KONTROL İŞLEMLERİ

Ardışık İşlem ve Kontrol İşlemleri

Yapısal Programlama başlığı altında programın ana fonksiyondan başlayacağı ve programlamanın ise birbirini çağıran fonksiyonlarla yapıldığı anlatılmıştı. Yapısal programlamada, ana fonksiyon dahil tüm fonksiyonlarda ilk önce işlenecek verileri taşıyan veri yapıları tanımlanır ve ardından bu verileri işleyen kontrol yapılarına ilişkin talimatlar yazılır.

Şu ana karar örneğini verdiğimiz kodlarda veri yapısı olarak sadece değişkenler kullanılmıştır. Sonrasında ise giriş çıkış işlemleri talimatlar içeren programlar yazılmıştır. Programın icrası ilk talimattan başlar ve sırasıyla program bitene kadar devam eder. İşte programın icrasını değiştirmeyen bu tür talimatlara ardışık işlem (sequential operation) adı verilir. Ardışık işlemler aşağıdaki üç tür talimattan (statement) oluşur.

1. Değişkenlerin kimliklendirildiği değişken tanımlamaları (identifier definition):

```
int yariCap=3;
const float PI=3.14;
float daireninAlani,daireninCevresi;
```

2. İfadelerden (expression) yani sabit, değişken ve operatör içeren sözdizimleri:

```
daireninAlani=PI*yariCap*yariCap;
float daireninCevresi=2.0*PI*yariCap;
```

3. Klavyeden veri okume ve konsola veri yazma gibi giriş çıkış işlemleri (input output operation):

```
printf("Kapasite Oranını (0.00-1.00) Giriniz:");
scanf("%f",&kapasiteOrani);
```

Kontrol işlemleri (control operation) ise programın icra sırasını değiştiren kontrol yapıları olup bu talimatlar (statement) olup üç türü vardır;

1. Duruma göre seçimler (conditional choice): `if`, `if-else` ve `switch` talimatları.
2. İlişkisel döngü (relational loop): `while`, `do-while` ve `for` talimatları.
3. Dallanmalar (jump): `continue`, `break`, `goto` ve `return` talimatları.

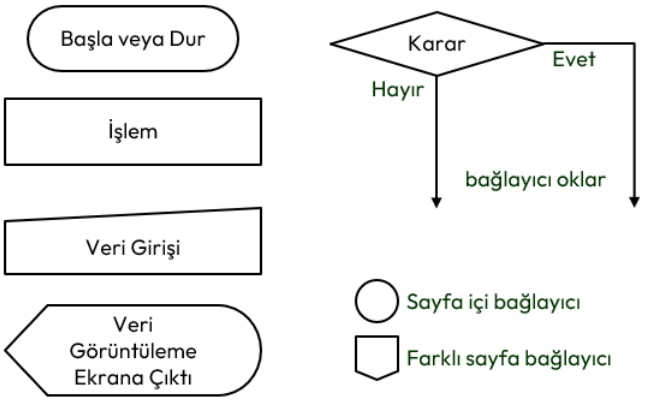
Kontrol işlemlerinde; talimat olarak yazılmış mantıksal satırlar (logical sequence) ile fiziksel olarak icra edilen satırlar (physical sequence) birbirinden farklıdır.

Akış Diyagramları

Kontrol işlemi içeren programın icrasını anlamak için akış diyagramlarını da anlamak gerekir. Akış diyagramı, adımların grafiksel gösterimleridir. Algoritmaları ve programlama mantığını temsil etmek için bir araç olarak bilgisayar biliminden ortaya çıkmıştır. Ancak diğer tüm işlem türlerinde kullanılmak üzere genişletilmiştir.

Günümüzde akış diyagramları, bilgileri göstermede ve akıl yürütmeye yardımcı olmada son derece önemli bir rol oynamaktadır. Karmaşık süreçleri görselleştirmemize veya sorunların ve görevlerin yapısını açık hale getirmemize yardımcı olurlar. Bir akış diyagramı, uygulanacak bir süreci veya projeyi tanımlamak için de kullanılabilir. Akış diyagramı çizilirken aşağıdaki kurallara uyulur;

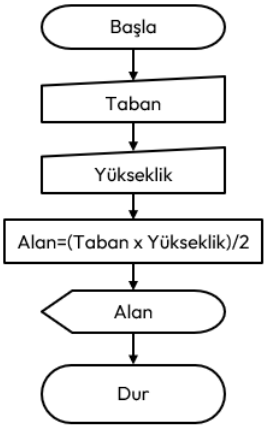
- Akış şemalarında tek bir başlangıç simgesi olmalıdır
- Bitiş simgesi birden çok olabilir.
- Karar simgesinin haricindeki simgelere her zaman tek giriş ve tek çıkış yolu bulunur.
- Bağlaç simgesi sayfanın dolmasından ötürü parçalanan akış şemasının öğelerini birleştirmede kullanılır.
- Simgeler birbirleri ile tek yönlü okla bağlanırlar.
- Okların yönü algoritmanın mantıksal işlem akışını tanımlar.



Şekil 4. Akış Diyagramları Genel Şekilleri

Yukarıda tüm programlama dillerinde akış diyagramları için ortak kullanılan temel şekiller verilmiştir.

Aşağıda taban ve yüksekliği klavyeden girilen bir üçgenin alanını hesaplamak için bir akış diyagramı örneği verilmiştir.



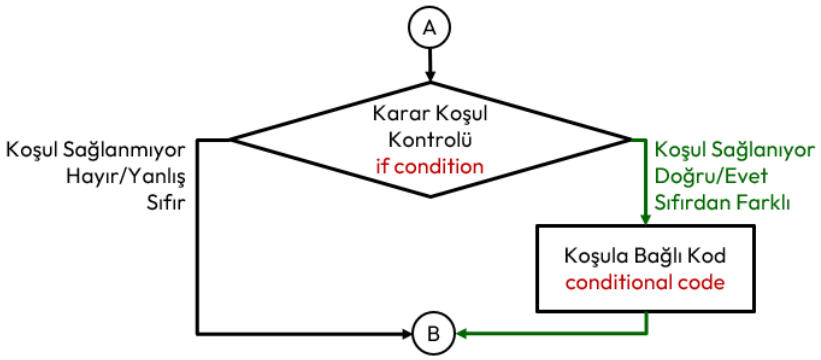
Şekil 5. Örnek Akış Diyagramı

Duruma Göre Seçimler

Duruma göre seçimler (conditional choice) programın akışını bir koşula göre değiştiren talimatlardır.

If Talimatı

If talimatı, karar vermeye (decision-making) ilişkin bir talimat olup bir koşula bağlı olarak programın icrasını değiştirir. Koşul (condition) ifadesi `true` ise koşul kodu (conditional code) icra edilir değilse icra edilmez. Koşul ifadesi (expression) test edilir ve sıfırdan farklı ise `true`, aksi halde `false` kabul edilir.



