ARDUINO RADIONICA
PROGRAMIRANJE

Proces kreiranja programa

- Programiranje se može posmatrati kao niz/serija procesa. Upotreba procesa je tehnika koja se koristi u kontroli kvaliteta kako bi se osiguralo da se zadaci obavljaju na isti način svaki put. Iako ovo može na prvi pogled da zvuči pomalo dosadno i monotono, **ukoliko se razvijaju programi na dosledan i standardizovan način, smanjuju se poznati nedostaci a dobija sistematičan način sprečavanja da se nove greške ponovo ponavljaju**.

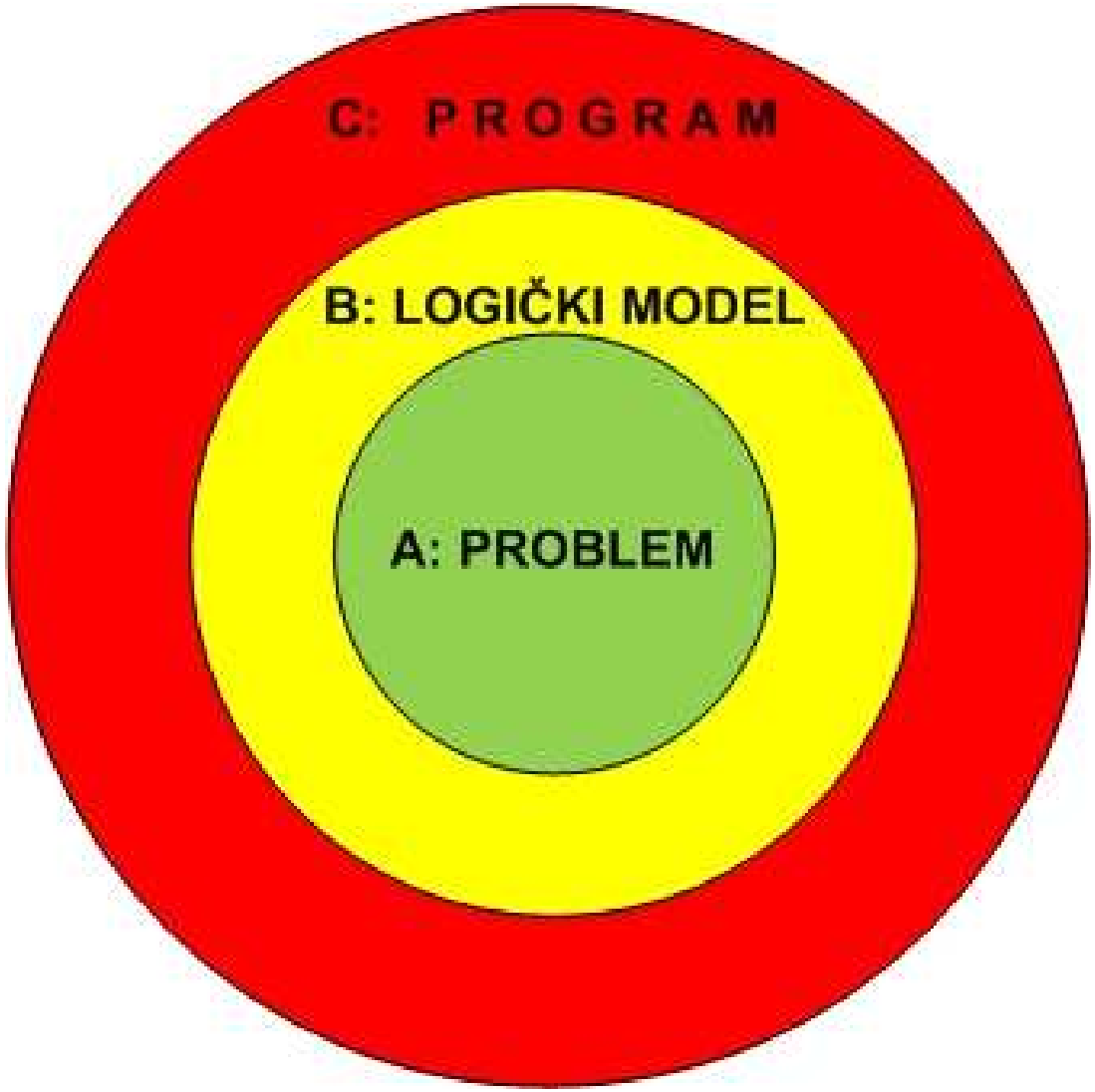
- Proces se može posmatrati kao kontrolni spisak (kontrolna lista). Da bi proces bio uspešan, mora se obaviti u korektnom redosledu niz zadataka. Kako se ovde ne radi o programiranju mehaničkih i nefleksibilnih aktivnosti, već se radi sa mašinama koje se zovu računari čiji rad je regulisan strogim pravilima, **programi moraju biti vrlo precizni**.

