

# Nesne Yönelimli Programlama

# Nesne Yönelimli Programlama

## Ders İçeriği

### Hafta 2: Kontrol Yapıları ve Döngüler

- Koşul İfadeleri ( `if` , `else if` , `else` )
- `switch-case` Yapısı
- Döngüler: `for` , `while` , `do-while`
- Break, Continue ve Return Kullanımı
- Nested (İç İç) Döngüler

# Giriş

Kontrol yapıları ve döngüler, programların kararlar almasını ve tekrarlayan işlemleri gerçekleştirmesini sağlar. Niçin kontrol yapıları ve döngüler kullanılır. Programlamada niçin önemlidir?

## 1. Karar Verme ve Program Akışını Yönetme

- **Kontrol yapıları** (if-else, switch-case) programların farklı durumlarda nasıl davranması gerektiğini belirler. Bu sayede programlar, kullanıcı girdilerine, değişkenlerin değerlerine veya farklı koşullara göre esnek ve dinamik tepkiler verebilir.
- Örneğin, bir not sisteminde kullanıcıdan alınan notun hangi harf notuna denk geldiği kontrol yapılarına göre belirlenir.

## 2. Kodun Tekrarını Engelleme ve Verimlilik

- **Döngüler** tekrarlı işlemlerin pratik bir şekilde yapılmasını sağlar. Aynı işlemi tekrar tekrar manuel olarak yazmak yerine, döngüleri kullanarak bu işlemleri otomatik hale getirebiliriz.
- Örneğin, bir dizinin tüm elemanlarını sıralamak, hesaplamak veya ekrana yazdırmak için döngüler kullanılır. Böylece aynı işlemi yüzlerce kez kodlamak yerine bir döngüyle tamamlamak mümkün olur.

### **3. Programların Modüler ve Okunabilir Olmasını Sağlama**

- Kontrol yapıları ve döngüler, kodun düzenli ve modüler olmasına katkıda bulunur. Karmaşık problemler küçük parçalara ayrılarak kontrol yapılarına ve döngülere yerleştirilebilir.
- Bu da hem kodun okunabilirliğini artırır hem de hata ayıklamayı ve geliştirmeyi kolaylaştırır.

### **4. Kullanıcı Girdilerine Dinamik Yanıt Verme**

- Programlar, kullanıcı girdisine göre farklı işlemler gerçekleştirmek zorundadır. Kontrol yapıları sayesinde kullanıcıdan gelen veriler analiz edilerek farklı yollar izlenir.
- Döngüler ise kullanıcıya sürekli aynı işlemi tekrar yaptırmak zorunda kalmadan dinamik girdilerle işlemlerin devam etmesini sağlar.

## **5. Veri İşleme ve Büyük Miktarda Bilgi ile Çalışma**

- Döngüler büyük veri kümeleriyle çalışırken oldukça verimlidir. Bir dosyadaki verileri okuma, bir dizideki öğeleri sıralama, bir veritabanında kayıtları listeleme gibi işlemler döngülerle yapılır.
- Kontrol yapıları ise bu veri üzerinde işlem yapılırken farklı koşullara göre karar alınmasını sağlar.

## **6. Otomasyon ve Tekrar Eden Görevler**

- Döngüler, bir işlemin belirli bir sayıda tekrarlanması gereken durumlarda kullanılır. Bu da programları otomatik hale getirir.
- Örneğin, bir hesaplama işlemini yüzlerce kez yapmanız gerektiğinde, döngüler sayesinde bu işlemi tek satırla çözmek mümkün olur.

## **7. Programları Daha Akıllı ve Etkileşimli Yapma**

- Kontrol yapıları sayesinde programlar, koşullara göre farklı sonuçlar üreterek daha "akıllı" hale gelir.
- Örneğin, bir oyun programında oyuncunun kazandığı puana göre seviye atlama veya oyunun sonlanması kontrol yapılarıyla sağlanır.

## **8. Daha Az Kodla Daha Fazla İş Yapabilme**

- Döngüler ve kontrol yapıları, daha az kod yazarak daha karmaşık işlemleri gerçekleştirmemizi sağlar. Aynı işlemi birden fazla yazmak yerine, döngüler sayesinde işlemleri verimli şekilde tekrarlarız.

# Koşullu İfadeler

## if-else Yapısı

`if-else` yapısı, bir koşulun doğruluğunu kontrol ederek programın iki farklı yolda ilerlemesini sağlar.