Şekil 6. If Talimatı İcra Akışı

Sözde kodu aşağıda verilmiştir;

```
if (koşul)
    koşul-kodu;
```

		yas	cinsiyet
	`#include <iostream>`		
	`using namespace std;`		
	`int main() {`		
1	`int yas=18;`	18	?
2	`char cinsiyet='K';`	18	'K'
3	`if (yas<30)`	18	'K'
4	`cout << "Genç";`	18	'K'
5	`if (cinsiyet=='E')`	18	'K'
	`cout << "Erkek";`	18	'K'
	`}`		

Tablo 16. If Talimatı İçeren Bir C Programı İcra Sırası

Programın icrası, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `yas` değişkenine `18` değeri atanır.
2. `cinsiyet` değişkenine `'K'` atanır.
3. `if` talimatındaki koşul (condition) test edilir. `yas<30` yani `18<30` testinde küçüktür işleci `true` verir. Bu nedenle izleyen koşul kodu (conditional code) icra edilir.
4. `Cout << "Genç";` koşul kodu olduğundan icra edilir.
5. `if` talimatındaki koşul test edilir. `cinsiyet=='E'` yani `'E'=='K'` testinde eşit mi işleci `false` verir. Dolayısıyla izleyen koşul kodu icra edilmez.

Bu durumda programın çalışması sonucu aşağıdaki çıktı elde edilir.

```
Genç

...Program finished with exit code 0
```

`if` talimatında koşul kodu (conditional code) her zaman ardışık işlem (sequential operation) olmaz. `if` gibi bir kontrol işlemi (control operation) de olabilir. Aşağıda kademeli (compound/cascade) if kullanımına ilişkin kod örneğinde ikinci `if`, birinci `if` talimatının koşul kodudur.

```
    if (yas<30)
        if (yas<7)
            cout << "Çoçuk ";
```

Koşul kodu (conditional code) birden fazla talimattan oluşacak ise kod bloğu `{ }` içine alınır. Aşağıda verilen kod örneğinde ilk `if` talimatında `yas<30` ile test edilen koşul doğrulandığında kod bloğunun içindeki tüm talimatlar icra edilir.

```
    if (yas<30) {
        if (yas<7)
           cout << "Çoçuk ";
        if (yas<18)
           cout << "Genç ";
        if (yas>60)
           printf("Yaşlı ");
    }
```

Bahsedilen iki durum iç içe de olabilir. Buna ilişkin kod örneği de aşağıda verilmiştir.

```
    if (cinsiyet=='E') {
        if (yas<7)
           cout << "Erkek Çocuk ";
        if (yas<18)
           cout << "Genç Delikanlı ";
        if (yas>60)
           cout << "Yaşlı Adam ";
    }
```

```
    if (cinsiyet=='K') {
        if (yas<3)
           cout << "Kız Bebek ";
        if (yas<7)
           cout << "Kız Çoçuk ";
        if (yas<18)
           cout << "Genç Kız ";
        if (yas>60)
           cout << "Yaşlı Kadın ";
    }
```

If talimatında koşul (condition) her zaman tek bir test ifadesinden oluşmayabilir. Bahsedilen duruma ilişkin kod örneği de aşağıda verilmiştir.

```
    if ((cinsiyet=='E') && (yas<18))
       cout << "Genç Delikanlı ";
    if ((cinsiyet=='K') && (yas>60))
       cout << "Yaşlı Kadın ";
    // if (10>rakam>5) // HATA!. Böyle bir şart yazılamaz.
       cout << "rakam 10 ile 5 arasında ";
    // if (rakam>10>rakam>5) // HATA!. Böyle bir şart yazılamaz.
       cout << "rakam 10 ile 5 arasında ";
```

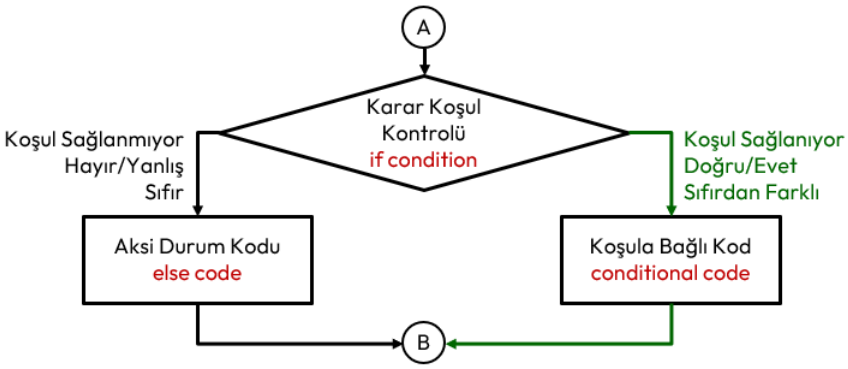
Bazen programcı tarafından aşağıdaki gibi `if` talimatları yazabilir.

```
    if (1)
      cout << "Bu metin konsola her zaman yazılır.";
    if (0)
      cout << "Bu metin konsola hiçbir zaman yazılmaz!";
```

If-Else Talimatı

If-Else Talimatı, bir başka karar verme (decision-making) talimatı olup bir koşula bağlı olarak programın icrasını değiştirir. If talimatına benzer şekilde koşul (condition) ifadesi `true` ise koşul kodu (conditional code) icra edilir, değilse aksi durum kodu (else code) icra edilir.

```
if (koşul)
    koşul-kodu;
else
    aksi-durum-kodu;
```



Şekil 7. If-Else Talimatı Sözde Kodu ve İcra Akışı

		yas	cinsiyet
	`#include <iostream>`		
	`using namespace std;`		
	`int main() {`		`?`
1	`int yas=65;`	65	`?`
2	`char cinsiyet='K';`	65	`'K'`
3	`if (yas<50)`	65	`'K'`
	`cout << "Genç ";`	65	`'K'`
4	`else`	65	`'K'`
5	`cout << "Yaşlı ";`	65	`'K'`

	}		

Tablo 17. If-Else Talimatı İçeren Bir C Programı İcra Sırası

Programın icrası, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `yas` değişkenine `65` değeri atanır.
2. `cinsiyet` değişkenine `'K'` atanır.
3. if talimatındaki koşul (condition) test edilir. `yas<50` yani `65<50` testinde küçüktür işleci `false` verir. Test sonucu `false` olduğundan aksi durum kodu (else code) icra edilir.
4. Programın icrası `else` ifadesinden devam eder.
5. `Cout <<"Yaşlı ";` aksi durum kodu olduğundan icra edilir.

