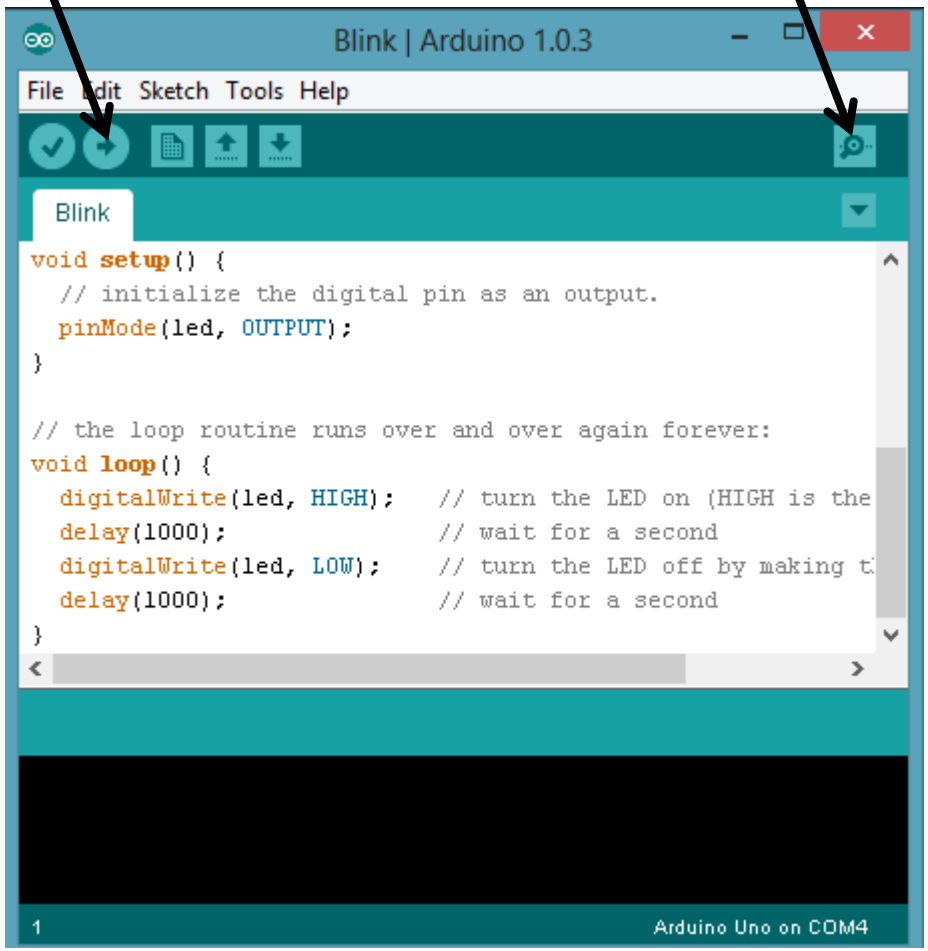
Arduino programiranje

Naloži

Serijski vmesnik- monitor



C jezik Arduina

Arduino je programiran v C jeziku.

Več informacij in znanja o Arduino lahko pridobite na http://arduino.cc/

Zgradba Arduino programa

Vsak Arduino program (pogosto imenovan sketch) ima dve zahtevani funkciji (imenovani tudi rutini).