Yapı:

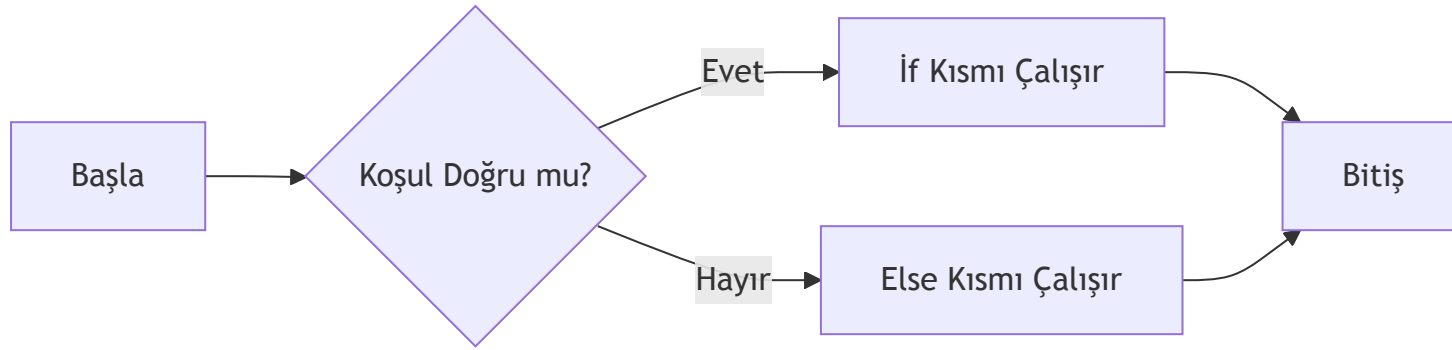
```
1  if (koşul) {  
2      // Koşul doğruysa yapılacak işlemler  
3  } else {  
4      // Koşul yanlışsa yapılacak işlemler  
5  }
```

```
1  int sayi = 10;
2  if (sayi > 0) {
3      cout << "Sayı pozitiftir." << endl;
4  } else {
5      cout << "Sayı negatiftir veya sıfırdır." << endl;
6  }
```



## else-if Yapısı

`else-if` yapısı, birden fazla koşulun kontrol edilmesi gerektiğinde kullanılır.



## Kod Yapısı (Syntax):

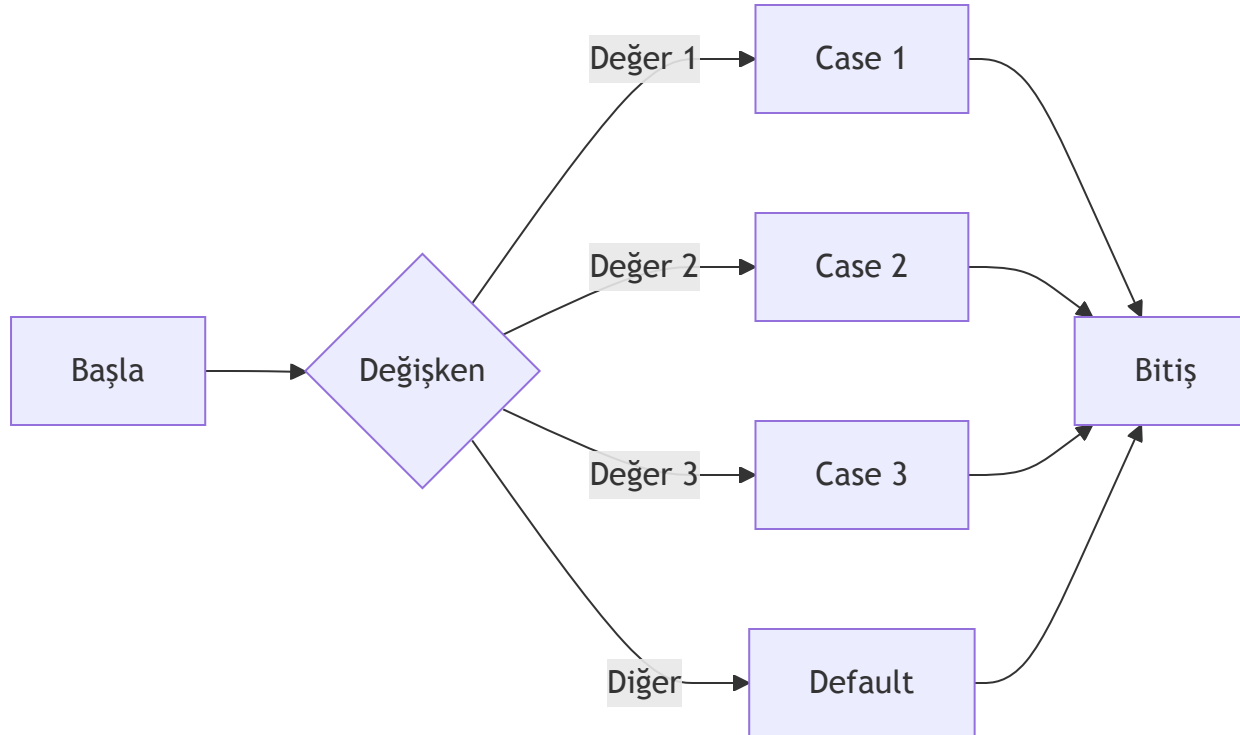
```
1   Kodu kopyala
2   if (koşul1) {
3       // Koşul1 doğruysa yapılacak işlemler
4   } else if (koşul2) {
5       // Koşul2 doğruysa yapılacak işlemler
6   } else {
7       // Hiçbir koşul doğru değilse yapılacak işlemler
8   }
```

