T.C.

GİRESUN ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

IEEE STUDENT BRANCH ÖĞRENCİ TOPLULUĞU

ARDUINO ATÖLYESİ ÇALIŞMA FÖYÜ

Bülent Çobanoğlu'na, "Derinlemesine Arduino" (Abaküs Yayınları) kitabından alıntılama yapmamıza izin verdiği için teşekkür ederiz.

BÖLÜM 1: Mikrodenetleyici Nedir?

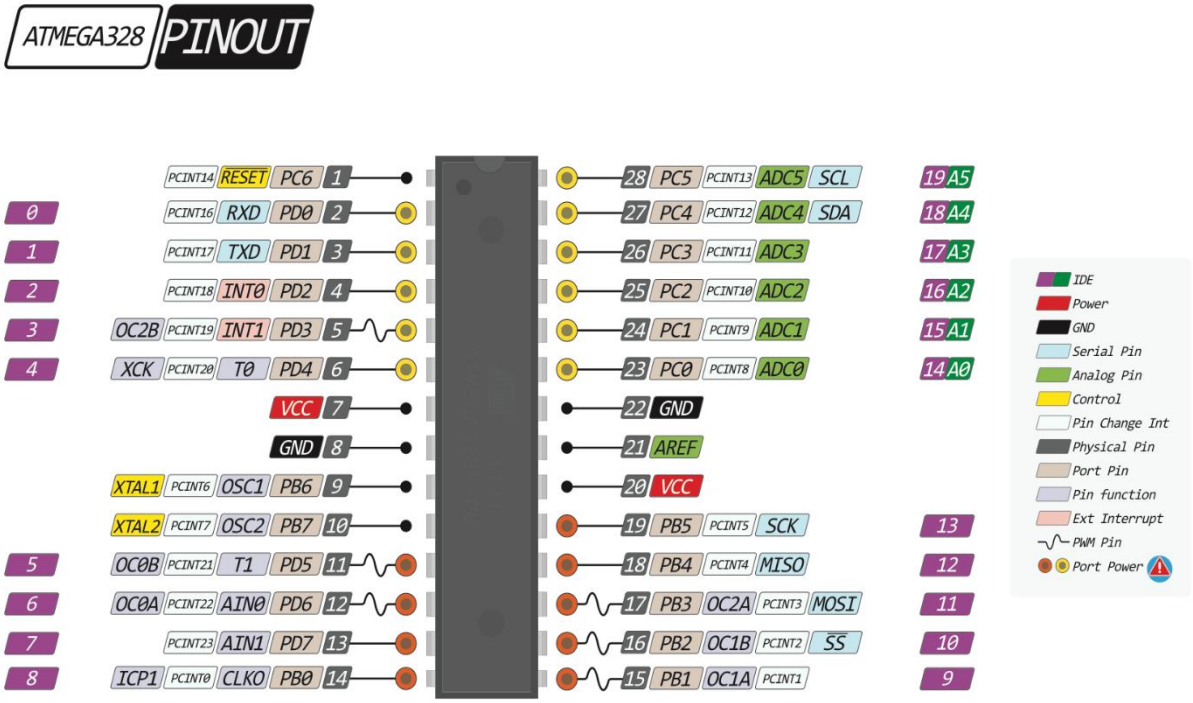
Çevresel arabirimlerden veya sensörlerden alınan harici sinyalleri işleyen ve kullanıcısının daha önce yazdığı komutlara göre karar veren elektronik devre elemanlarıdır.

Mikrodenetleyiciler tek bir yonga içerisinde düşük bir bilgisayar olarak düşünülebilir. Ancak bu kadar küçük olduklarından bazı PC özelliklerinden feragat edilmiştir (hız, kapasite vb..). Bu yüzden mikro denetleyiciler daha çok gömülü sistemlerde kullanılır.

Bir mikrodenetleyicide aşağıdaki birimler bulunur:

1. Merkezi İşlem Birimi (Central Processing Unit - CPU)
2. Hafıza Birimleri (Memory Units)
3. İletişim Yolları (Buses)
4. Giriş-Çıkış Birimleri (I/O Ports)

Arduino ise doğrudan bir mikrodenetleyici değildir. İçerisinde ATMEL ATmega Series mikrodenetleyicilerinin olduğu bir mikrodenetleyici platformudur. Bu yüzden Arduino üzerinde bir mikrodenetleyiciden çok daha fazla eleman vardır. Bunların bazıları programlama kartı işlevi görürken bazıları da mikrodenetleyicinin çalışması için gerekli osilatör, regülatör gibi devrelerdir.



Şekil 1.1

Şekil 1.1'de Arduino UNO'da kullanılan ATMEL ATmega 328'in pin yapısı verilmiştir.

1.2 Arduino UNO'nun Donanımsal Özellikleri

14 adet dijital giriş çıkış, 6 analog giriş pinine sahip sahiptir.

16MHz dahili kristal osilatör kullanılmıştır.

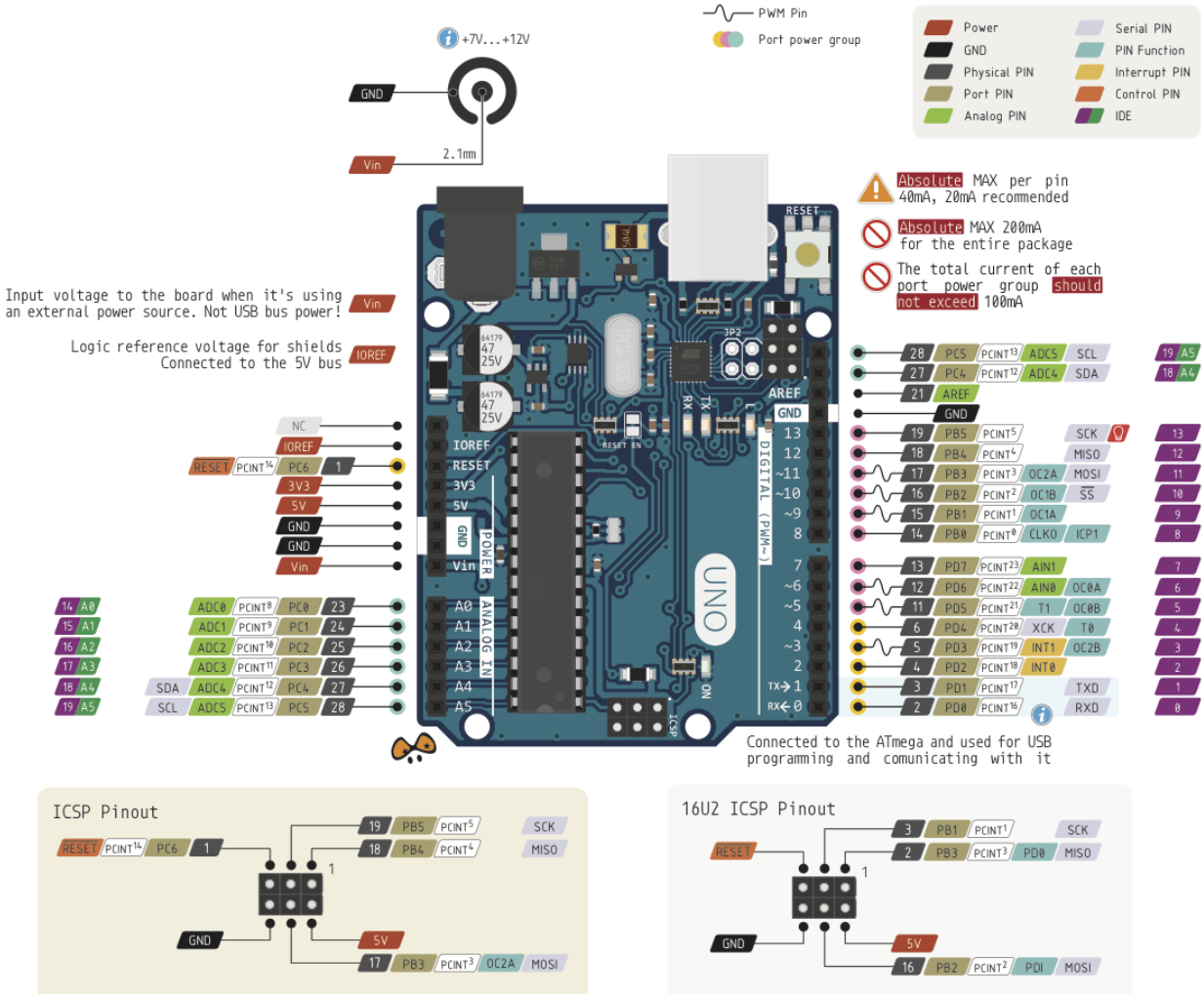
32KB ROM, 1KB EEPORM, 2KB SRAM hafızasına sahiptir.