Prva zahtevana funkcija je **void setup(){ }**

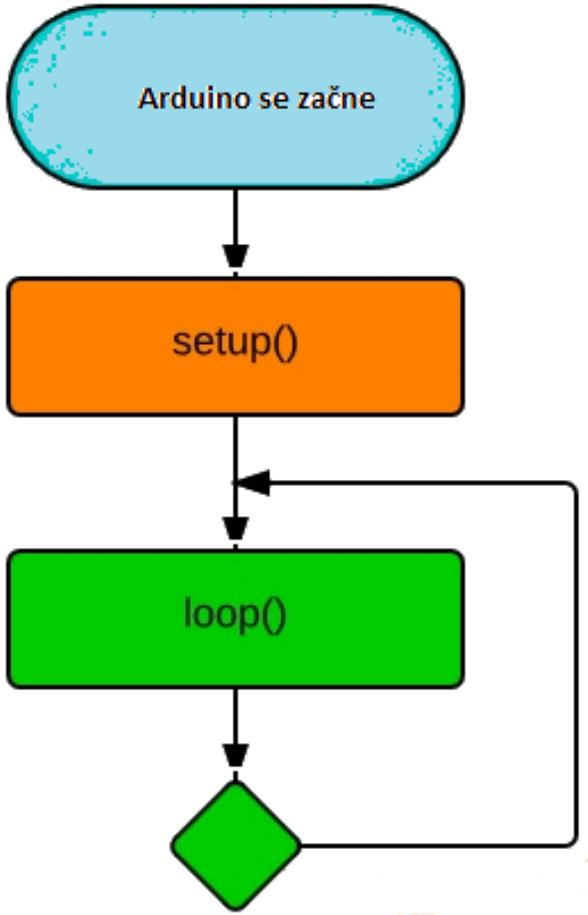
Druga zahtevana funkcija je **void loop(){ }**

Funkcija **void setup(){ }**

Vsa koda med tema dvema zavitima oklepajema, se bo izvedla le enkrat, takrat ko bo Arduino program prvič stekel. Če bomo Arduino izključili in ponovno vključili bo setup() zagnan ponovno.

Funkcija **void loop(){ }**

Ta funkcija imenovana tudi zanka steče po tem, ko je končan void setup(). Po tem, ko bo ta funkcija izvedena, se bo le void loop() ponavljal, dokler ne odklopimo vira energije.



Komentarji

// (enovrstični komentar)

Pogosto je koristno, da napišete opombe, ker je koristno, da veste, kaj katera vrstica v kodi naredi. Komentar naredimo z dvema poševnicama, in vse od tod dalje v tej vrstici, ne bo del programa

/* */(večvrstični komentar)

Če je komentar daljši (več vrstični -mogoče na začetku), uporabimo najprej /* in na koncu */ . Vse kar je med tema dvema znakoma ne bo del programa.

Zavita oklepaja in podpičje

{ } (zavita oklepaja)

Uporablja za opredelitev, kadar se del kode začne in konča (uporabljen je v funkcijah, in tudi zanki).

; (podpičje)

Vsaka vrstica kode, se mora končati z podpičjem (torej ne večvrstični komentarji).

Spremenljivke

Spremenljivka je sestavljena iz

imena spremenljivke, ki jo določimo sami in

tipa spremenljivk, ki je lahko:

- **int** (integer)
- **long** (long)
- **float** (float)
- **boolean** (boolean)
- **char** (character)

Naloga

Katere vrednosti lahko zavzame tip **integer**, če mu je dodeljen 2 bajta pomnilnika in tip **long**, če mu je dodeljen 4 bajte pomnilnika?

Tipi spremenljivk int, long, float

Tip spremenljivke **int** zavzame 2 bajta (16 bitov) in je brez decimalnih mest in se bo shranila v vrednost med -32 768 in 32 767.

Tip spremenljivke **long** zavzame 4 bajte (32 bitov) in je brez decimalnih mest in se bo shranila v vrednost med -2 147 483 648 in 2 147 483 647.

Tip spremenljivke **float** se uporablja za plavajoče matematične točke (decimalke). Porabi 4 bajte (32 bitov) pomnilnika in ima razpon med -3.4028235E+38 in 3.4028235E+38.

Tipa spremenljivk boolean in char

Tip spremenljivke **boolean** zavzame vrednost 1 ali 0. Uporablja le 1 bajt (8 bitov) pomnilnika.

Tip spremenljivke **char** shrani en znak z ASCII kodo (torej 'A' =65). Uporablja 1 bajt (8 bitov) pomnilnika.

Uporaba serijskega monitorja

S serijskem monitorjem enostavno komuniciramo z Arduinovo ploščo.