## Örnek:

```
1  int notDegeri = 85;
2  if (notDegeri >= 90) {
3      cout << "Harf Notu: A" << endl;
4  } else if (notDegeri >= 80) {
5      cout << "Harf Notu: B" << endl;
6  } else {
7      cout << "Harf Notu: C" << endl;
8  }
```

# switch-case Yapısı

`switch-case` yapısı, bir değişkenin belirli bir değere sahip olup olmadığını kontrol eder ve bu değerlere göre farklı işlemler yapar.



## Kod Yapısı (Syntax):

```
1  switch (değişken) {
2      case değer1:
3          // Değer1'e göre yapılacak işlemler
4          break;
5      case değer2:
6          // Değer2'ye göre yapılacak işlemler
7          break;
8      default:
9          // Hiçbir değer eşleşmezse yapılacak işlemler
10 }
```

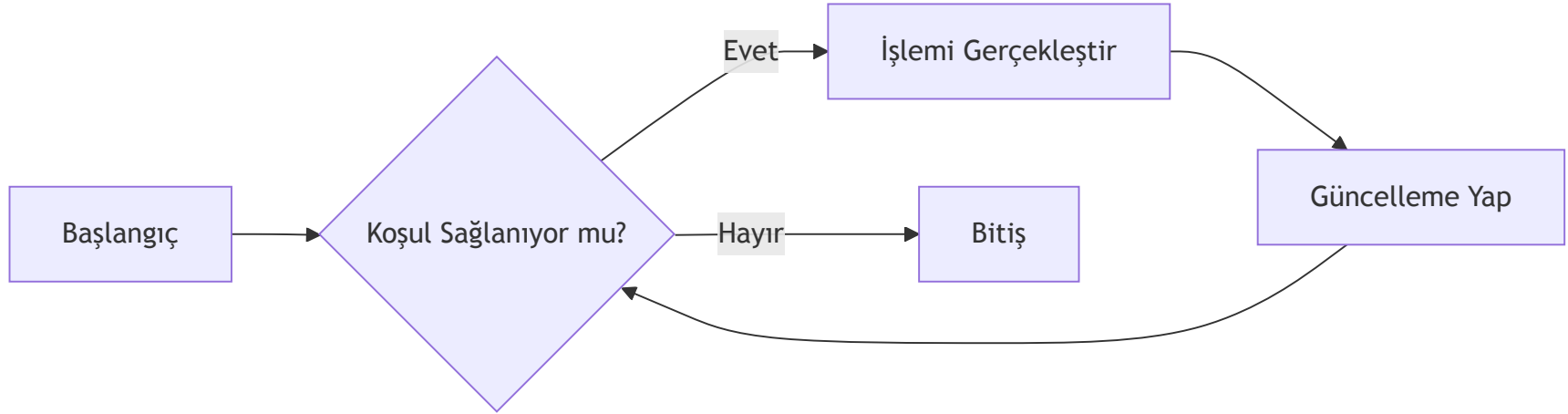
## Örnek:

```
1  char harfNotu = 'B';
2  switch (harfNotu) {
3      case 'A':
4          cout << "Mükemmel!" << endl;
5          break;
6      case 'B':
7          cout << "İyi!" << endl;
8          break;
9      case 'C':
10         cout << "Orta!" << endl;
11         break;
12     default:
13         cout << "Geçersiz not!" << endl;
14 }
```

# Döngüler

## while Döngüsü

`while` döngüsü, bir koşul doğru olduğu sürece sürekli olarak bir işlemi tekrarlar.