Bu durumda programın çalışması sonucu aşağıdaki çıktı elde edilir.

```
Yaşlı

...Program finished with exit code 0
```

If talimatında olduğu gibi bu talimatta da koşul kodu (conditional code) ya da aksi durum kodu (else code) birden fazla talimattan oluşacak ise kod bloğu `{ }` içine alınır. Buna ilişkin kod örneği aşağıda verilmiştir.

```
    if (cinsiyet=='E') {
        if (yas<7)
           cout << "Erkek Çocuk ";
        if (yas<18)
           cout << "Genç Delikanlı ";
        if (yas>60)
           cout << "Yaşlı Adam ";
    } else {
        if (yas<3)
           cout << "Kız Bebek ";
        if (yas<7)
           cout << "Kız Çoçuk ";
        if (yas<18)
           cout << "Genç Kız ";
        if (yas>60)
           cout << "Yaşlı Kadın ";
    }
```

Aşağıdaki program örneği incelendiğinde else, hangi if talimatına aittir? Böyle bir kod programcı tarafından yazılabilir ve derleyici buna hiçbir sıkıntı çıkarmaz.

```
    if (cinsiyet=='E') if (yas<18) cout << "Delikanlı"; else cout << "Adam";
```

Bu durumda kodu aşağıdaki gibi okunaklı hale getirdiğimizde ilk `if` talimatının koşul kodunun (conditional code) ikinci if-else talimatı olduğu görülmektedir. Bu problem sarkan else (dangling else) problemi olarak bilinir. Kod bloğu kullanılmadan yazılan bu tip kodlarda else her zaman kendinden bir önceki if talimatına aittir.

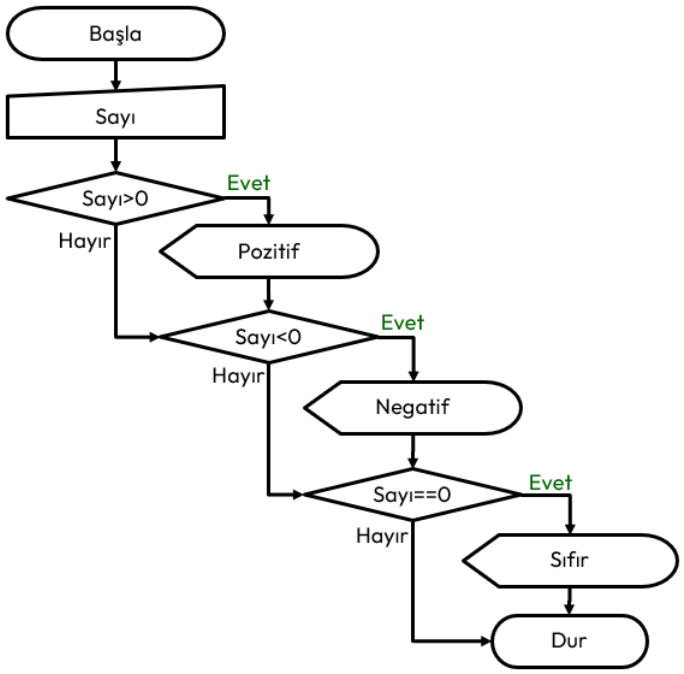
```
    if (cinsiyet=='E')
        if (yas<18)
            cout << "Delikanlı";
        else
            cout << "Adam";
```

Örnek olarak klavyeden girilen bir sayının sıfırdan küçük mü? Büyük mü? ya da sıfıra eşit mi? Olduğunu bulan bir C++ programı yazalım.

```
#include <iostream>
using namespace std;
```

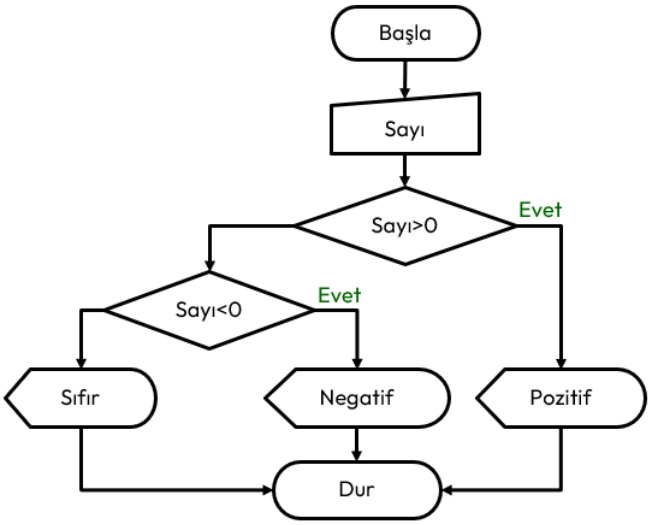
```
int main() {
    int sayi;
    cout << "Sayi Giriniz:";
    cin >> sayi;
    if (sayi>0) // sayının pozitif olup olmadığı test ediliyor.
        cout << "Pozitif"; // Burada sayının pozitif olduğu biliniyor.

    if (sayi<0) // sayının negatif olup olmadığı test ediliyor.
        cout << "Negatif"; // Burada sayının negatif olduğu biliniyor.

    if (sayi==0) // sayının sıfır olup olmadığı test ediliyor.
        cout << "Sıfır"; // Burada sayının sıfır olduğu biliniyor.
}
```

Programın icra akışı aşağıdaki şekilde olacaktır.



Şekil 8. Programın If Talimatlarıyla Yazılması Halinde İcra Akışı

Halbuki ilk if talimatında yapılan test doğru çıkarsa geri kalan testlerin yapılmasına gerek kalmaz. Benzer şekilde ilk iki test geçilirse üçüncü if talimatındaki teste gerek yoktur.



Şekil 9. Programın If-else Talimatlarıyla Yazılması Halinde İcra Akışı

Bu durumda aynı program if-else talimatlarıyla aşağıdaki şekilde yeniden yazabiliriz.

```
#include <iostream>
using namespace std;
int main() {
    int sayi;
    cout << "Sayi Giriniz:";
    cin >> sayi;
    if (sayi>0) { // sayının pozitif olup olmadığı test ediliyor.
        cout << "Pozitif"; // Burada sayının pozitif olduğu biliniyor.
    } else {  // Burada sayının pozitif olmadığı biliniyor.
        if (sayi<0) { // sayının negatif olup olmadığı test ediliyor.
            cout << "Negatif"; // Burada sayının negatif olduğu biliniyor.
        } else  { // Burada sayının pozitif ve negatif olmadığı test biliniyor.
            cout << "Sıfır"; // Burada sayının sıfır olduğu biliniyor.
        }
    }
}
```

