

T.C.
BİLECİK ŞEYH EDEBALI ÜNİVERSİTESİ
PAZARYERİ MESLEK YÜKSEKOKULU
BİLGİSAYAR PROGRAMCILIĞI BÖLÜMÜ



PROGRAMLAMA TEMELLERİ ÖDEVİ: KARAR YAPILARI, DÖNGÜLER VE DİZİLER

Öğrenci Zeliha Alıcık
Öğrenci Numarası



ÖĞRETİM GÖREVLİSİ
SERKAN SÖKMEN

BLP102 Programlama Dilleri

BİLECİK 2025

İÇİNDEKİLER

İÇİNDEKİLER	1
ÖNSÖZ	3
1 Giriş ve Proje Amacı	4
2 Operatörler	5
2.1 Aritmetik Operatörler	5
2.2 Atama Operatörleri	6
2.3 Karşılaştırma Operatörleri	6
2.4 Mantıksal Operatörler	7
2.4.1 VEYA Operatörü ()(OR)	8
2.4.2 VE Operatörü (&&)(AND)	8
2.4.3 DEĞİL Operatörü(!) (NOT)	8
3 Karar Yapıları (if-else, switch-case)	9
3.1 İf-else Yapısı	9
3.1.1 Söz Dizisi (Sytnax)	9
3.1.2 if-else Yapısı Örnek	10
3.2 else-if Yapısı	11
3.2.1 Söz Dizisi (Sytnax)	11
3.2.2 else-if Yapısı Örnek	12
3.3 switch-case Yapısı	13
3.3.1 Söz Dizisi (Sytnax)	13
3.3.2 Örnek Kod 1 (Normal Kullanım - break ile):	14
3.3.3 Örnek 2: Fall-through Durumu (break kullanılmadığında)	16
4 Döngü Yapıları	17
4.1 For Döngüsü	18
4.1.1 Söz Dizisi (Sytnax)	18
4.1.2 For Döngüsüne Örnek	18
4.2 while Döngüsü	20

4.2.1	Sözdizimi (Sytnax)	20
4.2.2	while Döngüsü Örnek	21
5	Kontrol İfadeleri: break ve continue	23
5.1	break İfadesi	23
5.1.1	Örnek kullanım mantığı (genel syntax/if karar yapısı ve while döngüsü ile):	23
5.2	continue İfadesi	23
5.2.1	Örnek kullanım mantığı (genel syntax/for döngüsü ile):	23
6	Diziler	24
6.1	Dizilerin Tanımlanması ve Başlatılması (Sözdizimi/Sytnax)	24
6.1.1	Genel Sözdizimi	24
6.1.2	veya başlatma ile birlikte Sözdizimi:	24
6.1.3	veya boyutu belirtmeden, başlangıç değerlerinden boyutunu çıkarmasına izin verme ile Sözdizimi:	24
6.1.4	Dizinin Elemanlarına Erişme	25
6.1.5	Diziler - Temel Örnek	25
6.1.6	Örnek Kod	25
6.2	Karakter Dizisi İşlemleri Örneği (gets() ve strlen() ile)	27
6.2.1	Örnek Kod:	27
6.3	Diziler ve Rastgele Sayılar: rand() ve srand() Örneği	29
6.3.1	Örnek Kod:	29
	KAYNAKLAR	31

1 Giriş ve Proje Amacı

Programlama dilleri, bilgisayarlarla insan etkileşimini sağlayan ve algoritmik düşüncüyü ifade etme imkanı sunan temel araçlardır. Bu bağlamda, her biri kendine özgü bir iletişim paradigması sunmaktadır.

Bu ödev kapsamında, programlama dillerinin temel kontrol yapıları arasında yer alan karar yapıları (if-else, switch-case), tekrar eden işlemleri yönetmeye yarayan döngüler (for, while, do-while) ve veri depolama ile erişimin etkili yöntemlerinden olan diziler detaylı bir şekilde incelenecektir. Konular, verilen örneklerle pekiştirilerek kavramsal ve uygulamalı anlayışın derinleştirilmesi hedeflenmektedir.

2 Operatörler

Programlama dillerinde, veriler üzerinde çeşitli işlemler gerçekleştirmek ve değerler arasındaki ilişkileri tanımlamak için operatörler kullanılır. Bu operatörler, programın mantıksal akışını şekillendiren koşullu ifadelerin ve tekrarlı işlemlerin temelini oluşturur. Bu bölümde, özellikle karar yapıları ve döngülerin anlaşılması için kritik öneme sahip olan başlıca operatör türleri detaylı bir şekilde incelenecektir

2.1 Aritmetik Operatörler

Aritmetik operatörler, temel matematiksel işlemleri gerçekleştirmek üzere tasarlanmıştır. Bu operatörler, programlama dillerinde geniş bir kullanım alanına sahip olup, karar yapılarında koşul ifadelerinin oluşturulmasında, döngülerde artırma ve azaltma işlemleriyle değişkenlerin manipülasyonunda ve dizi elemanları üzerinde matematiksel hesaplamaların yapılmasında etkin bir şekilde kullanılır.

Operatör	İngilizce Adı	Türkçe Adı	Açıklama	Örnek
+	Addition	Toplama	İki değeri birbirine ekler.	$x + y$
-	Subtraction	Çıkarma	Bir değerden diğerini çıkarır.	$x - y$
*	Multiplication	Çarpma	İki değeri birbiriyle çarpar.	$x * y$
/	Division	Bölme	Bir değeri diğerine böler.	x / y
%	Modulus	Mod alma / Kalan	Bölme işleminden kalanı döndürür.	$x \% y$
++	Increment	Artırma	Bir değişkenin değerini 1 artırır.	$++x$
--	Decrement	Azaltma	Bir değişkenin değerini 1 azaltır.	$--x$

Tablo 1: Aritmetik operatörler.

2.2 Atama Operatörleri

Programlama dillerinde, bir değişkene değer atamak veya mevcut değerini bir işlem sonucunda güncellemek temel bir gereksinimdir. Atama operatörleri, bu işlemleri gerçekleştirmek üzere tasarlanmış olup, değişkenlere doğrudan değer atamak ya da mevcut değerlerini aritmetik veya bit düzeyli işlemlerle değiştirmek için kullanılır. Bu operatörler, programın veri akışını ve değişkenlerin yaşam döngüsünü yönetmede merkezi bir role sahiptir.

Operatör	İngilizce Adı	Türkçe Adı/Anlamı	Açıklama	Örnek
=	Assignment	Atama	Sağdaki değeri soldaki değişkene atar.	$x = 5$
+=	Addition Assignment	Toplayarak Atama / Üzerine Ekleme	Değişkenin değerini sağdaki değer kadar artırır. ($x = x + 3$)	$x += 3$
-=	Subtraction Assignment	Çıkararak Atama /Üzerinden Çıkarma	Değişkenin değerini sağdaki değer kadar azaltır. ($x = x - 3$)	$x -= 3$
*=	Multiplication Assignment	Çarparak Atama /Kendisiyle Çarpma	Değişkenin değerini sağdaki değerle çarpır. ($x = x * 3$)	x / y
/=	Division Assignment	Bölerek Atama / Kendisini Bölme	Değişkenin değerini sağdaki değere böler. ($x = x / 3$).	$x /= 3$
%=	Modulus Assignment	Mod Alarak Atama/ Kalanı Atama	Değişkenin değerinin sağdaki değere bölümünden kalanı atar. ($x = x \% 3$)	$x \% = 3$

Tablo 2: Atama Operatörleri.