V funkcijo **void setup()** moramo vključiti stavek **Serial.begin(9600)**, ki omogča prenos podatkov s hitrostjo 9600 baudov.

Za izpis na serijski monitor uporabljamo stavka **Serial.print()** in **Serial.println()**

Serial.print()

Stavek **Serial.print()** izpiše podatke na serijski monitor in naslednji izpis nadaljuje v isti vrstici.

Če so podatki med narekovaji, potem gre za besedila, drugače pa gre za vrednosti spremenljivk.

Primer:

Koda

```
void setup() {
Serial.begin(9600);
Serial.print("Dobrodosel");
Serial.print("Dobrodosel");
}
void loop() {
}
```

Serijski monitor

```
DobrodoselDobrodosel
```

Serial.println()

Stavek **Serial.println()** izpiše podatke na serijski monitor in naslednji izpis nadaljuje v novi vrstici.

Koda

```
void setup() {
Serial.begin(9600);
Serial.println("Dobrodosel");
Serial.print("Dobrodosel");
Serial.println("Dobrodosel");
}
void loop() {
}
```

Serijski monitor

```
Dobrodosel
DobrodoselDobrodosel
```

Naloga_izpis na serijski monitor

Komentiraj in napiši izpis serijskega monitorja

```
void setup() {
Serial.begin(9600);
Serial.println("Dobrodosel v programu Arduino");
delay(5000);
}
void loop() {
Serial.println("Dobrodosel");
delay(1000);
}
```

Operatorji

Operatorje razvrstimo v:

- Aritmetične operatorje (+, - , *, / , %)
- Primerjalne operatorje (==, != , < , > , <= , >=)
- Logične operatorje (&& , || , !)
- Bitne logične operatorje (& , | , ^ , <<n , >>n , ~)

Aritmetične operatorje uporabljamo pri aritmetičnih izrazih. Primerjalne operatorje in logične operatorje uporabljamo pri odločitvenih in ponavljalnih stavkih. Bitne logične operatorje pri logičnih izrazih.

Aritmetični operatorji

Aritmetični operatorji izvajajo aritmetične operacije nad dvema ali več števil in vračajo kot rezultat številsko vrednost.

Simbol	Opis
+	Seštevanje
-	Odštevanje
*	Množenje
/	Deljenje
%	Modul (ostanek pri celoštevilskem deljenju)

Primerjalni operatorji

S primerjalnimi operatorji primerjamo dva izraza. V primeru, da je odgovor na vprašanje pritrdilno, je vrednost primerjalnega izraza DA(1), sicer je NE(0).

Simbol	Opis
==	Je enako
!=	Ni enako
<	Manjše kot
>	Večje kot
<=	Manjše ali enako
>=	Večje ali enako

Logični operatorji

Logični operatorji izvajajo logične operacije nad dvema ali več izrazoma in vračajo kot rezultat vrednost DA(1), oziroma NE (0).

Simbol	Opis
&&	Logični IN
||	Logični ALI
!	Logični NE

Bitni logični operatorji

Bitni logični operatorji izvajajo logične operacije nad posameznimi biti in vračajo kot rezultat logično vrednost.

Simbol	Opis
&	IN
|	ALI
^	XOR
<<n	pomik bitov v levo za n mest
>>n	pomik bitov v desno za n mest
~	eniški komplement (negacija bitov)

Eniški komplement in dvojiški komplement

Eniški komplement števila izvede negacijo bitov in predstavlja nasprotno število minus ena.

Dvojiški komplement =Eniški komplement + 1

Primeri

Izraz	dvojiško	desetiško
a=5	0000 0101	5
~5=-5-1	1111 1010	-6 =-5-1
~(-6)	0000 0101	5 =6-1
~(-6) +1	0000 0110	6 =6-1+1

Pomen dvojiškega komplementa

Dvojiški komplement števila vrne rezultat nasprotno število.

Krmilne strukture

Program sestavljajo različni stavki.