Program incelendiğinde sayının pozitif girilmesi halinde sadece ilk if talimatındaki test geçecek ve else sonrası aksi durum koduna ilişkin blok icra edilmeyecektir. Sayının negatif girilmesi halinde ilk if talimatının else bloğuna girilecek ve blok içindeki ilk if talimatındaki test geçecek ve bu if talimatının else kısmı çalıştırılmayacaktır. Sayı sıfır girilmiş ise blok içindeki else kodu çalıştırılacaktır. Bunların dışında blok içinde tek bir if-else talimatı bulunmaktadır. Dolayısıyla blok içine almaya gerek de yoktur. Bu durumda kodun nihai hali aşağıda verilmiştir.

```
    if (sayi>0)
        cout << "Pozitif";
    else if (sayi<0)
        cout << "Negatif";
    else
        cout << "Sıfır";
```

Sarkan Else

Aşağıdaki program örneği incelendiğinde else, hangi if talimatına aittir? Böyle bir kod programcı tarafından yazılabilir ve derleyici buna hiçbir sıkıntı çıkarmaz.

```
if (cinsiyet=='E') if (yas<18) std::cout<<"Delikanlı"; else std::cout<<"Adam";
```

Bu durumda kodu aşağıdaki gibi okunaklı hale getirdiğimizde ilk `if` talimatının koşul kodunun (conditional code) ikinci if-else talimatı olduğu görülmektedir. Bu problem sarkan else (dangling else) problemi olarak bilinir. Kod bloğu kullanılmadan yazılan bu tip kodlarda else her zaman kendinden bir önceki if talimatına aittir.

```
    if (cinsiyet=='E')
        if (yas<18)
            std::cout<<"Delikanlı";
        else
            std::cout<<"Adam";
```

Switch Talimatı

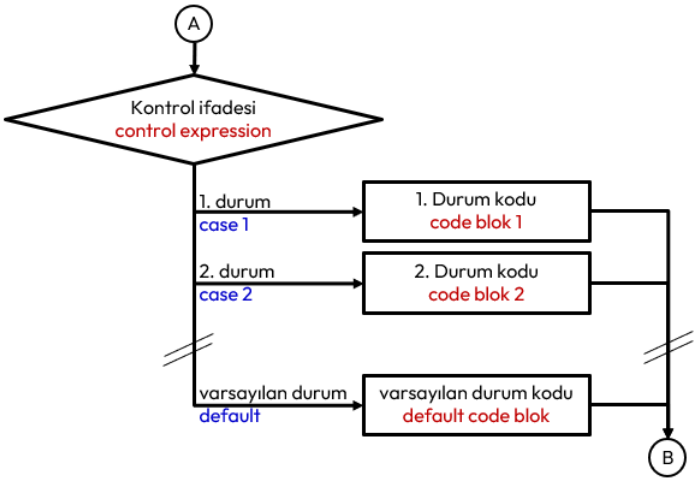
Switch talimatı, bir kontrol ifadesi (expression) sonucunda birden fazla alternatif arasında seçim yapılmasını sağlar. Sözde kodu aşağıda verilmiştir;

```
switch (kontrolifadesi) {
   case alternatif-1:
      alternatif-1-kodu;
      break;
   case alternatif-2:
      alternatif-2-kodu;
      break;
   // ...
```

```
   default:
      varsayılan-alternatif-kodu;
}
```

Switch talimatına ilişkin kurallar;

1. Kontrol ifadesi (control expression), sonucu tamsayı olan ifadedir ve bir kez değerlendirilir.
2. Switch bloğu zorunludur. Yani bloğu olmayan bir switch talimatı yazılamaz!
3. Her bir alternatif tamsayı değişmezdir (integer literal) izleyen ve iki nokta ile biten bir etiket (label) olarak tanımlanır. `case 1:`, `case 0:`, `case 'A':` gibi. Alternatif, bir değişken olamaz.
4. Blok için yazılan kod sonuna `break;` talimatı yazılarak switch bloğu dışına çıkılır. Zorunlu değildir, yazılmaz ise bir sonraki durum için yazılan kod icra edilir.
5. Tüm tamsayılar içerecek şekilde alternatiflerin tümü yazılmayabilir.
6. Blok sonunda yer alacak şekilde üzerinde yer alan alternatifler dışında kalan tüm alternatifler için `default:` etiketli alternatif yazılabilir. Zorunlu değildir.



Şekil 10. Switch Talimatı Sözde Kodu ve İcra Akışı

Aşağıda menü seçimi için yazılmış bir program örneği verilmiştir.

```
#include <iostream>
using namespace std;
int main() {
    int menu;
    cout << "Menü İçin Rakam Giriniz:";
    cin >> menu;

    switch(menu) {
    case 1:
        cout << "1 Numaralı Menüyü Seçtiniz.";
        cout << "Hamburger ve Ayran Hazırlanak.";
        break;
    case 2:
        cout << "2 Numaralı Menüyü Seçtiniz.";
        cout << "Patates Kızartması ve Kola Hazırlanacak.";
        break;
    case 3:
    case 4:
        cout << "3 veya 4 Numaralı Menüyü Seçtiniz.";
        break;
    default:
        cout << "1,2, 3 veya 4 Dışında Menü Seçtinniz.";
        cout << "Böyle bir Menümüz Yok!.";
    }
}
```

Programı aşağıdaki durumlar için çalıştırabiliriz;

1. Programda klavyeden `1` girildiğinde switch talimatında kontrol ifadesi test edilir ve sonucu `1` olduğuna karar verilir. Bu durumda `case 1:` etiketine gidilir ve bu alternatife ilişkin kodlar icra edilir. Yani ilk önce `cout << "1 Numaralı Menüyü Seçtiniz.";` icra edilir ve bir sonraki talimata geçilir. `cout << "Hamburger ve Ayran Hazırlanak.";` icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi switch bloğunun sonundan dışına çıkarır.
2. Programda klavyeden `2` girildiğinde switch talimatında kontrol ifadesi test edilir ve sonucu `2` olduğuna karar verilir. Bu durumda `case 2:` etiketine gidilir ve bu alternatife ilişkin kodlar icra edilir. Yani ilk önce `cout << "2 Numaralı Menüyü Seçtiniz.";` icra edilir ve bir sonraki talimata geçilir. `cout << "Patates Kızartması ve Kola Hazırlanacak.";` icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi switch bloğunun sonundan dışına çıkarır.
3. Programda klavyeden `3` girildiğinde switch talimatında kontrol ifadesi test edilir ve sonucu `3` olduğuna karar verilir. Bu durumda `case 3:` etiketine gidilir ve bu alternatife ilişkin icra edilecek kod yoktur. Sonrasında bulunan ilk talimat olan `cout << "3 veya 4 Numaralı Menüyü Seçtiniz.";` talimatıdır ve icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi switch bloğunun sonundan dışına çıkarır.
4. Programda klavyeden `4` girildiğinde switch talimatında kontrol ifadesi test edilir ve sonucu `4` olduğuna karar verilir. Bu durumda `case 4:` etiketine gidilir ve bir üst maddede belirtilen icra gerçekleşir.
5. Programda klavyeden `-1` , `0`, `12` ve `300` gibi bir sayı girildiğinde switch talimatında kontrol ifadesi test edilir ve sonucu `1`,`2`,`3`,`4` alternatiflerinden biri olmadığına karar verilir. Bu durumda `default:` etiketine gidilir. Bu etiket sonrası ilk önce `cout << "1,2, 3 veya 4 Dışında Menü Seçtinniz.";` icra edilir ve bir sonraki talimata geçilir. `cout << "Böyle bir Menümüz Yok!.";` icra edilir. Zaten blok sonuna ulaşılmıştır. Bloktan çıkılır.