2.3 Karşılaştırma Operatörleri

Programlamada, iki değer arasındaki ilişkiyi (eşitlik, eşitsizlik, büyüklük, küçüklük vb.) kontrol etmek ve bu ilişkiye göre program akışını yönlendirmek büyük önem taşır. Karşılaştırma operatörleri, bu tür ilişkileri değerlendirmek ve sonucunda doğru (true) veya yanlış (false)

gibi bir mantıksal değer döndürmek amacıyla kullanılır. Özellikle karar yapılarında (if-else, switch-case) koşullu ifadeler oluşturmaının temelini bu operatörler oluşturur.

Operatör	İngilizce Adı	Türkçe Adı	Açıklama	Örnek
==	Equal to	Eşittir	Değerler eşitse true (doğru) döndürür.	x == y
!=	Not equal	Eşit Değildir	Değerler eşit değilse true (doğru) döndürür	(x = x + 3)
>	Greater than	Büyüktür	İlk değer ikinci değerden büyükse true (doğru) döndürür.	x > y
<	Less than	Küçüktür	İlk değer ikinci değerden küçükse true (doğru) döndürür.	x < y
>=	Greater than or equal to	Büyük Eşittir	İlk değer ikinci değerden büyük veya eşitse true (doğru) döndürür.	x >= y
<=	Less than or equal to	Küçük Eşittir	İlk değer ikinci değerden küçük veya eşitse true (doğru) döndürür.	x <= y

Tablo 3: Karşılaştırma Operatörleri.

2.4 Mantıksal Operatörler

Programlamada, birden fazla koşulun aynı anda değerlendirilmesi veya koşulun tam tersi bir durumun kontrol edilmesi sıklıkla ihtiyaç duyulan bir durumdur. Mantıksal operatörler, bu tür karmaşık mantıksal ifadeleri oluşturmak ve birden fazla koşulun true (doğru) ya da false (yanlış) sonuçlarını birleştirerek tek bir mantıksal değer elde etmek amacıyla kullanılır. Özellikle if-else gibi karar yapılarında daha esnek ve güçlü koşullar tanımlamanın temel araçlarıdır.

Operatör	İngilizce Adı	Türkçe Adı	Açıklama	Örnek
	OR	VEYA	Her iki ifade de doğruysa true (doğru) döndürür.	x < 5 x < 4
&&	AND	VE	Her iki ifade de doğruysa true (doğru) döndürür.	x < 5 && x < 10
!	NOT	DEĞİL	İfadenin mantıksal sonucunu tersine çevirir. Sonuç true ise false, false ise true döndürür.	!(x < 5 && x < 10)

Tablo 4: Mantıksal Operatörler.

2.4.1 VEYA Operatörü (||)(OR)

Bu operatör, iki Boole ifadesini karşılaştırır. İfadelerden en az birinin `true` olması durumunda sonuç `true` olarak elde edilir. Yalnızca her iki ifadenin de `false` olması durumunda sonuç `false` olarak elde edilir.

Girdi	Çıktı(! Girdi)
<code>true true</code>	<code>true</code>
<code>true false</code>	<code>true</code>
<code>false true</code>	<code>true</code>
<code>false false</code>	<code>false</code>

Tablo 5: VEYA Operatörü (||)(OR)

2.4.2 VE Operatörü (&&)(AND)

Bu operatör, iki Boole (mantıksal) ifadesini karşılaştırır. Yalnızca her iki ifadenin de `true` olması durumunda sonuç `true` olarak elde edilir. İfadelerden en az birinin `false` olması durumunda sonuç `false` olarak elde edilir.

Girdi	Çıktı(! Girdi)
<code>true && true</code>	<code>true</code>
<code>true && false</code>	<code>false</code>
<code>false && true</code>	<code>false</code>
<code>false && false</code>	<code>false</code>

Tablo 6: VE Operatörü (&&)(AND))

2.4.3 DEĞİL Operatörü(!) (NOT)

Bu operatör, tek bir Boole ifadesi üzerinde işlem yaparak ifadenin mantıksal değerini tersine çevirir. Eğer ifade `'true'` ise sonuç `'false'`, eğer ifade `'false'` ise sonuç `'true'` olarak elde edilir.

Girdi	Çıktı(! Girdi)
<code>true</code>	<code>false</code>
<code>false</code>	<code>true</code>

Tablo 7: DEĞİL Operatörü(!) (NOT)

3 Karar Yapıları (if-else, switch-case)

Herhangi bir programda, belirli koşullara bağlı olarak farklı eylemler gerçekleştirmek, yapısal programlama dillerinin temel ve ortak bir prensibidir. Programın akışını dinamik hale getiren ve çeşitli senaryolarla uyum sağlamasına olanak tanıyan karar yapıları, bu noktada kritik bir rol oynar.

Karar yapıları temel olarak if-else ve switch-case olmak üzere iki ana kategoriye ayrılır. Bu bölümde, söz konusu karar yapıları detaylı bir şekilde açıklanacak, kullanım şekilleri örneklerle pekiştirilecek ve özellikle if-else yapısının farklı varyasyonlarından biri olan else-if yapısı da ek olarak incelenecektir.

3.1 If-else Yapısı

Programlamada, belirli bir koşulun doğruluğuna veya yanlışlığına göre farklı kod bloklarının çalıştırılması, algoritmaların temelini oluşturan en kritik prensiplerden biridir. if-else yapısı, programın bu tür koşullu dallanmaları yönetmesini sağlayan temel bir karar kontrol mekanizmasıdır. Bu yapı sayesinde, tanımlanan bir koşul true (doğru) ise belirli bir kod bloğu işletilirken, koşul false (yanlış) ise alternatif bir kod bloğu veya hiçbir işlem yapılmadan program akışına devam edilir.

3.1.1 Söz Dizisi (Syntax)

if-else yapısının genel sözdizimi aşağıdaki gibidir:

```
1 if (koşul) {  
2     // Koşul doğru (true) ise çalışacak kod bloğu  
3 } else {  
4     // Koşul yanlış (false) ise çalışacak kod bloğu  
5 }
```

Açıklama:

- **if**: Bu anahtar kelime, koşullu bloğu başlatır.
- **(koşul)**: Parantez içinde değerlendirilmesi gereken mantıksal ifade (karşılaştırma operatörleri ve/veya mantıksal operatörler içeren bir ifade) bulunur. Bu koşul true (doğru) veya false (yanlış) bir değer döndürmelidir.
- **{ }**: Küme parantezleri, koşul doğru olduğunda çalıştırılacak olan kod bloğunu (ifadesini

veya ifadelerini) kapsar. Tek bir ifade varsa bu parantezler bazı dillerde isteğe bağlı olabilir, ancak birden fazla ifade için zorunludur ve kodun okunabilirliği açısından her zaman kullanılması önerilir.

- **else:** if bloğundaki koşul false (yanlış) olarak değerlendirildiğinde çalıştırılacak olan alternatif kod bloğunu belirtir. else bloğu, isteğe bağlı bir yapıdır; programın, yalnızca belirli bir koşul doğru olduğunda işlem yapmasını, aksi takdirde herhangi bir özel eylemde bulunmamasını istediğimiz durumlarda kullanılmayabilir
- **else ... :** Bu küme parantezleri de else durumu için çalışacak kod bloğunu içerir.

