Bela Nemet, dipl.ing.

ARDUINO - PROGRAMIRANJE

Robotika i mikrokontroleri postali su (sa znatnim zakašnjenjem u odnosu na naprednije susjede) i kod nas popularni, barem u školama i radionicama tehničke kulture. Pa ipak, starije generacije stručnjaka – pa i inženjera i inovatora, da ne govorimo o drugima, o tome često ne znaju ništa, pa je vrijeme za cjeložvotno učenje.

ARDUINO - PROGRAMIRANJE

Napomena:

Ako ste potpuni početnik, dobro će biti da prije ovog tutoriala proučite osnove dostupne (engleski) na: (https://www.arduino.cc/en/Tutorial/Foundations#basics). Upišite tu web-adresu u adresnu traku, ako link ne radi sa ovog teksta. Osnove sadrže nekoliko cjelina, nažalost samo engleski. U uvodu https://www.arduino.cc/en/Guide/Introduction objašnjava što je Arduino i za što se koristi. Na istoj web-adresi umjesto /Introduction upišite /BoardAnatomy za upoznavanje anatomije razvojnih ploča na kojima ćete razvijati svoje projekte, a /Environment objašnjava tzv. IDE softversko okruženje koje koristi Arduino. O osnovama kodiranja, tj. pisanja programa za Arduino govori uputa https://www.arduino.cc/en/Tutorial/Sketch.
Skraćeni prijevod prikaza osnovnih sastavnica mikrokontrolera, tj. vrsta nožica (/DigitalPins umjesto /Sketch u gornjem linku, /AnalogInputPins i /PWM) uklopljen je na početku ovih uputa, a za vrste raspoloživih memorija mjesto /Sketch u zadnjem linku upišite /Memory .

Neki domaći tutorijali, većinom za pojedine specijalnosti mogu se naći na Internetu uglavnom za hrvatsku verziju Arduina, CRODUINO.

Rijeka, siječanj 2019.

ARDUINO - PROGRAMIRANJE

Arduino (ili hrvatska verzija – Croduino) - "instant" razvojno-eksperimentalna platforma dostupna je danas u prodaji u jednostavnijim i složenijim verzijama s bogatom ponudom ugradbenih elemenata, uputama i besplatnim programskim alatima, pa su ostvareni svi uvjeti za cjeloživotno učenje. Kad ga već koriste djeca i hobisti, vrijeme je da ih upoznaju i stručnjaci i inovatori, kojima Arduino otvara široko polje kreativnosti, tim prije, što je uz pomoć ovog hardvera i softvera moguće razviti i ozbiljnije inovacijske projekte računalno upravljanih uređaja ili instrumenata.
Hardverski paket Arduina sadrži odabranu verziju Arduino mikroprocesora s potrebnim sklopovima za povezivanje s računalom, oscilatorom za generiranje takta, modemom za bežičnu vezu i drugim osnovnim sklopovima, ovisno o odabranoj verziji. Verzije idu od najjeftinije početničke verzije Uno, do skupljih i moćnijih verzija s kojima je moguće složiti i upravljati primjerice i 3D printer.
U sve verzije je uključena i razvojna ploča s nekoliko osnovnih elektroničkih elemenata za sastavljanje pokaznih primjera uključenih u isporuku. Ostali elementi iz galerije nude se On-Line.

Učenje se može početi i bez izdataka, na svakom Windows ili Linux baziranom računalu, slažući uređaje na virtualnoj razvojnoj ploči prikazanoj na ekranu računala, koristeći neki od Arduino-simulatora, primjerice Fritzing koga se može besplatno preuzeti na ***http://fritzing.org/download/***

Simulator može prikazati:

- *razvojnu ploču (breadboard) sa galerijom elektroničkih elemenata (parts) koji se na nju mogu slagati, a po potrebi se mogu čak i kreirati, ili*
- *šematski prikaz sklopa koga kreiramo (shematic), ili*
- *tiskanu pločicu (PCB – Printed circuit board) s potrebnom skicom za izradu. Izradu tiskane pločice izvest će neki od za to specijaliziranih servisa. Jedan od njih je i Fritzing- Fab.*

Fritzing automatski generira shemu i tiskanu pločicu na temelju sklopa sastavljenog na razvojnoj ploči u pogledu Breadboard, u kome prikaz sliči fizičkoj izvedbi stvarnog uređaja/projekta. Svaka promjena u bilo kojem od ta tri prikaza, automatski se prenosi u ostala dva.