Bir adet seri haberleşme portu bulunur. 0 -> (Receiver) RX,1 -> (Transmitter) TX pini olarak belirlenmiştir. Bu pinler TTL seri haberleşmesi için kullanılır.

14 dijital pinin 6 tanesi PWM (Pulse With Modulation) çıkış üretebilir. PWM özelliği daha çok sayısal analog dönüşüm işlemlerinde motor sürme, LED şiddeti gibi alanlarda kullanılır.

10, 11, 12, 13'nolu pinler SPI haberleşme pinleridir.

A0-A5 arası 6 adet analog giriş pini 10 bit çözünürlükte analog veri okuyabilir. Yani $2^{10} = 1024$ farklı değer elde edilebilir.



Şekil 1.2

Şekil 1.2'de Arduino UNO'nun pin çıkışları verilmiştir. Bazı önemli pinler için aşağıda özellikleri verilmiştir:

Vin Pini: Arduino kartlarına harici güç vermek için kullanılan pindir.

5V Pini: Bu pin regülatör devresinden 5V almak için kullanılır. 200mA'e kadar çıkış verebilir.

3.3V Pini: Bu pin 3.3V güç kaynağıyla çalışan sensörler içindir. Bu pinden 50mA'e kadar çıkış alabilirsiniz.

GND Pini: 0V referans gerilimidir. (GROUND/Şase)

IOREF Pini: Bu pin mikrodenetleyicinin çalıştığı referans gerilimini verir.

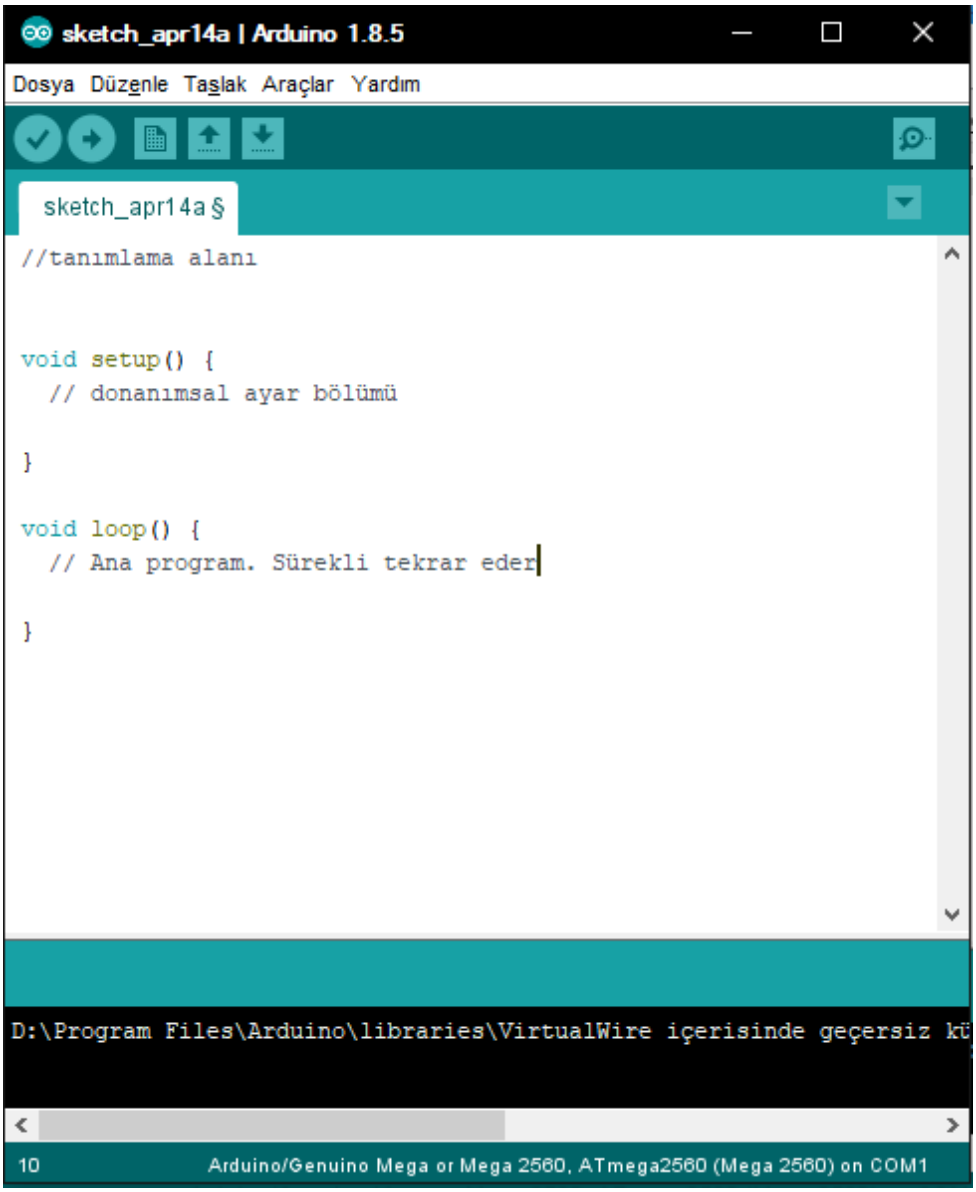
AREF Pini: AREF Analog/Digital dönüşümleri için bir referans pinidir ve analog ölçümlerde hassasiyeti arttırmak için kullanılabilir. AREF bağlantısı kullanılarak ölçüm aralığı 0-5V arasında değiştirilebilir. Örneğin 0-2.5V'luk ölçümler gerçekleştirilecekse AREF pinine 2.5V bir gerilim uygulanarak ölçüm hassasiyeti 2 kat arttırılabilir.

Reset Pini: Arduino'yu en baştan başlatmak için kullanılır. RAM'deki veriler silinir ancak ROM ve EEPROM'a dokunulmaz.

BÖLÜM 2: Arduino Temel Komutları

Her Arduino programı 3 temel bölümden oluşur;

1. Değişkenlerin, fonksiyonların (veya prototiplerinin) ve kütüphanelerin tanımlandığı tanımlanma alanı
2. Gerekli donanımsal ayarların yapıldığı setup() fonksiyonu
3. Ana programın olduğu ve sürekli çalışan loop() fonksiyonu



Şekil 2.1

Şekil 2.1'de Arduino kartları programlamak için geliştirilen Arduino IDE ortamı gösterilmiştir. Buradaki bazı simgeleri ve görevlerini incelersek;

 Verify(Doğrula): Kodu doğrular varsa yazım hatalarını belirtir.

 Upload(Yükle): Kodu derler ve eğer hata yoksa Arduino'ya yükler.

 New(Yeni): Yeni bir sketch (kod sayfası) açar.

 Open(Aç): Tüm Sketchbook'taki örnek uygulama listesi gösterilir.

 Save(Kaydet): Programınızı kaydeder.

 Serial Monitor: Seri haberleşme ekranını açar.

Şekil 2.1 verilen IDE ekranının sağ en altında Arduino modeli ve bağlı olduğu seri port (COM) yazılıdır.

2.1 Tanımlama Bloğu

Programda kullanılacak global değişkenlerin, harici kütüphanelerin, sabitlerin ve nesne tanımlamalarının yapıldığı alandır. Örneğin:

```
1 #include <Servo.h>    //Servo kütüphanesi eklendi
2 Servo sg90;           //sg90 adında Servo nesnesi oluşturuldu
3 int ledPin = 0;       //0.Pine tanımlayıcı atandı
4 int sayac;            //sayac adlı global değişken oluşturuldu.
```

2.2 Kütüphane Dosyaları

Bir Arduino programında, derleyici ilk önce # ile başlayan komutları ön işlemciye alır. Eğer IDE ile birlikte gelen dışında kütüphane kurmak için **"Sketch->Import Library"** yolunu takip edebilirsiniz.