3.1.2 if-else Yapısı Örnek

Aşağıdaki örnekte, bir kişinin yaşının 18'den büyük veya eşit olup olmadığını kontrol eden basit bir if-else yapısı gösterilmiştir. Eğer yaş 18 veya daha büyükse "Resitsiniz.", aksi takdirde "Resit degilsiniz." mesajı ekrana yazdırılacaktır.

```
1 #include <stdio.h> // printf ve scanf fonksiyonları için gerekli kütüphane
2
3 int main() {
4     int yas; // Yas değişkeni tanımlandı
5
6     printf("Yasınızı giriniz: "); // Kullanıcıdan yaş girmesini iste
7     scanf("%d", &yas); // Kullanıcının girdiği yaşı oku ve 'yas' değiş
8     kenine ata
9
10    if (yas >= 18) {
11        printf("Resitsiniz.\n"); // Koşul doğruysa bu blok çalışır
12    } else {
13        printf("Resit degilsiniz.\n"); // Koşul yanlışsa bu blok çalışır
14    }
15 }
```

Örnek Kodu Açıklaması:

1. **#include <stdio.h> :**Standart giriş/çıkış fonksiyonlarını (örneğin printf ve scanf) kullanabilmek için gerekli olan başlık dosyası dahil edilir.
2. **int main() ... :** C programlarının ana yürütme bloğudur. Program buradan başlar.
3. **int yas; :** yas adında, tam sayı (integer) tipinde bir değişken tanımlanmıştır.

4. **printf("Yasinizi giriniz: ");**: Ekrana "Yasinizi giriniz: " mesajını yazdırır. \n yeni satıra geçmek içindir, ancak burada kullanılmadı, çünkü kullanıcı girdisi aynı satırda bekleniyor.
5. **scanf("%d", &yas);**: Kullanıcının klavyeden girdiği tam sayıyı (%d format belirleyicisi ile) okur ve bu değeri yas değişkeninin bellekteki adresine (&yas) kaydeder.
6. **if (yas >= 18)**: Burada bir koşul ifadesi (yas >= 18) kontrol edilir. Bu ifade, yas değişkeninin değerinin 18'e eşit veya 18'den büyük olup olmadığını sorgular. Bu, bir karşılaştırma operatörü (>=) kullanılarak yapılan mantıksal bir testtir.
7. **printf("Resitsiniz.\n");**: Eğer yas >= 18 koşulu true (doğru) olarak değerlendirilirse, yani kişinin yaşı 18 veya daha büyükse, bu kod satırı çalışır ve ekrana "Resitsiniz." yazdırılır. \n yeni satıra geçmeyi sağlar.
8. **else**: Eğer if parantezindeki koşul false (yanlış) olarak değerlendirilirse, yani kişinin yaşı 18'den küçükse, program akışı else bloğuna geçer.
9. **printf("Resit degilsiniz.\n");**: else bloğu içindeki bu kod satırı çalışır ve ekrana "Resit degilsiniz." yazdırılır.

3.2 else-if Yapısı

Bir programda birden fazla koşulu ardışık olarak kontrol etmek ve her bir koşula özel eylemler tanımlamak gerektiğinde, sadece if ve else yapıları yeterli olmayabilir. Bu gibi durumlarda, else-if (veya bazı dillerde elif) yapısı, zincirleme koşul kontrolleri için esnek bir çözüm sunar. Bu yapı sayesinde, ilk koşul yanlış olduğunda bir sonraki koşul kontrol edilir ve bu şekilde doğru bir koşul bulunana veya tüm koşullar tükenene kadar devam edilir.

3.2.1 Söz Dizisi (Syntax)

else-if yapısının genel sözdizimi aşağıdaki gibidir:

```
1 if (ilk_koşul) {  
2     // İlk koşul doğruysa çalışacak kod bloğu  
3 } else if (ikinci_koşul) {  
4     // İlk koşul yanlış, ancak ikinci koşul doğruysa çalışacak kod bloğu  
5 } else if (üçüncü_koşul) {  
6     // İlk ve ikinci koşullar yanlış, ancak üçüncü koşul doğruysa çalışacak  
7     kod bloğu  
8 } else {
```

```
8 // Yukarıdaki koşulların hiçbirisi doğru değilse çalışacak kod bloğu (
   isteğe bağlı)
9 }
```

Açıklama:

- **if (ilk_koşul):** Zincirin başlangıcıdır. Eğer ilk_koşul doğruysa, bu blok çalışır ve diğer else-if veya else blokları kontrol edilmez.
- **else if (ikinci_koşul):** if bloğundaki ilk_koşul yanlış olduğunda devreye girer. Eğer ikinci_koşul doğruysa, bu blok çalışır ve sonraki kontroller durur. Bir if yapısının ardından birden fazla else if bloğu eklenebilir.
- **else:** Tüm if ve else-if koşullarının yanlış olduğu durumda çalışan isteğe bağlı bir son bloktur. Eğer hiçbir koşul sağlanmazsa varsayılan bir eylem gerçekleştirmek için kullanılır.

3.2.2 else-if Yapısı Örnek

Aşağıdaki örnekte, bir öğrencinin sınav notuna göre farklı harf notları (A, B, C, D veya F) atayan bir else-if zinciri gösterilmiştir.

```
1 #include <stdio.h> // printf ve scanf fonksiyonları için gerekli kütüphane
2
3 int main() {
4     int not; // Not değişkeni tanımlandı
5
6     printf("Öğrencinin notunu giriniz: "); // Kullanıcıdan not girmesini
       iste
7     scanf("%d", &not); // Kullanıcının girdiği notu oku ve 'not' değiş
       kenine ata
8
9     if (not >= 90) {
10         printf("Harf Notu: A\n");
11     } else if (not >= 80) {
12         printf("Harf Notu: B\n");
13     } else if (not >= 70) {
14         printf("Harf Notu: C\n");
15     } else if (not >= 60) {
16         printf("Harf Notu: D\n");
```

```

17     } else {
18         printf("Harf Notu: F\n");
19     }
20
21 }

```

Örnek Kodu Açıklaması:

1. **#include <stdio.h> ve int main() ...**: Önceki örnekteki gibi standart kütüphane ve ana fonksiyon.
2. **int not;**: not adında bir tam sayı değişkeni tanımlanmıştır.
3. **printf("Ogrencinin notunu giriniz: "); ve scanf("%d", ¬);**: Kullanıcıdan not girmesini ister ve girilen notu not değişkenine atar.
4. **if (not >= 90)**: Program, not değişkeninin 90 veya daha büyük olup olmadığını kontrol eder. Eğer koşul doğruysa, "Harf Notu: A" yazdırılır ve diğer else if blokları kontrol edilmez. Koşul yanlışsa, akış bir sonraki else if bloğuna geçer.
5. **else if (not >= 80)**: Eğer ilk koşul yanlışsa, bu koşul kontrol edilir. not 80 veya daha büyükse "Harf Notu: B" yazdırılır ve sonraki bloklar atlanır. Bu süreç, not >= 70 ve not >= 60 koşulları için de devam eder.
6. **else**: Eğer yukarıdaki if veya hiçbir else if koşulu doğru değilse (yani not 60'ın altındaysa), bu else bloğu çalışır ve ekrana "Harf Notu: F" yazdırılır.

