, toplam, carpim, ortalama

   write(*,"(a)", advance="no") "Birinci sayıyı giriniz: "
   read(*,*) x
   write(*,"(a)", advance="no") "İkinci sayıyı giriniz: "
   read(*,*) y
   write(*,"(a)", advance="no") "Üçüncü sayıyı giriniz: "
   read(*,*) z

   call islemler(x, y, z, toplam, carpim, ortalama)

   write(*,*) "Toplam: ", toplam
   write(*,*) "Çarpım: ", carpim
   write(*,*) "Ortalama: ", ortalama

contains

   subroutine islemler(a, b, c, t, cr, o)
```

```
      implicit none
      real, intent(in) :: a, b, c
      real, intent(out) :: t, cr, o

      t = a + b + c
      cr = a * b * c
      o = (a + b + c) / 3.0

   end subroutine islemler

end program alt_yordamlar_1
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Birinci sayıyı giriniz: 30
İkinci sayıyı giriniz: 32.5
Üçüncü sayıyı giriniz: 40
 Toplam:    102.500000
 Çarpım:    39000.0000
 Ortalama:    34.1666679
```

Değişkenlerin kendiliğinden tanımlanmasını önlemek için implicit none deyimi alt yordam içinde de kullanılabilir.

Ana program biriminin dışında alt yordamlar tanımlamak için fonksiyonlardaki gibi arayüz belirtimi yapılmalıdır.

```
interface Arayüz İsmi
   Veri Türü subroutine Alt Yordamın Adı(Parametreler)
      Parametre Bildirimleri
   end subroutine Alt Yordamın Adı
end interface Arayüz İsmi
```

Aşağıda, bir önceki örnek programla aynı işlevi gören fakat alt yordamın ana program biriminin dışında tanımlandığı bir örnek program verilmiştir:

```
program alt_yordamlar_2

   implicit none

   interface arayuz
      subroutine islemler(a, b, c, t, cr, o)
         real, intent(in) :: a, b, c
         real, intent(out) :: t, cr, o
      end subroutine islemler
   end interface arayuz

   real :: x, y, z, toplam, carpim, ortalama

   write(*,"(a)", advance="no") "Birinci sayıyı giriniz: "
   read(*,*) x
   write(*,"(a)", advance="no") "İkinci sayıyı giriniz: "
   read(*,*) y
   write(*,"(a)", advance="no") "Üçüncü sayıyı giriniz: "
   read(*,*) z

   call islemler(x, y, z, toplam, carpim, ortalama)

   write (*,*) "Toplam: ", toplam
   write (*,*) "Çarpım: ", carpim
   write (*,*) "Ortalama: ", ortalama

end program alt_yordamlar_2
```

```
subroutine islemler(a, b, c, t, cr, o)

   implicit none
   real, intent(in) :: a, b, c
   real, intent(out) :: t, cr, o

   t = a + b + c
   cr = a * b * c
   o = (a + b + c) / 3.0

end subroutine islemler
```

Yinelemeli fonksiyonlarda olduğu gibi yinelemeli alt yordamlar da tanımlanabilir.

Çok alt yordam çağırısı, çok fonksiyon çağırısı gibi yığın taşmasına yol açabilir.

Örnek olarak, girilen bir sayının iki tabanında karşılığını hesaplayan bir program aşağıda verilmiştir (mod fonksiyonu için Ek 3'e bakılmalıdır.):

```
program yinelemeli_alt_yordamlar

   implicit none
   integer :: sayi

   write(*,"(a)", advance="no") "Sayıyı giriniz: "
   read(*,*) sayi

   call ikili_yaz(sayi)

   write(*,*)

contains
```

```
   recursive subroutine ikili_yaz(sayi)

      implicit none
      integer, intent(in) :: sayi

      if (sayi > 1) call ikili_yaz(sayi / 2)

      write (*,"(i1)", advance="no") mod(sayi, 2)

   end subroutine ikili_yaz

end program yinelemeli_alt_yordamlar
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Sayıyı giriniz: 12
1100
```

Alt yordamlarda parametreler tanımlanırken fonksiyonlarda olduğu gibi, her parametre zorunlu olmayabilir ve zorunlu olmayan parametre tanımlanırken optional niteleyicisi kullanılır. Örnek olarak, ikiden daha çok sayının toplama ve çarpma işlemlerini yapabilen bir alt yordamı içeren bir program aşağıda veirlmiştir:

```
program alt_yordamlar_optional

   implicit none
   integer :: toplam, carpim

   call islemler(toplam, carpim, 14, -388, 56)
   write (*,*) toplam, carpim
```

```
   call islemler(toplam, carpim, 65, -78, 56, 0, 11)
   write (*,*) toplam, carpim

   call islemler(toplam, carpim, 876, 2, 36)
   write (*,*) toplam, carpim

contains

   subroutine islemler(top, car, a, b, c, d, e)

      implicit none
      integer, intent(out) :: top, car
      integer, intent(in), optional :: a, b, c, d, e

      top = a
      top = top + b

      car = a
      car = car * b

      if(present(c)) then
         top = top + c
         car = car * c
      end if

      if(present(d)) then
         top = top + d
         car = car * d
      end if

      if(present(e)) then
         top = top + e
```

```
         car = car * e
      end if

   end subroutine islemler

end program alt_yordamlar_optional
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
        -318     -304192
          54           0
         914       63072
```

8.2.2. Dâhilî Alt Yordamlar

Dâhilî alt yordamlar, Fortran standardında ve derleyicide ön tanımlı olan alt yordamlardır, bu alt yordamlar daha önceden tanımlanmış oldukları için kullanıcı tarafından tanımlanmasına gerek yoktur. Fortran'da birçok dâhilî alt yordam tanımlanmıştır. Dâhilî alt yordamlarla ilgili daha çok bilgi bu kitabın eklerinde verilmiştir.

8.3. Özet

Alt programlar, programı yinelenen kod satırlarını bir kere yazılacak şekilde gruplayan yapılardır. Fortran'da alt programlar **fonksiyonlar** ve **alt yordamlar** olarak ikiye ayrılır. Fonksiyonlar, kullanıcı tanımlı fonksiyonlar ve dâhilî fonksiyonlar olmak üzere ikiye ayrılır. Alt yordamlar, aynı şekilde kullanıcı tanımlı alt yordamlar ve dâhilî alt yordamlar olmak üzere ikiye ayrılır. Bir fonksiyon veya bir alt yordam, ana program biriminin dışında ya da ana program biriminin içinde tanımlanabilir. Ana program biriminin dışında olacak alt programların ana program bölümünde **arayüz** kullanılarak bildirilmesi gerekir. Bir fonksiyon veya bir alt yordam, ana program biriminin olduğu dosyadan farklı bir dosyada bulunabilir.

8.4. Sorular

8.1) Alt programlar nedir?

8.2) Fortran programlama dilinde alt programların türlerini yazınız.

8.3) Fortran programlama dilinde fonkisyon ile alt programların farklarını belirtiniz.

8.4) Fortran programlama dilinde fonksiyonlar kaça ayrılır?

8.5) Fortran programlama dilinde alt programlar kaça ayrılır?

8.6) Fortran programlama dilinde fonksiyon tanımlanmasını açıklayınız.

8.7) Fortran programlama dilinde alt program tanımlanmasını açıklayınız.

8.9) Ana program biriminin dışındaki fonksiyonların tanımlanmasında ana program biriminde alt programlar bildirilmesi için ne kullanılır?

8.10) Verilen iki sayının permütasyonunu alan bir programı Fortran programlama diliyle yazınız. Permütasyon, bir fonksiyon aracılığıyla alınacak.

8.11) Verilen iki sayının kombinasyonunu alan bir alt programı Fortran programlama diliyle yazınız. Kombinasyon, bir fonksiyon aracılığıyla alınacak.

8.12) Uzunluğu girilen fibbonacci serisini hesaplayan bir alt programı Fortran programlama diliyle yazınız. Fibbonacci serisi, bir fonksiyon aracılığıyla hesaplanacak.

9. Bölüm: Göstericiler

Göstericiler, belirli bir bellek adresini saklayan değişkenlerdir. Bu bellek adresi bir değişkenin adresi olabilir.

Göstericiler kullanılarak, belleği etkin bir biçimde kullanan programlar yazılabilir.

Göstericileri kullanarak birçok veri yapısı oluşturulabilir. Bu veri yapılarına örnek olarak bağlı listeler verilebilir. Bu tür veri yapıları oluşturulması bu kitabın konusu dışındadır.

9.1. Göstericilerin Bildirilmesi

Göstericilerin bildirilmesi, değişken bildirimi gibidir, göstericilerin bildirilmesinde hangi veri türündeki değişken adresini saklaması için veri türü belirtilir. Bildirmede, değişken bildiriminden ayrı olarak **pointer** niteleyicisi kullanılır.

```
Veri Türü, pointer :: Tanımlayıcı
```

Örnekler:

```
integer, pointer :: sayi

character(25), pointer :: ad, soyad

real, pointer :: a, b, c
```

Gösterici bildirimlerinde türetilmiş veri türleri de veri türü olarak belirtilebilir.

Göstericiler, dizilerin de adreslerini saklayabilir. Eğer bir gösterici, bir dizinin adresini saklayacaksa bildirimde **dimension** niteleyicisiyle, hangi dizinin adresini saklayacağı belli

olmadığı için adresini tutacağı bir dizinin kaç boyutlu olacağı kadar virgüllerle ayrılan iki nokta kullanılır.

```
Veri Türü, pointer, dimension(Boyut Sayısı Kadar İki Nokta) :: Tamımlayıcı
```

Örnekler aşağıda verilmiştir:

```
complex, pointer, dimension(:) :: i, j, k

integer, pointer, dimension(:,:) dizi_1, dizi_2

real, pointer, dimension(:,:,:) ortam
```

Göstericilerin bir değişkenin veya bir dizinin adresini saklaması için o değişken veya o dizi bildiriminde **target** niteleyicisi kullanılır.

```
Veri Türü, target :: Tanımlayıcı
```

Örnek olarak:

```
real, target :: a = 21

integer, target, dimension(4,3) :: matris
```

9.2. Göstericilere Değişkenlerin Atanması

Göstericilerin adreslerini saklayacağı değişkenlerin adreslerini saklaması için bir eşittir işaretinden ve bir büyüktür işaretinden oluşan adres atama işleci kullanılır. Böylece gösterici belirtilen değişkenin adresini saklar.

Gösterici => Değişken

Bir gösterici yalnızca bir değişkenin adresini saklar.

Gösterici kullanılarak göstericinin adresini sakladığı değişkene değer atanabilir ya da adresi saklanılan değişkenin değeri kullanılabilir.

Örnek program aşağıda verilmiştir:

```
program gostericiler_1

   implicit none
   integer, pointer :: gosterici
   integer, target :: gosterilen = 5

   write(*,*) "Gösterilen değişken: ", gosterilen

   gosterici => gosterilen

   gosterici = 28

   write(*,*) "Gösterilen değişken: ", gosterilen

end program gostericiler_1
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 Gösterilen değişken =           5
 Gösterilen değişken =          28
```

Eğer göstericiye değişken gibi bir değer atanması için o göstericiye bellekte yer ayrılması gerekir, bu ayrılma da gösterici bildirimiyle yapılmaz. Göstericinin bir değişkenin adresini

saklamdan göstericiye bellekte yer ayrılması için **allocate**, bellekte yer ayrılan gösterici için ayrılan yerin serbest bırakılması için **deallocate** deyimi kullanılır. Örnek program aşağıda verilmiştir:

```
program gostericiler_2

   implicit none
   integer, pointer :: x

   allocate(x)

   x = -4307
   print *, x

   deallocate(x)

end program gostericiler_2
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
       -4307
```

9.3. Göstericinin Bir Değişkenin Adresini Saklayıp Saklamadığını Belirleme ve Göstericinin Adresini Sakladığı Değişkeni Adresini Saklamasının Sonlandırılması

associated() fonksiyonu ile göstericinin bir değişkenin adresini saklayıp saklamadığı belirlenebilir. Bu fonksiyon, belirtilen gösterici, bir değişkenin adresini saklıyorsa **.true.**, bir değişkenin adresini saklamıyorsa **.false.** değerini döndürür, eğer bu fonksiyonda bir değişken de belirtilmişse göstericinin o değişkenin adresini saklayıp saklamadığı da belirlenebilir. Bir

göstericinin bir değişkenin adresini saklamasının sonlandırılması **nullify** deyimi ile yapılır. Örnek program aşağıda verilmiştir:

```
program gostericiler_3

   implicit none
   character(10), pointer :: ileti
   character(10), target :: ileti_1 = "Deneme", ileti_2 = "12345"

   ileti => ileti_1

   if(associated(ileti)) then
      write(*,*) "Gösterici şu değeri gösteriyor: ", ileti
   else
      write(*,*) "Gösterici bir değer göstermiyor."
   end if

   nullify(ileti)

   if(associated(ileti)) then
      write(*,*) "Gösterici şu değeri gösteriyor: ", ileti
   else
      write(*,*) "Gösterici bir değer göstermiyor."
   end if

   ileti => ileti_2

   if(associated(ileti, ileti_1)) then
      write(*,*) "Gösterici ", ileti, "değerini gösteriyor."
   else
      write(*,*) "Gösterici ", ileti, "değerini göstermiyor."
   end if
```

```
end program gostericiler_3
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 Gösterici şu değeri gösteriyor: Deneme
 Gösterici bir değer göstermiyor.
 Gösterici 12345     değerini göstermiyor.
```

9.4. Özet

Göstericiler, belirli bir bellek adresini saklayan değişkenlerdir. Göstericiler kullanılarak, ana belleği etkin bir biçimde kullanan programlar yazılabilir, türlü veri yapıları oluşturulabilir. Göstericilerin tanımlanması, normal değişken tanımlamasından biraz farklıdır.

9.5. Sorular

9.1) Fortran programlama dilinde göstericileri açıklayınız.

9.2) Fortran programlama dilinde gösterici tanımlaması nasıl yapılır?

9.3) Fortran programlama dilinde göstericilerin, bir değişkenin veya bir dizinin adresini saklaması için o değişken veya dizi nasıl tanımlanmalıdır?

9.4) Fortran programlama dilinde göstericilerin değişkenlere atanması nasıl yapılır?

9.5) Fortran programlama dilinde göstericinin bir değişkenin adresini saklamadan bellekte yer ayrılması için hangi deyim kullanılır?

9.6) Fortran programlama dilinde bellekte yer ayrılan göstericiye ayrılan yerin serbest bırakılması için hangi deyim kullanılır?

9.7) Fortran programlama dilinde bir göstericinin bir değişkenin adresini saklayıp saklamadığı hangi fonksiyonla belirlenir?

9.8) Fortran programlama dilinde bir göstericinin bir değişkenin adresini saklamasının sonlandırılması hangi deyimle yapılır?

10. Bölüm: Modüller

Modüller, ana program biriminden ayrı olarak tanımlanan bağımsız program birimleridir. Fortran programlama dilinde modül kullanılarak kütüphaneler yazılabilir. Modüller, ana programdan ayrı olarak derlenebilir. Modüller, ana program biriminin olduğu dosyada veya başka dosyalarda tanımlanabilir. Bundan dolayı daha az çabayla etkili Fortran programları yazılabilir. Modüller tanımlanmadan, ana program biriminin olmadığı dosyalarda alt programlar tanımlayarak da kitaplıklar oluşturulabilir ancak kitaplıklar oluştururken modül kullanılması önerilir.

Modüller, çok sayıda programın ve alt programın kullanacağı değişkenlerin, sabitlerin, fonksiyonların ve alt yordamların tanımlanmasında kullanılır. Bazı algoritmaların dağıtımında modüller kullanılır.

10.1. Modüllerin Yapısı

Modüller, **module** deyimi ve **end module** deyimiyle tanımlanır. Modüllerde, modüller için genel değişken tanımlamalarıyla program birimleri dışında tanımlanan alt programların arayüz belirtimleri vardır, ana program biriminde olduğu gibi contains deyiminden sonra alt program tanımlamaları yapılabilir.

```
module Modülün Adı
   Genel Değişken Tanımlamaları ile Arayüz Belirtimleri
contains
   Alt Program Tanımlamaları
end module Modülün Adı
```

end module deyiminden sonra modül adı veya bunun yanında module anahtar sözcüğü belirtilmeyebilir, ancak bunları belirtmek program kodunun okunurluğunu arttırır.

Modüllerde de genel değişken tanımlamaları ve arayüzlerden önce implicit deyimi kullanılabilir.

Ana programda hangi modüllerin kullanılacağının belirtilmesi **use** deyimiyle yapılır. Bu deyimle modül belirtimleri ana programın başında, implicit none deyiminden önce yapılmalıdır.

```
use Modül Adı
```

Modüllerde bulunan değişkenlerin, sabitlerin ve alt programların, ana program biriminde use deyimi kullanılsa bile modül dışından erişilebilir olup olmayacağı belirtilebilir. Bu belirtim için modülün genel değişken tanımlamaları bölümünde **public** ve **private** deyimleri ve niteleyicileri kullanılır. public, belirtilenlerin modül dışından da erişilebilir olduğunu; private, belirtilenlerin modül dışından erişilemez olduğunu belirtir.

Eğer bir modül kullanıldığında yalnızca belirli elemanların kullanılması isteniyorsa use deyiminin yanında **only** anahtar sözcüğü de kullanılır. only anahtar sözcüüğnün kullanılması aşağıda verilmiştir:

```
use Modül Adı, only:Elemanlar
```

Elemanlar, virgül işaretleriyle birbirlerinden ayrılır.

Aşağıda örnek modül olarak bir matematik kütüphanesi verilmiştir:

```
module matematik_kutuphanesi

   real, public :: pi = 3.14159, e = 2.71828
   public faktoriyel

contains
```

```
   subroutine faktoriyel(n, sonuc)

      integer, intent(in) :: n
      integer, intent(out) :: sonuc
      integer :: i

      sonuc = 1

      if (n >= 2) then
         do i = 1, n
            sonuc = sonuc * i
         end do
      end if

      if (n < 0) then
         print *, "Negatif tamsayı girildi, program sonlandırılacak."
         stop
      end if

   end subroutine faktoriyel

end module matematik_kutuphanesi
```

Aşağıda, bu örnek modülün uygulandığı örnek bir program verilmiştir:

```
program moduller

  use matematik_kutuphanesi

  implicit none
  integer :: x, faktoriyel_x
```

```
  write(*,"(a)", advance="no") "Bir doğal sayı giriniz: "
  read(*,*) x

  call faktoriyel(x, faktoriyel_x)

  write(*,*) "Girilen sayının faktöriyeli = ", faktoriyel_x
  write(*,*) "Yarıçapı girilen bir sayı olan çemberin alanı = ",
real(x) ** 2.0 * pi
  write(*,*) "Girilen sayının Euler sayısından farkı = ", real(x) - e

end program moduller
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
Bir doğal sayı giriniz: 5
 Girilen sayının faktöriyeli =          120
 Yarıçapı girilen bir sayı olan çemberin alanı =    78.5397491
 Girilen sayının Euler sayısından farkı =    2.28171992
```

10.2. Modüllerin Derlenmesi ve Bağlanması

Modüllerin derlenmesi aşağıdaki şekilde yapılır:

```
gfortran -c Modüllerin Kaynak Kodlarının Bulunduğu Dosyalar
```

Dosyalar, birbirilerinden boşluklarla ayrılır.

Derleme sonucunda aynı adlı, bir ara kod dosyası (**.o** uzantılı) ve bir modül tanımalama dosyası (**.mod** uzantılı) oluşur. Modül tanımlama dosyası, ara kod dosyası ve kaynak koddan oluşur.

Ana programın derlenmesi aşağıdaki şekilde yapılır:

```
gfortran -c Ana Program Kaynak Kod Dosyası Adı
```

Derleme sonucunda aynı adlı bir ara kod dosyası (**.o** uzantılı) oluşur. Bu dosyayı oluşturmak için modül tanımlama dosyası gerekir.

Bütün ara kod dosyalarının bağlanması ise aşağıdaki şekilde yapılır:

```
gfortran -c Ara Kod Dosyaları [-o Program Dosyası Adı]
```

Tümleşik geliştirme ortamları; ana programı ve modülleri kendiliğinden derleyip bağlar.

10.3. Özet

Modüller, ana programdan bağımsız olarak derlenebilen, değişkenler, sabitler, fonksiyonlar, alt yordamlar içeren yapılardır. Modüller, ana programdan farklı dosyalarda bulunabilir. Modüller, **module** deyimi ve **end module** deyimiyle tanımlanır. Ana programda, **use** deyimiyle ana programda kullanılacak modüller belirtilir. İstenirse **only** anahtar sözcüğü de kullanılarak modüllerin yalnızca istenilen elemanlarının kullanılması sağlanır. Modüllerdeki elemanlar, **public** ya da **private** deyimleriyle veya niteleyicileriyle dışarıdan erişilebilir olup olmayacağı belirlenebilir. public belirtimi, elemanların modül dışından erişilebilir olacağını; private belirtimi, elemanların modül dışından erişilebilir olmayacağını belirtir. Modüller, ana programdan önce derlenir ve ana program, o modülleri gerektirirse o modüllerle birlikte bağlanır.

10.4. Sorular

10.1) Fortran programlama dilinde modül nedir?

10.2) Fortran programlama dilinde modüller hangi amaçlarla kullanılır?

10.3) Fortran programlama dilinde modül tanımlaması nasıl yapılır?

10.4) Fortran programlama dilinde ana programın bir modülü kullanması için hangi deyim ya da deyimler kullanılır?

10.5) Fortran programlama dilinde use deyimi nasıl kullanılır?

10.6) Fortran programlama dilinde use deyimiyle only deyimi nasıl kullanılır?

10.7) Fortran programlama dilinde pi sayısı, Euler sayısı, faktoriyel hesaplama fonksiyonu, permütasyon hesaplama fonksiyonu, kombinasyon hesaplama fonksiyonu içeren bir modülü Fortran programlama dilinde yazınız.

Ek 1: Sözce-Sayı Dönüşümleri

Fortran'da tamsayı veya gerçel bir değişkeni sözceye dönüştürmek veya bir sözceyi sayısal bir değişkene dönüştürmek için read deyimi ve write deyimi kullanılır.

1. Sözceden Sayıya Dönüşüm

Bir sözcedeki karakterleri tamsayı bir değişkene ya da gerçel bir değişkene aktarmak için read deyimi kullanılır.

`read(`**`Dönüştürülecek Sözce`**`, `**`Biçim Belirtimi`**`, iostat=`**`Durum Değişkeni`**`)`
`Sayı Değişkeni`

Eğer okuma işlemi başarılı ise durum değişkenine sıfır, okuma işlemi başarılı değilse sıfırdan farklı bir değer atanır. Biçim belirtimi, belirtilen sayı değişkeninin biçimini belirtir, bunun belirtilmesi zorunludur. Biçim belirtimi, belirtilen sayı için, büyük olabilir ancak küçük olmamalıdır. Okuma işleminin başarılı olması için sözcede, harf olması, birden fazla nokta bulunması gibi sayı değişkenlerinde olmayacak durumlar olmamalıdır. Örnek program:

```
program sozceden_sayiya_donusum

   implicit none
   character(10) :: sozce = "123456.789"
   real :: sayi
   integer :: durum

   read(sozce, "(f10.3)", iostat=durum) sayi

   if (durum == 0) then
      print *, sayi
   else
      print *, "Sözceden sayıya dönüşümde hata var."
   end if

end program sozceden_sayiya_donusum
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
   123456.789
```

2. Sayıdan Sözceye Dönüşüm

Tamsayı bir sayı değişkenindeki ya da gerçel bir sayı değişkenindeki değeri, karakterler olarak sözceye aktarmak için write deyimi kullanılır.

```
write(Sözce, Biçim Belirtimi, iostat=Durum Değişkeni) Sayı Değişkeni
```

Eğer okuma işlemi başarılı ise durum değişkenine sıfır, okuma işlemi başarılı değilse sıfırdan farklı bir değer atanır. Biçim belirtimi, belirtilen sayı değişkeninin biçimini belirtir, bunun belirtilmesi zorunludur. Biçim belirtiminde, karakter sayısı, dönüştürülecek sayı değişkeni için yazılan karakter kadar olmalıdır ve ondalık kısım için ayrılan basamak sayısına dikkat edilmelidir. Örnek program:

```
program sayidan_sozceye_donusum

   implicit none
   character(10) :: sozce
   real :: sayi=123456.789
   integer :: durum

   write(sozce, "(f10.3)", iostat=durum) sayi

   if (durum == 0) then
      print *, sozce
   else
      print *, "Sözceye aktarma işleminde hata var."
   end if
```

```
end program sayidan_sozceye_donusum
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa şu çıktıyı verir:

```
 123456.789
```

Ek 2: Fortran'da Anahtar Sözcükler

Bu ekte, Fortran'da kulanılan anahtar sözcüklerin bir listesi verilmiştir.

```
abstract
allocatable
allocate
assign
assignment
associate
asynchronous
backspace
bind
block
block data
call
case
character
class
close
codimension
common
complex
contains
contiguous
continue
critical
cycle
data
deallocate
default
deferred
dimension
do
do concurrent
double precision
elemental
else
else if
else where
elseif
elsewhere
end
end block data
end do
end function
end if
end interface
end module
end program
end select
end subroutine
end type
end where
endfile
endif
endwhere
entry
enum
enumerator
equivalence
error
exit
extends
external
final
flush
forall
format
function
generic
go to
goto
if
implicit
import
in
include
inout
inquire
integer
intent
interface
intrinsic
kind
```

```
len                 private             submodule
lock                procedure           subroutine
logical             program             sync all
module              protected           sync images
namelist            public              sync memory
non_overridable     pure                target
nopass              read                then
nullify             real                type
only                recursive           unlock
open                result              use
operator            return              value
optional            rewind              volatile
out                 rewrite             wait
parameter           save                where
pass                select              while
pause               select case         write
pointer             sequence
print               stop
```

Ek 3: Dâhilî Sayısal Alt Programlar

1. abs() Fonksiyonu

Belirtilen sayının mutlak değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
abs(a)
```

a parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, parametre, tamsayıysa tamsayı döndürür, gerçel sayı ya da karmaşık sayı ise gerçel sayı döndürür.

2. aimag() Fonksiyonu

Belirtilen karmaşık sayının sanal kısmını döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
aimag(z)
```

z parametresi, belirtilen karmaşık sayıdır. Döndürülen değer gerçel sayıdır.