Razdelimo jih lahko na stavke, ki se izvršijo

- enkrat, ko je pogoj izpolnjen (stavek **if** , stavek **switch**)
- večkrat, dokler je pogoj izpolnjen (stavek **while**, stavek **do … while**, stavek **for**)

Pravilo pisanja stavka if

if (pogoj**)**

{stavki_pogojda**;}**

else

{stavki_pogojne**;}**

/*Stavki_pogojda se izvršijo, če je pogoj izpolnjen, sicer se izvršijo stavki_pogojne. */

Diagram poteka stavka if

```
if (pogoj)
{
stavek1;
stavek2;
}
else
{
stavek3;
stavek4;
}
```

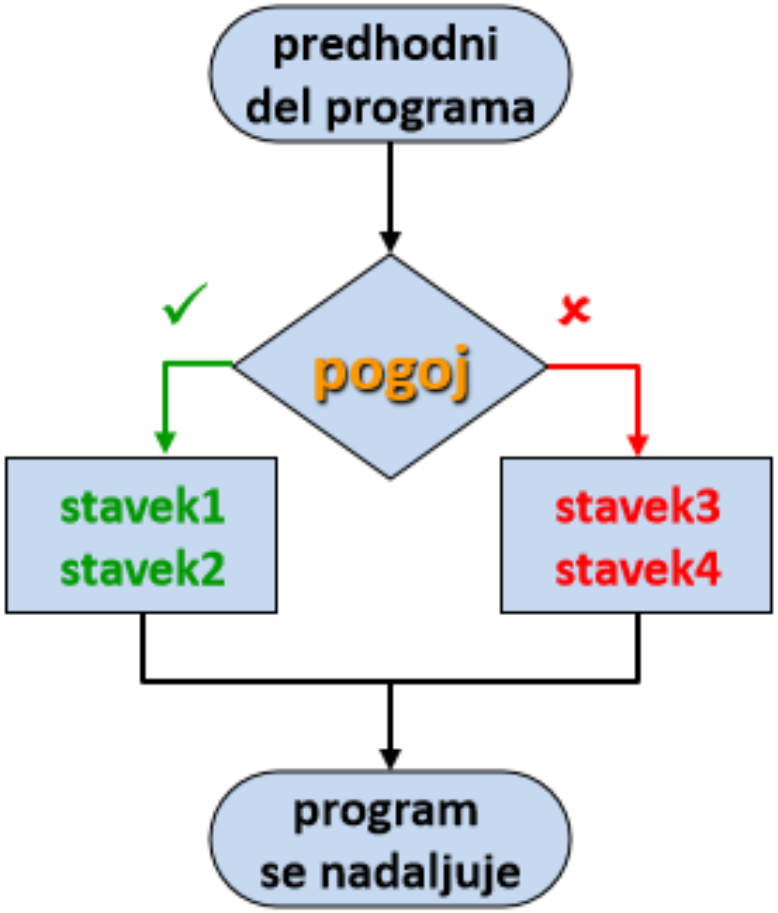


Diagram poteka stavka if

Pravilo pisanja stavka switch

switch (izraz**) {**
case vrednost_1**:**
{ stavki_1**; }**
break;
. . .

default:
{ stavki_sicer**; }**
}

/*Izvršijo se stavki_x, katerega izraz=vrednost_x. Če nobena vrednost_x ni enaka izrazu, se izvršijo stavki_sicer. */

Diagram poteka vgnezdenega stavka if

```
if (pogoj1)
{stavki_1;}
else
if (pogoj2)
{ stavki_2; }
else
if …
…
else
{ stavki_sicer; }
```