Aşağıda klavyeden girilen bir sayının 4 rakamına bölünmesinde kalanı gösteren bir program verilmiştir.

```
#include <iostream>
using namespace std;
int main() {
    int sayi;
    cout << "Bir Sayı Giriniz:";
    cin >> sayi;
    switch(sayi%4) { //konrol ifadesi bir işlem içerir
    case 3:
        cout << "Sayı 4 rakamına bölündüğünde kalan 3 dir.";
        break;
    case 2:
        cout << "Sayı 4 rakamına bölündüğünde kalan 2 dir.";
        break;
    case 1:
        cout << "Sayı 4 rakamına bölündüğünde kalan 1 dir.";
        break;
    default: // Başka alternatif yoktur.
        cout << "Sayı 4 Rakamına TAM Bölünür";
        break;
    }
}
```

Üçlü Şart İşleci

Daha önce *İşleçler* başlığı altında işlenenlere ek olarak, ardışık işlemlerde (sequential operation) if talimatlarına gerek kalmadan bir koşula göre ifadeler (expression) işlenecek ise üçlü şart işleci (conditional tenary operator) kullanılır. Aşağıda üçlü işlecin iki sözdizimi verilmiştir;

```
koşul ? doğruifadesi : yanlışifadesi;
değişken = koşul ? doğruifadesi : yanlışifadesi;
```

Aşağıda oy kullanma durumu bu işleçle işlenmiştir;

```
#include <iostream>
using namespace std;
int main() {
    int yas;
    cout << "Yaşınız?: ";
    cin >> yas;
    (yas >= 18) ? cout << "Oy Kullanabilirsin." : cout << "Oy Kullanamazsın!";
}
```

Aşağıda başka kullanımlarına ilişkin kod örneği verilmiştir;

```
    int sayi,tek,cift;
    cout << "Bir Sayı Giriniz: ";
    cin >> sayi;
    tek= (sayi%2) ? 1 : 0;
    cift= tek ? 0 : 1;
    int sonuc1=tek ? sayi+30 : sayi+40;
    int sonuc2=cift ? sayi*30 : sayi*40;
```

İlişkisel Döngüler

İşlemciler iyi bir sayıcıdır. CPU içindeki kaydediciler (registers) sayma işlevini birçok matematiksel işlemi yapmak için de kullanılır. Program akışında belli problemlerin çözümünde, gerçekleştirilen belli adımların tekrarlanması sonucu gidilir. Bu tip problemler şimdiye kadar olan yöntemlerle yapılırsa, tekrar tekrar aynı kodu yazmak zorunda kalırız. Yazdığımız kodları tekrar tekrar icra edebilmek için ilişkisel döngü (relational loop) talimatlarını kullanırız.

Konsola 10 defa ismimizi yazdıran bir program örneğini ele alalım. Burada konsolda satır başı (new line) yapmak için konsola `std::endl` gönderilir.

```
#include <iostream>
using namespace std;
int main() {
    int yas;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
    cout << "ILHAN" << endl;
}
```

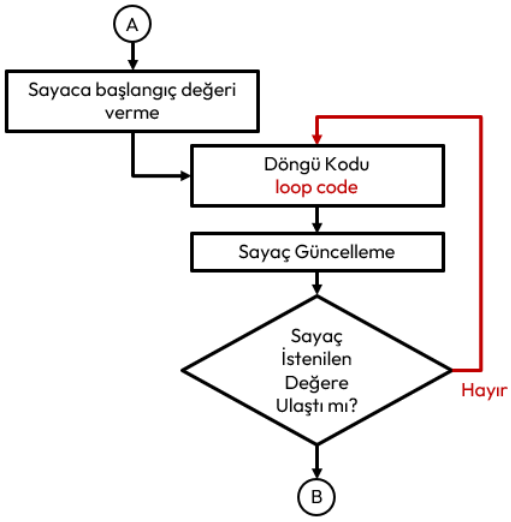
Program incelendiğinde aynı `cout` talimatının tekrar tekrar yazıldığını görürüz. Bu istenmeyen bir durumdur. Bunu yapmamak için sayaç olarak kullanılan bir değişken kullanırız.

Sayaç Kontrollü Döngüler

Belirlenen bir sayaç (counter) değişkeninin istenilen bir değere ulaşıp ulaşmadığı kontrol ederek kodun tekrar tekrar icra edilmesi sağlanır.

C ce C++ programlama dili ara seviye bir dil olduğundan `goto` talimatı desteklenir. 1968 Yılında *Edsger W. Dijkstra* GOTO/JUMP TO ifadelerini zararlı olarak ilan edilmiştir[15]. GOTO kullanmamak için;while, do-while ve for kontrol talimatları yapısal programlamaya eklenmiştir.

[15] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf



Şekil 11. Sayaç Kontrollü Döngü İcra Akışı

Yapısal programlama öncesinde bu döngü `goto` talimatıyla sağlanır. İcra sırasını etiketlenen bir yer olarak değiştirmek için `goto` talimatı kullanılır. Etiketlere kimlik verilirken yine değişken kimliklendirme (identifier definition) kuralları uygulanır.

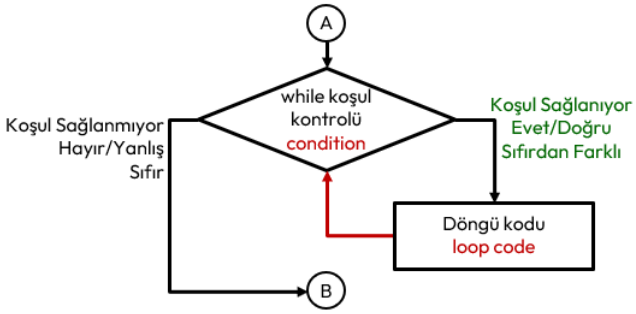
```
#include <iostream>
using namespace std;
int main() {
    int sayac=0; //Sayaca İlk Değer 0 Olarak Verildi
    Etiket: //Tekrar Edilecek Kodun Başı ETİKETLENİR. Etilet kimliği "Etiket"
        cout << "ILHAN" << endl;
        sayac++; //Sayaç Güncelleme
    if (sayac<10) //Sayaç Kontrolü
        goto Etiket; // İstenilen değere ulaşılmamış ise etikete git.
}
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu üç adet talimat içeren mantıksal satır (logical sequence) olmasına karşın 30 adet fiziksel satır (physical sequence) icra edilmiştir.

While Talimatı

While döngüsü, koşul (condition) `true` olduğu sürece tekrarlanacak kodu (loop code) icra eder. Tekrarlanacak kod, tek bir talimat (statement) olabileceği gibi bir kod bloğu da olabilir. Sözde kodu aşağıda verilmiştir;