3.3 switch-case Yapısı

Programlamada, bir değişkenin alabileceği farklı değerlere göre farklı kod bloklarının çalıştırılması gereken durumlar sıkça ortaya çıkar. Bu tür senaryolarda, özellikle birden fazla else-if koşulu yerine, daha düzenli ve okunabilir bir alternatif olarak switch-case yapısı kullanılır. switch-case yapısı, belirli bir ifadenin değerini birden fazla sabit değerle karşılaştırır ve eşleşen değere karşılık gelen kod bloğunu yürütür.

3.3.1 Söz Dizisi (Syntax)

switch-case yapısının genel sözdizimi aşağıdaki gibidir:

```

1 switch (ifade) {
2     case sabit_değer1:
3         // ifade, sabit_değer1 ile eşleşirse çalışacak kod bloğu

```

```

4      break; // Bu case'den çıkmak için kullanılır
5  case sabit_değer2:
6      // ifade, sabit_değer2 ile eşleşirse çalışacak kod bloğu
7      break;
8      // ... daha fazla case bloğu olabilir ...
9  default:
10     // ifadenin hiçbir case değeriyle eşleşmemesi durumunda çalışacak
        kod bloğu (isteğe bağlı)
11     break; // default'tan sonra da kullanılabilir
12 }

```

Açıklama:

- **switch (ifade):** Kontrol edilecek değişken veya ifadenin yer aldığı kısımdır. Bu ifade genellikle bir tam sayı, karakter veya bazen bir string (dile bağlı olarak) tipinde olmalıdır.
- **case sabit_değerN: :** switch parantezindeki ifadenin alabileceği olası sabit bir değeri belirtir. Eğer ifadenin değeri sabit_değerN ile eşleşirse, bu case bloğunun altındaki kodlar çalıştırılmaya başlanır.
- **break; :** Her case bloğunun sonunda kullanılan break anahtar kelimesi çok önemlidir. break, eşleşen case bloğunun yürütülmesi tamamlandıktan sonra switch yapısından çıkılmasını sağlar. Eğer break kullanılmazsa, program akışı bir sonraki case bloğuna "düşer" (fall-through) ve eşleşmeyen case'lerin kodları da çalışmaya devam eder, bu da beklenmedik sonuçlara yol açabilir.
- **default: :** İsteğe bağlı bir bloktur. Eğer switch ifadesinin değeri hiçbir case sabitiyle eşleşmezse, default bloğu içindeki kodlar çalıştırılır. Bu, if-else if zincirindeki son else bloğuna benzer bir işlev görür. default bloğunun konumu genellikle en sondadır, ancak zorunlu değildir.

3.3.2 Örnek Kod 1 (Normal Kullanım - break ile):

```

1  #include <stdio.h> // printf ve scanf fonksiyonları için gerekli kütüphane
2
3  int main() {
4      int gunNo;
5
6      printf("1-3 arasi bir sayi giriniz (Haftanin gununu ogrenmek icin): ");

```

```

7   scanf("%d", &gunNo);
8
9   switch (gunNo) {
10      case 1:
11          printf("Bugun Pazartesi.\n");
12          break; // Burada switch'ten çıkılır
13      case 2:
14          printf("Bugun Sali.\n");
15          break; // Burada switch'ten çıkılır
16      case 3:
17          printf("Bugun Carsamba.\n");
18          break; // Burada switch'ten çıkılır
19      default:
20          printf("Gecersiz giris yaptiniz. Lutfen 1, 2 veya 3 giriniz.\n"
21                );
22          break; // Default bloğundan sonra da break kullanmak iyi bir
23                pratiktir
    }
}

```

Örnek Kodu Açıklaması:

- **int gunNo;** : Kullanıcının gireceği gün numarasını tutmak için bir değişken tanımlanır.
- **scanf("%d", &gunNo);** : Kullanıcıdan bir tam sayı alınıp gunNo değişkenine atanır.
- **switch (gunNo):** gunNo değişkeninin değeri kontrol edilmek üzere switch yapısına verilir.
- **case 1:, case 2:, case 3:** : gunNo değişkeninin alabileceği olası değerler (1, 2, 3) ile karşılaştırılır. Eğer gunNo değeri bir case ile eşleşirse, o case bloğunun altındaki kod (printf) çalıştırılır.
- **break;** : Her case bloğunun sonunda kullanılan break; ifadesi, ilgili case bloğu çalıştıktan sonra programın switch yapısından tamamen çıkmasını sağlar. Bu sayede, yalnızca eşleşen case'e ait kodlar yürütülür.
- **default:** : Eğer gunNo değeri hiçbir case sabitiyle eşleşmezse, default bloğu çalışır ve "Gecersiz giris yaptiniz..." mesajı ekrana yazdırılır.

Beklenen Çıktı (gunNo = 3 için):

```
1-3 arasi bir sayi giriniz (Haftanin gununu ogrenmek icin):3
Bugun Pazartesi.
```

3.3.3 Örnek 2: Fall-through Durumu (break kullanılmadığında)

Bu örnek, bilerek break anahtar kelimesini kullanmayarak fall-through (düşme) durumunu göstermektedir. Bu durumda, bir case eşleştiğinde, o case'den sonraki tüm case blokları (break görene kadar veya switch bitene kadar) sırayla çalışmaya devam eder.

```
1 #include <stdio.h> // printf fonksiyonu için gerekli kütüphane
2
3 int main() {
4     int sehirKodu = 9; // Aydın'ın kodu
5
6     printf("Sehir kodu: %d\n", sehirKodu);
7
8     switch (sehirKodu) {
9         case 1:
10             printf("Adana\n");
11             // break; -- Buraya break konulmadığı için fall-through olacak
12         case 9:
13             printf("Aydın\n");
14             // break; -- Buraya break konulmadığı için fall-through olacak
15         case 6:
16             printf("Ankara\n");
17             break; // Burada break olduğu için sonraki case'ler çalışmaz
18         case 34:
19             printf("İstanbul\n");
20             break;
21         default:
22             printf("Bilinmeyen Sehir Kodu.\n");
23             break;
24     }
25
26 }
```

Örnek Kodu Açıklaması:

- **int sehirKodu = 9;** sehirKodu adında bir değişken tanımlanmış ve değeri 9 olarak atanmıştır (Aydın'ın plakası).
- **switch (sehirKodu):** sehirKodu değeri switch ifadesine verilir.
- **case 9:** ile eşleşme bulunur. Bu case altındaki printf("Aydın\n"); kodu çalışır.
- **break;** kullanılmadığı için program akışı case 9: bloğundan hemen sonraki case 6: bloğuna düşer (fall-through).
- **case 6:** bloğunun altındaki printf("Ankara\n"); kodu da çalışır.
- **case 6:** bloğunun sonunda break; olduğu için, program switch yapısından çıkar.

Beklenen Çıktı (sehirKodu = 9 için):

```
Sehir kodu: 9
Aydın
Ankara
```

Görüldüğü gibi, sehirKodu 9 olmasına rağmen Aydın ile birlikte Ankara da yazdırılmıştır. Bu durum, break kullanılmadığında fall-through mekanizmasının nasıl çalıştığını göstermektedir ve genellikle istenmeyen bir durumdur. Ancak bazı özel durumlarda (break kullanmadan birden fazla case için aynı kodu çalıştırmak gibi) kasıtlı olarak kullanılabilir.

