ARDUINO RADIONICA
PROGRAMIRANJE

Ардуино радионица

Aritmetički operatori

Aritmetički operatori su sabiranje, oduzimanje, množenje i deljenje. Kao rezultat vraćaju zbir, razliku, proizvod ili količnik (respektivno) dva operanda.

```
y = y + 3;
x = x - 7;
i = j*6;
r = r/5;
```

Operacija je podržana **tipom podatka operanada**, tako na primer, izračunavanje izraza 9/4 daje kao rezultat vrednost 2 umesto 2.25 pošto su 9 i 4 celobrojne vrednosti. Ovo takođe znači da prilikom izvođenja aritmetičke operacije može doći do tzv.

greške prepunjavanja, tj. ukoliko je rezultat izračunavanja veći nego što može da se smesti u memorijsku lokaciju čija veličina je određena definisanim tipom podatka.

...

Ako su operandi različitog tipa, veći tip se koristi za izračunavanje. Na primer, ako je jedan od brojeva (operanada) tipa float a drugi tipa integer, koristiće se izračunavanje sa pokretnom zapetom.

Veličinu promenljive (tj. tip) treba tako izabrati da bude dovoljne veličine da može da čuva najveću vrednost iz predviđenih izračunavanja.

Napomena:

Preporučuje se korišćenje **operatora cast**, tj. (int)myFloat za konvertovanje „u hodu” jednog tipa podatka u drugi.

Na primer,

i = (int) 3.6

će postaviti da i bude jednako 3.

Ардуино радионица

Složena dodeljivanja

Ova dodeljivanja kombinuju aritmetičku operaciju sa dodeljivanjem vrednosti promenljivoj. To se često sreće u for petljama. Slede najčešće korišćena složena dodeljivanja:

```
x++          // isto što x = x + 1, ili inkrementiranje x za 1
x--          // isto što x = x - a, ili dekrementiranje x za 1
x+= y        // isto što x = x + y, ili inkrementiranje x za y
x-= y        // isto što x = x - y, ili dekrementiranje x za y
x*= y        // isto što x = x * y, ili množenje x sa y
x/= y        // isto što x = x / y, ili deljenje x sa y
```

Napomena:

Na primer, izraz

```
x* = 3
```

će utrostručiti vrednost od x i tu vrednos dodeliti opet promenljivoj x.

Operatori poređenja

Poređenje jedne promenljive ili konstante sa drugom promenljivom ili konstantom često se koristi u naredbama if za proveru da li je specificirani uslov ispunjen.

```
x == y          // x je jednako y
x != y          // x nije jednako y
x < y           // x je manje od y
x > y           // x je veće od y
x <= y          // x je manje ili jednako y
x >= y          // x je veće ili jednako y
```

Logički operatori

Ovi operatori se obično koriste za poređenje dva izraza a kao rezultat daju vrednost TRUE ili FALSE u zavisnosti od korišćenog operatora. Postoje tri logička operatora, AND, OR i NOT, koji se često koriste u **if** naredbama:

Logičko AND:

```
if (x > 0 && x < 5)          // tačno samo ako su oba izraza tačna
```

Logičko OR:

```
if (x > 0 || y > 0)          // tačno ako je bilo koji izraz tačan
```

Logičko NOT:

```
if (!x > 0)                  // tačno ako je izraz netačan
```

Konstante

Arduinov jezik ima nekoliko **unapred definisanih vrednosti** koje se nazivaju **konstante**. One se koriste da bi program učinile čitljivijim. Konstante su klasifikovane u grupe.

true/false

To su Bulove konstante koje definišu logčke nivoe. FALSE se lako definiše kao 0 (nula), dok se TRUE često definiše kao 1, ali može biti bilo šta drugo izuzev nule. Prema tome, u Bulovom smislu, -1, -2 i -200 se mogu definisati kao TRUE.

```
if (b == TRUE)
{
        radiNesto;
}
```

...

high/low

Ove konstante definišu nivoe pinova i često se koriste kada se vrši očitavanje ili upisivanje digitalnih pinova. HIGH se definiše kao logčki nivo 1 ili 5V, dok je LOW logički nivo 0, odnosno OFF ili 0V.

```
digitalWrite(13, HIGH);
```

input/output

Ove konstante koristi funkcija **pinMode()** za definisanje režima (načina) rada digitalnih pinova bilo kao INPUT ili kao OUTPUT.

```
pinMode(13,OUTPUT);
```