...

- Jedna od manje očiglednih pogodnosti vezanih za korišćenje procesa je to da proces takođe može biti standard. Praksa pokazuje da često neki program tokom njegove eksploatacije održava više programera. Kada programer radi sa nečijim tuđim programom, ukoliko taj program nije razvijen u skladu sa industrijskim standardom, tada su male šanse da će novi programer bez problema moći da prepravlja preuzeti program. **Kao što se kućna električna instalacija razvodi prema standardu** tako da kasnije može da je prepravlja bilo koji električar, tako **i u slučaju kreiranja programa treba se pridržavati standarda da bi se olakšalo potonje održavanje.**
- Proces koji koriste programeri za kreiranje i održavanje računarskih programa je **proces razvoja programa**. Ako se taj proces poštuje i koriste njegovi glavni koraci, on deluje kao kontrolni spisak koji će u velikoj meri povećati šanse za uspeh.

Disciplinovan pristup programiranju

Počinje se sa dobro definisanim **model**om, zatim sledi **dokumentovanje** modela i okončava sa kreiranjem **programa**



Ардуино радионица

Modeli - putanje ka savršenom rešenju

- Modeli treba da predstavljaju idealno rešenje. Model treba da uzme u obzir sve zahteve bez pristrasnosti i da identifikuje šta bi bilo najbolje moguće logičko rešenje. **Ponekad se model naziva konceptualnim pogledom (idealni svet)**

  dok je realni program fizički pogled ili primena (implementacija) logike.

- **Hoće li definisani model odslikavati konačno rešenje?**

- Možda da, možda ne. Često idealno rešenje je teško realizovati programskim jezikom, a ponekad idealno rešenje se ne izvršava najefikasnije. Takođe, različiti programski jezici imaju različite prednosti i slabosti.

...

- **Model**: model je apstrakcija stvarnosti/realnosti. Kod modelovanja projekta koriste se zahtevi i objedinjuju da bi se dobilo rešenje. Model omogućava da se eksperimentiše i fino podešava vizija rešenja sve dok se ne ostvari zadovoljavajuća tačnost. Prilikom toga treba obratiti pažnu da nije ostao neprimećen neki zahtev. Logički model treba posmatrati kao vodilju ka savršenom programu.

- **Logički put**: logička putanja je jednostavo sekvenca/niz instrukcija i iskaza. Ponekad se putanja logičkih iskaza naziva algoritam. Logički put ili algoritam je skup koraka/instrukcija za obavljanje konkretnog zadatka. Program se obično sastoji od više serija logičkih puteva kada je u pitanju rešavanje velikih problema.

...

- Jedna od tehnika koje pomažu programeru da modelira logiku je **dijagram toka (flow chart).** Mada je dijagram toka vizuelno lako čitljiv, nije ga lako sastaviti. Ukoliko osoba ne raspolaže bogatom praksom i nema smisla za aranžiranje blokova, korišćenje papira i olovke ili šablona dijagram toka u odgovarajućem editoru biće veoma zamoran proces.

- **Dijagrami tokova sastoje se od standardnih grafičkih simbola povezanih linijama kod kojih svaki simbol odražava logiku operacije**. Pored toga, svaki simbol ne samo da je standardizovan da predstavlja operaciju (npr. IF naredbu odlučivanja), već i označava logiku operacije sa kratkim objašnjenjem šta ta logička instrukcija obavlja.