2.3 setup() Fonksiyonu

Her Arduino programında bulunan gerekli donanımsal ayarların yapıldığı fonksiyondur. Bu fonksiyon Arduino çalıştırıldığında bir kere çalışır. Genellikle, seri haberleşme hızı için kullanılan **Serial.begin(baundRate)** veya pinlerin giriş çıkış olmasını ayarladığımız **pinMode(pin_no, pin_Satate)** gibi tek seferlik çalıştırılması gereken kodlar bu kısımda yer alır.

2.4 loop() Fonksiyonu

Her Arduino programında bulunan esas işi yapan komutların yer aldığı fonksiyondur. Bu fonksiyon içine yazılan komutlar sonsuz döngü halindedir.

2.5 Temel Arduino Fonksiyonları

Fonksiyon	Açıklama

pinMode(pin, Mode)	Bir dijital pin için giriş veya çıkış pini seçmemizi sağlayan komuttur. **pin** kısmına hangi pin için atama yapılacağı, **Mode** kısmına ise çıkış için **OUTPUT**, giriş için **INPUT** deyimi yazılır.
	
	`pinMode(6, OUTPUT);  //6.pin çıkış`
	`pinMode(7, INPUT);  //7.pin giriş`

digitalWrite(pin, value)	Bir dijital çıkış pininin lojik seviyesini değiştirmek için kullanılır. **pin:** Değiştirilmek istenen pin **value:** pine atanmak istenen lojik seviye: **HIGH:** pini lojik 1 seviyesine getirir. **LOW:** pini lojik 0 seviyesine getirir. `digitalWrite(7, HIGH);  //7.pin 1(+5V)`
digitalRead(pin)	Bir dijital giriş pinine uygulanan lojik seviye eğer 1(+5V) ise **HIGH**, 0(0V) ise **LOW** geri çevirir. Genellikle bir değişken yardımıyla kullanılır. `value = digitalRead(8);`
delay(value)	Programı **value** milisaniye boyunca bekletir. `delay(250);  //250mS bekler`

Serial.begin(baundRate)	Seri haberleşme hızını baundRate'e ayarlar. Arduino 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200 hızlarını destekler.
	
	`Serial.begin(9600);`
	`Serial.begin(115200);`

analogRead(pin)	Bir analog giriş pininden alınan veriyi 0(0V)-1023(5V) arası olacak şekilde dijitale çevirerek geri gönderir. `analogVal = analogRead(A3);`
analogWrite(pin, value)	Bir pindeki **PWM** modülasyonunu kullanmayı sağlar. Yani **PWM** olarak tanımlanmış dijital bir **pin**e **value** ile tanımlanmış analog (0 ila 255) değer yazar. `analogWrite(10, 135);`

analogReference(type)	Analog giriş için referans gerilimini **type**'a göre değiştirir. **type** Arduino UNO için; **DEFAULT:** Referans gerilimini 5V yapar. **INTERNAL:** Referans gerilimini 1.1V yapar. **EXTERNAL:** Referans gerilimini **AREF** pinine uygulanan 0-5V arasındaki gerilime eşitler. `analogReference(DEFAULT);`

PWM (Pulse With Modulation)

PWM (Darbe Genişlik Modülasyonu) üretilecek dijital darbelerin genişliklerini kontrol ederek, üretilmek istenen analog sinyalin dijital pin çıkışında görülmesini sağlayan bir yöntemdir. PWM elektrik-elektronikte birçok farklı alanda kullanılır. Darbelerin genişliğini ayarlamak için **analogWrite()** fonksiyonu kullanılır.

LED_BUILTIN

Arduino UNO'da 13.dijital pine bağlı, TX ve RX LED'leri üzerinde 'L' harfi ile gösterilen bir LED bulunmaktadır. Bu LED programlarda **LED_BUILTIN** deyimi ile ifade edilir. Eğer 13.pin HIGH olursa bu LED'de yanar.

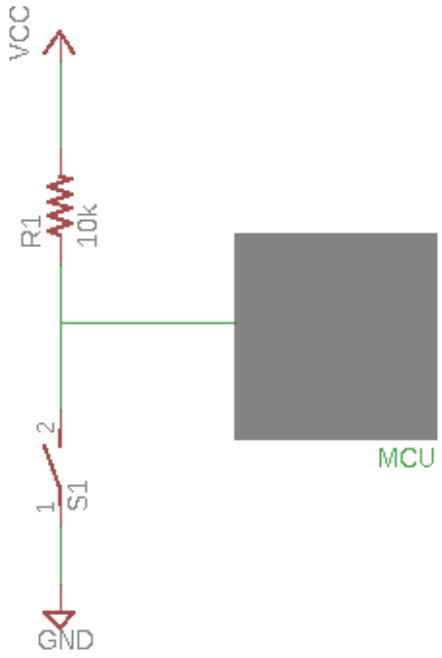
Örnek 1: Blink

Arduino UNO'nun 13.dijital pinine bağlı bir LED'i (LED_BUILTIN) 1sn aralıklarla yakıp söndürecek bir program yazınız.

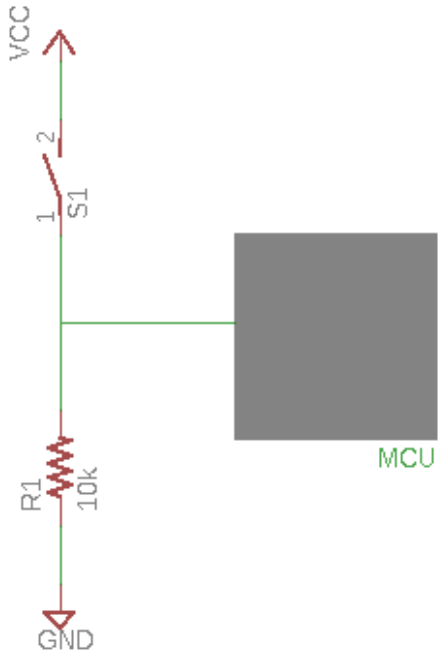
```
1  void setup() {
2       pinMode(13, OUTPUT);  //13.pin çıkış pini yapılıyor
3  }
4  void loop() {
5       digitalWrite(13, HIGH);    //13.pin 1 (HIGH, +5V) yapılıyor
6       delay(1000);          //1000mS = 1sn bekleniyor
7       digitalWrite(13, LOW);     //13.pin 0 (LOW, 0V) yapılıyor
8       delay(1000);          //1sn bekleniyor
9  }
```

Pull-Up ve Pull-Down Dirençleri

Pull-Up ve **Pull-Down** butonlu devreler için tasarlanmış özel bir direnç devresidir. 8-Bit AVR Arduino kartlar dahili **Pull-Up** direncine sahiptir.



Şekil 2.2



Şekil 2.3

Yukarıda Şekil 2.2'de Pull-Up, Şekil 2.3'te Pull-Down bağlantıları görünmektedir.

Bu bağlantıların yapılmazsa, butona basılmadığında mikrodenetleyiciye kararsız bir lojik sinyal gidecektir bu da programların hatalı çalışmasına kararsız yapıların oluşmasına neden olur.

Pull-Up bağlantıda butona basılmadığı zaman Vcc'den çıkan akım mikrodenetleyiciye girer. Yani mikrodenetleyicinin ilgili pini 1(HIGH) olmuş olur. Butona basıldığında ise mikrodenetleyicinin pini doğrudan toprağa bağlanmış olduğundan pinde okunan değer 0(LOW) seviyesindedir.

Pull-Down bağlantıda ise butona basılmadığında mikrodenetleyicinin pini 10k'luk dirençle toprağa bağlandığından pin 0(LOW) seviyesindedir. Butona basıldığında ise doğrudan Vcc'ye bağlanmış olduğundan 1(HIGH) seviyesindedir.

Özetle **Pull-Up**'ta butona basılmadığında **LOW**, basıldığında **HIGH**

Pull-Down'da butona basılmadığında **HIGH**, basıldığında **LOW** seviyesi mikrodenetleyici pinine gönderilir.

Bağlantı şekillerinin bu özelliklerine kodlama yaparken dikkat ediniz.

Eğer dahili Pull-Up direnci kullanılacaksa dirence gerek kalmadan direk butonun bir ucunu mikrodenetleyicinin pinine, diğerini de toprağa(GND) bağlanır. Ve pin tanımlamasında `pinMode(pin, INPUT_PULLUP);` kullanılır.