Struktura IF

Struktura (naredba) **if** testira da li je konkretan uslov zadovoljen, recimo da li je analogna vrednost veća od nekog konkretnog broja, i izvršava sve naredbe unutar vitičastih zagrada ako je to tačno. Ako uslov nije ispunjen, program preskače naredbe koje se nalaze unutar vitičastih zagrada. Oblik naredbe testiranja je:

```
if (nekaPromenljiva ?? vrednost)
{
        radiNesto;
}
```

Prethodni primer poredi promenljivu nekaPromenljiva sa nekom vrednošću koja može biti konstanta ili promenljiva. Ako je poređenje zadovoljeno (tj. uslov u zagradi ispunjen), naredba unutar vitičastih zagrada se izvršava. Ako uslov nije ispunjen, program skače na prvu naredu koja sledi posle vitičastih zagrada.

...

Napomena:

Treba voditi računa o korektnoj upotrebi znakova, naime izraz

```
if (x=10)
```

je tehnički gledano valjan, tj. definiše da promenljiva x dobija vrednost 10 i kao rezultat to je uvek tačno. Međutim, u naredbi ovog tipa korektno je umesto toga koristiti znak **==**, tj.

```
if (x == 10)
```

a time se samo testira da li je x jednako vrednosti 10 ili ne.

Posmatrajte znak '=' kao 'jednako'

a znak '==' kao 'jednako sa'.

Struktura if ... else

Struktura if ... else omogućava da se donese odluka tipa ili-ili. Na primer, ako se testira digitalni ulaz, jedna stvar se radi ako je ulaz u stanju HIGH a neka druga ako je ulaz u stanju LOW i to se može napisati na sledeći način:

```
if (inputPin == HIGH)
{
        radiStvarA;
}
else
{
        radiStvarB;
}
```

...

Takođe, else može da prethodni još jednom if testu, uzajamno ekskluzivni testovi mogu se izvršavati u isto vreme. Čak je omogućeno da postoji neograničen broj tih else grananja. Treba imati u vidu da samo jedan skup naredbi će moći da se izvrši u zavisnosti od ispunjenosti uslova testova:

```
if (inputPin < 500)
{
        radiStvarA;
}
else if (inputPin >= 1000)
{
        radiStvarB;
}
else
{
        radiStvarC;
}
```

...

Napomena:

Naredba if jednostavno testira da li je izraz (uslov) u zagradi tačan ili netačan.

Izraz može biti bilo koja važeća C naredba, kao što je slučaj u prethodnom primeru, tj.

```
if (inputPin == HIGH).
```

U ovom primeru, naredba if

samo proverava da li je zaista specificirani ulaz na logičkom nivou HIGH, tj. naponu 5V.

Struktura (naredba) for

Naredba for se koristi za ponavljajuće izvršavanje bloka naredbi obuhvaćenih vitičastim zagradama specificirani broj puta. Inkrementalni brojač se često koristi za inkrementiranje i izlazak iz petlje. Zaglavlje naredbe for ima tri dela (parametra) koji su međusobno odvojeni oznakom **;** (tačka-zapeta):

```
for (inicijalizacija; uslov; izraz)
{
        radiNesto;
}
```

Brojač petlje (lokalna promenljiva) prvo se inicijalizuje i to samo jednom. Pri svakom prolasku kroz petlju, testira se uslov. Ako je uslov tačan, sledeće naredbe i izraz se izvršavaju i uslov ponovo testira itd. **Kada uslov postane netačan, izlazi se iz petlje**.

...