```
while (koşul)
    döngü-kodu;
```



Şekil 12. While Talimatı İcra Akışı

Sayaç Kontrollü Döngüler başlığında verilen döngü, While talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <iostream>
using namespace std;
int main() {
    int sayac=0; //Sayaca İlk Değer 0 Olarak Verildi
```

```
    while (sayac<10) // While koşul kontrolü
    {  // döngü bloğu
        cout << "ILHAN" << endl;
        sayac=sayac+1; //Sayaç Güncelleme
    }
}
```

Sayaç güncelleme ifadesi koşul ifadesine eklenerek program daha da kısaltılabilir. Her koşul kontrolü sonrası sayaç artırılacaktır.

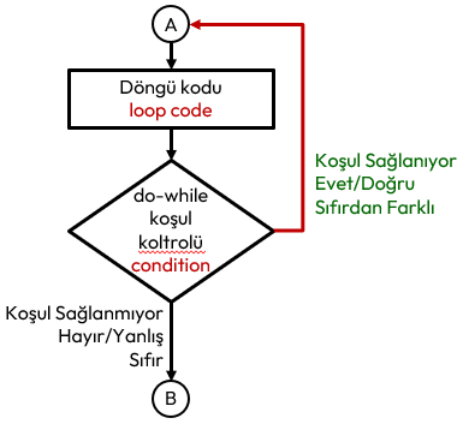
```
#include <iostream>
using namespace std;
int main() {
    int sayac=0; //Sayaca İlk Değer 0 Olarak Verildi
    while (sayac++<10) // hem sayaç güncelleme hem de koşul kontrolü
        cout << "ILHAN" << endl; // tek talimat olduğundan blok kullanılmadı
}
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu bir adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

Do-While Talimatı

Do-While döngüsü, `do` ile `while` saklı kelimeleri arasındaki tekrarlanacak kodu (loop code) koşul (condition) `true` olduğu sürece tekrarlayarak icra eder. Tekrarlanacak kod tek bir talimat (statement) olabileceği gibi bir kod bloğu da olabilir. While döngüsünde koşul kontrolü en başta yapılır, bu döngüde ise döngü bloğu en az bir kez çalıştırıldıktan sonra koşul kontrolü yapılır. Sözde kodu aşağıda verilmiştir;

```
do
    döngü-kodu;
while (koşul);
```



Şekil 13. Do-While Talimatı ve İcra Akışı

Sayaç Kontrollü Döngüler başlığında verilen döngü, Do-While talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <iostream>
using namespace std;
int main() {
    int sayac=0; //Sayaca İlk Değer 0 Olarak Verildi
    do {  // döngü bloğu
        cout << "ILHAN" << endl;
        sayac=sayac+1; //Sayaç Güncelleme
    } while (sayac<10); // Do-While koşul kontrolü
}
```

Sayaç güncelleme ifadesi koşul ifadesine eklenerek program daha da kısaltılabilir.

```
#include <iostream>
```

```
using namespace std;
int main() {
    int sayac=0; //Sayaca İlk Değer 0 Olarak Verildi
    do
        cout << "ILHAN" << endl;
    while (sayac++<10); // Do-While koşul kontrolü ve sayaç güncelleme
}
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu 1 adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

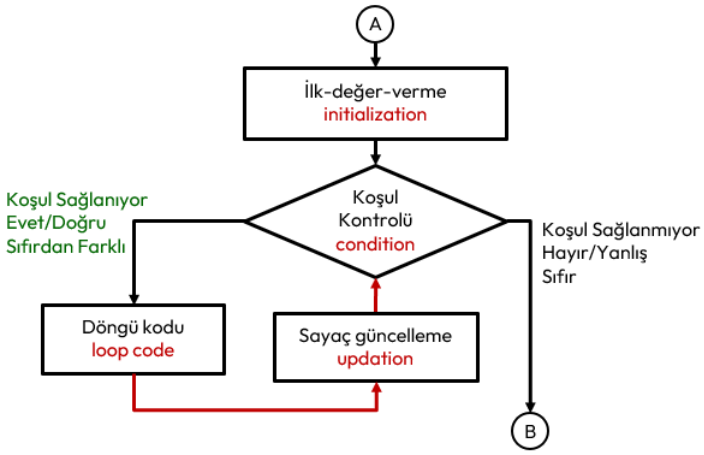
For Talimatı

For döngüsü özellikle tekrar edilen işlemlerin sayısı belli olduğunda kullanılan bir döngü yapısıdır. Sayaç kontrollü döngüler için en iyi seçimdir. Sözde kodu aşağıda verilmiştir.

```
for (ilk-değer-verme; koşul-kontrolü; sayaç-güncelleme)
    döngü-kodu;
```

For talimatı aşağıdaki şekilde icra edilir;

1. İlk değer verme ifadesi bir kez icra edilir.
2. Sonra koşul kontrolü yapılır. Eğer test sonucu `true` ise döngü kodu icrasına geçilir. Aksi halde döngüden çıkılır.
3. Döngü kodu icra edilir.
4. Döngü kodu icrası bitince sayaç güncellemeleri yapılır ve tekrar ikinci adımdaki koşul kontrolüne dönülür.



Şekil 14. For Talimatı İcra Akışı

Sayaç Kontrollü Döngüler başlığında verilen döngü, For talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <iostream>
using namespace std;
int main() {
    int sayac;

    for (sayac=1; sayac<10; sayac++)
        cout << "ILHAN" << endl;
}
```

Buradaki döngü mantıksal olarak iki satır talimattan oluşmaktadır. Ancak fiziksel olarak 10 satır icra edilmiştir. Aşağıda konsola yazdığımız her bir satırın başına satır numarası koyan bir program gösterilmektedir. Ayrıca sayaç sadece for bloğunda kullanılacak ise ilk değer verme ifadesinde tanımlanabilir.

```
#include <iostream>
using namespace std;
int main() {
```

```
    for (int sayac=1; sayac<=10; sayac++)
        cout << sayac << ". ILHAN" << endl;
}
/* Program çalıştırıldığında:
01-ILHAN
02-ILHAN
03-ILHAN
04-ILHAN
05-ILHAN
06-ILHAN
07-ILHAN
08-ILHAN
09-ILHAN
10-ILHAN

...Program finished with exit code 0
*/
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu bir adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

For döngüsünde bir sayaç ile çalışılabileceği gibi birden fazla sayaç ile de çalışılabilir. Hem ilk değer verme hem de sayaç güncellemede birden fazla sayaca ilişkin işlem yapılabilir. Bunun için virgül işleci (comma operator) kullanılır.