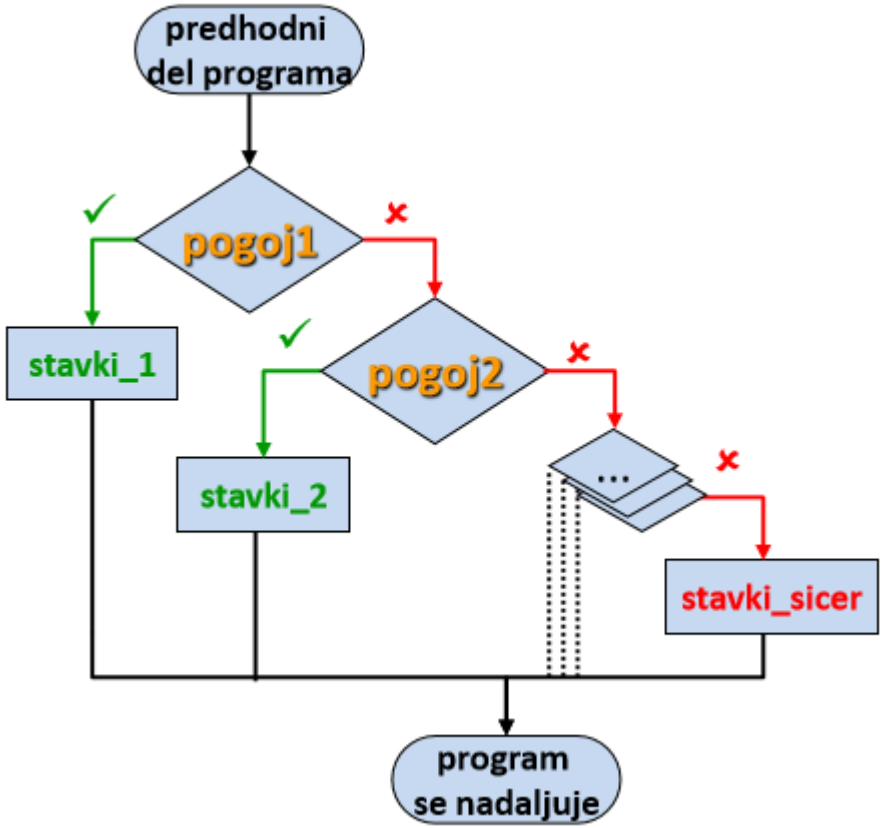


Diagram poteka vgnezdena stavka if

Primerjava stavka switch in vgnezdenega stavka if

```
switch (izraz)
{
   case vrednost_1:
  { stavki_1; }
   break;
  case vrednost_2:
  { stavki_2; }
  break;
  default:
  { stavki_sicer; }
 }
```

```
if (izraz = vrednost_1)
     { stavki_1; }
else if(izraz = vrednost_2)
     { stavki_2; }
    else
          { stavki_sicer; }
```

Naloge – napiši izpise na serijskem monitorju

```
//N1
int a,b;
void setup() {
Serial.begin(9600);
a=19;
b=5;
if (a > b)
        {
        Serial.print("a+b=");
        Serial.println(a+b);
        }
        else
        {
        Serial.print("a-b=");
        Serial.println(a-b);
        }
}
void loop() {
}
```

//N2

```
int a,b,c;
void setup() {
Serial.begin(9600);
a=19; b=14; c=24;
if (a > b && a>c)
{
Serial.println("Stevilo a je najvecje.");
Serial.print("a=");
Serial.println(a);
}
else
if (b>c)
```

```
 {
Serial.println("Stevilo b je najvecje.");
Serial.print("b=");
Serial.println(b);
}
else
{
Serial.println("Stevilo c je najvecje.");
Serial.print("c=");
Serial.println(c);
delay(2000);
}
}
void loop() {
}
```

//N3

```
void setup() {
Serial.begin(9600);
switch (5)
{
case 1:
Serial.println("Izraz je Vrednost 1");
break;
case 2:
Serial.println("Izraz je Vrednost 2");
break;
default:
Serial.println("Izraz ni enak Vrednosti");
}
}
void loop() {
}
```

Krmilne strukture

Program sestavljajo različni stavki.
Razdelimo jih lahko na stavke, ki se izvršijo