...

Simboli dijagrama toka su:

- **Pravougaonik sa zaobljenim temenima**: koristi se za označavanje početka i kraja logičkog toka. Ovaj simbol se zove i simbol terminator.
- **Krug**: koristi se za povezivanje dijagrama koji se protežu na više stranica papira i pomažu kod povezivanja logičkih putanja sa jedne strane na drugu. Krug označava kraj dijagrama na strani i sadrži unutar sebe identifikacioni broj. Sledeća strana počinje sa krugom sa istim identifikacionim brojem. Ovo pomaže da se korektno povežu dijagrami sa dve stranice.
- **Pravougaonik**: koristi se za prikazivanje jednog koraka u logičkom modelu. Na primer, neka aktivnost, izračunavanje itd.

...

- **Romb**: **koristi se za predstavljanje odluke**. Ishod odluke može biti tačan/netačan (true/false) i obično se u programskom kodu implementira kao IF naredba.

- **Trapez**: **predstavlja logički ulaz ili izlaz** (tj. čitanje sa tastature/senzora ili iz fajla ili ispisivanje na ekranu/štampaču).

- **Linije**: **pokazuju kretanje logike i pravac informacije** od simbola do simbola.

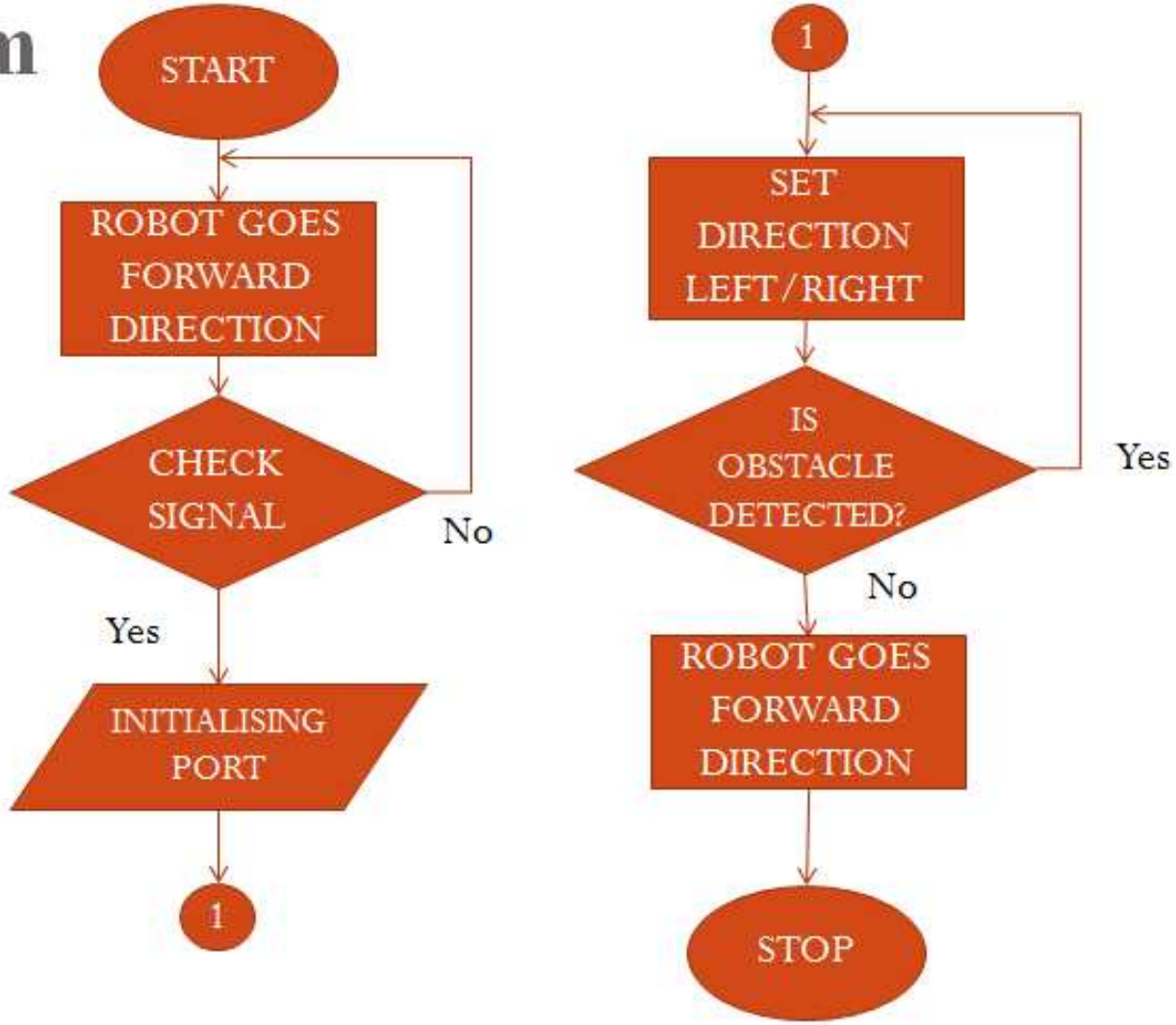
Prednosti dijagrama toka

- Najveća prednost dijagrama toka proističe iz njegove jednostavnosti. Oni su jednostavni za razumevanje i kreiranje. Budući da većina programera (i ljudi uopšte) ima težnju da lakše prihvata koncepte koji se izlažu vizuelno, dijagrami toka su u skladu sa tim. Kada čovek vidi ispred sebe logički dijagram, mnogo mu je laške da raspravlja o svrsi programa i programskim koracima. Korišćenjem Microsoft Visio aplikacije mogu se lako, brzo i efikasno kreirati dijagrami toka.

Nedostaci dijagrama toka

- Dijagrami toka imaju tri slabosti/nedostatka.
- **Prvi nedostatak** se ogleda u teškoći crtanja sofisticirane logike koja se proteže na nekoliko listova papira.Da bi se to prevazišlo ili svaki simbol mora biti iscrtan u mikroskopskoj razmeri ili čitalac dijagrama mora imati na raspolaganju ogromnu radnu površinu kako bi mogao da rasprostre više listova papira koji obrazuju dijagram toka.
