**STRUKTURA**

**setup()** - ova funkcija se poziva kada sketch počinje. Koristi se da se inicijaliziraju varijable, stanja pinova, da se počnu koristiti „libaries“ itd. SETUP funkcija se pokreće samo jednom, poslje svakog pokretanja ili resetiranja Arduino pločice.

**Primjer**

int buttonPin = 3;

void setup()

{

Serial.begin(9600);

pinMode(buttonPin, INPUT);

}

void loop()

{

// ...

}

**loop()** – poslije stvaranja setup() funkcije, loop() funkcija radi upravo ono što joj ime govori, vrti se uzastopno, dozvoljavajući programu da se mijenja i odgovara. Koristi se za aktivnu kontrolu Arduino pločice.

**Primjer**

const int buttonPin = 3;

// setup initializes serial and the button pin

void setup()

{

Serial.begin(9600);

pinMode(buttonPin, INPUT);

}

// petlja provjerava pin prekidača svaki put, i ispisuje ako je pritisnut ili ne

void loop()

{

if (digitalRead(buttonPin) == HIGH)

Serial.write('H');

else

Serial.write('L');

delay(1000);

}

**KONTROLNE STRUKTURE**

**If -** koristi se u vezi sa operatorom usporedbe, ispituje jeli neko određeno stanje dostignuto.

**Primjer 1**

if (someVariable > 50)

{

// ako je neka varijabla veća od 50 izvršava neku radnju

}

Ako je tvrdnja u zagradama istinita, tvrdnje unutar zagrada se izvršavaju. Ako nije kod se preskače.

Zagrade se mogu izostaviti nakon if tvrdnje. Ako je tako, sljedeći red (definiran sa (;)) postaje jedina uvjetovana tvrdnja.

**Primjer 2**

if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)

digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){

digitalWrite(LEDpin1, HIGH);

digitalWrite(LEDpin2, HIGH);

} //sve tvrdnje su točne

Tvrdnje koje se evaluiraju unutar zagrada zahtjevaju korištenje jednog ili više operatora:

x == y (x je jednako y)

x != y (x nije jednako y)

x < y (x je manji od y)

x > y (x je veći od y)

x <= y (x je manji ili jednak y)

x >= y (x je veći ili jednak y)

**PAŽNJA!!!**

Paziti na korištenje jednog znaka jednakosti (npr. If(x=10)). Jedan znak jednakosti je operator dodjele, tj. postavlja x na vrijednost 10.

**If/else** - dozvoljava veću kontrolu nad tokom koda nego osnovna **if** varijabla i to dozvoljavajući višestruke testove grupirane zajedno.

**Primjer 1**

if (pinFiveInput < 500)

{

// action A

}

else

{

// action B

}

**else** može izvršiti drugi **if**test, tako da se višestruki, međusobno nepovezani testovi mogu izvršavati istovremeno.

Svaki test će se nastaviti dok istiniti test nije dostignut. Kada je istinit test dostignut, kod s kojim je taj test povezan se izvršava, program potom preskače do reda prateći cijelu **if/else** konstrukciju. Ako nijedan test nije istinit zadani **else** blok se izvršava, ako je prisutan, i postavlja se na zadano ponašanje.

**else if** blok se može koristiti sa ili bez završnog **else** bloka i obrnuto. Neograničen broj takvih **else if**  grana je dopušten

**Primjer 2**

if (pinFiveInput < 500)

{

// izvrši radnju A

}

else if (pinFiveInput >= 1000)

{

// izvrši radnju B

}

else

{

// izvrši radnju C

}

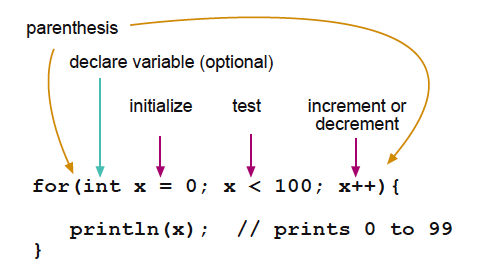
**for –** koristi se za ponavljanje bloka izjava zatvorenog u vitičastim zagradama. Inkrementni brojač se često koristi za povećanje i prekidanje petlje. **for** se koristi za sve ponavljajuće operacije i često se koristi u kombinaciji sa redovima (arrays) za operacije sa kolekcijama podataka/pinova.