3. aint() Fonksiyonu

Belirtilen sayının ondalık kısmını atar. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
aint(a[, kind])
```

a parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer gerçel sayıdır.

4. anint() Fonksiyonu

Belirtilen sayıyı en yakın tam sayıya yuvarlar. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
anint(a[, kind])
```

a parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer gerçel sayıdır.

5. ceiling() Fonksiyonu

Belirtilen sayının kendisinden büyük olan en küçük tamsayıyı döndürür. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. Fortran 95 ve sonraki sürümler içindir.

```
ceiling(a[, kind])
```

a parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

6. cmplx() Fonksiyonu

Belirtilen özelliklerde bir karmaşık sayı oluşturur. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
cmplx(x[, y, kind])
```

x parametresi, oluşturulacak karmaşık sayının gerçek kısımıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. y parametresi, oluşturulacak karmaşık sayının sanal kısımıdır, tamsayı ya da gerçel sayı olmalıdır, x parametresi karmaşık sayıysa belirtilmemelidir,

belirtilmediğinde x parametresi karmaşık sayı değilse döndürülen sayının sanal kısmı **0.0** olur. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer karmaşık sayıdır.

7. conjg() Fonksiyonu

Belirtilen karmaşık sayının eşleniğini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
conjg(z)
```

z parametresi, belirtilen karmaşık sayıdır. Döndürülen değer karmaşık sayıdır.

8. dble() Fonksiyonu

Belirtilen sayıyı çift duyarlıklı gerçel sayıya dönüştürür. FORTRAN 77 ve sonraki sürümler içindir.

```
dble(a)
```

a parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. Döndürülen değer çift duyarlıklı gerçel sayıdır.

9. dim() Fonksiyonu

Belirtilen iki sayının farkını alır. Fark, sıfırdan büyükse bu iki sayının farkını döndürür, sıfırdan büyük değilse sıfır sayısını döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
dim(x, y)
```

x parametresi eksilendir, y parametresi çıkandır. Bu parametreler tamsayı, gerçel sayı veya karmaşık sayı olmalıdır. Fonksiyon, parametreler, tamsayıysa tamsayı, gerçel sayıysa gerçel sayı döndürür.

10. dprod() Fonksiyonu

Belirtilen iki sayının çarpımını döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
dprod(x, y)
```

x ve y parametreleri, belirtilen sayılardır, gerçel sayı olmalıdır. Döndürülen değer çift duyarlıklı gerçel sayıdır.

11. floor() Fonksiyonu

Belirtilen sayının kendisinden küçük olan en büyük tamsayıyı döndürür. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. Fortran 95 ve sonraki sürümler içindir.

```
floor(a[, kind])
```

a parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

12. int() Fonksiyonu

Belirtilen sayıyı tamsayıya dönüştürür. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
int(a[, kind])
```

a parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

13. max() Fonksiyonu

Belirtilen sayıların en büyüğünü döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
max(a1, a2[, a3 [, ...]])
```

a1 parametresi, belirtilen birinci sayıdır, tamsayı ya da gerçel sayı olmalıdır. Diğer parametreler (a2, a3, ...) a1 parametresi ile aynı türden olmalıdır. Fonksiyon, parametreler, tamsayıysa tamsayı, gerçel sayıysa gerçel sayı döndürür.

14. min() Fonksiyonu

Belirtilen sayıların en küçüğünü döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
min(a1, a2[, a3 [, ...]])
```

a1 parametresi, belirtilen birinci sayıdır, tamsayı ya da gerçel sayı olmalıdır. Diğer parametreler (a2, a3, ...) a1 parametresi ile aynı türden olmalıdır. Fonksiyon, parametreler, tamsayıysa tamsayı, gerçel sayıysa gerçel sayı döndürür.

15. mod() Fonksiyonu

Belirtilen sayıların birincisinin ikincisine bölümünden kalanı verir. FORTRAN 77 ve sonraki sürümler içindir.

```
mod(a, p)
```

a parametresi, birinci sayıdır, tamsayı ya da gerçel sayı olmalıdır. p parametresi, ikinci sayıdır, a parametresiyle aynı türden olmalıdır. Fonksiyon, a parametresi, tamsayıysa tamsayı döndürür, gerçel sayıysa gerçel sayı döndürür. `a - (int(a / p) * p)` ifadesi de aynı sonucu döndürür.

16. modulo() Fonksiyonu

Belirtilen sayıların birincisinin ikincisine bölümünden kalanı verir. Fortran 95 ve sonraki sürümler içindir. mod() fonksiyonundan farklı olarak negatif değer döndürmez.

```
modulo(a, p)
```

a parametresi, birinci sayıdır, tamsayı ya da gerçel sayı olmalıdır. p parametresi, ikinci sayıdır, a parametresiyle aynı türden olmalıdır. Fonksiyon, a parametresi, tamsayıysa `a – p * q` ifadesiyle (q bölümdür.) aynı sonucu döndürür, gerçel sayıysa `a - floor(a / p) * p` ifadesiyle aynı sonucu döndürür.

17. nint() Fonksiyonu

Belirtilen sayıyı kendisine en yakın tamsayıya dönüştürür. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir, Fortran 90 sürümüyle birlikte kind parametresi desteği gelmiştir.

```
nint(a[, kind])
```

a parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

18. real() Fonksiyonu

Belirtilen sayıyı gerçel sayıya dönüştürür. İstenirse sonucun bayt cinsinden büyüklüğü belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
real(a[, kind])
```

a parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır, karmaşık sayıysa bunun gerçek kısmını döndürür. kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer gerçel sayıdır.

19. sign() Fonksiyonu

Belirtilen sayıların birincisini, ikincisinin işaretiyle çarpar, çarpım sonucunu döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
sign(a, b)
```

a parametresi, birinci sayıdır, tamsayı ya da gerçel sayı olmalıdır. b parametresi, a parametresiyle aynı türden olmalıdır. Fonksiyon, a parametresi tamsayıysa tamsayı döndürür, gerçel sayıysa gerçel sayı döndürür.

Ek 4: Dâhilî Matematiksel Alt Programlar

Bu alt programlarda açı cinsi radyandır.

1. acos() Fonksiyonu

Belirtilen sayının arkkosinüsünü döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği getirilmiştir.

```
acos(x)
```

x parametresi, beirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür. Döndürülen değerin gerçek kısmı, radyan cinsindendir, [0, π] aralığındadır.

2. asin() Fonksiyonu

Belirtilen sayının arksinüsünü döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği getirilmiştir.

```
asin(x)
```

x parametresi, beirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür. Döndürülen değerin gerçek kısmı, radyan cinsindendir, [-π/2, π/2] aralığındadır.

3. atan() Fonksiyonu

Belirtilen sayının arktanjantını döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği ve çift parametre desteği getirilmiştir.

```
atan(x)
```

x parametresi, beirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür. Döndürülen değerin gerçek kısmı, radyan cinsindendir, [-π/2, π/2] aralığındadır.

```
atan(y, x)
```

Bu fonksiyon, belirtilen karmaşık sayının argümentinin asıl değerini döndürür. y parametresi, beirtilen karmaşık sayının gerçek kısmıdır, gerçel sayı olmalıdır. x parametresi, belirtilen karmaşık sayının sanal kısmıdır, gerçel sayı olmalıdır. Fonksiyon, atan2(y, x) ifadesiyle aynı sonucu döndürür.

4. atan2() Fonksiyonu

Belirtilen karmaşık sayının argümentinin asıl değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
atan2(y, x)
```

x parametresi, beirtilen karmaşık sayının gerçek kısmıdır, gerçel sayı olmalıdır, y parametresinin değeri sıfırsa sıfır olmamalıdır. y parametresi, belirtilen karmaşık sayının sanal kısmıdır, gerçel sayı olmalıdır. Fonksiyon, gerçel sayı döndürür.

5. cos() Fonksiyonu

Belirtilen sayının kosinüsünü döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
cos(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

6. cosh() Fonksiyonu

Belirtilen sayının hiperbolik kosinüsünü döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği getirilmiştir.

```
cosh(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

7. exp() Fonksiyonu

Euler sayısının belirtilen sayıdan kuvvetini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
exp(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, parametre, gerçel sayıysa gerçel sayı, karmaşık sayıysa karmaşık sayı döndürür.

8. log() Fonksiyonu

Belirtilen sayının doğal logaritmasını döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
log(x)
```

x parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, parametre, gerçel sayıysa gerçel sayı, karmaşık sayıysa karmaşık sayı döndürür.

9. log10() Fonksiyonu

Belirtilen sayının on tabanındaki logaritmasını döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
log10(x)
```

x parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, parametre, gerçel sayıysa gerçel sayı, karmaşık sayıysa karmaşık sayı döndürür.

10. sin() Fonksyonu

Belirtilen sayının sinüsünü döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
sin(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

11. sinh() Fonksyonu

Belirtilen sayının hiperbolik sinüsünü döndürür. Fortran 95 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği getirilmiştir.

```
sinh(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

12. sqrt() Fonksiyonu

Belirtilen sayının karekökünü döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
sqrt(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, parametre, gerçel sayıysa gerçel sayı, karmaşık sayıysa karmaşık sayı döndürür.

13. tan() Fonksiyonu

Belirtilen sayının tanjantını döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008'le karmaşık sayı desteği getirilmiştir.

```
tan(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

14. tanh() Fonksiyonu

Belirtilen sayının hiperbolik tanjantını döndürür. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2008 ile karmaşık sayı desteği getirilmiştir.

```
tanh(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Fonksiyon, gerçel sayı belirtilmişse gerçel sayı, karmaşık sayı belirtilmişse karmaşık sayı döndürür.

Ek 5: Dâhilî Sayısal Sorgu Alt Programları

1. digits() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre anlamlı ikili basamak sayısını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
digits(x)
```

x parametresi, belirtilen sayıdır, tamsayı ya da gerçel sayı olmalıdır. Döndürülen değer tamsayıdır.

2. epsilon() Fonksiyonu

1 + a > 1 ve bayt olarak büyüklüğü belirtilen sayının büyüklüğü olacak şekilde en küçük a sayısını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
epsilon(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer gerçel sayıdır.

3. huge() Fonksiyonu

Belirtilen sayının veri türüne ve o veri türünün bayt sayısı olarak büyüklüğüne göre en büyük sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
huge(x)
```

x parametresi, belirtilen sayıdır, tamsayı ya da gerçel sayı olmalıdır. Döndürülen değer belirtilen sayıyla aynı veri türündendir.

4. maxexponent() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre asgari ikili kuvvetini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
maxexponent(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer tamsayıdır.

5. minexponent() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre azami ikili kuvvetini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
minexponent(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer tamsayıdır.

6. precision() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre onluk duyarlılığını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
precision(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı ya da karmaşık sayı olmalıdır. Döndürülen değer tamsayıdır.

7. radix() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre tabanını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
radix(x)
```

x parametresi, belirtilen sayıdır, tamsayı ya da gerçel sayı olmalıdır. Döndürülen değer tamsayıdır.

8. range() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre onluk kuvvet aralığını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
radix(x)
```

x parametresi, belirtilen sayıdır, tamsayı, gerçel sayı ya da karmaşık sayı olmalıdır. Döndürülen değer tamsayıdır.