- **Drugi nedostatak** se javlja zbog potrebne crtačke veštine koja može predstavljati problem za neke programere. Crtački softver tipa Microsoft Visio umnogome ublažava ovaj nedostatak.
- **Treći nedostatak** može biti poteškoća u prevođenju logike dijagrama toka u naredbe programskog jezika.

Flowchart's Program



Arduino programi

- Arduino programi se sastoje od tri sekcije:
  - **strukture**,
  - **funkcija** i
  - **vrednosti** (promenljive i konstante).

Struktura

Osnovna struktura Arduino programskog jezika je prilično jednostavna i **sastoji se iz dva dela**. Ta dva neophodna dela, ili funkcije, obuhvataju blokove naredbi.

```
void setup()
{
        naredbe;
}

void loop()
{
        naredbe;
}
```

Može se reći da je setup() **pripremni** a loop() **izvršni deo programa**. Obe funkcije su neophodne da bi program radio.

...

Funkcija setup() sledi posle deklaracija promenljivih i samim tim nalazi se na samom početku programa. To je prva funkcija koju program izvršava i izvršava se samo jedanput a koristi se za postavljanje režima rada nekog od pinova Arduino pločice ili za inicijalizaciju serijske komunikacije. Mora postojati u programu čak i ako nema naredbi koje bi trebalo da izvrši.

```
void setup()
{
        pinMode(pin, OUTPUT); // postavlja 'pin' kao izlaz
}
```

...

Sledeća funkcija koja se po redu pojavljuje u programu je funkcija loop(). Ona radi baš ono što njen engleski naziv i implicira, ciklično/kružno izvršava naredbe koje obuhvata. Omogućava programu da reaguje i upravlja Arduino pločicom.

```
void loop()
{
        digitalWrite(pin, HIGH);  // uključuje 'pin'
        delay(1000);              // zadrška od 1 sekunde
        digitalWrite(pin, LOW);   // isključuje 'pin'
        delay(1000);              // zadrška od 1 sekunde
}
```

Funkcije

Funkcija je blok naredbi koji ima svoj naziv/ime, tj. to je blok programskog koda koji se izvršava kada se funkcija pozove. Funkcije setup() i loop() su već objašnjene.