Tri su dijela za **for** petlju**:**

for (initialization; condition; increment) {

//statement(s);

}



„Initialization“ se događa prva i samo jednom. Svaki put kroz petlju, „condition“ je provjeravana; ako je istinit, blok i „increment“ je izvršen, nakon toga „condition“ se provjerava ponovno. Kada „condition“ postane lažan petlja završava.

**Primjer**

// Dim an LED using a PWM pin

int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()

{

// no setup needed

}

void loop()

{

for (int i=0; i <= 255; i++){

analogWrite(PWMpin, i);

delay(10);

}

}

**Savjeti za kodiranje**

**for** petlja iz programskog jezika C je puno fleksibilnija od petlji u nekim drugim programskim jezicima, uključujući BASIC. Neki ili svi od tri elementa zaglavlja mogu biti izostavljeni, iako se zahtjeva korištenje točke-zareza. Isto tako izjave za „initalization“, „condition“ i „increment“ mogu biti bilo koje važeće C izjave sa nevezanim varijablama, i koriste bilo koju C vrstu podataka uključujući decimalni tip podataka (floats). Ovi tipovi neobičnih **for** izjava mogu dati riješenja nekim rijetkim programskim problemima.

**Primjer 2**

Korištenje množenja u redu za povećanje će generirati logaritamski porast:

for(int x = 2; x < 100; x = x \* 1.5){

println(x);

}

Generira se: 2,3,4,6,9,13,19,28,42,63,94

**Primjer 3**

Pojačava i smanjuje svjetlost LED-ice sa jednom **for** petljom:

void loop()

{

int x = 1;

for (int i = 0; i > -1; i = i + x){

analogWrite(PWMpin, i);

if (i == 255) x = -1; // zamjeni smjer na najvećoj vrijednosti

delay(10);

}

}

**switch case** – kontrolira tok programa dozvoljavajući programeru da odredi različite kodove koje bi trebalo izvršiti u promjenjivim uvjetima. Posebno, „switch“ uspoređuje vrijednosti varijabli sa vrijednostima određenim u izjavama slučaja. Kada je izjava slučaja pronađena čija se vrijednost podudara sa varijablom, kod u toj izjavi se pokreće.

**break** riječ izlazi iz „switch“ izjave, i obično se koristi na kraju svakog slučaja (case). Bez „break“ izjave „switch“ izjava će se nastaviti izvršavati sljedeće izraze („falling through“) sve do „break“ ili kada je kraj „switch“ izjave dostignut.

**Primjer**

switch (var) {

case 1: //slučaj 1

//do something when var equals 1

break;

case 2:

//do something when var equals 2

break;

default:

// if nothing else matches, do the default

// default is optional

break;

}

Voditi računa da su potrebne zagrade kako bi se objavile varijable unutar slučaja. Kao npr.

switch (var) {

case 1:

{

//do something when var equals 1

int a = 0;

.......

.......

}

break;

default:

// if nothing else matches, do the default

// default is optional

break;

}

**SINTAKSA**

switch (var) {

case label:

// statements

break;

case label:

// statements

break;

default:

// statements

break;

}

**PARAMETRI**

**var-** varijabla čija se vrijednost uspoređuje sa različitim slučajem

**label-** vrijednost s kojom se uspoređuje varijabla

**while()-** ova petlja se ponavalja stalno i beskonačno, sve dok izraz unutar zagradi () postane lažan. Nešto mora promijeniti testiranu varijablu ili **while**  petlja neće nikada završiti.