9. tiny() Fonksiyonu

Belirtilen sayının, veri türüne ve bayt sayısı olarak büyüklüğüne göre en küçük sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
tiny(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer belirtilen sayıyla aynı veri türündendir.

Ek 6: Kayan Noktalı Sayılarla İlgili İşlem Yapan Dâhilî Alt Programlar

1. exponent() Fonksiyonu

Belirtilen sayının üstel kısmını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
exponent(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer, tamsayıdır, belirtilen sayı sıfırsa sıfır değerini döndürür.

2. fraction() Fonksiyonu

Belirtilen sayının kesirli kısmını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
fraction(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer, tamsayıdır, belirtilen sayı sıfırsa sıfır değerini döndürür. Döndürülen değer, gerçel sayıdır, `x * radix(x) ** (-exponent(x))` değerine eşittir.

3. nearest() Fonksiyonu

Belirtilen ikinci sayının işareti yönünde belirtilen birinci sayıya en yakın işlemci eliyle belirtilebilir sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
nearest(x, s)
```

x parametresi, belirtilen birinci sayıdır, gerçel sayı olmalıdır; s parametresi, belirtilen ikinci sayıdır, gerçel sayı olmalıdır ve sıfır olmamalıdır. Döndürülen değer, gerçel sayıdır, belirtilen

ikinci sayı, pozitifse belirtilen birinci sayıdan büyük, negatifse belirtilen birinci sayıdan küçük değer döndürür.

4. rrspacing() Fonksiyonu

Belirtilen sayının yakınındaki sayıların göreceli boşluklamanın resiprokalini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
rrspacing(x)
```

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer gerçel sayıdır, `abs(fraction(x)) * float(radix(x))**digits(x)` değerine eşittir.

5. scale() Fonksiyonu

Belirtilen x sayısının `x * radix(x) ** i` değerini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
scale(x, i)
```

x parametresi, belirtilen x sayısıdır, gerçel sayı olmalıdır; i parametresi, sonuç ifadesindeki i sayısıdır, tamsayı olmalıdır. Döndürülen değer gerçel sayıdır.

6. set_exponent() Fonksiyonu

Kesirli kısmı belirtilen birinci sayı, üstel kısmı belirtilen ikinci sayı olan sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
set_exponent(x, i)
```

x parametresi, belirtilen birinci sayıdır, gerçel sayı olmalıdır; i parametresi, belirtilen ikinci sayıdır, tamsayı olmalıdır. Döndürülen değer, gerçel sayıdır, `fraction(x) * radix(x) ** i` değerine eşittir.

7. spacing() Fonksiyonu

Belirtilen sayıyla kendisine aynı veri türünde en yakın komşu sayı arasındaki uzaklığı döndürür. Fortran 95 ve sonraki sürümler içindir.

`spacing(`**`x`**`)`

x parametresi, belirtilen sayıdır, gerçel sayı olmalıdır. Döndürülen değer gerçel sayıdır.

Ek 7: Bit Düzeyimde İşlem Yapan Dâhilî Alt Programlar

1. bit_size() Fonksiyonu

Belirtilen değerin, ondalık noktası ve işaret biti dahil olmak üzere bit sayısını döndürür. Fortran 95 ve sonraki sürümler içindir.

```
bit_size(i)
```

i parametresi, belirtilen değerdir, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

2. btest() Fonksiyonu

Belirtilen sayının belirtilen sırada olan bitinin değerini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
btest(i, pos)
```

i parametresi, belirtilen değerdir, tamsayı olmalıdır; pos parametresi, belirtilen sıra numarasıdır, tamsayı olmalıdır. Sıra numaraları sıfırdan başlar. Döndürülen değer mantıksaldır.

3. iand() Fonksiyonu

Belirtien iki sayının bit düzeyinde ve işlemi ile olan sonucunu döndürür. Fortran 95 ve sonraki sürümler içindir.

```
iand(i, j)
```

i ve j parametreleri, belirtilen iki sayıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

4. ibclr() Fonksiyonu

Belirtilen sayının belirtilen sırada olan bitinin değerini sıfır yaparak yeni sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
ibclr(i, pos)
```

i parametresi, belirtilen değerdir, tamsayı olmalıdır; pos parametresi, belirtilen sıra numarasıdır, tamsayı olmalıdır. Sıra numaraları sıfırdan başlar. Döndürülen değer tamsayıdır.

5. ibits() Fonksiyonu

Belirtilen sayının belirtilen sıradan itibaren belirtilen sayıdaki bitlerinin değerlerini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
ibits(i, pos, len)
```

i parametresi, belirtilen değerdir, tamsayı olmalıdır; pos parametresi, belirtilen sıra numarasıdır, tamsayı olmalıdır, len parametresi değeri döndürülecek bit sayısıdır. Pos ile len parametrelerinin toplam sayı değeri `bit_size(i)` değerini aşmamalıdır. Sıra numaraları sıfırdan başlar. Döndürülen değer tamsayıdır.

6. ibset() Fonksiyonu

Belirtilen sayının belirtilen sırada olan bitinin değerini bir yaparak yeni sayıyı döndürür. Fortran 95 ve sonraki sürümler içindir.

```
ibset(i, pos)
```

i parametresi, belirtilen değerdir, tamsayı olmalıdır; pos parametresi, belirtilen sıra numarasıdır, tamsayı olmalıdır. Sıra numaraları sıfırdan başlar. Döndürülen değer tamsayıdır.

7. ieor() Fonksiyonu

Belirtien iki sayının bit düzeyinde özel veya işlemi ile olan sonucunu döndürür. Fortran 95 ve sonraki sürümler içindir.

```
ieor(i, j)
```

i ve j parametreleri, belirtilen iki sayıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

8. ior() Fonksiyonu

Belirtien iki sayının bit düzeyinde veya işlemi ile olan sonucunu döndürür. Fortran 95 ve sonraki sürümler içindir.

```
ior(i, j)
```

i ve j parametreleri, belirtilen iki sayıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

9. ishft() Fonksiyonu

Belirtilen sayının bitlerine belirtilen basamak kadar bitirme kaydırması uygular. Belirtilen basamak sayısı sıfırdan büyükse sola kaydırır. Belirtilen basamak sayısı sıfırsa kaydırma olmaz. Belirtilen basamak sayısı sıfırdan küçükse sağa kaydırır. Belirtilen basamak sayısının mutlak değeri belirtilen sayının bit sayısından büyük olmamalıdır. Kaydırma sonucu boşalan bitler sıfır değerindedir. Fortran 95 ve sonraki sürümler içindir.

```
ishft(i, shift)
```

i parametresi, belirtilen sayıdır, tamsayı olmalıdır; shift parametresi, belirtilen basamak sayısıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

10. ishftc() Fonksiyonu

Belirtilen sayının bitlerini belirtilen basamak kadar dairesel kaydırır. Kaydırılacak en sağdaki bit sayısı da belirtilebilir. Kaydırılacak en sağdaki bit sayısı belirtilmezse bu değer belirtilen sayının bit sayısına eşittir. Belirtilen basamak sayısı sıfırdan büyükse sola kaydırır. Belirtilen basamak sayısı sıfırsa kaydırma olmaz. Belirtilen basamak sayısı sıfırdan küçükse sağa kaydırır. Belirtilen basamak sayısı, kaydırma taşması büyüklüğünden küçük olmalıdır. Belirtilen basamak sayısının mutlak değeri belirtilen sayının bit sayısından büyük olmamalıdır. Fortran 95 ve sonraki sürümler içindir.

```
ishftc(i, shift[, size])
```

i parametresi, belirtilen sayıdır, tamsayı olmalıdır; shift parametresi, belirtilen basamak sayısıdır, tamsayı olmalıdır; size parametresi, kaydırılacak en sağdaki bitler sayısıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

11. mvbits() Alt Yordamı

Belirtilen iki sayıdan birinin belirtilen sıra numarasından itibaren, belirtilen sayıda bitlerini diğerinin belirtilen konumuna taşır. Fortran 95 ve sonraki sürümler içindir.

```
mvbits(from, frompos, len, to, topos)
```

from parametresi, belirtilen birinci sayıdır, tamsayı olmalıdır; frompos parametresi, belirtilen birinci sayıda olan konum belirtimidir, tamsayı olmalıdır; len parametresi, taşınacak bit sayısıdır, tamsayı olmalıdır; to parametresi, belirtilen ikinci sayıdır, tamsayı olmak zorundadır; topos parametresi, belirtilen ikinci sayıda olan konum belirtimidir, tamsayı olmalıdır. Sıra numaraları sıfırdan başlar. frompos + len – 1, birinci sayının bit sayısından; topos + len – 1, ikinci sayının bit sayısından büyük olmamalıdır.

12. not() Fonksiyonu

Belirtien sayının bit düzeyinde değil işlemi ile olan sonucunu döndürür. Fortran 95 ve sonraki sürümler içindir.

```
not(i)
```

i parametresi, belirtilen sayıdır, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

Ek 8: Karakterlerle İlgili Dâhilî Alt Programlar

1. achar() Fonksiyonu

Belirtilen tamsayının ASCII kodunda olan karakterini döndürür. İstenirse sonucun bayt cinsinden büyüklüğü de belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
achar(i[, kind])
```

i parametresi, belirtilen tamsayıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer karakter türündedir.

2. adjustl() Fonksiyonu

Belirtilen dizginin solunda olan boşlukların silinip dizginin sağına eklenmesiyle oluşan yeni dizgiyi döndürür. Fortran 90 ve sonraki sürümler içindir.

```
adjustl(string)
```

string parametresi, belirtilen dizgidir. Döndürülen değer karakter türündedir.

3. adjustr() Fonksiyonu

Belirtilen dizginin sağında olan boşlukların silinip dizginin soluna eklenmesiyle oluşan yeni dizgiyi döndürür. Fortran 95 ve sonraki sürümler içindir.

```
adjustr(string)
```

string parametresi, belirtilen dizgidir. Döndürülen değer karakter türündedir.

4. char() Fonksiyonu

Belirtilen tamsayıya göre sistemin ön tanımlı karakter setinden olan bir karakter döndürür. İstenirse sonucun bayt cinsinden büyüklüğü de belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir.

```
char(i[, kind])
```

i parametresi, belirtilen tamsayıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer karakter türündedir.

5. iachar() Fonksiyonu

Belirtilen karakterin ASCII kodunu döndürür. İstenirse sonucun bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
iachar(c[, kind])
```

c parametresi, belirtilen karakterdir; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

6. ichar() Fonksiyonu

Belirtilen karakterin, sistemin ön tanımlı karakter setinden olan kodunu döndürür. İstenirse sonucun bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
ichar(c[, kind])
```

c parametresi, belirtilen karakterdir; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

7. index() Fonksiyonu

Belirtilen bir dizgi içinde belirtilen bir alt dizginin ilk bulunduğu konumunu döndürür. Konum belirtimi birden başlar. İstenirse aramanın geriden yapılacağı ve döndürülecek değerin bayt cinsinden büyüklüğü de belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
index(string, substring[, back, kind])
```

string parametresi, belirtilen dizgidir; substring parametresi, belirtilen alt dizgidir; back parametresi, geriden arama belirtimidir, mantıksal olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, tamsayı değer döndürür, eğer belirtilen alt dizgi belirtilen dizgi içinde bulunmazsa sıfır değerini döndürür.

8. len() Fonksiyonu

Belirtilen dizginin veya belirtilen karakter dizisinin bir ögesinin uzunluğunu döndürür. İstenirse döndürülen değerin bayt cinsinden büyüklüğü de belirtilebilir. FORTRAN 77 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
len(string[, kind])
```

string parametresi, belirtilen dizgi veya belirtilen karakter dizisinin bir ögesidir; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

9. len_trim() Fonksiyonu

Belirtilen dizginin sondaki boşluklarının olmadığı hâlinin uzunluğunu döndürür. İstenirse döndürülen değerin bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
len(string[, kind])
```

string parametresi, belirtilen dizgidir; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Döndürülen değer tamsayıdır.

10. lge() Fonksiyonu

Belirtilen iki dizginin birincisi, ikincisinden sözlük değeri olarak büyük ya da eşitse **.true.**, değilse **.false.** değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
lge(string_a, string_b)
```

string_a parametresi, belirtilen birinci dizgidir; string_b parametresi, belirtilen ikinci dizgidir. Döndürülen değer mantıksaldır.

Sözlük değerlerini karşılaştıran fonksiyonlarda sözlük değeri olarak ASCII sıralaması dikkate alınır.