```
#include <iostream>
using namespace std;
int main() {
    int sayac1,sayac2;

    for (sayac1=0,sayac2=10; sayac1<10; sayac1++,sayac2--)
        cout << sayac1 << "-" <<sayac2 <<"- ILHAN" << endl;
}
/* Program çalıştırıldığında:
0-10- ILHAN
1-9- ILHAN
2-8- ILHAN
3-7- ILHAN
4-6- ILHAN
5-5- ILHAN
6-4- ILHAN
7-3- ILHAN
8-2- ILHAN
9-1- ILHAN

...Program finished with exit code 0
*/
```

Aşağıda klavyeden girilen iki sayı aralığında tek ve çift sayıların toplamını bulan ve ekrana yazan programı verilmiştir.

```
#include <iostream>
using namespace std;
int main() {
   int sayac, sayi1,sayi2;
   int tekToplam=0,ciftToplam=0;
   cout << "İki Sayı Giriniz: ";
   cin >> sayi1 >> sayi2;
   if (sayi2<sayi1) { //sayi2, sayi1 den büyük olmalı.
      int temp=sayi1; //küçük ise yer değiştiriyoruz.
      sayi1=sayi2;
      sayi2=temp;
```

```
   }
   for (sayac=sayi1; sayac<=sayi2; sayac=sayac+1) {
      if (sayac%2==1) //sayac tek mi?
         tekToplam+=sayac;
      else
         ciftToplam+=sayac;
   }
   cout << "Tek Toplam:" << tekToplam << " Çift Toplam:" << ciftToplam;
}
/* Program çalıştırıldığında:
İki Sayı Giriniz: 9 17
Tek Toplam: 65 Çift Toplam: 52

...Program finished with exit code 0
*/
```

İç İçe Döngü Kodu

Döngüler içinde yer alan döngü kodu (loop code) bir başka döngüyü içerebilir. Örneğin ekrana satır ve sütun içeren bir şekil oluşturmak istediğimizde iç içe döngü kullanırız. Aşağıda buna ilişkin bir örnek verilmiştir. İlk for döngüsünün tekrarlanan kodu ikinci for döngüsüdür.

```
#include <iostream>
using namespace std;
int main() {
  int satir,sutun;
  for (satir=0; satir<5; satir++)
  {
     for (sutun=0; sutun<4; sutun++)
        cout << "*";
     cout << endl;
  }
}
/* Program çalıştırıldığında:
****
****
****
****
****

...Program finished with exit code 0
*/
```

Bir başka örnek olarak elemanları, satır ve sütun çarpımları olan 4x5 boyutlarındaki matrisi ekrana yazdıran program;

```
#include <iostream>
using namespace std;
int main() {
  int satir,sutun;

  //Başlığı Yazdıran Kısım
  cout << "  Sutun: ";
  for(sutun=0; sutun<3;sutun++)
   cout << sutun+1 << " ";
  cout << endl;

  //Matrisi Yazdıran Kısım
  for (satir=0; satir<5; satir++) {
    cout << satir+1 <<".Satir:" << " ";
    for(sutun=0; sutun<3;sutun++)
      cout << sutun << " ";
```

```
    cout << endl;
  }
}
/* Program çalıştırıldığında:
  Sutun: 1 2 3
1.Satir: 0 1 2
2.Satir: 0 1 2
3.Satir: 0 1 2
4.Satir: 0 1 2
5.Satir: 0 1 2

...Program finished with exit code 0
*/
```

Gözcü Kontrollü Döngüler

Döngüler her zaman bir sayaca bağlı çalıştırılmaz. Bir döngüden çıkış koşulu döngü kodu içerisinde üretilen bir değere bağlı ise bu değere gözcü değeri (sentinel value) adı verilir. Buradaki gözcü değeri beklenen bir değer dışındaki bir değerdir. Buna örnek olarak klavyeden `0` girilene kadar girilen rakamları toplayan bir program verilmiştir.

```
#include <iostream>
using namespace std;
int main() {

   int sayi; /* okunan tamsayı */
   int toplam=0; /* girilen sayıların toplamı. başlangıçta 0*/
   do {
      cout << "Bir sayı giriniz (Bitirmek için 0): ";
      cin >> sayi;
      toplam+=sayi; // sayı 0 girilse bile toplam etkilenmez
   } while (sayi != 0); //sentinel value 0

   cout << "Girilen Sayıların Toplamı:" << toplam;
}
```

Aşağıda pozitif sayı girilmesini sağlayan bir kod örneği verilmiştir.

```
#include <iostream>
using namespace std;
int main() {
    int sayi,pozitifSayi=0;
    do { /* Pozitif girilmesini zorluyoruz */
        cout << "Lütfen pozitif sayı giriniz: ";
        cin >> sayi;
        if (sayi<=0)
            cout << "HATA:Pozitif Sayı GİRMEDİNİZ!" << endl;
    } while (sayi <= 0);
    pozitifSayi=sayi;
    cout <<"Girilen pozitif tamsayı: " << pozitifSayi;
}
```

Aynı örnek bir başka şekilde aşağıdaki gibi kodlanabilir.