**SINTAKSA**

while(expression){

// statement(s)

}

**PARAMETRI**

expression- C izjava koja se procjnjuje na istinu ili laž

**Primjer**

var = 0;

while(var < 200){

// do something repetitive 200 times

var++;

}

**do-while -**  **do** petlja radi isto što i **while** petlja, razlika je u tome što se stanje provjerava na kraju petlje, tako da će se **do** petlja uvijek izvršiti barem jednom.

do

{

// statement block

} while (test condition);

**Primjer**

do

{

delay(50);// čekaj da se senzori stabiliziraju

x = readSensors(); // provjeri senzore

} while (x < 100);

**break;-** koristi se za izlazak iz **do,for ili while** petlji, zaobilazeći normalno stanje petlje. Koristi se i za izlazak iz **switch** izjave.

**Primjer**

for (x = 0; x < 255; x ++)

{

analogWrite(PWMpin, x);

sens = analogRead(sensorPin);

if (sens > threshold){ // bail out on sensor detect

x = 0;

break;

}

delay(50);

}

**continue;-** preskače ostatak iteracije trenutne petlje (**do, for, while**). Nastavlja provjeravajući uvjetni izraz petlje, i nastavlja sa naknadnom iteracijom.

**Primjer**

for (x = 0; x < 255; x ++)

{

if (x > 40 && x < 120){ // create jump in values

continue;

}

analogWrite(PWMpin, x);

delay(50);

}

**return-** završava funkciju i vraća vrijednost funkcije u pozivajuću funkciju.

**SINTAKSA**

return;

return value; // obe forme su pravilne

**PARAMETRI**

**value:** bilo koja varijabla ili konstanta

**Primjer**

int checkSensor(){

if (analogRead(0) > 400) {

return 1;

else{

return 0;

}

}

**goto-** prebacuje tok programa na određeno mjesto u programu

**SINTAKSA**

label:

goto label; // sends program flow to the label

**Primjer**

for(byte r = 0; r < 255; r++){

for(byte g = 255; g > -1; g--){

for(byte b = 0; b < 255; b++){

if (analogRead(0) > 250){ goto bailout;}

// more statements ...

}

}

}

bailout:

**VARIJABLE**

**KONSTANTE-** predefinirani izrazi u Arduino jeziku. Koriste se kako bi programi bili lakši za čitanje.

**HIGH/LOW-** pinovi mogu čitati/biti postavljeni na samo dvije vrijednosti **HIGH** ili **LOW**

**HIGH-** značenje (za pinove) je drugačije ovisno o tome dali je pin postavljen na INPUT ili OUTPUT. Kada je postavljen na INPUT sa pinMode() i čita sa digitalRead(), Arduino će izvijestiti **HIGH** ako:

-je voltaža veća od 3V dana na pinu (5V pločice)

-je voltaža veća od 2V dana na pinu (3.3V pločice)

Pin može biti postavljen kao INPUT sa pinMode() i zatim na **HIGH** sa digitalWrite().

To će „upaliti“(enable) unutarnji 20K „pullup“ otpornik, koji će „povući“ (pullup) ulazni pin na **HIGH** osim ako nije „povučen“ na **LOW** vanjskim strujnim krugovima.

Kada je pin postavljen na OUTPUT sa pinMode() i na **HIGH** sa digitalWrite(), pin je na:

-5 volti (5V pločice)

-3.3 volti (3.3V pločice)

**LOW-** značenje (za pinove) je drugačije ovisno o tome dali je pin postavljen na INPUT ili OUTPUT. Kada je postavljen na INPUT sa pinMode() i čita sa digitalRead(), Arduino će izvijestiti **LOW** ako:

-je voltaža manja od 3V dana na pinu (5V pločice)

-je voltaža manja od 2V dana na pinu (3.3V pločice)

Kada je pin postavljen na OUTPUT sa pinMode() i na **LOW** sa digitalWrite(), pin daj 0 volti.

**INPUT, INPUT\_PULLUP, OUTPUT**

digitalni pinovi pinovi se mogu koristiti kao **INPUT, INPUT\_PULLUP, OUTPUT.** Mijenjanje pina sa pinMode() mijenja električno ponašanje pina.