11. lgt() Fonksiyonu

Belirtilen iki dizginin birincisi, ikincisinden sözlük değeri olarak büyükse **.true.**, değilse **.false.** değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
lgt(string_a, string_b)
```

string_a parametresi, belirtilen birinci dizgidir; string_b parametresi, belirtilen ikinci dizgidir. Döndürülen değer mantıksaldır.

12. lle() Fonksiyonu

Belirtilen iki dizginin birincisi, ikincisinden sözlük değeri olarak küçük ya da eşitse **.true.**, değilse **.false.** değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
lle(string_a, string_b)
```

string_a parametresi, belirtilen birinci dizgidir; string_b parametresi, belirtilen ikinci dizgidir. Döndürülen değer mantıksaldır.

13. llt() Fonksiyonu

Belirtilen iki dizginin birincisi, ikincisinden sözlük değeri olarak küçükse **.true.**, değilse **.false.** değerini döndürür. FORTRAN 77 ve sonraki sürümler içindir.

```
llt(string_a, string_b)
```

string_a parametresi, belirtilen birinci dizgidir; string_b parametresi, belirtilen ikinci dizgidir. Döndürülen değer mantıksaldır.

14. repeat() Fonksiyonu

Belirtilen dizginin belirtilen sayı kadar kopyasını sıralar. Fortran 95 ve sonraki sürümler içindir.

```
repeat(string, ncopies)
```

string parametresi, belirtilen dizgidir; ncopies parametresi belirtilen kopya sayısıdır, tamsayı olmalıdır. Döndürülen değer karakter türündedir.

15. scan() Fonksiyonu

Belirtilen birinci dizgide belirtilen ikinci dizgideki karakterleri arar, bu karakterlerin herhangi birinin ilk bulunma yerinin konumunu döndürür. Konum belirtimi birden başlar. İstenirse aramanın geriden yapılacağı ve döndürülecek değerin bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir, Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
scan(string, set[, back, kind])
```

string değişkeni, belirtilen birinci dizgidir; set parametresi, belirtilen ikinci dizgidir; back parametresi, geriden arama belirtimidir, mantıksal olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, tamsayı değer döndürür, eğer belirtilen karakterler belirtilen dizgi içinde bulunmazsa sıfır değerini döndürür.

16. trim() Fonksiyonu

Belirtilen dizginin, sonunda olan boşlukları silerek yeni hâlini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
trim(string)
```

string değişkeni, belirtilen dizgidir. Döndürülen değer karakter türündedir.

17. verify() Fonksiyonu

Belirtilen birinci dizgide belirtilen ikinci dizgideki karakterleri arar, bu karakterlerin herhangi birinin bulunmadığı ilk yerin konumunu döndürür. Konum belirtimi birden başlar. İstenirse aramanın geriden yapılacağı ve döndürülecek değerin bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
verify(string, set[, back, kind])
```

string değişkeni, belirtilen birinci dizgidir; set parametresi, belirtilen ikinci dizgidir; back parametresi, geriden arama belirtimidir, mantıksal olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, tamsayı değer döndürür, eğer belirtilen karakterlerin herhangi birinin bulunmadığı yer yoksa sıfır değerini döndürür.

Ek 9: Bayt Olarak Büyüklükle İlgili Dâhilî Alt Programlar

1. kind() Fonksiyonu

Belirtilen ifadenin bayt sayısı olarak büyüklüğünü döndürür. Fortran 95 ve sonraki sürümler içindir.

```
kind(x)
```

x parametresi, belirtilen ifadedir, herhangi bir temel veri türünden olmalıdır. Döndürülen değer tamsayıdır.

2. selected_char_kind() Fonksiyonu

Belirtilen karakter seti adının bayt olarak büyüklüğünü döndürür. Fortran 2003 ve sonraki sürümler içindir.

```
selected_char_kind(name)
```

name parametresi, belirtilen karakter seti adıdır, alabileceği değerler şunlardır: **"ascii"** (ASCII için.), **"default"** (Ön tanımlı karakter seti için.), **"iso_10646"** (UCS-4 için.). Fonksiyon, eğer bir karakter seti, belirtilen adla desteklenmiyorsa **-1** değerini döndürür. Döndürülen değer tamsayıdır.

3. selected_int_kind() Fonksiyonu

10'un, belirtilen tamsayının kuvvetiyle belirtilen tamsayının negatifinin kuvveti arasındaki tüm tamsayıları gösterebilecek en küçük bayt olarak büyüklük değerini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
select_int_kind(r)
```

r parametresi, belirtilen tamsayıdır. Fonksiyon, belirtilen sayıyla bu sayı gibi bir bayt olarak büyüklük değeri yoksa **-1** değerini döndürür. Döndürülen değer tamsayıdır.

4. selected_real_kind() Fonksiyonu

Ondalık duyarlılığı en az belirtilen birinci sayı, üstel aralığı en az belirtilen ikinci sayı, tabanı en az belirtilen üçüncü sayı olan gerçel sayının gösterilebilecek en küçük bayt olarak büyüklüğünü döndürür. Fortran 95 ve sonraki sürümler içindir. Fortran 2008'le birlikte radix parametresi desteği gelmiştir.

```
selected_real_kind([p, r, radix])
```

p parametresi, belirtilen birinci sayıdır, tamsayı olmalıdır; r parametresi, belirtilen ikinci sayıdır, tamsayı olmalıdır; radix parametresi, belirtilen üçüncü sayıdır, tamsayı olmalıdır. Fortran 2008'den önce belirtilen birinci sayı ve belirtilen ikinci sayıdan en az biri belirtilmelidir, Fortran 2008'le birlikte bu belirtimlerin ikisi de yoksa bu belirtimler sıfır olarak kabul edilmektedir. Fonksiyon, belirtilen sayılarla gösterilebilecek böyle bir bayt olarak büyüklük değeri yoksa şu değerleri döndürür: **-1** (İşlemci böyle gerçel sayıları desteklemiyorsa, yalnızca ikinci sayı belirtilmişse ve diğer belirtimler karşılanabiliyorsa.), **-2** (İşlemci böyle gerçel sayıları desteklemiyorsa, yalnızca birinci sayı belirtilmişse ve diğer belirtimler karşılanabiliyorsa.), **-3** (Üçüncü belirtim karşılanabiliyor ve diğer belirtimler karşılanamıyorsa.), **-4** (Üçüncü belirtim karşılanabiliyor ve diğer belirtimlerin ikisinden biri karşılanabiliyorsa.), **-5** (Üçüncü belirtim karşılanamıyorsa.).

Ek 10 - Dâhilî Mantıksal Alt Programlar

1. logical() Fonksiyonu

Belirtilen mantıksal değeri bayt olarak belirtilen büyüklükte bir mantıksal değere dönüştürür. Fortran 95 ve sonraki sürümler içindir.

```
logical(l[, kind])
```

l parametresi, belirtilen mantıksal değerdir; kind parametresi, bayt olarak büyüklük belirtimidir, tamsayı olmalıdır. Fonksiyon, mantıksal değer döndürür, bayt olarak büyüklük belirtimi yapılmadıysa ön tanımlı büyüklükte bir değer döndürür.

Ek 11 - Seçimlik Değiştirgenlerle İlgili Dahili Alt Programlar

1. present() Fonksiyonu

Belirtilen seçimlik alt program değiştirgeninin var olup olmadığını saptar. Fortran 95 ve sonraki sürümler içindir.

```
present(a)
```

a parametresi; bir işaretçi, bir sayıl ya da bir dizi olabilir, herhangi bir veri türünden olabilir. Fonksiyon, belirtilen böyle bir değiştirgen varsa **.true.**, yoksa **.false.** değerini döndürür.

Ek 12 - Diziyle İlgili Dâhilî Alt Programlar

1. Vektör ve Matris Çarpımıyla İlgili Dizi Alt Programları

1.1. dot_product() Fonksiyonu

Belirtilen iki vektörü skaler olarak çarpar. Fortran 95 ve sonraki sürümler içindir.

```
dot_product(vector_a, vector_b)
```

vector_a ve vector_b parametreleri, belirtilen vektörlerdir, aynı sayıda öge içermelidir, tamsayı, gerçel, karmaşık ya da mantıksal olmalıdır. Döndürülen değer, eğer vektörler, tamsayı ya da gerçelse sonuç sum(vector_a * vector_b), karmaşıksa sonuç sum(conjg(vector_a) * vector_b), mantıksalsa any(vector_a .and. vector_b) olur.

1.2. matmul() Fonksiyonu

Belirtilen iki matrisi çarpar. Fortran 95 ve sonraki sürümler içindir.

```
matmul(matrix_a, matrix_b)
```

matrix_a ve matrix_b parametreleri, belirtilen bu iki matristir, tek boyutlu dizi olabilir. matrix_a, tamsayı, gerçel, karmaşık ya da mantıksal olmalıdır. matrix_b, matrix_a sayısalsa sayısal, matrix_a mantıksalsa mantıksal olur. Döndürülen dizinin türü, * ve .and. işleçlerine giren değer türlerine göre olur.

2. Dizi Redüksiyon Fonksiyonları

2.1. all() Fonksiyonu

Belirtilen maske dizisinde bulunan bütün ögeler doğru ise **.true.**, yanlış ise **.false.** değerini döndürür. İstenirse boyut numarası da belirtilebilir. Fortran 95 ve sonraki sürümler içindir.

```
all(mask[, dim])
```

mask parametresi maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, maske dizisi mantıksal bir dizidir; dim parametresi boyut numarası belirtimidir, tamsayı olmalıdır. Fonksiyon, boyut numarası belirtilmemişse tamsayı sayıl, boyut numarası belirtilmişse tamsayı dizi döndürür.

Dizi fonksiyonlarında (all, any, count, maxloc, maxval, minloc, minval, product, sum fonksiyonlarında) boyut numarası belirtimi ile sonuç arasındaki ilişkiyi bir örnekle açıklayalım. Bir a maske dizisi `x * y * z` boyutlarında tanımlansın. `any(a, dim=2)` ifadesi `any(a(i,:,j))` ifadesine eşittir ve bu ifadenin sonucu `x * z` boyutlarında bir dizidir.

2.2. any() Fonksiyonu

Belirtilen maske dizisinde bulunan en az bir öge doğru ise **.true.**, yanlış ise **.false.** değerini döndürür. İstenirse boyut numarası da belirtilebilir. Fortran 95 ve sonraki sürümler içindir.

```
any(mask[, dim])
```

mask parametresi maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, maske dizisi mantıksal bir dizidir; dim parametresi boyut numarası belirtimidir, tamsayı olmalıdır. Fonksiyon, boyut numarası belirtilmemişse tamsayı sayıl, boyut numarası belirtilmişse tamsayı dizi döndürür.

2.3. count() Fonksiyonu

Belirtilen maske dizisindeki **.true.** olan ögelerin sayısını döndürür. İstenirse boyut numarası veya sonucun bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
count(mask[, dim, kind])
```

mask parametresi maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, maske dizisi mantıksal bir dizidir; dim parametresi boyut numarası belirtimidir, tamsayı olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, boyut numarası belirtilmemişse tamsayı sayıl, boyut numarası belirtilmişse tamsayı dizi döndürür.

2.4. maxval() Fonksiyonu

Belirtilen dizide bulunan en büyük ögeyi ya da en büyük ögeleri döndürür. İstenirse boyut numarası veya maske dizisi de belirtilebilir. Fonksiyonda, maske dizisi belirtiminde **.true.** olan ögeler dikkate alınır, **.false.** olanlarsa dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
maxval(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı ya da gerçel sayı olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, burada maske dizisi, boyutları dizinin boyutları kadar olan mantıksal bir dizidir. Fonksiyon, boyut numarası belirtilmemişse belirtilen diziyle aynı veri türünde olan bir sayıl, boyut numarası belirtilmişse belirtilen diziyle aynı veri türünde olan bir dizi döndürür.

2.5. minval() Fonksiyonu

Belirtilen dizide bulunan en küçük ögeyi ya da en küçük ögeleri döndürür. İstenirse boyut numarası veya maske dizisi de belirtilebilir. Fonksiyonda, maske dizisi belirtiminde **.true.** olan ögeler dikkate alınır, **.false.** olanlarsa dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
minval(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı ya da gerçel sayı olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, burada maske dizisi, boyutları dizinin boyutları kadar olan mantıksal bir dizidir. Fonksiyon, boyut numarası belirtilmemişse belirtilen diziyle aynı veri türünde olan bir sayıl, boyut numarası belirtilmişse belirtilen diziyle aynı veri türünde olan bir dizi döndürür.

2.6. product() Fonksiyonu

Belirtilen sayısal dizide bulunan ögeleri çarpar. İstenirse boyut numarası veya maske dizisi de belirtilebilir. Fonksiyonda, maske dizisi belirtiminde **.true.** olan ögeler dikkate alınır, **.false.** olanlarsa dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
product(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı, gerçel ya da karmaşık olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, mantıksal ve belirtilen diziyle aynı biçimde olmalıdır. Fonksiyon, belirtilen dizinin türünden olan döndürür.

2.7. sum() Fonksiyonu

Belirtilen dizide bulunan ögelerin toplamlarını döndürür. İstenirse maske dizisi veya boyut numarası belirtimleri de yapılabilir. Maske dizisi belirtiminde, **.true.** değerlerinin karşılığındaki ögeler dikkate alınır, **.false.** değerlerinin karşılığındaki ögeler dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
sum(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı, gerçel ya da karmaşık olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, mantıksal ve belirtilen diziyle aynı biçimde olmalıdır. Döndürülen dizi belirtilen diziyle aynı veri türündendir.

3. Dizi Sorgu Alt Programları

3.1. allocated() Fonksiyonu

Belirtilen dinamik dizinin bellekte ayrılıp ayrılmadığını belirten fonksiyondur. Belirtilen dinamik dizi için bellekte yer ayrılmışsa **.true.**, ayrılmamışsa **.false.** değerini döndürür. Fortran 95 ve sonraki sürümler içindir. Fortran 2003'le birlikte dizi yerine sayıl belirtimi desteği de gelmiştir.