4 Döngü Yapıları

Programlamada, belirli bir işlem kümesinin veya kod bloğunun belirli bir koşul sağlandığı sürece veya belirli sayıda tekrarlanması gereken durumlar sıkça karşılaşılan bir ihtiyaçtır. Bu tekrarlı işlemleri manuel olarak yazmak yerine, kodun daha verimli, okunabilir ve yönetilebilir olmasını sağlayan döngü yapıları kullanılır. Döngüler, bir görevi belirlenen sayıda veya bir koşul yanlış olana dek otomatik olarak yineleyerek programcının iş yükünü önemli ölçüde azaltır.

Bu bölümde, programlama dillerinde en yaygın kullanılan döngü yapıları olan for döngüsü, while döngüsü ve do-while döngüsü' detaylı bir şekilde incelenecek, çalışma mantıkları açıklanacak ve her biri için C dilinde örnekler sunulacaktır.

4.1 For Döngüsü

for döngüsü, özellikle belirli bir kod bloğunun önceden belirlenmiş sayıda veya kesin olarak bilinen bir aralıkta tekrar etmesi gereken durumlarda kullanılan yapısal bir kontrol mekanizmasıdır. Bu döngü türü, döngünün başlangıç değerini, sonlanma koşulunu ve her iterasyondaki (tekrarlama) artış veya azalış miktarını tek bir satırda, döngü tanımının içerisinde barındırır. Bu entegre yapısı sayesinde, bir kod bloğunun kaç kez yineleneyeceği net bir şekilde bilindiğinde, while döngüsüne kıyasla daha kompakt ve düzenli bir çözüm sunar.

4.1.1 Söz Dizisi (Syntax)

For Döngüsünün genel sözdizimi aşağıdaki gibidir:

```
1 for (başlangıç_değeri; koşul; artış_azalış) {  
2     // Koşul doğru olduğu sürece tekrar edecek kod bloğu  
3 }
```

Açıklama:

- **başlangıç_değeri (Initialization):** Döngü başlamadan önce sadece bir kez çalışan kısımdır. Genellikle bir döngü değişkeninin (sayaç) başlangıç değeri bu bölümde tanımlanır ve atanır. Örneğin: `int i = 0;`
- **koşul (Condition):** Her döngü iterasyonunun başında kontrol edilen mantıksal ifadedir. Eğer bu koşul `true` (doğru) olarak değerlendirilirse, döngü bloğu çalıştırılır. Koşul `false` (yanlış) olduğunda döngü sona erer. Örneğin: `i < 10;`
- **artış_azalış (Increment/Decrement):** Döngü bloğu her çalıştıktan sonra yürütülen kısımdır. Genellikle döngü değişkeninin değerini artırmak veya azaltmak için kullanılır. Örneğin: `i++` (artırma) veya `i--` (azaltma).
- **{ }:** Küme parantezleri, koşul doğru olduğu sürece tekrar edilecek olan kod bloğunu kapsar.

4.1.2 For Döngüsüne Örnek

```
1 #include <stdio.h> // printf fonksiyonu için gerekli kütüphane  
2  
3 int main() {  
4     int i; // Döngü sayacı olarak kullanılacak değişken  
5 }
```

```

6   printf("1'den 10'a kadar sayılar:\n");
7
8   // Döngü tanımı:
9   // Başlangıç: i = 1 (i'yi 1'den başlat)
10  // Koşul: i <= 10 (i 10'a eşit veya küçük olduğu sürece devam et)
11  // Artış: i++ (Her döngü sonunda i'yi 1 artır)
12  for (i = 1; i <= 10; i++) {
13      printf("%d ", i); // i'nin değerini ekrana yazdır ve bir boşluk bırak
14  }
15
16  printf("\n"); // Son sayının ardından yeni satıra geç
17 }

```

Beklenen Çıktı

```

1'den 10'a kadar sayılar:
1 2 3 4 5 6 7 8 9 10

```

Örnek Kodu Açıklaması:

- **#include <stdio.h> ve int main() { ... }:** Standart kütüphane ve ana fonksiyon.
- **int i;** Döngü kontrol değişkeni i tanımlanır.
- **printf("1'den 10'a kadar sayılar:\n");** Kullanıcıya neyin yazdırılacağını belirten bir başlık yazdırılır.
- **for (i = 1; i <= 10; i++):** Bu satır for döngüsünü başlatır:
- **i = 1:** Döngü başlamadan önce i değişkenine 1 değeri atanır. Bu, döngünün ilk çalışacağı andaki başlangıç noktasıdır.
- **i <= 10:** Her döngü iterasyonundan önce bu koşul kontrol edilir. Eğer i'nin değeri 10'dan küçük veya 10'a eşitse, koşul doğru (true) kabul edilir ve döngü bloğu çalıştırılır. Aksi takdirde (i 10'dan büyük olduğunda), koşul yanlış (false) olur ve döngü sona erer.
- **i++:** Döngü bloğu her çalıştırıldıktan sonra, i değişkeninin değeri 1 artırılır. Bu, döngünün her adımda bir sonraki sayıya geçmesini sağlar.
- **printf("%d ", i);** Döngü her çalıştığında, o anki i değeri ekrana yazdırılır. %d format belirleyicisi tam sayı çıktısı için kullanılır. (boşluk), sayıların arasında boşluk bırakılma-

sını sağlar.

- **printf("\n");** : Tüm sayılar yazdırıldıktan sonra ekranda temiz bir görünüm için yeni bir satıra geçilir.

4.2 while Döngüsü

while döngüsü, belirli bir kod bloğunun, tanımlanan bir koşul true (doğru) olduğu sürece sürekli olarak yürütülmesini sağlayan temel bir kontrol yapısıdır. for döngüsünden farklı olarak, while döngüsü genellikle döngünün kaç kez tekrar edeceğinin başlangıçta kesin olarak bilinmediği durumlarda tercih edilir. Döngü, koşul false (yanlış) olana kadar veya döngü içinden bir break ifadesi ile manuel olarak sonlandırılana kadar çalışmaya devam eder. Koşul, döngü bloğu çalıştırılmadan önce her iterasyonun başında kontrol edilir.

4.2.1 Sözdizimi (Syntax)

while döngüsünün genel sözdizimi aşağıdaki gibidir:

```
1 while (koşul) {  
2     // Koşul doğru olduğu sürece tekrar edecek kod bloğu  
3     // Döngüyü sonlandıracak bir mekanizma (koşulu değiştiren bir ifade)  
4     burada olmalıdır  
5 }
```

Açıklama:

- **while:** Döngüyü başlatan anahtar kelimedir.
- **(koşul):** Parantez içinde yer alan mantıksal ifadedir. Döngü bloğu, bu koşul true olduğu sürece tekrar tekrar çalıştırılır. Koşul false olduğunda döngü sona erer.
- **{ }:** Küme parantezleri, koşul doğru olduğu sürece tekrar edilecek olan kod bloğunu kapsar.
- **Döngüyü Sonlandıracak Mekanizma:** while döngüsünün sonsuz bir döngüye girmesi için, döngü bloğu içinde koşulu zamanla false yapacak (örneğin bir sayacı artırma veya kullanıcıdan çıkış onayı alma gibi) bir mekanizmanın bulunması zorunludur. Aksi takdirde, koşul her zaman doğru kalacak ve program sonsuz döngüye girecektir.