Sledeći primer startuje brojač petlje **i** vrednošću 0,

testira da li je još **i** uvek manje od 20 i ako je to tačno, inkrementira/uvećava **i** za 1 i izvršava naredbe unutar vitičastih zagrada:

```
for (int i=0; i<20; i++)                // deklarisanje i, testiranje da li je
{                                       // manje od 20 i inkrementiranje za 1
        digitalWrite(13, HIGH);         // uključuje pin 12
        delay(250);                     // zadrška od 250ms
        digitalWrite(13, LOW);          // isključuje pin 13
        delay(250);                     // zadrška od 250ms
}
```

Naredba while

Ova petlja **kruži** kontinualno kroz naredbe unutar zagrada sve dok uslov unutar zagrada ne postane netačan. Nešto mora promeniti vrednost testirane promenljive ili će while petlja kružiti beskonačno. Ta promena se ostvaruje unutar programskog koda ili inkrementiranjem vrednosti promenljive ili spoljašnjim uslovom, kao što je testiranje vrednosti koju daje neki senzor.

```
while (nekaPromenljiva ?? vrednost)
{
        radiNesto;
}
```

...

U sledećem primeru testira se da li je Promenljiva1 manja od 200 i ako je to tačno izvršavaju se naredbe unutar vitičastih zagrada i nastavlja kruženje u petlji sve dok Promenljiva1 ne postane jednaka ili veća od vrednosti navedene u uslovu testiranja.

```
while (Promenljiva1 < 200)
{
        radiNesto;
        Promenljiva1++;
}
```

Struktura do ... while

Petlja do-while je petlja sa proverom uslova na kraju petlje, tako da se zbog takve strukture uvek izvršava bar jednom.

```
do
{
        radiNesto;
} while (nekaPromenljiva ?? vrednost);
```

U sledećem primeru funkcija očitavanja nekog senzora, readSensors(), dodeljuje očitanu vrednost promenljivoj x, zatim sledi zadrška od 50 ms, a na kraju se proverava vrednost promenljive x i kruženje u petlji se nastavlja ukoliko je ta vrednost manja od 100:

```
do
{
        x = readSensors();          // dodeljivanje vrednosti sa senzora
                                    // promenljivoj x
        delay(50);                  // zadrška od 50ms
} while (x<100);                    // kruži u petlji ako je x<100
```

Naredba pinMode(pin, mode)

Ova funkcija/naredba se koristi u okviru funkcije setup() za konfigurisanje **navedenog pina** kao ulaznog (INPUT) ili izlaznog (OUTPUT).

```
piMode(pin, OUTPUT);      // postavlja 'pin' kao izlaz
```

Arduinovi digitalni pinovi su **fabrički postavljeni kao ulazi** (ulazni), stoga nije ih potrebno eksplicitno deklarisati kao ulaze korišćenjem funkcije pinMode(). Kada su pinovi konfigurisani kao ulazi to znači da se nalaze u tzv. **stanju visoke impedanse**.

...

Takođe, Atmega mikroprocesorski čip ima **ugrađen** odgovarajući tzv. ***pull-up* otpornik od 20 kilooma** (kΩ)

kome se može softverski pristupiti. Tim otpornicima se pristupa na sledeći način:

```
pinMode(pin, INPUT);        // postavlja 'pin' kao ulaz
digitalWrite(pin, HIGH);    // uključuje pull-up otpornike
```

Pull-up otpornici se obično koriste kada se na ulaze Arduino pločice priključuju ulazni uređaji tipa **prekidača**.

U zadnjoj naredbi prethodnog primera treba obratiti pažnju na to da pin nije naredbom konvertovan u izlaz (OUTPUT), **već je naredba korišćena samo kao metoda za aktiviranje internih pull-up otpornika**.

...

Kada se pinovi konfigurišu kao izlazi, to znači da su u tzv. **stanju niske impedanse** i mogu da obezbede struju od 40 mA uređajima koji su na njih priključeni. To je dovoljna jačina struje da izazove jasno svetljenje svetlosne diode (na red sa svetlosnom diodom treba vezati zaštitni otpornik)

ali nije dovoljna da aktivira kalem (tzv. špulnu) releja ili da aktivira mali elektromotor.