Namenske funkcije se pišu za obavljanje ponavljajućih zadataka kako bi se smanjila nečitljivost programa.

Funkcije se definišu tako što se prvo deklariše **tip funkcije**.

Tip funkcije se odnosi na tip vrednosti koji vraća funkcija po svom izvršavanju. Ukoliko funkcija po izvršavanju vraća celobrojnu vrednost onda ona treba da se deklariše da je tipa 'int', a ukoliko uopšte ne vraća vrednost tada treba da se deklariše da je tipa 'void'.

Osim tipa funkcije, **deklariše se naziv funkcije** a u zagradama parametri koji se prosleđuju funkciji.

...

Primer funkcije

```
type functionName(parameters)
{
        int v;                          // kreira privremenu promenljivu 'v'
        v = analogRead(pot)     // čita analognu vrednost 'pot'
        v/= 4;                  // konvert. opseg 0 do 1023 u opseg 0 do 255
        return v;                       // vraća konačnu vrednost
}
```

Vitičaste zagrade - {}

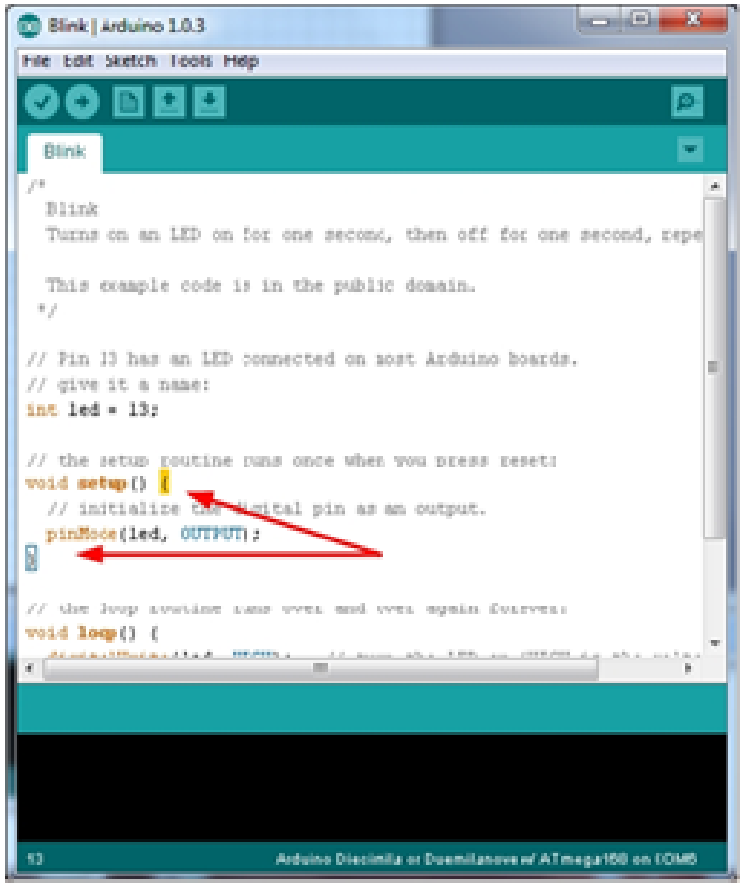
Vitičaste zagrade definišu početak i kraj funkcijskog bloka ili bloka naredbi koji se nalazi u sklopu for i if naredbe.

```
type function()
{
        naredbe;
}
```

Leva (otvarajuća) vitičasta zagrada mora uvek biti praćena desnom (zatvarajućom) vitičastom zagradom, tj. **zagrade moraju biti izbalansirane**. Neizbalansirane zagrade dovode do grešaka prilikom prevođenja programa a i takve programe je teško logički ispratiti.

...