## Kod Yapısı (Syntax):

```
Kodu kopyala  
while (koşul) {  
    // Koşul doğru olduğu sürece yapılacak işlemler  
}
```

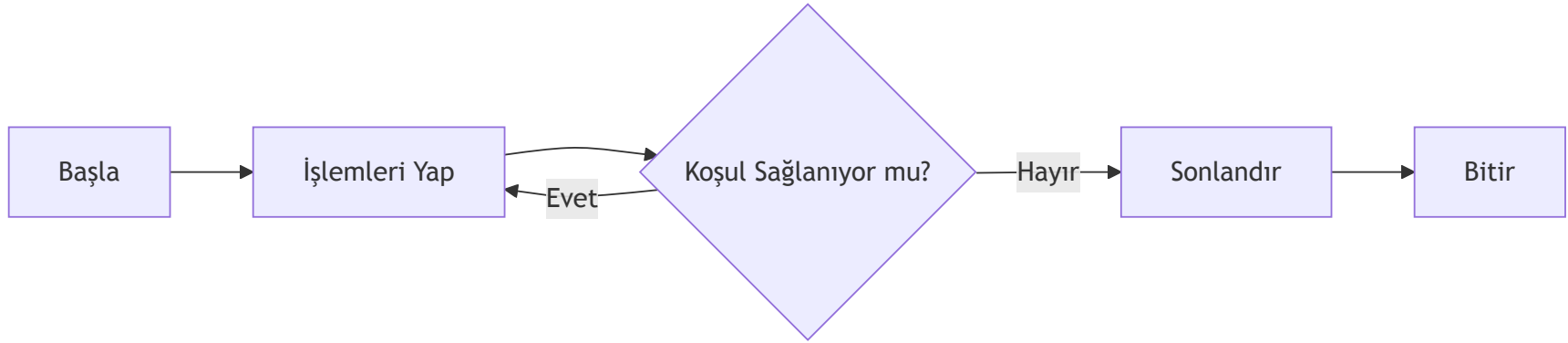


## Örnek:

```
1  int i = 0;
2  while (i < 5) {
3      cout << "i: " << i << endl;
4      i++;
5  }
```

# do-while Döngüsü

`do-while` döngüsü, işlemleri en az bir kere gerçekleştirir ve sonra koşulu kontrol eder.



## Kod Yapısı (Syntax):

```
1  do {  
2      // Koşul kontrol edilmeden önce yapılacak işlemler  
3  } while (koşul);
```

## Örnek:

```
1  int i = 0;
2  do {
3      cout << "i: " << i << endl;
4      i++;
5  } while (i < 5);
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int sayi;
6      int sayiAdedi = 0;
7      int toplam = 0;
8      int enBuyuk = 0;
9      int enKucuk;
```

```
// Kullanıcıdan pozitif sayılar alın
do {
    cout << "Pozitif bir tam sayı girin (durdurmak için 0 girin): ";
    cin >> sayi;

    if (sayi > 0) {
        sayiAdedi++;
        toplam += sayi;

        if (sayiAdedi == 1) {
            enKucuk = sayi; // İlk sayı en küçük olarak atanır
        }

        if (sayi > enBuyuk) {
            enBuyuk = sayi; // En büyük sayı güncellenir
        }

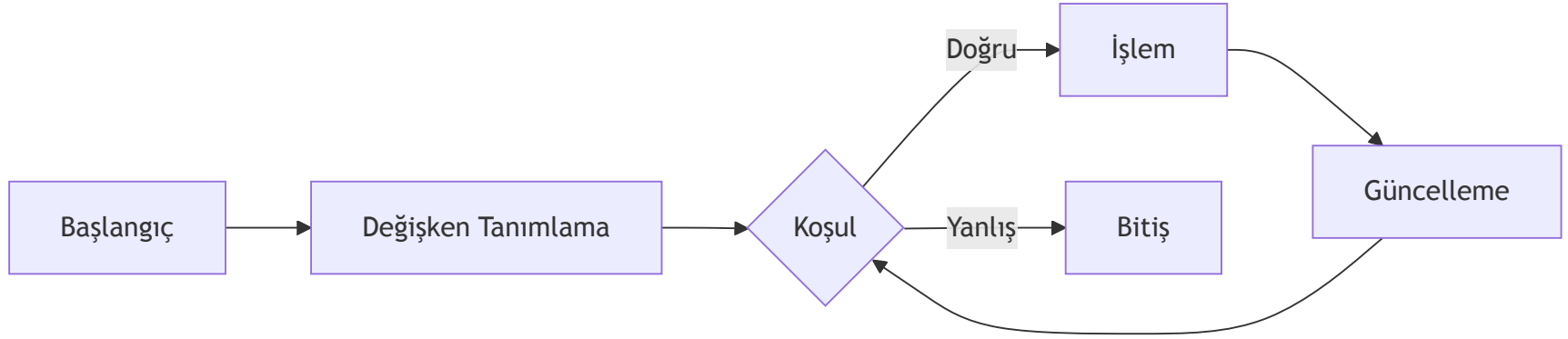
        if (sayi < enKucuk) {
            enKucuk = sayi; // En küçük sayı güncellenir
        }
    }
} while (sayi != 0); // Kullanıcı 0 girmedikçe döngü devam eder
```

```
// İstatistikleri yazdır
if (sayiAdedi > 0) {
    cout << "Toplam sayı adedi: " << sayiAdedi << endl;
    cout << "En büyük sayı: " << enBuyuk << endl;
    cout << "En küçük sayı: " << enKucuk << endl;
    cout << "Ortalama: " << (static_cast<double>(toplama) / sayiAdedi) << endl;
} else {
    cout << "Hiç pozitif sayı girilmedi." << endl;
}

return 0;
}
```