- enkrat, ko je pogoj izpolnjen (stavek **if** , stavek **switch**)
- večkrat, ko je pogoj izpolnjen (stavek **while**, stavek **do … while**, stavek **for**)

Za stavke, ki se izvršijo enkrat , ko je pogoj izpolnjen, uporabimo

- **if (**pogoj**) {**stavki_pogojda**;} else {**stavki_pogojne**;}**
- **switch (**izraz**) { case** vrednost_1**: {** stavki_1**; } break;**

```
        . . .
    default: { stavki_sicer; }
}
```

Za stavke, ki se lahko izvršijo večkrat , dokler je pogoj izpolnjen, uporabimo

- **while (**pogoj**) {**stavki_pogojda**;}**
- **do {**stavki_pogojda**;} while (**pogoj**)**
- **for (**stevec**;** pogoj stevca**;** stevec++ **) {** stavki_pogoj stevca**; }**

Zanke

Za stavke, ki se lahko izvršijo večkrat , dokler je pogoj izpolnjen, uporabimo

- **while (**pogoj**) {**stavki_pogojda**;}**
- **do {**stavki_pogojda**;} while (**pogoj**)**
- **for (**stevec**;** pogoj stevca**;** stevec++ **) {**
  stavki_pogoj stevca**; }**

Zanka **while**

Dokler je izpolnjen **pogoj**, se **stavki** v zanki ponavljajo.

```
while (pogoj)
{
stavki;
}
```

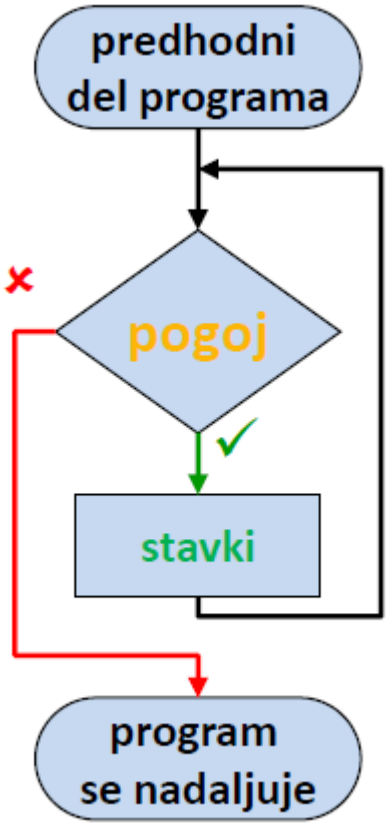


Diagram poteka zanke while

Primer **while**

```
int Stevec;
void setup() {
Serial.begin(9600);
}
void loop() {
Stevec=1;
while (Stevec<=5)
{
Serial.print("Stevec=");
Serial.println(Stevec);
delay(1000);
Stevec=Stevec+1;
}
}
```

Zanka **do…while**

Stavki v zanki se ponavljajo, **dokler** je **pogoj** izpolnjen.

```
do
{
stavki;
}
while (pogoj);
```

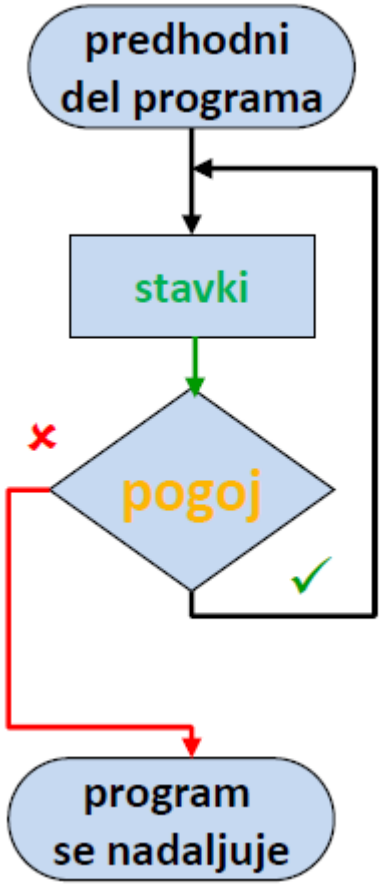


Diagram poteka zanke do … while

Primer **do** … **while**

```
int Stevec;
void setup() {
Serial.begin(9600);
}
void loop() {
Stevec=1;
do
{
Serial.print("Stevec=");
Serial.println(Stevec);
delay(1000);
Stevec=Stevec+1;
}
while (Stevec<=5);
}
```