4.2.2 while Döngüsü Örnek

Aşağıdaki örnekte, kullanıcı "q" karakterini girene kadar ekrana bir mesaj yazdıran ve kullanıcıdan giriş almaya devam eden bir while döngüsü gösterilmiştir. Bu örnek, döngünün kaç kez çalışacağını önceden belli olmadığı durumlarda nasıl kullanıldığını güzelce açıklar.

```
1 #include <stdio.h> // printf ve getchar fonksiyonları için gerekli kütü
   phane
2 #include <conio.h> // getch veya _getch fonksiyonu için (isteğe bağlı,
   Windows için)
3
4 int main() {
5     char karakter; // Kullanıcının gireceği karakteri tutacak değişken
6
7     printf("Programdan cikmak icin 'q' tusuna basin.\n");
8     printf("Devam etmek icin herhangi bir tusa basin...\n");
9
10    // getchar() yerine _getch() veya getch() kullanmak, enter'a basmadan
       karakter almayı sağlar.
11    // Ancak daha taşınabilir olması için getchar() tercih edilebilir.
12    // Eğer anında karakter girişi isteniyorsa, işletim sistemine özel kütü
       phaneler gerekebilir (örn. conio.h'deki getch()).
13    // Bu örnekte getchar() kullanıldı, dolayısıyla her karakterden sonra
       Enter'a basmak gerekecek.
14
15    karakter = getchar(); // İlk karakteri oku
16
17    while (karakter != 'q') { // Karakter 'q' olmadığı sürece döngü devam
       eder
18        printf("Devam ediliyor... Baska bir karakter girin (cikmak icin 'q
       ').\n");
19        karakter = getchar(); // Yeni karakteri oku (Enter'a basana kadar
       bekler)
20    }
21
22    printf("Programdan cikildi.\n");
23
24 }
```

Örnek Kodu Açıklaması:

1. **#include <stdio.h> ve int main() { ... }:** Standart kütüphane ve ana fonksiyon. (<conio.h> Windows için anında karakter girişi sağlamak amacıyla eklendi ancak getchar() kullanıldığı için zorunlu değil).
2. **char karakter;** : Kullanıcının gireceği tek bir karakteri tutmak için char (karakter) tipinde bir değişken tanımlanır.
3. **karakter = getchar();** : Programın başında kullanıcıdan ilk karakteri almasını sağlar. getchar() fonksiyonu, klavyeden bir karakter okur (ve Enter tuşuna basılana kadar bekler).
4. **while (karakter != 'q');** : Burası while döngüsünün koşuludur. Döngü, karakter değişkeninin değeri 'q' (küçük 'q' harfi) olmadığı sürece çalışmaya devam eder. Eğer kullanıcı 'q' girerse, koşul false olur ve döngü sona erer.
5.
 - **printf("Devam ediliyor... Baska bir karakter girin (cikmak icin 'q').\n");** : Kullanıcıya döngünün devam ettiğini ve çıkmak için 'q' girmesi gerektiğini bildiren bir mesaj yazdırılır.
 - **karakter = getchar();** : Bu satır çok önemlidir. Döngünün her iterasyonunda kullanıcıdan yeni bir karakter alınır ve karakter değişkeni güncellenir. Bu, döngüyü sonsuz bir döngüden kurtaran ve çıkış koşulunun zamanla true (yani karakter == 'q') olmasını sağlayabilen mekanizmadır. Eğer bu satır olmazsa, karakter'in değeri hiç değişmeyeceği için döngü sonsuza kadar çalışırdı.
6. **printf("Programdan cikildi.\n");** : Döngü sona erdiğinde (kullanıcı 'q' girdiğinde), bu mesaj ekrana yazdırılır.

5 Kontrol İfadeleri: break ve continue

Programlama dillerinde döngülerin varsayılan akışını değiştirmek için kullanılan iki önemli kontrol ifadesi bulunmaktadır: break ve continue. Bu ifadeler, döngünün tamamlanma şeklini veya mevcut iterasyonun nasıl ilerleyeceğini dinamik olarak belirlemeye olanak tanır.

5.1 break İfadesi

break ifadesi, kullanıldığı döngüden (örneğin for, while, do-while veya switch-case yapısından) anında çıkılmasını sağlar. break ile karşılaşıldığında, döngünün kalan iterasyonları atlanır ve program akışı, döngünün hemen dışındaki ilk komut satırından devam eder. Bu ifade genellikle belirli bir koşul karşılandığında döngüyü erken sonlandırmak için kullanılır.

5.1.1 Örnek kullanım mantığı (genel syntax/if karar yapısı ve while döngüsü ile):

```
1 while (koşul) {  
2     // Bazı işlemler...  
3     if (erken_cikis_koşulu) {  
4         break; // Döngüden çık  
5     }  
6     // Kalan işlemler...  
7 }
```

5.2 continue İfadesi

continue ifadesi, kullanıldığı döngüde mevcut iterasyonun geri kalan kısmını atlamasını ve döngünün bir sonraki iterasyonuna geçmesini sağlar. continue ile karşılaşıldığında, döngü bloğu içindeki continue'dan sonra gelen tüm kodlar atlanır ve döngünün koşulu (örneğin for döngüsünde koşul kısmı, ardından artış_azalış kısmı) tekrar kontrol edilerek bir sonraki döngü adımına geçilir. Bu ifade genellikle, belirli bir koşul sağlandığında o iterasyondaki bazı işlemleri pas geçip döngüye devam etmek için kullanılır.

5.2.1 Örnek kullanım mantığı (genel syntax/for döngüsü ile):

```
1 for (başlangıç_değeri; koşul; artış_azalış) {  
2     // Döngü başlangıcındaki işlemler...  
3     if (iterasyonu_atlama_koşulu) {  
4         continue; // Bu iterasyonun kalanını atla, bir sonraki iterasyona  
                    geç  
5     }  
6     // Döngü devamındaki işlemler...  
7 }
```

```

5     }
6     // Sadece atlama koşulu sağlanmadığında çalışacak işlemler...
7 }

```

6 Diziler

Programlamada, tekil değişkenler yerine aynı türden (örneğin, hepsi tamsayı veya hepsi karakter) çok sayıda veriyi bir arada, düzenli bir şekilde depolama ve bu verilere erişme ihtiyacı sıkça doğar. İşte bu noktada diziler (arrays) devreye girer. Bir dizi, aynı veri tipindeki elemanların bellekte ardışık (peş peşe) olarak depolandığı, sabit boyutlu bir veri yapısıdır. Bu yapı, benzer verilerin tek bir isim altında gruplanmasını sağlayarak, verilere indeks (konum) numaraları aracılığıyla kolayca erişim ve işlem yapma imkanı sunar.

Bu bölümde, dizilerin temel mantığı, tanımlama ve başlatma şekilleri, elemanlara erişim yöntemleri ve dizilerle gerçekleştirilebilecek yaygın işlemler C dili örnekleriyle açıklanacaktır.

6.1 Dizilerin Tanımlanması ve Başlatılması (Sözdizimi/Syntax)

Diziler, kullanılmadan önce tanımlanmalı ve bellek ayrılmalıdır. C dilinde bir dizi tanımlarken, dizinin saklayacağı veri tipini, dizinin adını ve köşeli parantez içinde eleman sayısını (boyutunu) belirtiriz. Dizilerin indeksleri (konum numaraları) 0'dan başlar ve boyut-1'e kadar devam eder.