Arduino okruženje poseduje pogodno svojstvo koje omogućuje automatsko praćenje balansiranja vitičastih zagrada. Naime, kada korisnik označi/selektuje zagradu ili postavi kursor miša (tzv.tačku umetanja) neposredno ispred ili iza vitičaste zagrade, okruženje ističe njenog logičkog pratioca.



Oznaka tačka-zapeta - ;

Oznaka ; se mora postaviti da označi kraj naredbe i na taj način razdvaja elemente programa. Takođe, ta oznaka se koristi da odvoji elemente u petlji for.

```
int x = 13;          // deklariše promenljivu 'x' kao celobrojnu vrednost 13
```

Napomena: Ukoliko se ne postavi oznaka kraja naredbe, kompajler će javiti grešku.

...

Blokovski komentar - /* ... */

Blokovski ili višeredni komentari su oblasti teksta koje program ignoriše i koriste se za opisivanje koda ili kao komentari koji pomažu drugima da shvate delove programa. Blok počinje oznakom /* a završava se oznakom */ i može da obuhvati više redova/linija.

```
/*  ovo je početak blokovskog
    komentara koji se proteže
    u više redova.
*/
```

Napomena: Dozvoljeno je da se linijski komentar ugnezdi unutar blokovskog komentara, ali nije dozvoljeno ugnježđivanje blokovskih komentara.

Linijski komentari - //

Komentar koji zahvata jednu liniju/red započinje sa oznakom // a završava se sa sledećom linijom programskog koda. Program ih ignoriše kao i blokovske komentare i ne zauzimaju memorijski prostor.

```
// ovo je jednoredni komentar
```

Ovaj tip komentara se često koristi za dodatno objašnjavanje onoga što radi naredba koja neposredno sledi.

Promenljive

Promenljiva (varijabla) je način da se smesti numerička vrednost za kasnije korišćenje od strane programa. Promenljive su brojevi koji se mogu kontinualno menjati nasuprot konstantama čije se vrednosti ne mogu nikada menjati. Promenljivu je potrebno deklarisati i opciono joj dodeliti vrednost koju treba da čuva. Sledeći segment koda deklariše promenljivu pod nazivom inputVariable a zatim joj dodeljuje vrednost koja se dobija sa analognog ulaznog pina pod rednim brojem 2:

```
int inputVariable = 0;                  // deklarisanje promenljive i
                                        // dodeljivanje vrednosti 0
inputVariable = analogRead(2);          // postavljanje promenljive na
                                        // vrednost sa analognog pina 2
```

Prvi red deklariše da će promenljiva 'inputVariable' sadržati celobrojnu vrednost (engl. integer; skraćenica int). Drugi red postavlja promenljivu na vrednost koja dolazi sa analognog pina 2. Ovime je omogućeno da vrednost očitana sa pina 2 bude dostupna svuda u programu.

...

Kada je promenljivoj dodeljena ili ponovo dodeljena vrednost, može se testirati njena vrednost da se proveri da li zadovoljava konkretne uslove a može se njena vrednost i direktno koristiti. U svrhu objašnjavanja rada sa promenljivama sledi primer sa **tri korisne operacije**.

Kôd testira da li je vrednost promenljive inputVariable manja od 100 i

ako je to tačno dodeljuje joj vrednost 100

a zatim postavlja zadršku čije je trajanje bazirano na vrednosti promenljive inputVariable:

```
if (inputVariable < 100)          // test da li je promenljiva manja od 100
{
        inputVariable = 100;   // ako je to tačno onda joj dodeli vrednost 100
}
delay(inputVariable);             // koristi vrednost promenljive za zadršku/pauzu
```

...

Napomena:

Promenljivama treba dodeljivati opisna imena/nazive kako bi kôd bio jasniji.

Promenljive tiltSensor ili startTaster

pomažu programeru, odnosno onome ko čita program da lakše shvati šta promenljiva predstavlja. Promenljivama se mogu dodeljivati proizvoljni nazivi izuzimajući službene reči Arduino programskog jezika.