Bölüm 3: Değişkenler

3.1 Arduino'da Değişken Tanımlanması

Değişkenler içerisinde veri saklamak için bellekten alınan, tipini tanımlarken bizim belirttiğimiz alanlardır. Değişkenler RAM'den boş bir alan alırlar ve bu alana bizim belirttiğimiz isimle ulaşmamızı sağlarlar.

Arduino söz dizimi olarak C/C++ ve JAVA dillerinin kurallarını kullanır. Dolayısıyla bir değişken tanımlarken bu dillerdeki kurallara dikkat edilmesi gerekmektedir.

	Veri Tipi	Boyutu	Değer Aralığı
Mantıksal	**bool**	1 Byte	**true, false**
Karakter	**char**	1 Byte	Rakamlar ve A'dan Z'ye karakterler
Tam Sayı	**unsigned char**	1 Byte	0 to 255
	short	2 Byte	-32,768 to 32,767
	unsigned int	2 Byte*	0 to 65,535
	int	2 Byte*	-32,768 to 32,767
	long	4 Byte	2,147,483,648 to 2,147,483,647
	unsigned long	4 Byte	0 to 4,294,967,295
	word	2 Byte*	0 to 65,535
Reel Sayı	**float**	4 Byte	±3.4028235E+38
	double	4 Byte	±3.4028235E+38

*Buradaki değerler Arduino UNO içindir. Bazı kartlarda farklılık gösterebilir

3.2 Değişkenlere İsim Verme Kuralları

- Bir değişkenin ilk harfi rakam olamaz.
- Değişken isimlerinde alt tire (_) hariç özel karakter bulunamaz.
- Türkçe karakter kullanılamaz. (ş, ö vs.)
- Arduino komutları (int, loop, float vb.) değişken adı olamaz.
- Arduino küçük-büyük harf duyarlı bir dildir. LED ile led farklı değişken isimleridir.

Kurallar haricinde programı daha okunaklı ve anlaşılır kılmak için bazı iyi programlama önerileri vardır.

Değişken isimleri anlamlı olmalı. Örneğin LED'in durumunu kontrol etmek için ledDurum adında bir değişken oluşturmak gibi.

Eğer değişken adı iki veya daha fazla kelimeden oluşacak ise baştaki kelime hariç diğer kelimelere büyük harfle başlanır. Örneğin ledPin, triggerPinDurum, potDegeri gibi

3.3 Değişkenlerin Faaliyet Alanları ve Ömürleri

Bir değişken tanımlandığı alana göre programın tamamında ya da belli bir fonksiyonda faaliyet gösterebilir. Bu faaliyet alanına göre iki tip değişkenden bahsedebiliriz. **Global (genel)** ve **local (yerel)** değişkenler.

Global Değişkenler

Herhangi bir fonksiyonun içinde değil en üstte tanımlanma bloğunda tanımlanan değişkenlerdir. Program bitene kadar bellekten alınan alan geri verilmez ve değişkenin değeri korunmaya devam eder. Bütün fonksiyonlarda kullanılabilir. Yerel fonksiyonlarda aynı isimle farlı değişken oluşturulamaz.

Local Değişkenler

Bir blok veya fonksiyon içinde tanımlanırlar. Sadece tanımlandıkları alanda kullanılırlar. Bir fonksiyonda tanımlandılarsa o fonksiyon bittikten sonra alınan alan belleğe geri verilir. Farklı fonksiyonlar içinde aynı ada sahip yerel değişkenler tanımlamak mümkündür.

```
1  int globalVar;
2
3  void setup() {
4       int localVar;
5  }
6  void loop() {
7
8  }
9
10 void fonksiyon(void) {
11       int localVar;
12
13 }
```

Yukarıdaki örnekte *globalVar* global bir değişkendir ve bütün fonksiyonların içinde aynı isimle kullanılabilir. *localVar*'lar ise yerel değişkenlerdir. Burada iki tane aynı isimle tanımlanmış *localVar* değişkeni var. Bu iki değişken birbirinden farklıdır.

3.4 Stringler

Stringler birden fazla karakteri veya bir metini bellekte tutmak için kullanılan **char** dizileridir. JAVA'da olduğu gibi Arduino'da da Stringler bir nesnedir (object). String'in varsayılan değeri **null**'dur.

String tanımlamak için farklı yollar vardır. Aşağıda belirtilen tüm ifadeler bir *ad* Stringi tanımlamak için kullanılabilir.

```
String ad = "Istanbul";
char ad[] = {'I', 's', 't', 'a', 'n', 'b', 'u', 'l', '\0'};
char ad[9] = {'I', 's', 't', 'a', 'n', 'b', 'u', 'l'};
char ad[] = "Istanbul";
char ad[9] = "Istanbul";
```

char ile tanımlama yaparken boyut belirtilmediğinde en sona **sonlandırıcı karakter '\0'** koyulduğunu unutmayın!

Bir kelime ile atamak için çift tırnak, harf harf atama yapmak için tek tırnak kullanıldığına dikkat edin!

3.5 Sabit Tanımlama

Değeri tüm program boyunca değişmeyen ifadelere sabit denir. Arduino'da C/C++ dillerinde olduğu gibi **const** ve **#define** olmak üzere iki tip sabit tanımlama şekli vardır.

Komut	Örnek
`#define sabitAdi degeri` `const veritipi sabitAdi = degeri;`	`#define ledPin 3` `const float pi = 3.14;`

#define kullanırken atama operatörü (**=**) ve noktalı virgül kullanılmadığına dikkat ediniz.

const kullanırken sabitin *veri tipini* belirtmeyi unutmayınız.

Sabitler programın okunabilirliğini artırır ve daha ölçeklenebilir kılar.

const tipi sabitler bellekte tutulurken, #define ile tanımlanan sabitler ön işlemci komutu (#) ile tanımlandığından programda sabitin ismi görünen yere sabitin değerini yazarak RAM'de alan harcamaz ve daha kullanışlıdır.

Bölüm 4: Seri Haberleşme Ekranının Kullanımı

4.1 Giriş

Kullanıcı ile Arduino arasında veri alışverişi yapmak için kullanılır. Seri haberleşme ekranına, Arduino kartından bilgi aktarılacaksa program içinde seri haberleşme, **Serial.begin**() komutu ile başlatılmalıdır. Bu komutun kullanımını Bölüm 2.5'te anlatmıştık.

Arduino UNO içerisinde bir adet seri haberleşme portu bulunmaktadır. Bu portlardan Receiver (RX) 0, Transmitter (TX) 1 numaralı pindir. Eğer seri haberleşme kullanılacaksa bu pinler ile ilgili bir dijital giriş çıkış yapılamaz. Ayrıca bu pinler kullanılmayacaksa bile Arduino'ya yükleme yapılırken bu pine bağlı bir komponent varsa devreden çıkarılmalı ve TX(0), RX(0) bacakları boşta bırakılmalıdır.

Seri haberleşme ekranına (Serial Monitor) Şekil 2.1'de verilen Seri Port Ekranı ikonundan ulaşabilirsiniz.

Serial Port Ekranında mesajlar veya değişken içeriklerini aşağıdaki komutlar yardımıyla gösterebilirsiniz.

Fonksiyon	Açıklama

Serial.print(val)	**val** ile belirtilen parametreyi ekrana yazdırır. Burada bir parametredir. int, float, double String gibi değişkenler olabilir. Veya sadece karakter ya da metin olabilir.
	
	`Serial.print("Merhaba"); //Merhaba yazar`
	`Serial.print(x); //x'in değerini yazar`

Serial.print(val, format)	Serial.print ile aynı görevdedir. Ancak format ile belirtilen tipte çıktı üretir. **format** için:
	BIN: val'ı binary olarak yazar.
	OCT: val'ı octal olarak yazar.
	DEC: val'ı decimal olarak yazar.
	HEX: val'ı hexadecimal olarak yazar.
	0: **val**'ın virgülden sonra 0 rakamını gösterir.
	1: **val**'ın sıfırdan sonra 1 rakamını gösterir.
	N: **val**'ın sıfırdan sonra N rakamını gösterir.
	
	`Serial.print(3, BIN); //11 yazar`
	`Serial.print(78, HEX); //4E yazar`
	`Serial.print(3.1415, 1); //3.1 yazar`
	`Serial.print(2.7175, 0); //2 yazar`

Serial.println(val)	**Serial**.print ile aynı görevdedir. Tek farkı her **val** değerini yazdıktan sonra bir satır aşağı geçer. Aynı şekilde formatlı bir şekilde de kullanılabilir.
Serial.end(void)	Seri haberleşmeyi bitirir.