```
#include <iostream>
using namespace std;
int main() {
    int sayi,pozitifSayi=0;
    cout << "Lütfen pozitif sayı giriniz: ";
    cin >> sayi;
    while (sayi <= 0) /* Pozitif girilmesini sağlıyoruz */
    {
          cout << "HATA:Negatif sayı giriniz!" << endl;
```

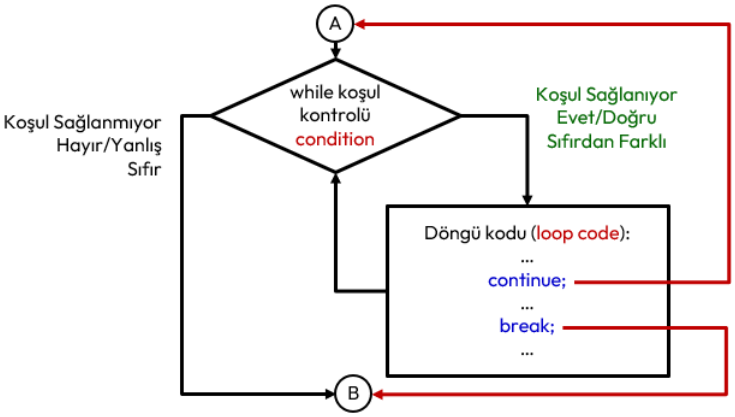
```
        cout << "Lütfen pozitif sayı giriniz: ";
        cin >> sayi;
    }
    pozitifSayi=sayi;
    cout <<"Girilen pozitif tamsayı: " << pozitifSayi;
}
```

Dallanmalar

Dallanmalar (jump) programın akışını bir başka talimata yönlendiren talimatlardır.

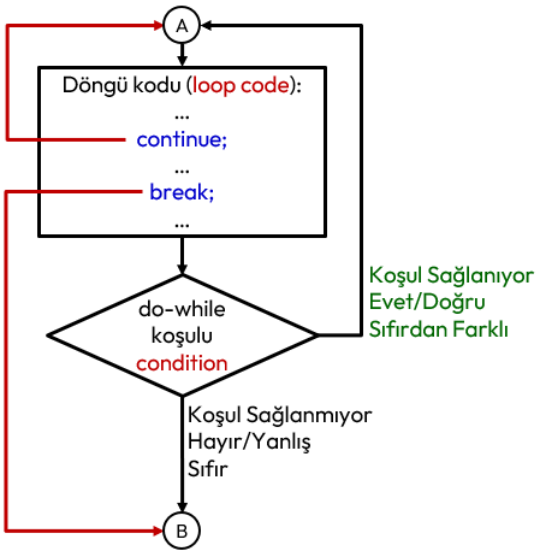
Continue ve Break Talimatları

Döngüdeki `continue` talimatı, geçerli yinelemeyi (iteration) sonlandırır ve bir sonraki yinelemeye devam edilir. Dolayısıyla sadece döngü kodları (loop code) içinde kullanılabilir. `break` Talimatı döngüyü sonlandırır ve program icrası döngü sonrası ilk talimattan devam eder. Bunu daha önce switch talimatında görmüştük. Bu talimat, aynı şekilde `while`, `do-while` ve `for` döngüleri ile kullanılabilir. Bu talimatlarının döngü kodlarında kullanılması halinde icra akışı aşağıdaki şekilde verilmiştir.



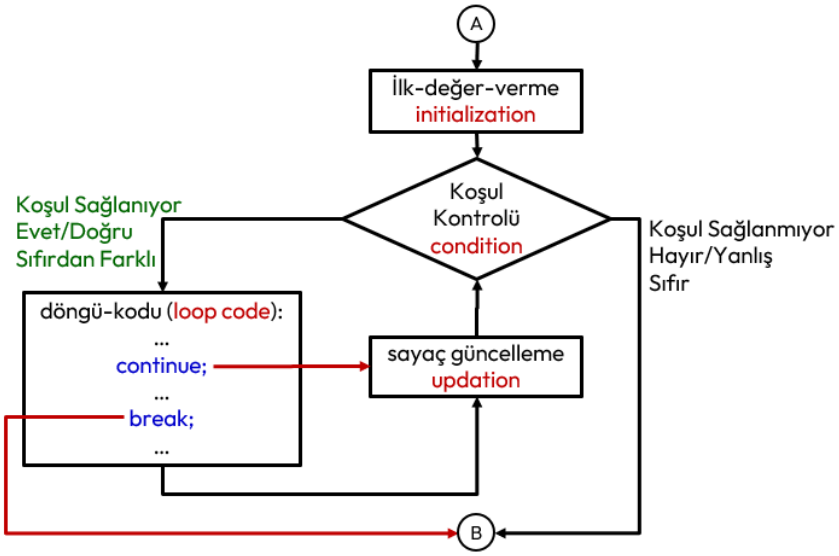
Şekil 15. While Döngü Kodunda Break ve Continue Talimatları İcra Akışı

While döngüsünde, döngü bloğunda `continute` kullanmadan önce koşulu değiştirecek bir talimat vermeliyiz. Aksi halde sonsuz döngüye girmiş oluruz. Do döngüsünde eğer `continue` talimatı bir şarta bağlanmalıdır. Yoksa sonsuz döngüye girilebilir.



Şekil 16. Do-While Döngü Kodunda Break ve Continue Talimatları İcra Akışı

Benzer durum for döngüsü için de geçerlidir. Ancak for döngüsünde `continue` talimatı sonrasında bir sonraki yineleme için önce sayaç güncelleme ifadeleri icra edilir ve sonra koşul testi yapılır. Bu durum göz önüne alınarak bu talimat kullanılmalıdır. Bu talimatlar döngü kodu içinde amacımıza uygun olarak birden çok kullanabiliriz.



Şekil 17. For Döngü Kodunda Break ve Continue Talimatları İcra Akışı

Aşağıda `0` ile `15` arasındaki sayıların beşe bölünenleri ve `10` dışındaki sayılar ekrana yazdıran bir program örneği verilmiştir.

```
#include <iostream>
using namespace std;
int main() {
    int i;
    for (i=0; i<=15; i=i+1)
    {
        if (i<7) //i'nin değeri 7 den küçük ise
            continue;  //bir sonraki yinelemeye (iteration) geç

        if (i==10) //i'nin değeri 10 ise
            continue;  //bir sonraki yinelemeye (iteration) geç
        if (i%5==0) // i'nin değeri 5 in katı ise
            continue;  //bir sonraki yinelemeye (iteration) geç
        cout << "i=" << i << endl;
   }
}
/*Program Çıktısı:
i = 7
i = 8
i = 9
i = 11
i = 12
i = 13
i = 14

...Program finished with exit code 0
*/
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu üç adet talimat içeren mantıksal satır (logical sequence) olmasına karşın 42 adet fiziksel satır (physical sequence) icra edilmiştir. Bazı `continue` talimatları `cout` talimatının icrasını engellemiştir.

Aşağıda pozitif sayı girilmesini zorlayan sonsuz döngü içeren program verilmiştir. Programda döngü, pozitif sayı girildiğinde break talimatı ile kırılmaktadır.

```
#include <iostream>
using namespace std;
int main() {
    int sayi;
    int pozitifSayi=0;
```

```
    while (1) /* Sonsuz döngü */
    {
        cout << "Lütfen pozitif sayı giriniz: ";
        cin >> sayi;
        if (sayi<=0)
            cout << "HATA:Pozitif Sayı GİRMEDİNİZ!" << endl;
        else
            break; //döngüden çık
    }
    pozitifSayi=sayi;
    cout << "Girilen pozitif tamsayı: " << pozitifSayi;
}
```

Return Talimatı

Fonksiyonlarda çokça kullanacağımız `return` talimatı bir fonksiyonun üreteceği ya da belirlenen değeri geri döndürür. Kullanıldığı yerde belirlenen değer ile içinde bulunduğu fonksiyon bloğu dışına çıkılır. *FONKSİYONLAR* başlığı altında detaylı incelecek olup bir örneği *En Basit C++ Programı* başlığında bir örneği verilmiştir.

Goto Talimatı

Tanımlı bir etikete program akışını yönlendiren `goto` talimatıdır. *Sayaç Kontrollü Döngüler* başlığında örneği verilmiştir.