Deklaracija promenljive

Sve promenljive moraju da se deklarišu pre nego što se koriste. Deklarianje promenljive znači definisanje njenog tipa vrednosti, specificiranje naziva i opciono dodeljivanje inicijalne vrednosti. Ovo je potrebno uraditi samo jednom u okviru programa, dok se vrednost promenljive može menjati u zavisnostiod logike programa.

Sledeći primer deklariše promenljivu inputVariable kao int, tj. celobrojnog tipa i dodeljuje joj inicijalnu vrednost 0. To je primer jednostavnog dodeljivanja.

```
int inputVariable = 0;
```

Promenljiva se može deklarisati na različitim mestima u programu a mesto gde je deklaracija postavljena određuje koji delovi programa mogu koristiti tu promenljivu.

Oblast važenja promenljive

Promenljiva se može deklarisati:
na početku programa pre funkcije setup(),
lokalno unutar funkcije/funkcija i
ponekad unutar bloka naredbi, recimo unutar for petlje.

Mesto gde je promenljiva deklarisana određuje oblast važenja (ili domet) promenljive, tj. mogućnost da konkretni delovi programa mogu da koriste tu promenljivu.

Globalna promenljiva je ona promenljiva koju može da vidi i koristi svaka funkcija i naredba u programu. **Takva promenljiva se deklariše na početku programa**, **pre funkcije** setup().

...

Lokalna promenljiva je promenljiva koja je definisana unutar funkcije ili unutar for petlje.

Ona je vidljiva i može se koristiti samo unutar funkcije u kojoj je deklarisana.

Međutim, moguće je da program ima dve ili više promenljivih sa istim nazivom **u svojim različitim delovima** koji koriste različite vrednosti. Osiguravajući da samo jedna funkcija ima pristup svojim promenljivama pojednostavljuje program i redukuje potencijalne programerske greške.

...

Sledeći primer pokazuje kako može da se deklariše više različitih tipova promenljivih i demonstrira vidljivost svake promenljive:

```
int value;                          // 'value' je vidljivo
                                    // za sve funkcije
void setup()
{
        // setup nije potreban
}

void loop()
{
        for (int i=0; i<20;)        // 'i' je vidljivo samo
        {                           // unutar for petlje
                i++;
        }
        float f;                    // 'f' je vidljivo samo
}                                   // unutar loop funkcije
```

Ардуино радионица

Tipovi podataka

Svakom računaru je važno sa kakvim podacima barata. U Arduino okruženju razlikuju se sledeći tipovi podataka:

kratki celobrojni - **byte**,

celobrojni - **int**,

prošireni celobrojni - **long**,

realni - **float**.

Svaki tip podatka zauzima određenu veličinu (memoriju) u računaru i ima određeni opseg vrednosti koji pokriva.

...

byte

Promenljiva tipa Byte smešta 8-bitnu numeričku vrednost bez decimalne tačke. Ima opseg vrednosti od 0 do 255.

```
byte nekaPromenljiva = 180;        // deklariše promenljivu 'nekaPromenljiva'
                                   // da je tipa byte
```

int

Promenljiva tipa int (integer) je primarni tip podatka za smeštanje brojeva bez decimalne tačke i prihvata 16-bitne vrednosti koje mogu biti u opsegu od 32767 do -32768.

```
int nekaPromenljiva = 1500;        // deklariše promenljivu 'nekaPromenljiva'
                                   // da je tipa int
```

...

Napomena:

Celobrojne (integer) promenljive će prebaciti vrednost na drugi kraj opsega ukoliko budu primorane da premaše minimalnu ili maksimalnu vrednost prilikom nekog dodeljivanja vrednosti ili poređenja. Na primer, ako je x = 32767 a sledeća naredba dodaje 1 na x, tj. x = x + 1 ili je x++, x će se prebaciti na drugi kraj opsega i biti jednako -32768.