Kratak spoj na pinovima Arduino pločice ili prevelika struja na njima može izazvati oštećenje izlaznih pinova ili oštećenje kompletnog Atmega čipa. Da bi se to izbeglo **preporučuje se**

da se na OUTPUT pin, na koji se priključuje spoljni uređaj, **redno veže otpornik od 470 oma ili 1 kiloom**.

digitalRead(pin)

Očitava vrednost na specificiranom pinu, koja može biti HIGH ili LOW. Pin se može specificirati kao promenljiva ili kao konstanta (0 - 13).

```
value = digitalRead(Pin);          // postavlja 'value' da je jednako sa
                                   // ulaznim pinom
```

Ардуино радионица

digitalWrite(pin, value)

Prosleđuje logički nivo, HIGH ili LOW (uključuje ili isključuje), na **specificirani digitalni pin**. Pin se može specificirati ili kao promenljiva ili kao konstanta (0 - 13).

```
digitalWrite(pin, HIGH);      // postavlja 'pin' na visoki logički nivo
```

Sledeći primer očitava stanje tastera priključenog na digitalni ulaz i uključuje svetlosnu diodu priključenu na digitalni izlaz kada je taster pritisnut:

```
int led = 13;                 // povezuje LED sa pinom 13
int pin = 7;                  // povezuje taster sa pinom 7
int value = 0;                // promenljiva za čuvanje očitane vrednosti
```

...

```
void setup()
{
        pinMode(led, OUTPUT);          // postavlja pin 13 kao izlaz
        pinMode(pin, INPUT);           // postavlja pin 7 kao ulaz
}

void loop()
{
        value = digitalRead(pin);      // postavlja 'value' da je jednako
                                       // ulaznom pinu
        digitalWrite(led, value);      // postavlja 'led' na vrednost
}                                      // koja odgovara stanju tastera
```

Ардуино радионица

analogRead(pin)

Očitava vrednost sa specificiranog analognog pina sa **10-bitnom rezolucijom**. Ova funkcija se odnosi samo na **analogne pinove (0 – 5; A0 – A5)**. Rezultantna celobrojna vrednost je u opsegu od 0 do 1023.

```
vrednost = analogRead(pin);  // postavlja 'vrednost' da bude jednaka sa 'pin'
```

Napomena:

Za razliku od digitalnih pinova, **analogni pinovi** ne moraju da se prvo deklarišu ni kao INPUT niti kao OUTPUT.

analogWrite(pin, value)

Upisuje pseudo-analognu vrednost koristeći hardverski omogućenu impulsnu širinsku modulaciju (engl. Pulse Width Modulation - PWM) na **izlazni pin označen kao PWM**. Kod Arduino pločica sa čipom ATmega168 ova funkcija radi na pinovima 3, 5, 6, 9, 10 i 11. Kod starijih Arduino pločica, sa čipom ATmega8, podržani su samo pinovi 9, 10 i 11. Parametar value može biti specificiran kao promenljiva ili konstanta u opsegu od 0 do 255.

```
analogWrite(pin, value);          // upisuje 'value' na analogni 'pin'
```

Vrednost 0 generiše stabilni izlaz od 0 volti na specificiranom pinu; vrednost od 255 generiše stabilan izlaz od 5 volti na specificiranom pinu. Za vrednosti između 0 i 255, pin će brzo naizmenično menjati vrednosti između 0 i 5 volti, što je viša vrednost, pin će češće biti u stanju HIGH (izlaz 5 volti). Na primer, za vrednost 64 izlaz će biti 0 volti tri četvrtine vremena a 5 volti jednu četvrtinu vremena. Za vrednost 128 izlaz će biti na 0 volti polovinu vremena a 5 volti polovinu vremena.

...

Kako se radi o hardverskoj funkciji, pin će generisati stabilan talasni oblik u pozadini posle poziva analogWrite sve do sledećeg poziva analogWrite (ili poziva digitalRead ili digitalWrite **na istom pinu**).

Napomena:

Za razliku od digitalnih pinova, analogni pinovi ne moraju da se prvo deklarišu ni kao INPUT niti kao OUTPUT.

...

U sledećem primeru očitava se analogna vrednost sa analognog ulaznog pina, konvertuje vrednost deljenjem sa 4 i šalje kao PWM signal na PWM pin:

```
int led = 10;                   // LED sa otpornikom od 220 oma na pinu 10
int pin = 0;                    // potenciometar na analognom pinu 0
int value ;                     // vrednost za očitavanje

void setup() {}                 // nije potrebno nikakvo postavljanje

void loop()
{
        value = analogRead(pin);   // postavlja 'value' da je jednako 'pin'
        value /= 4;                // konvertuje opseg 0 - 1023 u opseg 0 - 255
        analogWrite(led, value);   // šalje PWM signal na svetlosnu diodu
}
```

delay(ms), millis()

delay(ms)