6.1.1 Genel Sözdizimi

```

1 veriTipi diziAdi[boyut]; // Diziye tanımlama

```

6.1.2 veya başlatma ile birlikte Sözdizimi:

```

1 veriTipi diziAdi[boyut] = {değer1, değer2, ..., değerN}; // Diziye tanımlama ve başlatma

```

6.1.3 veya boyutu belirtmeden, başlangıç değerlerinden boyutunu çıkarmasına izin verme ile Sözdizimi:

```

1 veriTipi diziAdi[] = {değer1, değer2, ..., değerN}; // Boyutu otomatik belirlenen dizi tanımlama ve başlatma

```

Açıklama:

- **veriTipi:** Dizinin depolayacağı elemanların veri tipini belirtir (örneğin int, char, float, double vb.). Bir dizideki tüm elemanlar aynı veri tipinde olmalıdır.
- **diziAdi:** Diziye verilen isimdir. Değişken isimlendirme kurallarına uymalıdır.
- **[boyut]:** Dizinin kaç tane eleman saklayacağını belirten tam sayı bir ifadedir. Bu, dizinin bellekte ne kadar yer kaplayacağını sabitler. Boyut, tanımlama sırasında bilinen bir sabit veya sabit bir ifade olmalıdır (C99 standardından sonra değişken boyutlu diziler mümkün olsa da, genelde sabit boyut tercih edilir).
- **= ve {} (Başlatma):** Dizi tanımlanırken aynı zamanda `={ }` süslü parantezler içinde başlangıç değerleri atanabilir. Eğer eleman sayısı boyuttan az ise, kalan elemanlar otomatik olarak 0 (veya ilgili veri tipinin varsayılan boş değeri) ile başlatılır.
- **diziAdi[] (Boyut Belirtmeme):** Eğer dizi tanımlanırken başlangıç değerleri de veriliyorsa, boyut kısmı boş bırakılabilir. Bu durumda derleyici, verilen başlangıç değerlerinin sayısına göre dizinin boyutunu otomatik olarak belirler.

6.1.4 Dizinin Elemanlarına Erişme

Dizi elemanlarına, dizinin adı ve elemanın indeks numarası (konumu) kullanılarak erişilir. İndeksler her zaman 0'dan başlar.

```
1 diziAdi[indeks]; // Belirli bir indeksteki elemana erişim
```

Örnek: `int sayilar[5];` tanımlanmış bir dizideki ilk elemana `sayilar[0]` ile, son elemana ise `sayilar[4]` ile erişilir.

6.1.5 Diziler - Temel Örnek

Aşağıdaki örnekte, 5 elemanlı bir tam sayı dizisi tanımlanmış, elemanlarına değerler atanmış ve ardından bir for döngüsü kullanılarak bu elemanlar ekrana yazdırılmıştır. Bu örnek, dizilerin nasıl başlatıldığını ve indeksler aracılığıyla elemanlarına nasıl erişildiğini gösterir.

6.1.6 Örnek Kod

```
1 #include <stdio.h> // printf fonksiyonu için gerekli kütüphane
2
3 int main() {
4     // 1. Diziyi tanımlama ve başlangıç değerleri ile başlatma
```

```

5      int sayilar[] = {10, 20, 30, 40, 50}; // Boyut belirtilmedi, derleyici
        5 olarak algılar
6
7      // Veya 5 elemanlı bir dizi tanımlayıp tek tek değer atama:
8      // int sayilar[5];
9      // sayilar[0] = 10;
10     // sayilar[1] = 20;
11     // sayilar[2] = 30;
12     // sayilar[3] = 40;
13     // sayilar[4] = 50;
14
15     int i; // Döngü sayacı
16
17     printf("Dizinin elemanlari:\n");
18
19     // Dizinin elemanlarını bir döngü ile ekrana yazdırma
20     // Dizinin boyutu 5 olduğu için indeksler 0'dan 4'e kadardır.
21     for (i = 0; i < 5; i++) {
22         printf("sayilar[%d]: %d\n", i, sayilar[i]);
23     }
24
25     // Tek bir elemana doğrudan erişim örneği
26     printf("\nDizinin 3. elemanı (indeks 2): %d\n", sayilar[2]);
27
28     // Bir elemanın değerini değiştirme örneği
29     sayilar[0] = 100;
30     printf("Dizinin ilk elemanı guncellendi: %d\n", sayilar[0]);
31
32 }

```

Beklenen Çıktı:

```

Dizinin elemanlari:
sayilar[0]: 10
sayilar[1]: 20
sayilar[2]: 30
sayilar[3]: 40

```

```
sayilar[4]: 50
```

```
Dizinin 3. elemani (indeks 2): 30
```

```
Dizinin ilk elemani guncellendi: 100
```

Örnek Kodu Açıklaması:

1. **#include <stdio.h> ve int main() { ... }:** Standart kütüphane ve ana fonksiyon.
2. **int sayilar[] = {10, 20, 30, 40, 50};:** sayilar adında bir tam sayı dizisi tanımlanmış ve başlangıç değerleriyle birlikte başlatılmıştır. Boyut belirtilmediği için derleyici, verilen eleman sayısına (5) göre dizinin boyutunu otomatik olarak belirler.
3. **for (i = 0; i < 5; i++):** Bir for döngüsü kullanılarak dizinin tüm elemanları gezilir. Döngü, indeksler 0'dan başladığı için i = 0 ile başlar ve dizi boyutu (5) kadar dönmek için i < 5 koşulu kullanılır.
4. **printf("sayilar[%d]: %d\n", i, sayilar[i]);:** Her iterasyonda, o anki i indeksindeki dizi elemanının (sayilar[i]) değeri ekrana yazdırılır.
5. **sayilar[2]:** Doğrudan indeks kullanarak dizinin üçüncü elemanına (indeks 2) erişilerek değeri yazdırılır.
6. **sayilar[0] = 100;:** Dizinin ilk elemanının (indeks 0) değeri 10'dan 100'e güncellenir. Bu, dizi elemanlarının okunabildiği gibi yazılabildiğini de gösterir.

6.2 Karakter Dizisi İşlemleri Örneği (gets() ve strlen() ile)

Aşağıdaki örnekte, kullanıcıdan bir isim girmesi istenir. gets() fonksiyonu ile bu isim okunur ve isim adındaki karakter dizisine atanır. Ardından strlen() fonksiyonu kullanılarak girilen ismin uzunluğu hesaplanır ve ekrana yazdırılır.

6.2.1 Örnek Kod:

```
1 #include <stdio.h>    // printf ve gets için
2 #include <string.h>   // strlen için
3
4 int main() {
5     char isim[50]; // En fazla 49 karakter + null sonlandırıcı için 50
6                 boyutunda bir karakter dizisi
```

```

7   printf("Lutfen isminizi giriniz: ");
8   // gets() fonksiyonu ile kullanıcıdan metin girişi alınır
9   // DİKKAT: gets() bellek taşması riski taşır. Gerçek uygulamalarda
    fgets() tercih edilmelidir.
10  gets(isim);
11
12  printf("Merhaba, %s!\n", isim); // Girilen ismi ekrana yazdırır
13
14  // strlen() fonksiyonu ile ismin uzunluğu bulunur
15  size_t uzunluk = strlen(isim);
16  printf("Isminizin uzunluğu: %zu karakter.\n", uzunluk); // %zu size_t
    tipi için format belirleyicidir
17
18  }

```

Beklenen Çıktı:

```

Lutfen isminizi giriniz: Zeliş
Merhaba, Zeliş!
Isminizin uzunluğu: 5 karakter.

```

(Not: 'ş' gibi Türkçe karakterler bazı ortamlarda farklı uzunluk gösterebilir, bu örnek ASCII varsayımına göre 5 karakter çıktısı verir.)