# for Döngüsü

`for` döngüsü, belirli bir sayıda tekrarlamak için kullanılır ve genellikle sayaç kontrollüdür.





## Kod Yapısı (Syntax):

```
1  for (başlangıç; koşul; güncelleme) {  
2      // Koşul doğru olduğu sürece yapılacak işlemler  
3  }
```

## Örnek:

```
1   Kodu kopyala
2   for (int i = 0; i < 5; i++) {
3       cout << "i: " << i << endl;
4   }
```

# break ve continue İfadeleri

## break İfadesi

`break` ifadesi, bir döngüyü erken sonlandırmak için kullanılır.

## Örnek:

```
1  for (int i = 0; i < 10; i++) {  
2      if (i == 5) {  
3          break; // i 5 olduğunda döngüden çık  
4      }  
5      cout << "i: " << i << endl;  
6  }
```

## continue İfadesi

`continue` ifadesi, döngünün geri kalan kısmını atlayarak bir sonraki yinelemeye geçmek için kullanılır.

### Örnek:

```
1  for (int i = 0; i < 10; i++) {  
2      if (i == 5) {  
3          continue; // i 5 olduğunda bu adımı atla  
4      }  
5      cout << "i: " << i << endl;  
6  }
```

## Alıştırma 1: Asal Sayı Kontrolü

**Soru:** Kullanıcıdan bir tam sayı alın ve bu sayının asal olup olmadığını kontrol edin. Asal sayı, yalnızca 1 ve kendisi dışında tam böleni olmayan sayıdır.

# Pesudo Kod

```
1   Başla
2   Değişkenler:
3       tamSayi, i (tam sayı)
4       asal (boolean) = doğru
5
6   Kullanıcıdan bir tam sayı al:
7       Yaz "Bir tam sayı girin: "
8       Oku tamSayi
9
10  Eğer tamSayi <= 1 ise
11      asal = yanlış // 1 ve daha küçük sayılar asal değildir.
12  Aksi durumda
13      döngü i = 2'den tamSayi / 2'ye kadar (dahil):
14          Eğer tamSayi % i == 0 ise
15              asal = yanlış // Sayı i ile tam bölünüyorsa asal değildir.
16              dur
17      Son döngü
18
19  Eğer asal ise
20      Yaz tamSayi + " bir asal sayıdır."
21  Aksi durumda
22      Yaz tamSayi + " bir asal sayı değildir."
23  Sonu
```

```
1  using namespace std;
2
3  int main() {
4      int sayi;
5      bool asal = true;
6
7      cout << "Bir tam sayı girin: ";
8      cin >> sayi;
9
10     if (sayi <= 1) {
11         asal = false; // 1 ve daha küçük sayılar asal değildir.
12     } else {
13         for (int i = 2; i <= sayi / 2; i++) {
14             if (sayi % i == 0) {
15                 asal = false; // Sayı i ile tam bölünüyorsa asal değildir.
16                 break;
17             }
18         }
19     }
20
21     if (asal) {
22         cout << sayi << " bir asal sayıdır." << endl;
23     } else {
24         cout << sayi << " bir asal sayı değildir." << endl;
25     }
26 }
```



## Alıştırma 2: Fibonacci Dizisi

# Fibonacci Dizisi

**Fibonacci Dizisi**, her bir terimin kendisinden önceki iki terimin toplamı olduğu bir sayı dizisidir. Dizinin ilk iki terimi genellikle 0 ve 1 olarak başlar. Fibonacci dizisi şu şekilde başlar:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

## Fibonacci Dizisinin Genel Tanımı

Fibonacci dizisinin matematiksel olarak tanımı aşağıdaki gibidir:

- (  $F(0) = 0$  )
- (  $F(1) = 1$  )
- (  $F(n) = F(n-1) + F(n-2)$  ) (  $n \geq 2$  )

# Fibonacci Formülü

Fibonacci dizisini hesaplamak için kullanılan formül:

$$[ F(n) = F(n-1) + F(n-2) ]$$

Bu formülde:

- (  $F(n)$  ):  $n$ 'inci Fibonacci sayısını temsil eder.
- (  $F(n-1)$  ):  $n-1$ 'inci Fibonacci sayısını temsil eder.
- (  $F(n-2)$  ):  $n-2$ 'inci Fibonacci sayısını temsil eder.