```
allocated(array)
```

```
allocated(scalar)
```

array parametresi, belirtilen dinamik dizidir; scalar parametresi, belirtilen sayıldır. Döndürülen değer mantıksaldır.

3.2. lbound() Fonksiyonu

Belirtilen dizinin boyutlarının en düşük sınırlarını döndürür. İstenirse dizinin hangi boyutunun en düşük sınırını döndüreceği belirtimi yapılabilir, sonucun bayt cinsinden büyüklüğü belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
lbound(array[, dim, kind])
```

array parametresi, belirtilen dizidir, herhangi bir veri türünden olabilir; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, boyut numarası belirtilmemişse tamsayı dizi, boyut numarası belirtilmişse tamsayı sayıl döndürür.

3.3. shape() Fonksiyonu

Belirtilen bir dizinin biçimini döndürür. İstenirse sonucun bayt cinsinden büyüklük belirtimi yapılabilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003'le birlikte kind parametresi desteği gelmiştir.

```
shape(source[, kind])
```

source parametresi, sayı ya da dizidir, herhangi bir veri türünden olabilir; kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Döndürülen değer bir boyutlu tamsayı dizidir.

3.4. size() Fonksiyonu

Belirtilen dizinin tamamının veya bir boyutunun kaç ögeli olduğunu döndürür. İstenirse dizinin hangi boyutunun kaç ögeli olacağı veya sonucun bayt cinsinden büyüklüğü de belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003'le birlikte kind parametresi desteği gelmiştir.

```
size(array[, dim, kind])
```

array parametresi, belirtilen dizidir, herhangi bir veri türünden olabilir; dim parametresi, dizinin hangi boyutunun kaç ögeli olacağı belirtimidir, tamsayı olmalıdır; kind parametresi, büyüklük belirtimidir, tamsayı olmalıdır. Fonksiyon, dim parametresi, belirtilirse belirtilen boyutun kaç ögeli olacağını, belirtilmezse tüm dizinin kaç ögeli olduğunu döndürür. Döndürülen değer tamsayıdır.

3.5. ubound() Fonksiyonu

Belirtilen dizinin boyutlarının en yüksek sınırlarını döndürür. İstenirse dizinin hangi boyutunun en yüksek sınırını döndüreceği belirtimi yapılabilir, sonucun bayt cinsinden büyüklüğü belirtilebilir. Fortran 95 ve sonraki sürümler içindir. Fortran 2003 ile birlikte kind parametresi desteği gelmiştir.

```
ubound(array[, dim, kind])
```

array parametresi, belirtilen dizidir, herhangi bir veri türünden olabilir; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; kind parametresi, sonucun bayt cinsinden büyüklüğüdür, tamsayı olmalıdır. Fonksiyon, boyut numarası belirtilmemişse tamsayı dizi, boyut numarası belirtilmişse tamsayı sayıl döndürür.

4. Dizi Yapım Alt Programları

4.1. merge() Fonksiyonu

Belirtilen iki dizinin ögelerinden maske dizisine göre yeni bir dizi oluşturur. Maske dizisindeki her **.true.** değeri için o değerlerin belirtilen birinci diziye göre karşılığı olan ögeler kullanılır, her **.false.** değeri içinse o değerlerin belirtilen ikinci diziye göre karşılığı olan ögeler kullanılır. Fortran 95 ve sonraki sürümler içindir.

```
merge(tsource, fsource, mask)
```

tsource parametresi, belirtilen birinci dizidir, herhangi bir veri türünden olabilir; fsource parametresi, belirtilen ikinci dizidir, birinci diziyle aynı veri türünden olmalıdır; mask parametresi, maske dizisi belirtimidir, mantıksal olmalıdır. Bu üç dizi de aynı biçimde olmalıdır. Döndürülen dizi, birinci diziyle aynı veri türündendir.

4.2. pack() Fonksiyonu

Belirtilen bir dizinin ögelerini bir tek boyutlu dizide saklar. Maske dizisindeki her **.true.** değeri için o değerlerin belirtilen diziye göre karşılığı olan ögeler kullanılır, her **.false.** değeri içinse, vektör belirtilmişse o değerlerin belirtilen vektöre göre karşılığı olan ögeler kullanılır, vektör dizisi belirtilmemişse bir şey kullanılmaz. Fortran 95 ve sonraki sürümler içindir.

```
pack(array, mask[, vector])
```

array parametresi, belirtilen dizidir, herhangi bir veri türünden olabilir; mask parametresi, maske dizisi belirtimidir, mantıksal ve belirtilen diziyle aynı biçimde olmalıdır; vector parametresi, vektör dizisi belirtimidir, belirtilen diziyle aynı veri türünde ve en az belirtilen dizinin öge sayısı kadar sayıda ögesi olmalıdır. Fonksiyon, belirtilen diziyle aynı veri türünde bir boyutlu dizi döndürür.

4.3. spread() Fonksiyonu

Belirtilen bir kaynak dizisini belirtilen boyut boyunca belirtilen bir sayıya çoğaltır. Fortran 95 ve sonraki sürümler içindir.

```
spread(source, dim, ncopies)
```

source parametresi, belirtilen kaynak dizisidir, herhangi bir veri türünden olabilir; dim parametresi, boyut belirtimidir, kaynak dizisinin boyut sayısının bir fazlası olabilir ve tamsayı olmalıdır; ncopies parametresi, belirtilen çoğaltma sayısıdır, tamsayı olmalıdır. Döndürülen dizi, kaynak dizisiyle aynı veri türünde ve boyut sayısı kaynak dizisininkinden bir fazlasıdır.

4.4. unpack() Fonksiyonu

Belirtilen bir vektörün ögelerini daha yüksek boyut sayılı bir diziye doldurur. Maske dizisi belirtiminde, **.true.** değerlerinin karşılığına gelen vektör belirtiminin ögeleri, **.false.** değerlerinin karşılığına gelen alan dizisi belirtiminin ögeleri dikkate alınır. Fortran 95 ve sonraki sürümler içindir.

```
unpack(vector, mask, field)
```

vector parametresi, vektör belirtimidir, herhangi bir veri türünden olabilir, en az maske dizisindeki **.true.** değerli öge sayısı kadar ögesi olmalıdır; mask parametresi, maske dizisi belirtimidir, mantıksal olmalıdır; field parametresi, belirtilen alan dizisidir, vektörle aynı veri türünde ve maske dizisiyle aynı biçimde olmalıdır. Döndürülen dizi, alan dizisi biçimindedir ve alan dizisiyle aynı veri türündendir.

5. Dizi Yeniden Biçimlendirme Alt Programları

5.1. reshape() Fonksiyonu

Belirtilen bir kaynak dizisini belirtilen biçim dizisine göre yeniden biçimlendirir. Gerekli olduğunda, sonuç olacak dizi belirtilen dolgu dizisinin ögeleriyle doldurulur veya sonuç olacak dizide sıralama, belirtilen sıralama dizisine göre yapılır. Fortran 95 ve sonraki sürümler içindir.

```
reshape(source, shape[, pad, order])
```

source parametresi, belirtilen kaynak dizisidir, herhangi bir veri türünden olabilir; shape parametresi, belirtilen biçim dizisidir, tamsayı ve tek boyutlu olmalıdır; pad parametresi, belirtilen dolgu dizisidir, belirtilen kaynak dizisiyle aynı türden olmalıdır; order parametresi, belirtilen sıralama dizisidir, tamsayı ve biçim dizisiyle aynı biçimde olmalıdır. Sıralama dizisi, bir sayısından biçim dizisinin öge sayısına kadar olan sayıların dizilmesinden oluşur, belirtilmemişse doğal sıralama yapılır. Fonksiyon, biçim dizisine göre biçimde ve kaynak dizisiyle aynı veri türünde bir dizi döndürür.

Sıralama dizisine örnek olarak kaynak dizisi bir matris dizisiyse sıralama dizisi, sütuna göre sıralama için **(1, 2)**, satıra göre sıralama içinse **(2, 1)** olmalıdır.

6. Dizi Manipülasyon Alt Programları

6.1. cshift() Fonksiyonu

Belirtilen dizide belirtilen sayı kadar dairesel kaydırma yapar. Belirtilen sayı; sıfırdan büyükse sola kaydırma, sıfırdan küçükse sağa kaydırma, sıfır ise kaydırmanın olmayacağı belirtilir. Dizi birden çok boyutlu ise kaydırma belirtimi bir dizi belirtimi gibi olur. İstenirse kaydırmanın hangi boyuttan yapılacağı belirtilebilir. Fortran 95 ve sonraki sürümler içindir.

```
cshift(array, shift[, dim])
```

array parametresi, kaydırma yapılacak dizi belirtimidir, herhangi bir veri türünden olabilir; shift parametresi, kaydırma belirtimidir, tamsayı olmalıdır; dim parametresi, kaydırma yapılacak boyut belirtimidir, tamsayı olmalıdır. Döndürülen dizi kaydırma yapılacak dizinin veri türündendir.

6.2. eoshift() Fonksiyonu

Belirtilen dizide belirtilen sayı kadar bitirme kaydırması yapar. Belirtilen sayı; sıfırdan büyükse sola kaydırma, sıfırdan küçükse sağa kaydırma, sıfır ise kaydırmanın olmayacağı belirtilir. Dizi birden çok boyutlu ise kaydırma belirtimi bir dizi belirtimi gibi olur. İstenirse kaydırmanın hangi boyuttan yapılacağı belirtilebilir, cshift() fonksiyonundan farklı olarak sınır belirtimi de yapılabilir. Sınır belirtiminde kaydırma sonucu boşalan yerlerin hangi ögelerle doldurulacağı belirtilir. Sınır belirtimi yapılmazsa kaydırma yapılacak dizinin türüne göre öge (sayısal ise **0**, mantıksal ise **.false.**, karakterse belirtilen sayıda boşluklar) doldurulur. Fortran 95 ve sonraki sürümler içindir.

```
eoshift(array, shift[, boundary, dim])
```

array parametresi, kaydırma yapılacak dizi belirtimidir, herhangi bir veri türünden olabilir; shift parametresi, kaydırma belirtimidir, tamsayı olmalıdır; boundary parametresi, sınır belirtimidir, kaydırma yapılacak dizinin veri türünden olmalıdır; dim parametresi, kaydırma yapılacak boyut belirtimidir, tamsayı olmalıdır. Döndürülen dizi kaydırma yapılacak dizinin veri türündendir.

6.3. transpose() Fonksiyonu

Belirtilen matrisin transpozesini döndürür. Fortran 95 ve sonraki sürümler içindir.

```
transpose(matrix)
```

matrix parametresi, belirtilen matristir, herhangi bir veri türünden olabilir. Döndürülen matris, matrix parametresiyle aynı türdendir.

7. Dizi Konum Alt Programları

7.1. maxloc() Fonksiyonu

Belirtilen dizide bulunan en büyük ögenin konumunu ya da en büyük ögelerin konumlarını döndürür. İstenirse boyut numarası veya maske dizisi de belirtilebilir. Fonksiyonda, maske dizisi belirtiminde **.true.** olan ögeler dikkate alınır, **.false.** olanlarsa dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
maxloc(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı ya da gerçel sayı olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, burada maske dizisi, boyutları dizinin boyutları kadar olan mantıksal bir dizidir. Fonksiyon, boyut numarası belirtilmemişse belirtilen diziyle aynı veri türünde olan bir sayıl, boyut numarası belirtilmişse belirtilen diziyle aynı veri türünde olan bir dizi döndürür.