PINOVI POSTAVLJENI NA **INPUT**

Arduino (Atmega) pinovima postavljenima kao **INPUT** sa pinMode() je rečeno da budu u high-impendance stanju. Pinovi postavljeni kao **INPUT** čine izuzetno male zahtjeve na strujnom krugu kojeg su dio. Jednako kao serijski postavljen otpornik od 100 Mohm ispred pina. Ovo ih čini korisnim za čitanje podataka sa senzora.

Ako imate pin postavljen kao **INPUT** i „čitate“ prekidač, kada je prekidač u otvorenom stanju ulazni pin će „plutati“(floating), što rezultira nepredvidljivim rezultatima. Kako bi osigurali odgovarajuće očitanje kada je prekidač otvoren, mora se koristiti pull-up ili pull-down otpornik. Svrha ovog otpornika je da „povuče“ pin na poznato stanje kada je prekidač otvoren. Obično se odabire 10K ohm jer je to dovoljno nisko da pouzdano spriječi plutajući ulaz, i u isto vrijeme dovoljno visoko da ne povuče previše struje kada je prekidač zatvoren.

Ako se koristi pull-down otpornik ulazni pin će biti LOW kada je prekidač otvoren i obratno.

Ako se koristi pull-up otpornik, ulazni pin će vidi HIGH kada je prekidač otvoren i obratno.

PINOVI POSTAVLJENI NA **INPUT\_PULLUP**

Koristi se u pinMode() kada ne želimo koristiti vanjski pull-up otpornik (Atmega mikrokontroler ima ugrađen pull-up otpornik).

Pinovi postavljeni kao ulazi sa **INPUT** ili **INPUT\_PULLUP** mogu biti uništeni ako se povežu na negativne voltaže ili iznad dozvoljene voltaže (5V ili 3V).

PINOVI POSTAVLJENI NA **OUTPUT**

Pinovima postavljenim na **OUTPUT**  sa pinMode() je rečeno da budu u low-impendance stanju. To znači da mogu pružiti znatnu količinu struje drugim strujnim krugovima. Atmega pinovi mogu dati struju ili apsorbirati struju do 40 mA drugim uređajima/strujnim krugovima. To ih čini korisnima za paljenje LED-ica zbog toga što LED-ice obično koriste manje od 40 mA. Za struje veće od 40 mA zahtjevaju tranzistor ili neki drugi strujni krug.

Pinovi postavljeni kao output mogu biti uništeni ako su povezani na minus ili plus.

**VRSTE PODATAKA**

**void-** koristi se samo u pojašnjenju funkcije. Indicira da funkcija ne vraća nikakvu informaciju funkciji sa koje je pozvana.

**Primjer**

// radnje su izvršene u funkcijama "setup" i "loop"

// ali ni jedna informacija nije prijavljena većem programu

void setup()

{

// ...

}

void loop()

{

// ...

}

**boolean-** sadržava jednu od dvije vrijednosti, true ili false

**Primjer**

int LEDpin = 5; // LED on pin 5

int switchPin = 13; // momentary switch on 13, other side connected to ground

boolean running = false;

void setup()

{

pinMode(LEDpin, OUTPUT);

pinMode(switchPin, INPUT);

digitalWrite(switchPin, HIGH); // turn on pullup resistor

}

void loop()

{

if (digitalRead(switchPin) == LOW)

{ // switch is pressed - pullup keeps pin high normally

delay(100); // delay to debounce switch

running = !running; // toggle running variable

digitalWrite(LEDpin, running); // indicate via LED

}

}

**char-** vrsta podataka koja sprema vrijednost simbola. Character literals? su napisani u jednostrukim navodnicima 'A' a za više simbola se koriste dvostruki navodnici "ABC".

Simboli su spremljeni i kao brojevi. Ovo znači da je moguće vršiti aritmetiku na simbolima, u kojima se koristi ASCII vrijednost signala ( 'A'+1 ima vrijednost 66, pošto je ASCII vrijednost velikog slova A jednaka 65).

**char** vrsta podataka je „signed type“, što znači da šifrira brojeve od -128 do 127. Za „unsigned“, jednobitne (8 bit) vrste podataka koriste se „byte“ vrste podataka.

**Primjer**

char myChar = 'A';

char myChar = 65; // both are equivalent