# Örnek Hesaplama

Fibonacci dizisindeki ilk birkaç terimi hesaplayalım:

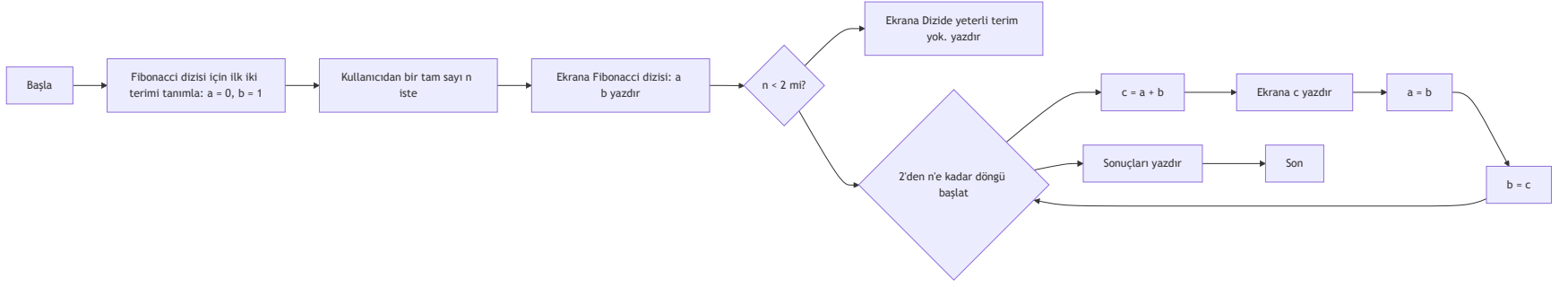
- (  $F(0) = 0$  )
- (  $F(1) = 1$  )
- (  $F(2) = F(1) + F(0) = 1 + 0 = 1$  )
- (  $F(3) = F(2) + F(1) = 1 + 1 = 2$  )
- (  $F(4) = F(3) + F(2) = 2 + 1 = 3$  )
- (  $F(5) = F(4) + F(3) = 3 + 2 = 5$  )
- (  $F(6) = F(5) + F(4) = 5 + 3 = 8$  )

Bu şekilde devam ederek Fibonacci dizisinin sonraki terimlerini elde edebilirsiniz.

**Soru:** Kullanıcıdan bir tam sayı alın ve bu sayıya kadar olan Fibonacci dizisini ekrana yazdırın.

## Pesudo Kod:

```
1  Başla
2  Fibonacci dizisi için ilk iki terimi tanımla: a = 0, b = 1
3  Kullanıcıdan bir tam sayı n iste
4  Ekrana "Fibonacci dizisi: a b" yazdır
5
6  Eğer n < 2 ise
7      Ekrana "Dizide yeterli terim yok." yazdır
8  Değilse
9      2'den n'e kadar döngü başlat
10         c = a + b
11         Ekrana c yazdır
12         a = b
13         b = c
14     Döngüyü bitir
15     Sonuçları yazdır
16
17  Son
```



## Cevap:

```
1  using namespace std;
2
3  int main() {
4      int n, a = 0, b = 1, c;
5
6      cout << "Bir tam sayı girin: ";
7      cin >> n;
8
9      cout << "Fibonacci dizisi: " << a << " " << b << " ";
10     for (int i = 2; i < n; i++) {
11         c = a + b;
12         cout << c << " ";
13         a = b;
14         b = c;
15     }
16
17     cout << endl;
18
19     return 0;
20 }
```