7.2. minloc() Fonksiyonu

Belirtilen dizide bulunan en küçük ögenin konumunu ya da en küçük ögelerin konumlarını döndürür. İstenirse boyut numarası veya maske dizisi de belirtilebilir. Fonksiyonda, maske dizisi belirtiminde **.true.** olan ögeler dikkate alınır, **.false.** olanlarsa dikkate alınmaz. Fortran 95 ve sonraki sürümler içindir.

```
minloc(array[, dim, mask])
```

array parametresi, belirtilen dizidir, tamsayı ya da gerçel sayı olmalıdır; dim parametresi, boyut numarası belirtimidir, tamsayı olmalıdır; mask parametresi, maske dizisi belirtimidir, bu parametreye dizi yerine koşul da belirtilebilir, burada maske dizisi, boyutları dizinin boyutları kadar olan mantıksal bir dizidir. Fonksiyon, boyut numarası belirtilmemişse belirtilen diziyle

aynı veri türünde olan bir sayıl, boyut numarası belirtilmişse belirtilen diziyle aynı veri türünde olan bir dizi döndürür.

Ek 13: Rastgele Sayı Oluşturma

Fortran 95'le birlikte Fortran'da, rastgele sayılar üretilebilir, rastgele sayı oluşturmak için **random_seed()** ve **random_number()** dahili alt yordamları kullanılır.

```
random_seed([size, put, get])
```

Yukarıdaki, sözde rastgele sayı üretecini yeniden başlatan veya sorgulayan bir alt yordamdır. Bu alt yordamda hiçbir parametre belirtilmezse sözde rastgele sayı üreteci ön tanımlı durumda başlar. size parametresi, tamsayı bir değişkendir, kaynağı tutmak için kullanılacak tamsayı sayısını belirtir. put parametresi, bir boyutlu tamsayı bir dizidir, kaynağın durumunu değiştirmeyi sağlar. get parametresi, bir boyutlu tamsayı bir dizidir, şimdiki kaynak durumunu belirtir.

```
random_number(harvest)
```

Yukarıdaki, bir sözde rastgele sayı veya bir sözde rastgele sayı dizisi döndüren bir alt yordamdır. harvest parametresi, gerçel bir sayıdır veya gerçel bir dizidir, döndürülen değerleri belirtir. Döndürülen değerler 1.0 hariç 0.0'dan 1.0'e kadar gerçel sayılar olabilir.

Örnek program olarak bir çift zarın benzetimini yapan bir program verilmiştir:

```
program zar_atma

   implicit none
   real :: a
   integer :: n = 1, saat, i
   integer, dimension(:), allocatable :: kaynak

   call random_seed(size=n)
```

```
   allocate(kaynak(n))

   call random_number(a)
   print *, int(a * 6.0) + 1

   call system_clock(count=saat)
   kaynak = saat * (/ (i, i=1, n) /)
   call random_seed(put=kaynak)

   call random_number(a)
   print *, int(a * 6.0) + 1

   deallocate(kaynak)

end program zar_atma
```

Bu örnek program derlendiğinde çalıştırılabilir dosyanın adı "Deneme" olarak varsayılırsa Linux'ta konsol çıktısı şöyle olabilir:

```
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           5
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           2
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           6
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           4
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
```

```
           6
           1
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           3
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           6
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           3
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           6
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           5
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           2
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           4
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           3
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           3
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
           6
           5
```

Ek 14: Sistem Bilgisi Dâhilî Alt Programları

Fortran'da sistem bilgisi alt programlarıyla işletim sisteminden ve donanımdan komut satırı değiştirgenleri, saat ve tarih gibi bilgiler edinilerek bunlar programda kullanılabilir.

1. Komut Satırı İşlemleriyle İlgili Alt Programlar

Fortran 2003'le birlikte Fortran'a komut satırını kullanmak için dâhilî alt programlar tanımlanmıştır.

1.1. get_command() Alt Yordamı

Bir programı çağırmak için olan tüm komut satırını getiren bir alt yordamdır. Fortran 2003 ve sonrası içindir.

```
get_command([command, length, status])
```

command parametresi, karakter bir değişkendir ve buna tüm komut satırı atanır. length parametresi, tamsayı bir değişkendir ve buna tüm komut satırının kaç karakterlik olduğu atanır. status parametresine, işlem başarılıysa 0, command parametresi tüm komut satırı için kısaysa -1, işlemde bir hata varsa pozitif bir sayı değeri atanır.

1.2. command_argument_count() Fonksiyonu

Bir programı çağırmak için olan komut satırındaki değiştirgen sayısını döndüren bir fonksiyondur. Bu fonksiyonun döndürdüğü değer tamsayıdır. Fortran 2003 ve sonrası içindir.

```
command_argument_count()
```

Bu fonksiyonun parametresi yoktur.

1.3. get_command_argument() Alt Yordamı

Komut satırının belirtilen sırada olan verisine erişilmesini sağlayan bir alt yordamdır. Fortran 2003 ve sonrası içindir.

```
get_command_argument(number [, value, length, status])
```

number parametresi, tamsayı bir değişkendir ve belirtilen sırada olan veriyi işaret eder, bu parametrenin değeri sıfırsa program adını işaret eder, sıfır değerinin program adını ataması için sistemin bu özelliği desteklemesi gerekir. value parametresi karakter türünde bir değişkendir ve buna belirtilen sırada olan veri atanır, eğer veri bu değişken için büyükse değişkene verinin bu değişkenin uzunluğu kadarı atanır. length parametresi tamsayı bir değişkendir ve buna belirtilen sıradaki verinin uzunluğu atanır. status parametresi tamsayı bir değişkendir ve buna işlem başarılıysa sıfır, value parametresinin uzunluğu belirtilen sıradaki verinin uzunluğundan azsa **-1**, işlemde bir hata varsa pozitif bir sayı değeri atanır.

Örnek program olarak verilen parametreleri teker teker standart çıktıya yazan bir program verilmiştir:

```
program komut

   implicit none
   character(50) :: parametre
   integer :: parametre_sayisi, i

   parametre_sayisi = command_argument_count()

   do i = 0, parametre_sayisi
      call get_command_argument(i, parametre)
      write(*,*) parametre
   end do
```

```
end program komut
```

Bu örnek program derlendiğinde çalıştırılabilir dosyanın adı "Deneme" olarak varsayılırsa Linux'ta konsol çıktısı şöyle olabilir:

```
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
 ./Deneme
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme 1 2 3
 ./Deneme
 1
 2
 3
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme 12345
 ./Deneme
 12345
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme rgerg ergewg
erwgewrg weewg ewrgewg
 ./Deneme
 rgerg
 ergewg
 erwgewrg
 weewg
 ewrgewg
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $ ./Deneme
12345678901234567890123456789012345678901234567890123456789012345
 ./Deneme
 12345678901234567890123456789012345678901234567890
eersoy93@Lenovo ~/Masaüstü/Deneme/bin/Debug $
```

2. Süre ve Zaman İşlemleriyle İlgili Alt Programlar

2.1. cpu_time() Alt Yordamı

Programın geçen CPU süresini döndüren bir alt yordamdır. Fortran 95 ve sonrası içindir.

```
cpu_time(time)
```

time parametresi, gerçel bir değişkendir ve mikro saniye cinsinden geçen CPU süresi atanır, eğer bir süre kaynağı yoksa buna -1.0 değeri atanır.

Örnek program olarak sıfırdan bir milyara kadar sayan bir döngünün çalışma süresini veren bir program verilmiştir:

```
program gecen_islemci_zamani

   implicit none
   integer :: i
   real :: sure_1, sure_2

   call cpu_time(sure_1)

   !Yürütülme süresi hesaplanacak bölüm
   do i = 0, 1000000000
   end do

   call cpu_time(sure_2)

   write(*,*) "Yürütülme süresi: ", sure_2 - sure_1

end program gecen_islemci_zamani
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
 Yürütülme süresi:    2.24399996
```

2.2. system_clock() Alt Yordamı

İşlemci saati bilgilerini döndüren bir alt yordamdır. Fortran 95 ve sonrası içindir.

```
system_clock([count, count_rate, count_max])
```

count parametresi, tamsayı bir değişkendir ve buna işlemci saati sayım değeri atanır. count_rate parametresi, tamsayı veya gerçel bir değişkendir ve buna bir saniyede olan işlemci saati sayım değeri atanır. count_max parametresi, tamsayı bir değişkendir ve buna azami işlemci saati sayım değeri atanır. Eğer işlemci saati bilgisi yoksa ya da işlemci saati bilgisi alınamazsa count parametresi `huge(count)` değerini alır, başka parametreler sıfır değerini alır.

Örnek program olarak işlemci saat bilgilerini standart çıktıya yazan bir program verilmiştir:

```
program islemci_saati

   implicit none
   integer :: sayim, sayim_hizi, sayim_en_cok
   call system_clock(sayim, sayim_hizi, sayim_en_cok)
   write(*,*) sayim, sayim_hizi, sayim_en_cok

end program islemci_saati
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
     2852371        1000  2147483647
```

2.3. date_and_time() Alt Yordamı

Tarih ile saat bilgisini alan bir alt yordamdır. date parametresi 8 karakterlik bir değişkendir ve buna tarih bilgisi atanır. Fortran 95 ve sonrası içindir.

```
date_and_time([date, time, zone, values])
```

Tarih bilgisi **ccyymmdd** (cc: yüzyıl, yy: yıl, mm: ay, dd: gün) biçimindedir. Time parametresi, 10 karakterlik bir değişkendir ve buna saat bilgisi atanır. Saat bilgisi **hhmmss.sss** (hh: 24 saatlik saat, mm: ay, ss: saniye, sss: mikro saniye) biçimindedir. zone parametresi, 5 karakterlik bir değişkendir ve buna konum bilgisi atanır. Konum bilgisi **(+/-)hhmm** (hh: saat, mm: dakika) biçimindedir. values parametresi sekiz ögeli tamsayı bir dizidir. Bu dizinin, birinci ögesine yıl bilgisi, ikinci ögesine ay bilgisi, üçüncü ögesine gün bilgisidir, dördüncü ögesine saatin Eş Güdümlü Evrensel Saat'ten olan farkın dakika olarak bilgisi, beşinci ögesine 24 saatlik saat bilgisi, altıncı ögesine dakika bilgisi, yedinci ögesine saniye bilgisidir, sekizinci ögesine mikro saniye bilgisi atanır.

Örnek program olarak tarih ve saat bilgilerini standart çıktıya yazdıran bir program verilmiştir:

```
program tarih_ve_saat_bilgisi

   implicit none
   character(8) :: tarih
   character(10) :: saat
   character(5) :: konum
   integer, dimension(8) :: degerler

   call date_and_time(tarih, saat, konum, degerler)
```

```
   write (*,"(a,2x,a,2x,a)") tarih, saat, konum
   write (*,"(8i4)") degerler

end program tarih_ve_saat_bilgisi
```

Bu örnek program, derlenip çalıştırıldığında standart çıkışa verdiği çıktı aşağıdaki gibi olabilir:

```
20160626  165319.970  +0300
2016   6  26 180  16  53  19 970
```

Ek 15: ASCII Tablosu

Bu ekte ASCII'deki kontrol karakterleriyle yazdırılabilir karakterlerin tablosu verilmiştir. Genişletilmiş ASCII karakterlerine burada yer verilmemiştir, bu karakterler karakter kümesine göre değişiklik göstermektedir.