Seri haberleşmede iletişim hızı "baund" adı verilen bir değerle ifade edilir. Bu değer saniyede gönderilen bit sayısını ifade eder (bits per second).

Başlangıç için 9600 BaundRate hızı idealdir ve genelde 9600 seçilir.

Örnek 4.1

```
1  int x = 5;
2  int y = 8;
3  int toplam;
4  void setup() {
5        Serial.begin(9600);     //Seri haberleşme başlatılıyor
6        Serial.print("x'in degeri = ");
7        Serial.println(x);
8        Serial.print("y'nin degeri = ");
9        Serial.println(y);
10       Serial.print("Toplam = ");
11       toplam = x + y;
12       Serial.println(toplam);
13       Serial.print("Toplam Binary = ");
14       Serial.print(toplam, BIN);
15 }
16 void loop() {
17       }
18
```

```
x'in degeri = 5
y'nin degeri = 8
Toplam = 13
Toplam Binary = 1101
```

Burada bütün program setup() fonksiyonunun içinde verilmiş, loop() fonksiyonu boş bırakılmıştır. Çünkü loop() içine yazılan ifadeler sonsuz kere tekrarlanacağından Seri Port Ekranında çok fazla veri yazılmış olacaktı.

Seri Ekrandan veri okuma ile ilgili fonksiyonlar aşağıda verilmiştir.

Fonksiyon	Açıklama
Serial.available(void)	Seri ekranına gönderilen bayt sayısını geri çevirir. Parametresi yoktur. Genelde Seri Port Ekranına veri gönderilme durumunu kontrol etmek için **if** ile kullanılır. `if(Serial.available() > 0)`
Serial.read(void)	Seri Port'a gelen veriye ait ilk baytı **int** veri tipinde (ASCII'ye göre) çevirir. Eğer veri yok ise -1 geri gönderir. Gelen veri ASCII kodu olduğundan **char**'a çevirmek daha anlamalı kılacaktır. `comByte = Serial.read(); /*Gelen veriyi comByte'a atar*/`

Serial.write(val)
Serial.write(str)
Serial.write(buf, len)
	Binary veriyi Seri Porta yazar. Parametresine göre;
	val: Tek bir bayt yazmak için kullanılır.
	str: Bir diziyi seri baytlar halinde göndermek için kullanılır.
	buf: len uzunluğunda bir diziyi Seri Porta yazmak için kullanılır.
	Tüm türlerde, geriye yazılan verinin baytını geri çevirir.
	
	`sentByte = Serial.write(1658);`
	`sentByte = Serial.write("Giresun");`
	`sentByte = Serial.write(array, 28);`

Serial.find(str)	Seri Porta gelen veriler içerisinde **str** ile belirtilen Stringi arar. Var ise **true**, yoksa **false** çevirir. Sadece **String** araması yaptığını unutmayın. `if(Serial.find("28")`

*Seri haberleşme için kullanılan diğer komutlar gerektiğinde gösterilecektir.

Örnek 4.2

```
1   int alinanByte = 0;
2   void setup() {
3     Serial.begin(9600);     //Seri haberleşme başlatılıyor
4     }
5
6   void loop() {
7     if( Serial.available() > 0 ) {  //Veri girişi kontrolü
8       alinanByte = Serial.read();  //Veriyi alinanByte'a yaz
9       Serial.print("Alinan byte = ");
10      Serial.println(alinanByte);
11      Serial.print("Char olarak = ");
```

```
      Serial.println(char(alinanByte));
      Serial.println("-----------------");
    }
 }
```

```
>> a
Alinan byte = 97
Char olarak = a
-----------------
>> B
Alinan byte = 66
Char olarak = B
-----------------
>> 9
Alinan byte = 57
Char olarak = 9
-----------------
```

* >> Seri Port'a klavyeden gönderilen verileri belirtmektedir.

Örnek 4.3

```
void setup() {
  Serial.begin(9600);     //Seri haberleşme başlatılıyor
  }

void loop() {
  if( Serial.available() > 0 ) {  //Veri girişi kontrolü
    if(Serial.find("a")) {   //a karakteri aranıyor
      Serial.println("Var");
    }
    else Serial.println("Yok");
  }
}
```

```
>> a
Var
>> b
Yok
>> ba
Var
>> ab
Var
Yok
>> A
Yok
>> aa
Var
Var
```

* >> Seri Port'a klavyeden gönderilen verileri belirtmektedir.

Büyük küçük harf duyarlı olmasına ve gelen verileri bitler halinde kontrol ettiğine dikkat edin.

Ayrıca sayı araması yapsaydık bile String halinde yazmamız gerekir. Yani Seri Porta gelen veriler içinde 13 sayısını aramak için if fonksiyonunu Serial.find("13"); şeklinde yazmamız gerekirdi.

Örnek 4.4

```
1  void setup() {
2    Serial.begin(9600);     //Seri haberleşme başlatılıyor
3    }
4
5  void loop() {
6    if( Serial.available() > 0 ) {  //Veri girişi kontrolü
7      if(Serial.find("13")) {   //a karakteri aranıyor
8        Serial.println("Var");
9      }
10     else Serial.println("Yok");
11   }
12 }
```

```
>> 13
Var
>> 45
Yok
>> 413
Var
>> 4135
Var
Yok
>> 41315
Var
Yok
```

* >> Seri Port'a klavyeden gönderilen verileri belirtmektedir.

Bölüm 5: Aritmetik ve Lojik Operatörler

5.1 Giriş

İşlem yapmamızı sağlayan işaretlere operatörlerden. Bu operatörleri Aritmetiksel ve Mantıksal olarak inceleyebiliriz.

5.2 Aritmetiksel Operatörler

İşlem	Operatör	Kod Gösterimi	Açıklama
Toplama	+	a+b	a ile b'yi topla
Çıkarma	-	a-b	a'dan b'yi çıkar
Çarpma	*	a*b	a ile b'yi çarp
Bölme	/	a/b	a'yı b'ye böl
Mod	%	a%b	a'nın b'ye bölümünden kalan
Atama	=	a = b	a'yı b'ye ata

Ayrıca bazı operatörlerin birlikte kullanılmasıyla bazı özel durumlar olmaktadır.

İşlem	Operatör	Kod Gösterimi	Açıklama
Bir arttır	++	a++	a = a + 1
Bir azalt	--	a--	a = a - 1
N arttır	+=	a+= N	a = a + N
N azalt	-=	a-= N	a = a - N
N ile çarp	*=	a*= N	a = a * N
N'e böl	/=	a/= N	a = a / N
N'e göre mod	%=	a%= N	a = a % N

5.3 Mantıksal(Lojik) Operatörler

İşlem	Operatör	Kod Gösterimi	Açıklama
Eşit mi?	==	a == b	a b'ye eşit mi?
Eşit değil mi?	!=	a != b	a b'ye eşit değil mi?
Büyük mü?	>	a > b	a b'den büyük mü?
Küçük mü?	<	a < b	a b'den küçük mü?
Büyük Eşit mi?	>=	a >= b	a b'den büyük eşit mi?
Küçük Eşit mi?	<=	a <= b	a b'den küçük eşit mi?
Ve(And)	&&	a>b && a>c	a b'den ve c'den büyük mü?
Veya(Or)	||	a>b || a<c	a b'den büyük veya c'den küçük mü?
Değil(Not)	!	!(a>b)	a b'den büyük değil mi?