Örnek Kodu Açıklaması:

1. **#include <stdio.h> ve #include <string.h>:** Gerekli kütüphaneler dahil edilir. string.h özellikle strlen gibi string fonksiyonları için önemlidir.
2. **char isim[50]; :** isim adında, en fazla 49 karakter ve sonuna eklenecek null sonlandırıcı (\0) için toplam 50 karakterlik bir karakter dizisi (string) tanımlanır.
3. **gets(isim); :** Kullanıcıdan bir satır metin alınır ve isim dizisine kaydedilir. Kullanıcı Enter tuşuna bastığında okuma durur. Güvenlik Uyarısı: gets() fonksiyonu, kullanıcının girdiğinin dizi boyutunu aşmadığından emin olamaz. Bu nedenle güvenlik açıkları oluşturabilir. Örneğin, eğer isim dizisi 50 karakterlik tanımlanmışken kullanıcı 100 karakterlik bir metin girerse, bu durum programın çökmesine veya güvenlik sorunlarına yol açabilir. Bu nedenle, gerçek projelerde fgets() gibi boyut kontrolü yapan fonksiyonlar tercih edilme-

lidir.

4. **printf("Merhaba, %s!\n", isim);** : %s format belirleyicisi kullanılarak isim karakter dizisinin içeriği ekrana yazdırılır.
5. **size_t uzunluk = strlen(isim);** : strlen() fonksiyonu, isim dizisinin içindeki null karakter (\0) öncesindeki karakter sayısını (yani metnin gerçek uzunluğunu) hesaplar ve uzunluk değişkenine atar. size_t genellikle platforma göre değişen, boyut bilgisi tutmak için kullanılan işaretli bir tam sayı tipidir.
6. **printf("İsminizin uzunluğu: %zu karakter.\n", uzunluk);** : Hesaplanan uzunluk ekrana yazdırılır.

6.3 Diziler ve Rastgele Sayılar: rand() ve srand() Örneği

Diziler, bazen belirli bir veri kümesi yerine rastgele değerlerle doldurulması gereken durumlarda da kullanılır. C dilinde sözde-rastgele sayı üretimi için rand() ve bu üretici her çalıştırmada farklı bir başlangıç noktasıyla tohumlamak için srand() fonksiyonları kullanılır. Genellikle srand() fonksiyonu time(NULL) ile tohumlanarak o anki sistem zamanından faydalanılır, böylece program her başlatıldığında farklı rastgele sayılar üretilir.

6.3.1 Örnek Kod :

```
1 #include <stdio.h> // printf için
2 #include <stdlib.h> // rand ve srand için
3 #include <time.h> // time fonksiyonu için (srand'ı tohumlamak için)
4
5 int main() {
6     int rastgeleSayilar[5]; // 5 elemanlı bir tam sayı dizisi
7     int i; // Döngü sayacı
8
9     // Rastgele sayı üreticinin tohumlanması
10    // srand(time(NULL)) fonksiyonu, rand() fonksiyonunun her program çalış-
11    // tırmada farklı
12    // bir rastgele sayı dizisi üretmesini sağlamak için o anki sistem
13    // zamanını tohum olarak kullanır.
14    srand(time(NULL));
15
16    printf("Rastgele Sayılar Dizisi:\n");
```

```

16 // Diziyi 1 ile 100 arasında rastgele sayılarla doldurma
17 for (i = 0; i < 5; i++) {
18     rastgeleSayilar[i] = (rand() % 100) + 1; // rand() ile 1-100 arası
19     sayı üret
20     printf("rastgeleSayilar[%d]: %d\n", i, rastgeleSayilar[i]);
21 }
22 }

```

Örnek Çıktı (Her çalıştırmada farklı olacaktır):

```

Rastgele Sayılar Dizisi:
rastgeleSayilar[0]: 45
rastgeleSayilar[1]: 12
rastgeleSayilar[2]: 87
rastgeleSayilar[3]: 3
rastgeleSayilar[4]: 66

```

Örnek Kodu Açıklaması:

1. **#include <stdlib.h> ve #include <time.h>** : rand(), srand() ve time() fonksiyonlarını kullanabilmek için gerekli başlık dosyaları.
2. **srand(time(NULL));** : Bu satır, rand() fonksiyonunun üreteceği sayı dizisini başlatmak için kullanılır. time(NULL) fonksiyonu, o anki sistem zamanını döndürerek her program başlangıcında farklı bir tohum değeri sağlar ve böylece her çalıştırmada farklı rastgele sayılar elde edilir.
3. **for (i = 0; i < 5; i++)** : Döngü, dizinin her bir elemanını doldurmak için 5 kez döner.
4. **rastgeleSayilar[i] = (rand() % 100) + 1;** : rand() fonksiyonu 0 ile RAND_MAX arasında bir sözde-rastgele sayı üretir. % 100 ile 0-99 arasına düşürülür, + 1 ile de 1-100 arasına getirilir. Bu sayı dizinin ilgili elemanına atanır.
5. **printf("rastgeleSayilar[%d]: %d\n", i, rastgeleSayilar[i]);** : Atanan rastgele sayı ekrana yazdırılır.

Kaynaklar

Kaynaklar

- [1] W3Schools. (t.y.). *C Operators*. Erişim Adresi: https://www.w3schools.com/c/c_operators.php[Ziyaret Tarihi: 22 Haziran 2025]
- [2] W3Schools. (t.y.). *C If...Else*. Erişim Adresi: https://www.w3schools.com/c/c_if_else.php[Ziyaret Tarihi: 22 Haziran 2025]
- [3] W3Schools. (t.y.). *C Switch*. Erişim Adresi: https://www.w3schools.com/c/c_switch.php[Ziyaret Tarihi: 22 Haziran 2025]
- [4] W3Schools. (t.y.). *C For Loop*. Erişim Adresi: https://www.w3schools.com/c/c_for_loop.php[Ziyaret Tarihi: 23 Haziran 2025]
- [5] W3Schools. (t.y.). *C While Loop*. Erişim Adresi: https://www.w3schools.com/c/c_while_loop.php[Ziyaret Tarihi: 23 Haziran 2025]
- [6] W3Schools. (t.y.). *C Break and Continue*. Erişim Adresi: https://www.w3schools.com/c/c_break_continue.php[Ziyaret Tarihi: 23 Haziran 2025]
- [7] W3Schools. (t.y.). *C Arrays*. Erişim Adresi: https://www.w3schools.com/c/c_arrays.php[Ziyaret Tarihi: 24 Haziran 2025]
- [8] W3Schools. (t.y.). *C Random Numbers*. Erişim Adresi: https://www.w3schools.com/c/c_random.php[Ziyaret Tarihi: 24 Haziran 2025]
- [9] cppreference.com. (t.y.). *std::rand*. Erişim Adresi: <https://en.cppreference.com/w/c/numeric/rand/rand>[Ziyaret Tarihi: 24 Haziran 2025]
- [10] cppreference.com. (t.y.). *std::srand*. Erişim Adresi: <https://en.cppreference.com/w/c/numeric/rand/srand>[Ziyaret Tarihi: 24 Haziran 2025]
- [11] cppreference.com. (t.y.). *std::gets*. Erişim Adresi: <https://en.cppreference.com/w/c/io/gets>[Ziyaret Tarihi: 24 Haziran 2025]
- [12] cppreference.com. (t.y.). *std::strlen*. Erişim Adresi: <https://en.cppreference.com/w/c/string/byte/strlen>[Ziyaret Tarihi: 24 Haziran 2025]