Sayısal Karşılığı	Karakter	Açıklama
0	NUL	Boş
1	SOH	Başlık başlangıcı
2	STX	Metin başlangıcı
3	ETX	Metin sonu
4	EOT	Aktarım sonu
5	ENQ	Sorgu
6	ACK	Bildirim
7	BEL	Zil
8	BS	Geri alma
9	HT	Yatay sekme
10	LF	Satır besleme
11	VT	Dikey sekme
12	FF	Form besleme
13	CR	Satır başı
14	SO	Dışarı kaydırma
15	SI	İçeri kaydırma
16	DLE	Veri bağlantısından çık
17	DC1	Aygıt denetimi – 1
18	DC2	Aygıt denetimi – 2
19	DC3	Aygıt denetimi – 3
20	DC4	Aygıt denetimi - 4
21	NAK	Olumsuz bildirim
22	SYN	Eş zamanlı boşta kalma
23	ETB	Aktarım bloğu sonu
24	CAN	İptal

25	EM	Ortam sonu
26	SUB	Yerine geçme
27	ESC	Çıkış
28	FS	Dosya ayırıcı
29	GS	Grup ayırıcı
30	RS	Kayıt ayırıcı
31	US	Birim ayırıcı
32	SP	Boşluk
33	!	Ünlem işareti
34	"	Çift tırnak işareti
35	#	Diyez
36	$	Dolar işareti
37	%	Yüzde
38	&	Ve işareti
39	'	Tek tırnak işareti
40	(	Ayraç (açma)
41	)	Ayraç (kapama)
42	*	Yıldız
43	+	Artı
44	,	Virgül
45	-	Eksi
46	.	Nokta
47	/	Eğik çizgi
48	0	Sıfır
49	1	Bir
50	2	İki
51	3	Üç
52	4	Dört
53	5	Beş
54	6	Altı
55	7	Yedi
56	8	Sekiz
57	9	Dokuz

58	:	İki nokta üst üste
59	;	Noktalı virgül
60	<	Küçüktür
61	=	Eşittir
62	>	Büyüktür
63	?	Soru işareti
64	@	Kuyruklu a
65	A	A harfi
66	B	B harfi
67	C	C harfi
68	D	D harfi
69	E	E harfi
70	F	F harfi
71	G	G harfi
72	H	H harfi
73	I	I harfi
74	J	J harfi
75	K	K harfi
76	L	L harfi
77	M	M harfi
78	N	N harfi
79	O	O harfi
80	P	P harfi
81	Q	Q harfi
82	R	R harfi
83	S	S harfi
84	T	T harfi
85	U	U harfi
86	V	V harfi
87	W	W harfi
88	X	X harfi
89	Y	Y harfi
90	Z	Z harfi

91	[	Köşeli ayraç (açma)
92	\	Ters eğik çizgi
93	]	Köşeli ayraç (kapama)
94	^	Şapka
95	_	Alt çizgi
96	`	Aksan işareti
97	a	a harfi
98	b	b harfi
99	c	c harfi
100	d	d harfi
101	e	e harfi
102	f	f harfi
103	g	g harfi
104	h	h harfi
105	i	i harfi
106	j	j harfi
107	k	k harfi
108	l	l harfi
109	m	m harfi
110	n	n harfi
111	o	o harfi
112	p	p harfi
113	q	q harfi
114	r	r harfi
115	s	s harfi
116	t	t harfi
117	u	u harfi
118	v	v harfi
119	w	w harfi
120	x	x harfi
121	y	y harfi
122	z	z harfi
123	{	Süslü açık ayraç

124	|	Dikey çizgi
125	}	Süslü kapalı ayraç
126	~	Yaklaşık işareti
127	DEL	Silme

Türkçe-İngilizce Sözlük

Bu bölümde, bu kitapta kullanılan birtakım Türkçe terimlerin İngilizce'deki karşılıkları verilmiştir. Bu eki yazılmasının nedeni, terimler hakkında kafa karışıklığını önlemektir.

- **Açıklama bölümü:** Comment
- **Alt dizgi:** Substring
- **Alt dizi:** Subarray
- **Alt program:** Procedure, subprogram
- **Alt yordam:** Subroutine
- **Anahtar sözcük:** Keyword
- **Ancak ve ancak (mantıksal ifadelerde):** Equivalent
- **Arayüz:** Interface
- **Ayırma (bellek ayırma):** Allocate
- **Belirteç:** Specifier
- **Bellek:** RAM
- **Biçim:** Format
- **Biçim (dizilerde):** Shape
- **Bildirim, bildirme:** Declaration
- **Bitirme kaydırması:** End-off shift
- **Boyut:** Dimension
- **Büyüklük (dizilerde):** Size
- **CPU süresi:** CPU time
- **Çapraz platform:** Cross-platform
- **Çevrim İçi:** Online
- **Çıkış:** Output
- **Çift duyarlıklı:** Double-precision
- **Çift duyarlıklı kayan noktalı sayı:** Double-precision floating-point number
- **Dâhilî alt program:** Intrinsic procedure, intrinsic subprogram

- **Dahilî alt yordam:** Intrinsic subroutine
- **Dâhilî fonksiyon:** Intrinsic function
- **Dairesel kaydırma:** Circular shift
- **Değişmez:** Constant
- **Değiştirgen:** Argument
- **Deyim:** Statement
- **Dikey aralık:** Vertical spacing
- **Dizgi:** String
- **Dizi:** Array
- **Dizi yapıcısı:** Array constructor
- **Dizisel programlama:** Array programming
- **Doğal:** Native
- **Dosya:** File
- **Döngü:** Loop
- **Duyarlık:** Precision
- **Düzenleme tanımlayıcıları:** Edit descriptors
- **Eniyileştirme:** Optimization
- **Eş Güdümlü Evrensel Saat:** Coordinated Universal Time
- **Eş zamanlı programlama:** Concurrent programming
- **Etiket:** Label
- **Fonksiyon Çevirici:** Wrapper
- **Form besleme:** Form feed
- **Gerçek sayı:** Real number
- **Giriş:** Input
- **Gösterici:** Pointer
- **İfade:** Expression
- **İlk örnek:** Prototype
- **İstem:** Prompt
- **İşleç:** Operator

- **İşlemci tarafından belirtilebilir sayı:** Processor-representable number
- **İşlemci saati:** Processor clock
- **Kapsam (dizilerde):** Extent
- **Karakter kümesi:** Character set
- **Karmaşık sayı:** Complex number
- **Kayan nokta:** Floating-point
- **Kaynak (rastgele sayı üretmede):** Seed
- **Kesirli kısım:** Fractional part
- **Kitaplık:** Library
- **Komşu sayı:** Adjacent number
- **Komut satırı:** Command line
- **Koşul:** Condition
- **Kuvvet:** Exponent
- **Mantıksal:** Logical
- **Mertebe (dizilerde):** Rank
- **Nesne tabanlı programlama:** Object oriented programming
- **Niteleyici:** Attribute
- **Öge:** Element
- **Ön tanımlı:** Default
- **Özel veya (mantıksal ifadelerde):** Non-equivalent
- **Sabit biçimlendirme:** Fixed formatting
- **Saf:** Pure
- **Satır başı:** Carriage return
- **Satır besleme:** Line feed
- **Satıra göre sıralama:** Row-major ordering
- **Sayıl:** Scalar
- **Seçmeli duyarlı matematik:** Arbitrary precision mathematic
- **Serbest bırakma (belleği serbest bırakma):** Deallocate
- **Serbest biçimlendirme:** Free formatting

- **Sözde rastgele:** Pseudorandom
- **Süre kaynağı:** Time source
- **Sütuna göre sıralama:** Column-major ordering
- **Tamsayı:** Integer
- **Tanımlama:** Definition
- **Tanımlayıcı:** Identifier
- **Taşınabilir:** Portable
- **Tek duyarlıklı:** Single-precision
- **Tek duyarlıklı kayan noktalı sayı:** Single-precision floating-point number
- **Tümleşik geliştirme ortamı:** Integrated development environment
- **Türetilmiş veri türü:** Derived data type
- **Üstel kısım:** Exponent part
- **Ve (mantıksal ifadelerde):** And
- **Veri:** Data
- **Veri türü:** Data type
- **Veri yapıları:** Data structures
- **Veya (mantıksal ifadelerde):** Or
- **Yapısal programlama:** Structured programming
- **Yığın:** Stack
- **Yığın taşması:** Stack overflow
- **Yinelemeli:** Recursive
- **Yordam:** Routine
- **Yürütülebilir:** Executable
- **Yürütülemeyen:** Non-executable
- **Zorunlu programlama:** Imperative programming

Kaynaklar

Akademik Makale Kaynakları

Floyd B. Hanson; Local Supercomputing Training in the Computational Sciences Using Remote National Centers; Yayıncı: Future Generation Computer Systems – Elsevier B.V.; Yayın Tarihi: Kasım 2003; Sayfalar: 1335-1347

J. W. BACKUS, R. J . BEEBER, S. BEST, R. GOLDBERG, L. M. HAIBT, H. L. HERRICK, R. A. NELSON, D. SAYRE, P. B. SHERİDAN, H. J. STERN, I. ZILLER, R. A. HUGHES ve R. NUTT; The FORTRAN Automatic Coding System; Yayıncı: In Proceedings Western Joint Computer Conference; Yayın Tarihi: 1957; Sayfalar: 188-198

John BACKUS; The History of FORTRAN I, II and III; Konferans: Proceedings First ACM SIGPLAN Conference on History of Programming Languages; Yayın Tarihi: 1978; Sayfalar: 165-180

Martin H. GREENFIELD; History of FORTRAN Standardization; Konferans: Proceedings of the 1982 National Computer Conference; Yayıncı: AFIPS Press; Yayın Tarihi: 7 Haziran 1982 – 10 Haziran 1982; Sayfalar: 817-824

S. GORN, E. L. LOHSE, R. V. SIMTH, J. F. TRAUB; “Character Structure And Character Parıty Sense For Serıal-By-Bıt Data Communıcatıon In Amerıcan Standard Code For Informatıon Interchange”; Yayıncı: Communications of the ACM, Sayfalar: 553-556; Yayın Tarihi: Eylül 1965; Sayfalar: 553–556

Xingfu WU, Valerie TAYLOR; Performance Modeling of Hybrid MPI/OpenMP Scientific Applications on Large-Scale Multicore Supercomputers; Yayıncı: Journal of Computer and System Sciences – Elsevier B. V.; Yayın Tarihi: Aralık 2013; Sayfalar: 1256-1268

Yuefan DENG, Alex KOROBKA; The Performance of a Supercomputer Built With Commodity Components; Yayıncı: Parallel Computing – Elsevier B. V.; Yayın Tarihi: Ocak 2001; Sayfalar: 91-108

Kitap Kaynakları

Bahattin KANBER, Andrew BEDDALL; An Introduction to Fortran 95; 1. Basım; Yayıncı: Gazi Kitapevi; Basım Tarihi: 2006

Ed JORGENSEN; Introduction to Programming using Fortran 95/2003/2008; 3.0.6. Sürüm; Yayın Tarihi: 2018; İnternet Adresi: http://www.egr.unlv.edu/~ed/fortranv3.pdf

John M. LEVESQUE, Joel W. WILLIAMSON; A Guidebook to Fortran on Supercomputers; 1. Basım; Yayıncı: Academic Press; Yayın Tarihi: 1989

Katherine HOLCOMB; Scientific Programming in Fortran 2003; İnternet Adresi: https://wiki.uiowa.edu/download/attachments/109785161/fortran-novella-Holcomb.pdf

Mithat UYSAL, S. Aynur UYSAL; Fortran 90 & 95 & 2000; 2. Basım; Yayıncı: Beta Yayınevi; Yayın Tarihi: 2004

Walter S. Brainerd; Guide to Fortran 2008 Programming; 1. Sürüm; Yayıncı: Springer, London; Yayın Tarihi: 2015

İnternet Kaynakları

Absoft; İnternet Adresi: https://www.absoft.com/

Coarray Fortran (GCC Wiki’deki bir madde): https://gcc.gnu.org/wiki/Coarray

Code::Blocks; İnternet Adresi: http://www.codeblocks.org/

Fortran Standards Technical Commite; İnternet Adresi: http://j3-fortran.org/

Fortran Wiki; İnternet Adresi: http://fortranwiki.org/

g95; İnternet Adresi: http://www.g95.org/

Gedit; İnternet Adresi: https://wiki.gnome.org/Apps/Gedit

GNU Emacs; İnternet Adresi: https://www.gnu.org/software/emacs/

GNU Fortran Manual; İnternet Adresi: https://gcc.gnu.org/onlinedocs/gfortran/

GNU Nano; İnternet Adresi: https://www.nano-editor.org/

GNU Wiki GFortran – Gfortran; İnternet Adresi: https://gcc.gnu.org/wiki/Gfortran

Intel Fortran Compiler; İnternet Adresi: https://software.intel.com/en-us/fortran-compilers

Intel Fortran Compiler; İnternet Adresi: https://software.intel.com/en-us/fortran-compilers

Kate; İnternet Adresi: https://kate-editor.org/

KWrite; İnternet Adresi: https://www.kde.org/applications/utilities/kwrite/

Microsoft Visual Studio; İnternet Adresi: https://www.visualstudio.com

Microsoft Windows; İnternet Adresi: https://www.microsoft.com/en-us/windows

NAG Fortran Compiler; İnternet Adresi: https://www.nag.co.uk/nag-compiler

Notepad++; İnternet Adresi: https://notepad-plus-plus.org/

Open64; İnternet Adresi: https://developer.amd.com/x86-open64-compiler-suite/

Open Watcom; İnternet Adresi: http://www.openwatcom.org/

Oracle Developer Studio; İnternet Adresi:
https://www.oracle.com/tools/developerstudio/index.html

Photran; İnternet Adresi: http://www.eclipse.org/photran/

The GNU Fortran Compiler, https://gcc.gnu.org/onlinedocs/gfortran/

University of Gaziantep Fortran Homepage, http://www.fortran.gantep.edu.tr/

Vim; İnternet Adresi: http://www.vim.org/

xed; İnternet Adresi: https://github.com/linuxmint/xed