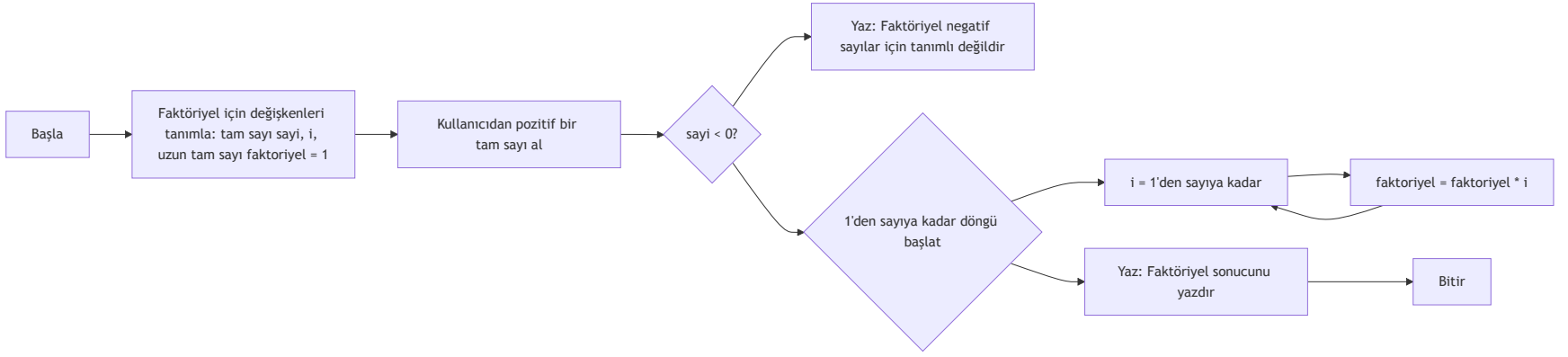
## Alıştırma 3: Faktöriyel Hesaplama

Faktöriyel, pozitif bir tam sayının kendisi ve kendisinden küçük pozitif tam sayıların çarpımını ifade eder. Genellikle "n!" şeklinde gösterilir.

Bir pozitif tam sayının faktöriyeli, o sayının ve altındaki tüm pozitif tam sayıların çarpımıdır. Örneğin:

- $0! = 1$  (Faktöriyel 0'ın tanımı gereği 1'dir)
- $1! = 1$
- $2! = 2 \times 1 = 2$
- $3! = 3 \times 2 \times 1 = 6$
- $4! = 4 \times 3 \times 2 \times 1 = 24$
- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

**Soru:** Kullanıcıdan pozitif bir tam sayı alın ve bu sayının faktöriyelini hesaplayın.





## Pesudo Kod:

```
1  Başla
2  Faktöriyel için değişkenleri tanımla:
3      tam sayı sayı
4      tam sayı i
5      uzun tam sayı faktoriyel = 1
6
7  Kullanıcıdan pozitif bir tam sayı al:
8      Yaz("Pozitif bir tam sayı girin:")
9      oku(sayı)
10
11  Eğer sayı < 0 ise
12      Yaz("Faktöriyel negatif sayılar için tanımlı değildir.")
13  Aksi halde
14      1'den sayı'ya kadar döngü başlat:
15          için i = 1'den sayı'ya kadar
16              faktoriyel = faktoriyel * i
17
18      Yaz("Faktöriyel: ", faktoriyel)
19
20  Bitir
```

## Cevap:

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int sayi;
6      long long faktoriyel = 1;
7
8      cout << "Pozitif bir tam sayı girin: ";
9      cin >> sayi;
10
11     if (sayi < 0) {
12         cout << "Faktöriyel negatif sayılar için tanımlı değildir." << endl;
13     } else {
14         for (int i = 1; i <= sayi; i++) {
15             faktoriyel *= i; // Faktöriyel hesaplama
16         }
17         cout << sayi << " sayısının faktöriyeli: " << faktoriyel << endl;
18     }
19
20     return 0;
21 }
```

## Alıştırma 4: En Büyük ve En Küçük Sayıyı Bulma

**Soru:** Kullanıcıdan belirli bir miktarda tam sayı alın ve bu sayılar arasında en büyük ve en küçük olanını bulun.

## Pesudo Kod:

```
1  Başla
2  Değişkenler:
3      n, sayı, enBuyuk, enKucuk: Tam sayı
4
5  Yaz "Kaç tane sayı gireceksiniz? "
6  Oku n
7
8  Yaz "Birinci sayıyı girin: "
9  Oku sayı
10 enBuyuk ← sayı
11 enKucuk ← sayı
```

## Pesudo Kod(devam):

```
Döngü i = 1'den n - 1'e kadar:  
  Yaz (i + 1) + ". sayıyı girin: "  
  Oku sayi
```

```
  Eğer sayi > enBuyuk ise:  
    enBuyuk ← sayi  
  Son Eğer
```

```
  Eğer sayi < enKucuk ise:  
    enKucuk ← sayi  
  Son Eğer
```

```
Son Döngü
```

```
Yaz "En büyük sayı: ", enBuyuk  
Yaz "En küçük sayı: ", enKucuk
```

```
Son
```

## Cevap:

```
1  using namespace std;
2  int main() {
3      int n, sayi, enBuyuk, enKucuk;
4      cout << "Kaç tane sayı gireceksiniz? ";
5      cin >> n;
6      cout << "Birinci sayıyı girin: ";
7      cin >> sayi;
8      enBuyuk = enKucuk = sayi; // İlk sayıyı hem en büyük hem de en küçük olarak ayarlayın.
9
10     for (int i = 1; i < n; i++) {
11         cout << (i + 1) << ". sayıyı girin: ";
12         cin >> sayi;
13
14         if (sayi > enBuyuk) {
15             enBuyuk = sayi; // En büyük sayıyı güncelle
16         }
17         if (sayi < enKucuk) {
18             enKucuk = sayi; // En küçük sayıyı güncelle
19         }
20     }
21
22     cout << "En büyük sayı: " << enBuyuk << endl;
23     cout << "En küçük sayı: " << enKucuk << endl;
24 }
```

## Alıştırma 5: Gün Hesaplama

**Soru:** Kullanıcıdan bir tam sayı alın ve bu sayının haftanın hangi günü temsil ettiğini belirleyin. (1: Pazartesi, 2: Salı, 3: Çarşamba, 4: Perşembe, 5: Cuma, 6: Cumartesi, 7: Pazar). Geçersiz bir sayı girilirse "Geçersiz gün" mesajını verin.

## Pesudo Kod:

```
1   Başla
2   Değişkenler:
3       gun: Tam sayı
4
5   Yaz "Lütfen bir gün numarası girin (1-7): "
6   Oku gun
7
8   Durum gun:
9       1:
10          Yaz "Gün: Pazartesi"
11       2:
12          Yaz "Gün: Salı"
13       3:
14          Yaz "Gün: Çarşamba"
15       4:
16          Yaz "Gün: Perşembe"
17       5:
18          Yaz "Gün: Cuma"
19       6:
20          Yaz "Gün: Cumartesi"
21       7:
22          Yaz "Gün: Pazar"
23   Diğer:
24       Yaz "Geçersiz gün"
```



```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int gun;
6
7      cout << "Lütfen bir gün numarası girin (1-7): ";
8      cin >> gun;
9
10     switch (gun) {
11         case 1:
12             cout << "Gün: Pazartesi" << endl;
13             break;
14         case 2:
15             cout << "Gün: Salı" << endl;
16             break;
17         case 3:
18             cout << "Gün: Çarşamba" << endl;
19             break;
20         case 4:
21             cout << "Gün: Perşembe" << endl;
22             break;
```

```
case 5:
    cout << "Gün: Cuma" << endl;
    break;
case 6:
    cout << "Gün: Cumartesi" << endl;
    break;
case 7:
    cout << "Gün: Pazar" << endl;
    break;
default:
    cout << "Geçersiz gün" << endl;
    break;
}

return 0;
}
```

# Ödev 1: Basamak Toplama

**Açıklama:** Kullanıcıdan bir tam sayı alın ve bu sayının basamaklarının toplamını hesaplayın. Örneğin, kullanıcı 1234 girdiğinde,  $1 + 2 + 3 + 4 = 10$  sonucu elde edilmelidir.

## Beklenen Sonuç:

- Kullanıcıdan bir tam sayı alın.
- Sayının her bir basamağını ayrıştırın ve toplama işlemi yapın.
- Toplam sonucu ekrana yazdırın.

## Yardımcı Kod Örneği:

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      // Kullanıcıdan alınacak sayıyı tanımlayın
6      // Döngü ile basamakları toplayın
7      // Sonucu yazdırın
8  }
```

## Ödev 2: FizzBuzz Problemi

**Açıklama:** Kullanıcıdan bir pozitif tam sayı alın ve 1'den bu sayıya kadar olan sayıları yazdırın. Ancak:

- 3'e bölünebilen sayılar için "Fizz",
- 5'e bölünebilen sayılar için "Buzz",
- Hem 3'e hem de 5'e bölünebilen sayılar için "FizzBuzz" yazdırın.

### Beklenen Sonuç:

- Kullanıcıdan bir pozitif tam sayı alın.
- 1 ile bu sayı arasındaki her sayıyı kontrol edin ve uygun "Fizz", "Buzz" veya "FizzBuzz" değerlerini yazdırın.

### Yardımcı Kod Örneği:

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      // Kullanıcıdan alınacak pozitif tam sayıyı tanımlayın
6      // Döngü ile sayıları kontrol edin
7      // Uygun sonucu yazdırın
8  }
```