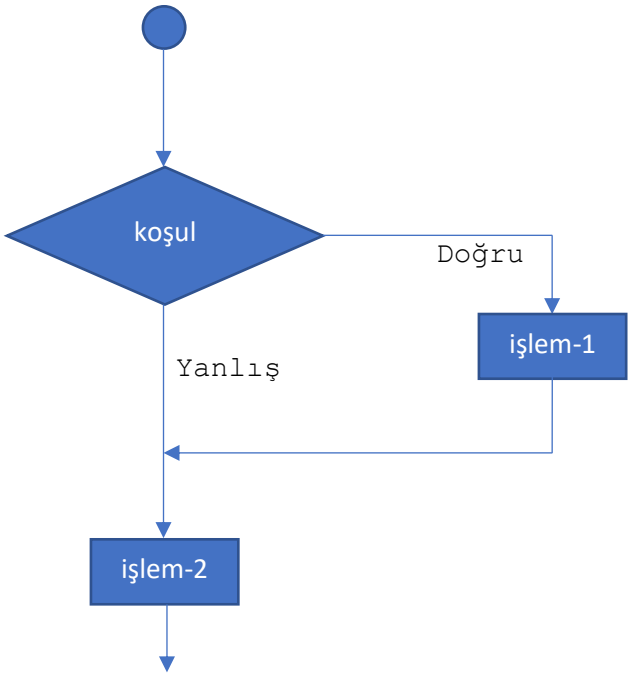
Bölüm 6: Karar ve Döngü Yapıları

6.1 Karar Yapıları

Şarta/koşula bağlı olarak iki veya daha fazla seçenekten birine dallanma işlemi gerçekleştiren komutlardır. Programın şartlı olarak hangi işlemi yapacağına karar vermemizi sağlar.

6.1a if Deyimi

Koşula bağlı olarak tek bir işlemi yerine getiren yapılardır. Eğer butona basılı ise LED'i yak, eğer uzaklık 30'dan az ise uyarı ver… gibi karar ifadelerinde kullanılır. Eğer koşul yanlış ise herhangi bir işlem yapılmaz.



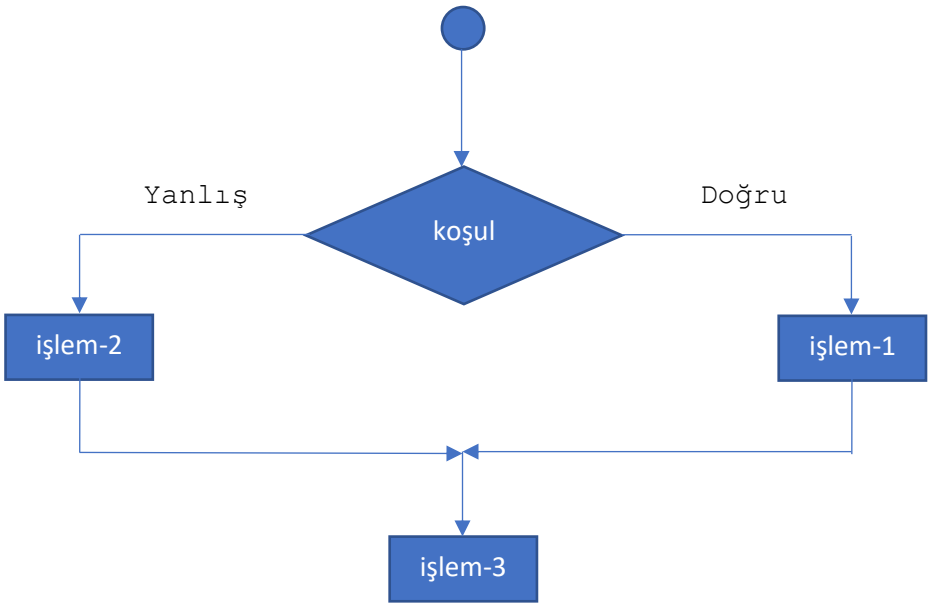
Şekil 6.1

Şekil 6.1'de tek seçimli if yapısının akış diyagramı verilmiştir. Aşağıda ise buna ait Arduino kodu yer almaktadır.

```
1  if(koşul) {
2       işlem -1
3
4  }
5  işlem -2
6  ...
```

6.1b if-else Deyimi

Koşula bağlı olarak iki işlem yerine getiren yapılardır. Yani koşul doğruysa şunu değilse bunu yap biçiminde çalışır. Eğer not ortalaması 60'dan yüksek ise geçtiniz, değilse kaldınız biçiminde ifadeleri oluşturmamızı sağlar.



Şekil 6.2

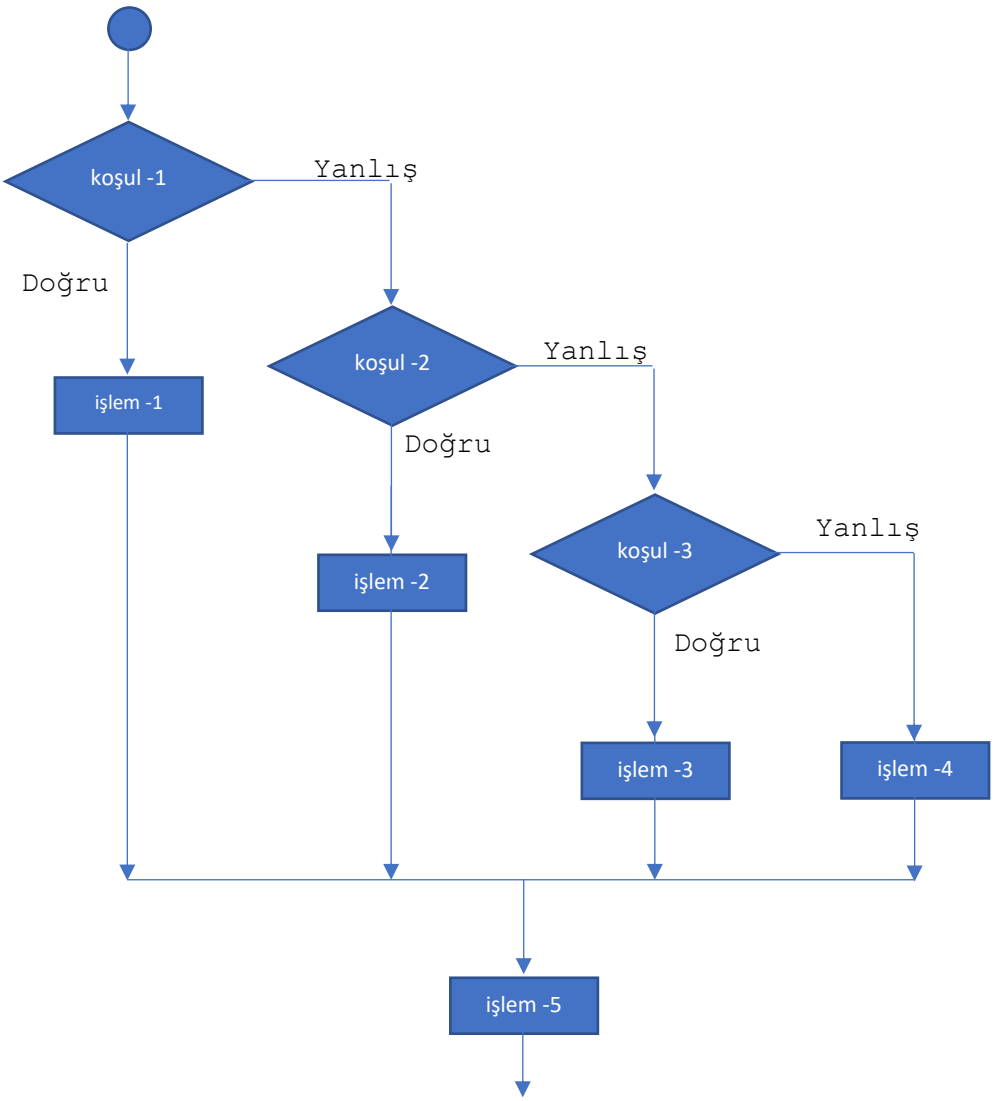
Şekil 6.2'de if-else deyiminin akış diyagramı verilmiştir. Aşağıda ise buna ait Arduino kodu yer almaktadır.

```
if(koşul) {
     işlem -1
}
else {
     işlem -2
}
işlem -3
...

```

6.1c İç İçe if-else Yapısı

Çok sayıda koşulu kontrol ederek iki veya daha fazla if-else if yapısından oluşan ifadelerdir. Örneğin 3 butonlu bir devrede 1.butonu kontrol et basılmadıysa ikinci butonu kontrol et, ona da basılmadıysa üçüncü butonu kontrol et biçiminde bir yapı kurmak istiyorsak if-else if deyimini kullanmamız gerekir.