...

long

Tip podatka koji obuhvata prošireno područje za celobrojne vrednost, bez decimalne tačke i smešta 32-bitnu vrednost u opsegu od 2 147 483 647 do - 2 147 483 648.

```
long nekaPromenljiva = 90000;       // deklariše promenljivu 'nekaPromenljiva'
                                    // da je tipa long
```

float

Tip podatka za realne brojeve ili brojeve koji imaju decimalnu tačku. Realni brojevi (ili brojevi sa pokretnom zapetom) imaju veću rezoluciju od celobrojnih brojeva i smeštaju se kao 32-bitne vrednosti u opsegu od 3.4028235E+38 do -3.4028235E-38.

```
float nekaPromenljiva = 3.14;       // deklariše promenljivu 'nekaPromenljiva'
                                    // da je tipa float
```

...

Napomena:

Brojevi u pokretnoj zapeti nisu baš egzaktni i mogu da daju neobične rezultate prilikom poređenja. Takođe, matematičke operacije sa ovim brojevima su mnogo sporije u odnosu na iste operacije sa celobrojnim brojevima, stoga je poželjno izbeći korišćenje ovih brojeva kadgod je to moguće.

...

Nizovi

Niz je kolekcija vrednosti kojima se pristupa korišćenjem indeksnog broja. Bilo koja vrednost u nizu može se pozvati navođenjem naziva niza i indeksnog broja te vrednosti. Nizovi se indeksiraju počevši od 0, tako da prva vrednost u nizu ima indeksni broj 0. Niz mora da se deklariše i mogu opciono da mu se dodele vrednosti pre nego što bude korišćen (pozivan/referenciran).

```
int myArray[ ] = {vrednost0, vrednost1, vrednost2, ... }
```

Takođe, moguće je deklarisati niz navođenjem tipa i veličine niza a kasnije dodeljivati vrednosti preko pozicije indeksa:

```
int myArray[5];          // deklariše celobrojni niz od 5 članova
myArray[3] = 10;         // dodeljuje 4-tom članu niza vrednost 10
```

...

U sledećem primeru dat je niz koji ima 10 članova. Poslednji član (element) niza se indeksira brojem 9. Prema tome:

```
int myArray[10] = {9,3,2,4,3,2,7,8,9,11};
// myArray[9]    sadrži vrednost 11
// myArray[10]   ovo referencira neku drugu memorijsku adresu a ne niz
// myArray
```

Nizovi se često koriste u for petljama gde se brojač inkrementalnog tipa koristi kao indeks pozicije za pristupanje vrednostima članova niza.

...

Sledeći primer koristi niz za blinkovanje (treperenje) svetlosne diode. Na vrhu for petlje, brojač počinje sa vrednošću 0 i ujedno indeksira niz flicker, tj. pokazuje na vrednost 180 a ta vrednost se dodeljuje pinu 10, zatim sledi zadrška od 200ms i ponovo se ide na vrh petlje for i inkrementira se brojač tako da sada pokazuje na sledeći član niza flicker itd.

```
inte ledPin = 10;                                   // LED na pinu 10
// sledi niz sa 8 članova
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
//
void setup()
{
        pinMode(ledPin, OUTPUT);                    // postavlja pin na
OUTPUT
}
//
```

...

```
void loop()
{
        for (int i=0; i<7; i++)                    // brojač petlje
        {                                          // se koristi za indeksiranje,
                analogWrite(ledPin, flicker[i]);   // tj. pristup članovima niza
                delay(200);                        // zadrška od 200ms
        }
}
```

Ova prezentacija je nekomercijalna.

Slajdovi mogu da sadrže materijale preuzete sa Interneta, stručne i naučne građe, koji su zaštićeni Zakonom o autorskim i srodnim pravima. Ova prezentacija se može koristiti samo privremeno tokom usmenog izlaganja nastavnika u cilju informisanja i upućivanja studenata na dalji stručni, istraživački i naučni rad i u druge svrhe se ne sme koristiti –

Č**lan 44** - Dozvoljeno je bez dozvole autora i bez plaćanja autorske naknade za nekomercijalne svrhe nastave: (1) javno izvođenje ili predstavljanje objavljenih dela u obliku neposrednog poučavanja na nastavi; - ZAKON O AUTORSKOM I SRODNIM PRAVIMA ("Sl. glasnik RS", br. 104/2009 i 99/2011)

Dragan S. Marković