Zaustavlja (zadržava) program za iznos vremena specificiran u milisekundama (1000ms je 1 sekunda).

```
delay(1000);                    // zadrška od 1 sekunde
```

millis()

Vraća broj milisekundi u formatu *unsigned long* pošto Arduino pločica počne da izvršava tekući program.

```
value = millis();    // postavlja 'value' da je jednako millis()
```

Napomena: Posle približno devet sati rada doći će do tzv. prepunjavanja (engl. overflow) ovog broja, tj. njegovog vraćanja („resetovanja”) na nulu.

min, max

min(x, y)

Izračunava minimum od dva broja bilo kog tipa i vraća manji broj.

```
value = min(value, 100);     // postavlja value na manji broj od brojeva
                             // 'value' i 100, obezbeđujući da se nikada neće
                             // dobiti veće od 100
```

max(x, y)

Izračunava maksimum od dva broja bilo kog tipa i vraća veći broj.

```
value = max(value, 100);     // postavlja value na veći broj od brojeva
                             // 'value' i 100, obezbeđujući da se dobije
                             // bar 100
```

randomSeed(seed)

Postavlja vrednost, ili seme (ili koren, tj. broj od koga počinju da se generišu slučajni brojevi), kao startnu tačku funkcije random().

```
randomSeed(value);          // postavlja value kao slučajno seme
```

Kako je Arduino nesposoban da generiše istinski slučajni broj, randomSeed omogućava da programer/korisnik postavi promenljivu, konstantu ili neku funkciju kao argument što doprinosi pouzdanijem generisanju slučajnog broja.

Postoji niz različitih semena ili funkcija koje se mogu koristiti u ovoj funkciji uključujući funkcije millis() ili čak analogRead() (koja recimo očitava električni šum na analognom pinu).

random(max), random(min, max)

Funkcija random vraća pseudo-slučajne brojeve unutar zadatog opsega specificiranog vrednostima min i max.

```
value = random(100, 200); // postavlja value na slučajni broj
                          // između 100 i 200
```

Napomena: Koristite ovu funkciju posle korišćenja funkcije randomSeed().

...

U sledećem primeru generiše se slučajna vrednost iz opsega 0 - 255 i šalje kao PWM signal na PWM pin:

```
int randNumber;          // promenljiva za čuvanje slučajne vrednosti
int led = 10;            // LED sa otpornikom od 220 oma na pinu 10

void setup() {}          // nije potrebno postavljanje

void loop()
{
        randomSeed(millis());          // postavlja millis() kao seme
        randNumber = random(255);      // slučajni broj između 0 i 255
        analogWrite(led, randNumber);  // šalje PWM signal
        delay(500);                    // zadrška od pola sekunde
}
```

Serial.begin(rate)

Otvara serijski port i postavlja serijsku brzinu prenosa podataka. Tipična brzina prenosa za komunikaciju sa personalnim računarom je 9600 bauda ali su podržane i druge brzine.

```
void setup()
{
        Serial.begin(9600);          // otvara serijski port i postavlja
}                                    // brzinu prenosa podataka na 9600
```

Napomena:

Kada se koristi serijska komunikacija, digitalni pinovi 0 (RX) i 1 (TX) ne mogu se **istovremeno** koristiti.

Serial.println(data)

Ispisuje (štampa) podatke na serijskom portu uz automatsko dodavanje znaka za kraj reda (tzv. karet, engl. carriage return) i znaka za novi red. Ova komanda ima sličan oblik kao i komanda Serial.print(), ali doprinosi boljoj čitljivosti ispisa podataka na softverskom serijskom monitoru (Serial Monitor).

```
Serial.println(analogValue);// šalje na ispis vrednost promenljive  'analogValue'
```

Napomena:

Više informacija o varijetetima funkcija Serial.println() i Serial.print() može se naći na službenom Arduino portalu.

...

U sledećem primeru očitava se vrednost na analognom pinu 0 i taj podatak se šalje personalnom računaru svake sekunde.

```
void setup()
{
        Serial.begin(9600);                    // postavlja brzinu serijske
komunikacija na 9600
}

void loop()
{
        Serial.println(analogRead(0));         // šalje analognu vrednost
        delay(1000);                           //zadrška od 1 sekunde
}
```