Şekil 6.3

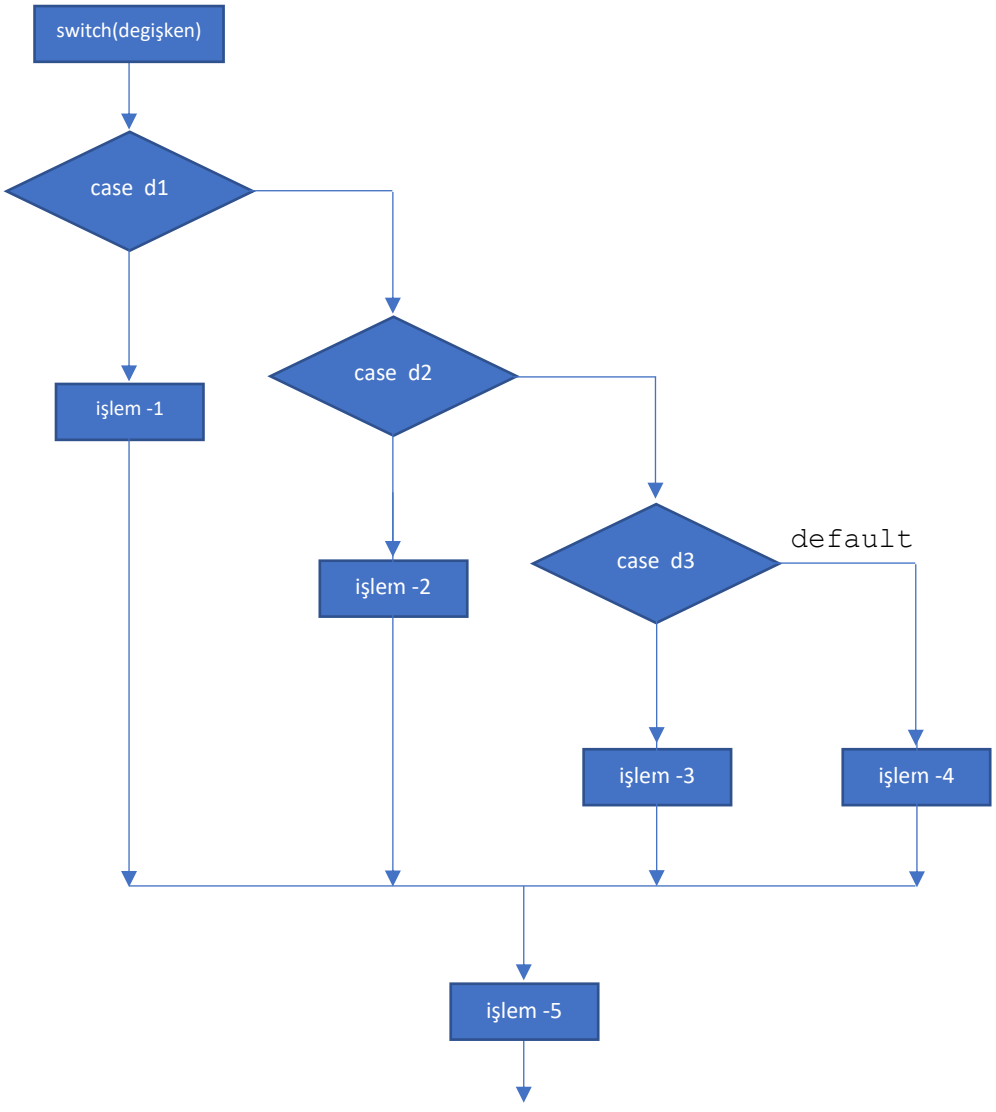
Şekil 6.3'te iç içe üç if-else if yapısının akış diyagramı verilmiştir. Aşağıda bu diyagrama ait Arduino kodları gösterilmiştir.

```
if(koşul -1) {
     işlem -1
}
else if(koşul -2) {
     işlem -2
}
else if(koşul -3) {
     işlem -3
}
else {
     işlem -4
}
işlem -5
...
```

Burada sadece tüm koşullar yanlış olduğunda işlem -4'ün gerçekleştirildiğine dikkat edin. Ayrıca işlem -5'in gerçekleştirilmesi koşullardan bağımsızdır.

6.1d switch-case Deyimi

Parametre olarak aldığı değişkenin durumuna göre farklı işlemlerin seçiminde kullanılır. İç içe if-else yapısı gibidir.



Şekil 6.4

Şekil 6.4'te switch-case yapısı gösterilmiştir. Aşağıda ise bu yapıya ait Arduino kodları verilmiştir.

```
switch(değişken) {
     case d1: {
          işlem -1
          break;
     }
     case d2: {
          işlem -2
          break;
     }
     case d3: {
          işlem -3
          break;
     }
     default: {
          işlem -4
     }
}
işlem -5
...
```

switch-case deyiminde switch'ten sonra kullanılan *değişken* değeri hangi **case** deyimindeki iki noktadan önce yazılan değere eşitse o bloktaki işlem yaptırılır. Eğer hiçbir **case** değerine eşit olmaz ise **default** ile belirtilen işlem yapılır.

Örneğin yukarıda switch ile verilen değişkenin değeri d1'e eşitse işlem -1, d2'ye eşitse işlem -2, d3'e eşitse işlem -3 bloğu çalıştırılır. Hiçbir değere eşit değilse işlem -4 gerçekleştirilir ve switch yapısından çıkılır.

switch yapısında kullanılan değişken sıralı(*tamsayı*(int, short, long) veya *karakter*) veri tipinde olmalıdır. Kesirli sayı değişkeni (float, double) veya String veri tipinde kullanılamaz.

Her **case** ve **default** bloğunun sonunda **break;** deyiminin olması gerekir. Aksi halde program hatalı çalışacaktır.

6.2 Döngü Yapıları

Yazdığımız kodda bazı komutların tekrar etmesini istiyorsak döngü yapılarını kullanmamız gerekir.

6.2a for Döngüsü

Temel olarak üç bölümde yazılır.

```
for(Başlangıç; Koşul; Artırım) {
     tekrarlı işlem
}
```

Başlangıç: Bu bölümde herhangi bir değişken veya sayaç değişkeni için başlangıç değeri atayabiliriz. Eğer atamaya ihtiyaç yok ise boş bırakılabilir. Başlangıç bölümü for döngüsü başında bir kere çalıştırılır. Döngü tekrar ettiği müddetçe sürekli başlangıç değeri ataması yapmaz.

Koşul: for döngüsünü sonlandıracak koşulu belirtir. Program for döngüsüne girmeden koşula bakar eğer koşul sağlanıyorsa döngüye girilir ve her bir döngü tamamlandığında koşulu tekrar kontrol eder. Eğer koşul hâlâ doğru ise döngüye devam eder.

Artırım: Burada herhangi bir değişken üzerinde işlem yapabilirsiniz. Genelde sayaç değişkenleri bir artırılır (sayac++) veya azaltılır (sayac--). Bu bölüm program tarafından ilk başta çalıştırılmaz. İlk döngü bittikten sonra koşul ifadesine bakmadan önce çalıştırılır ve koşul ifadesine öyle bakılır. Koşul sağlanıyorsa döngü tekrarlanacak ve tekrar koşula bakılmadan artırım bölgesindeki işlem yapılacaktır.

```
1  for(i = 1; i <= 10; i++) {
2        Serial.println("Merhaba");
3  }
4
```

Yukarıdaki örnek Seri Port'a 10 kere Merhaba yazdırır. Burada i değişkeninin sayaç görevi gördüğünü söyleyebiliriz.

Başlangıç bloğunda i'ye 1 atanmış. Bu atamadan sonra program ekrana bir kere Merhaba yazdırır ve Artırım bloğuna gider. Artırırım bloğunda i++ ifadesini görüyoruz. Bu ifade i'yi bir arttırır. Yeni i değerimiz 2'dir. Bu işlemden sonra program koşul bloğunu kontrol eder. i <= 10 ifadesi i=2 olduğunda doğrudur ve program ekrana tekrar Merhaba yazdırır. Bu şekilde i=11 olana kadar devam eder ve i=11 koşulu sağlamadığı için program bir daha döngüye girmez.

6.2b while Döngüsü

Sadece koşula bağlı olarak yazılır.

```
while(koşul) {
     tekrar eden işlemler
}
```

while döngüsünde program ilk başta koşula bakar eğer koşul doğru ise döngü içersine girer ve koşul yanlış olana kadar içerdeki işlemleri tekrarlar. Eğer içeride koşulu bozacak değişiklikler yapılmaz ise koşul sürekli doğru olacağında program sonsuz döngüye girmiş olur. Bazen gerekli olsa da çoğu zaman istenmeyen bir durumdur.