Zanka **for**

Zanka **for** se ponavlja, dokler je izpolnjen **pogoj stevca** .

```
for (stevec; pogoj stevca; stevec++ )
{
stavki;
}
```

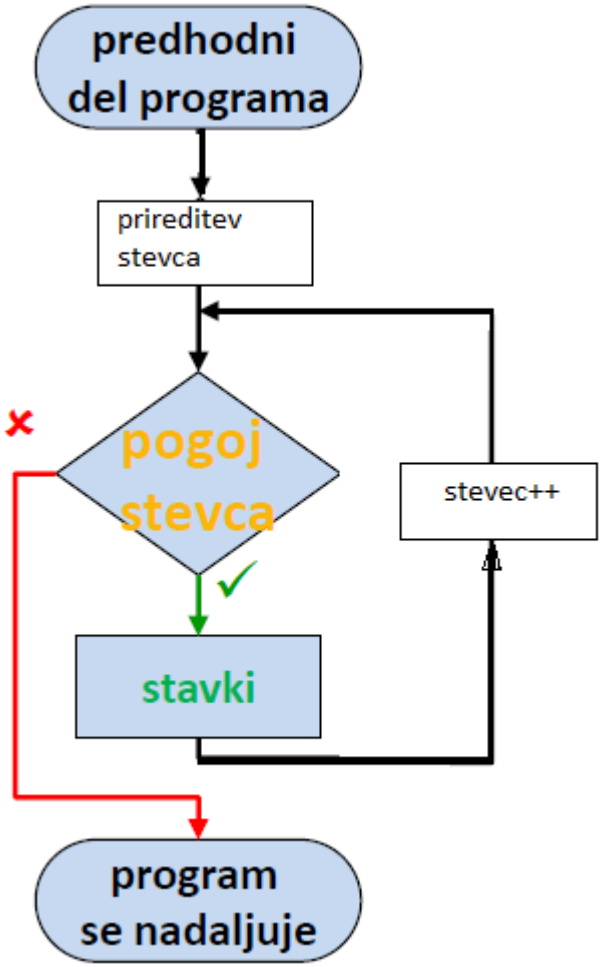


Diagram poteka zanke for

Primer **for**

```
int Stevec;
void setup() {
Serial.begin(9600);
}
void loop() {
Stevec=1;
for(Stevec;Stevec<=5;Stevec=Stevec+1)
{
Serial.print("Stevec=");
Serial.println(Stevec);
delay(1000);
}
}
```

Deklaracija lastnih funkcij

Lastne funkcije deklariramo za funkcijo setup in loop. Funkciji setup in loop sta tipa void, medtem ko so lahko lastne funkcije tipa int, long, float, char, boolean, …
Lastno funkcijo kličemo po imenu tako da v zaviti oklepaj setup-a ali loop-a napišemo ime lastne funkcije.

Primer deklaracije in klicanje lastne funkcije

```
void setup() {
MojaFunkcija();  //klicanje
}
void loop() {
}
int MojaFunkcija(){  }   //deklaracija
```

Primer deklaracije funkcije vsota

```
void setup() {                  //funkcija setup
Serial.begin(9600);     //nastavitev serijskega prenosa
vsota(5,7);                     //skok v funkcijo vsota
}
void loop() {                   //funkcija loop
}
int vsota(int a, int b){        //deklaracija funkcije vsota
Serial.print("a+b=");           //izpis besedila a+b
Serial.println(a+b);            //izpis vrednosti a+b
delay(1000);                    //premor 1000ms
}
```