Az önce for döngüsü ile yaptığımız işlemi while ile tekrar yapalım;

```
1  i = 1;
2  while( i <= 10 ) {
3       Serial.println("Merhaba");
4       i++;
5  }
```

6.2c do while Dönügüsü

while döngüsü ile benzerdir. Tek fark do while dönüsünde önce döngü içindeki işlem yapılır. Sonra koşula bakılır. while'da ise döngüye girmeden önce de koşula bakılır. İki döngü içinde eğer koşul doğru ise işlemler tekrarlanır ve her seferinde tekrar koşula bakılır.

```
do {
     tekrar edecek işlemler
}while(koşul);
```

Yine iki döngüde verdiğimiz örneği do while ile gerçekleştirirsek;

```
1  i = 1;
2  do {
3       Serial.println("Merhaba");
4       i++;
5  }while(i <= 10);
```

while ile do while arasındaki farka dikkat edin. do while'ın bu özelliği koşul sağlanmadan bir kere döngüdeki işlemlerin gerçekleştirilmesini sağlar.

Bölüm 7: Fonksiyon

7.1 Giriş

Fonksiyon, işlemlerin ana program (setup ve loop) dışında küçük program parçacıkları halinde yazılmasıdır. Programların anlaşılabilirliğini, hata takibini ve program üzerinde ileri-geri kod takibini kolaylaştırmaya yarar.

7.2 Fonksiyon Tanımlama

Programı küçük işler yapan parçalara ayırmak için işlevsel fonksiyonlar tanımlamak gerekir. Fonksiyonlar iki bölgede tanımlanabilirler.

1. Tanımlama Bloğunda: Fonksiyon kütüphane ve değişken tanımlamalarının hemen altında, setup fonksiyonunun üstünde tanımlanabilir.
2. En Alt Kısımda: loop fonksiyonunda sonraki alanla fonksiyon tanımlaması yapılabilir. Bu alanda fonksiyon tanımlanmışsa üst tarafta (tanımlama bloğunda) fonksiyon prototipinin belirtilmesi gerekir. Prototip geri çevrilecek tip fonksiyon adı ve parametre tiplerinden oluşur. Bundan ileride bahsedilecektir.

Bir fonksiyon tanımlama üç kısımdan oluşur.

1. Geri çevrilecek veri tipi: Fonksiyon çalışıp işlemleri yaptıktan sonra bir veri çevirmesi gerektiğinde bunun veri tipini fonksiyon tanımlarken belirtmemiz gerekir. Bu işlem matematikteki fonksiyonlara benzetirsek; R->R bir fonksiyon tanımlandığında bu fonksiyonun görüntü kümesinin Reel sayılar olduğunu görebiliyoruz. Yani fonksiyonlarda da geri çevrilecek tip, o fonksiyonun görüntü kümesi gibidir diyebiliriz.
2. Fonksiyon adı: Fonksiyonu ana programın içinde çağırırken kullanacağımız ismi belirtir. Yine matematikteki gibi her fonksiyonunun ayrı bir adı vardır. Örneğin f, g, h sin, cos... fonksiyonu gibi.
3. Parametre: Fonksiyona gönderilecek (varsa)verilerin veri tiplerinin ve isimlerinin belirtildiği yer. Örneğin bir matematik fonksiyonunda f(x): N->R tanımlaması gibi. Burada x fonksiyona gönderilen parametre. Değer kümesini belirten doğal sayılar ise o fonksiyona gönderilecek parametrenin veri tipidir.

Bir fonksiyon tanımlamak için yukarıdaki bilgiler ışığında;

```
geriÇevrilecekVeriTipi fonksiyonAdı(ParametreTip parametreAd) {
     işlemler
     return geriGönderilecekİfade;
}
```

return Deyimi

Fonksiyona geri çevirme işlemini yaptıran komuttur. Burada return'den sonra yazılan değişken veya sabit ana programda fonksiyonun olduğu yere geri gönderilir ve fonksiyon durur.

Örneğin iki int veri tipinde tamsayı alan ve bunları toplayıp geri gönderen bir toplam isminde fonksiyon oluşturalım;

```
int toplam(int sayi1, int sayi2) {
     return (sayi1 + sayi2);
}
```

Yukarıda görüldüğü gibi iki parametre alıyor ise bu parametreleri virgül ile ayırmamız gerekiyor. Bu fonksiyonu tanımladıktan sonra program içerisinde 5 ve 6'yı toplamak için şu şekilde kullanabiliriz:

```
sonuc = toplam(5,6);
```

Bu işlem sonucunda sonuc değişkeninin içine toplam fonksiyonun 5 ve 6 için ürettiği cevap atanacaktır.

Veya bir sayının karesini alan bir fonksiyon tanımlayalım;

```
int kare(int sayi1) {
     return (sayi1 * sayi1);
}
```

Aynı şekilde bu fonksiyonu program içinde çağırırken;

```
sonuc = kare(8);
```

sonuc değişkeninin içine 8*8=64 değeri atanacaktır.

void Deyimi

void deyimi hiçliği ifade etmek için kullanılır. Örneğin fonksiyon herhangi bir parametre almıyor ise parametre kısmına void yazılır;

```
int fonksiyon(void) {
     işlemler
}
```

Bu fonksiyon program içinde sonuc = fonksiyon(); biçiminde çağrılır.

Benzer şekilde program herhangi bir değer geri çevirmesini istemezsek;

```
void fonksiyonAdi(parametreler) {
     işlemler;
}
```

! Geri çevrilen tip void ise, yani fonksiyon geri bir değer çevirmiyorsa fonksiyon içinde return deyimi kullanılamaz.

Biçiminde tanımlamamız gerekiyor. Örneğin iki tam sayı alıp sonucu Seri Port'a yazan geri çevrimsiz bir fonksiyon tasarlarsak;

```
1  void topla(int sayi1, int sayi2) {
2       int toplam;
3       toplam = sayi1 + sayi2;
4       Serial.println(toplam);
5  }
```

Bu fonksiyonu 5 ve 6 için çağıracak olursak;

```
topla(5,6);
```

Komutunu kullanabiliriz.

Seri Port'a 10 kere merhaba yazdıran parametresi ve dönüş tipi void olan bir fonksiyon yazarsak;

```
1  void seriPortYaz(void) {
2       int i;
3       for(i = 1; i <=10; i++) {
4            Serial.println("Merhaba");
5       }
6  }
```

*Fonksiyonlar içinde tanımlanan değişkenler fonksiyon tamamlanınca belleğe geri verilir. Ayrıca parametre kısmında yazılan değişkenler fonksiyon içinde tanımlanmış gibi kullanılabilir.

seriPortYaz fonksiyonu için prototip şu şekilde olmalıdır;

```
void seriPortYaz(void);
```

Yukarıda yazılan bütün fonksiyonları tek bir programda toplayacak olursak;

```
int toplam(int sayi1, int sayi2);
int kare(int sayi1);
void topla(int sayi1, int sayi2);
void seriPortYaz(void);
int sonuc;
void setup() {
  Serial.begin(9600);
  }

void loop() {
  sonuc = toplam(5,6);
  Serial.print("Toplam = ");
  Serial.println(sonuc);
  sonuc = kare(8);
  Serial.print("Kare = ");
  Serial.println(sonuc);
  topla(5,6);
  seriPortYaz();
}

int toplam(int sayi1, int sayi2) {
  return (sayi1 + sayi2);
}

int kare(int sayi1) {
  return sayi1*sayi2;
}

void topla(int sayi1, int sayi2) {
  int toplam;
  toplam = sayi1 + sayi2;
  Serial.println(toplam);
}

void seriPortYaz(void) {
  int i;
  for(i = 1; i <= 10; i++) {
    Serial.println("Merhaba");
  }
}
```

```
Toplam = 11
Sonuc = 64
11
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
```

Kaynakça;

1. Bülent Çobanoğlu, "Derinlemesine Arduino", Abaküs Yayıncılık, 1.Baskı: Eylül 2017
2. Paul Deitel-Harvey Deitel, "C ile Programlama", Palme Yayıncılık 7.Baskı'dan Çeviri: 2016
3. https://www.arduino.cc/reference/en/
4. https://www.mobilhanem.com/arduino-dersleri-serial-port-ve-fonksiyonlari/