Uz simulator na računalo treba instalirati i programski paket Arduino preuzet sa https://download.freedownloadmanager.org/Windows-PC/Arduino/FREE-1.8.6.html (ili eventualno hrvatsku verziju Croduino – u tom slučaju treba instalirati i pripadajuće drajvere).

Fritzing je opremljen detaljnim HELP-om, tj. objašnjenjima (engleski) - osim programiranja - za izvođenje svih radnji s primjerima pokaznih projekata.
Budući da je bez upravljanja posloženi hardver samo mrtva skalamerija, Fritzing uz navedena tri pogleda uključuje i prozor CODE za pisanje upravljačkog programa. U pokaznim projektima, kodovi su već upisani u okviru Code. U vlastitim projektima, upravljanje će programirati, tj. kodove u prozor Code upisati korisnik – autor projekta, a to ćete biti Vi. Tome treba pomoći ovaj pregled programskih elemenata i naredbi. U okviru Code treba odabrati platformu (u našem slučaju to je Arduino) i verziju Arduino-hardvera koga smo nabavili ili namjeravamo nabaviti, te port preko koga se Arduino povezuje na računalo. Ovaj tekst je prerađeni hrvatski sažetak predzadnje verzije 1-8-7 iz 2018.g. Arduinovih pregleda naredbi, varijabli i funkcija Language reference. Nije ažuriran po najnovijoj verziji 1-8-8 s početka 2019.g..
Izvorna objašnjenja (engl.) mogu se naći na web-adresi: **https://www.arduino.cc/en/Main/Donate** *. Tko ne želi donirati iznos za razvoj Arduina, na toj stranici treba odabrati opciju JUST DOWNLOAD, Gornji link treba ukopirati u adresnu traku Internet-pretraživača, ako se ne otvara na klik u ovom tekstu i preuzeti detaljnije upute, koje ćete spremiti na disk za kasniju uporabu.*

Kad su elementi složeni na razvojnoj ploči i veze na priključke ploče povučene sa nožica elemenata (mišem), te napisan kod, odnosno upravljački program, treba ga aktivirati klikom na dugme Upload u donjem desnom uglu okvira Code. Time se pokreće simulacija, pa funkcioniranje sklopa možemo pratiti u pogledu breadboard.
Sustav je opremljen i dijagnostičkim alatom, te bilježi povijest izmjena s mogućnošću povratka na prijašnja stanja projekta.

S obzirom da se u sustavu pomoći Fritzing-a mogu naći detaljna objašnjenja o svim hardverskim sastavnicama simulatora i njegovom korištenju, u nastavku se ograničavamo na sažetiji prikaz Arduino programskih elemenata (varijabli i funkcija) te naredbi, a nešto detaljniji engleski original može se preuzeti kako smo naveli, na https://www.arduino.cc/en/Main/Donate .

SASTAVNICE MIKROKONTROLERA

Arduino razvojno-eksperimentana platforma u hardverskom dijelu sadrži neku od brojnih verzija razvojne ploče (Breadboard), temeljene na ugrađenom mikrokontroleru – mikrointegriranom čipu, najčešće neku verziju ATMEGA čipa. Nećemo se baviti onim ugrađenim skopovima koji rade automatski bez naše intervencije, poput unutarnje građe čipa, generatora takta, sklopova za vezu s računalom, napajanja i dr.
Podatke i naloge s mikrokontrolerom koji ih obrađuje razmijenjujemo slanjem na, ili očitavanjem sa nožica mikrokontrolera (pin-ovi), uz pomoć naloga uklopljenih u program (Sketch) za upravljanje uređajima ili instrumentima ili elektroničkim elementima složenim ili priključenim na razvojnu ploču.

DIGITAL PINS

*Nožice (pins) Arduina se mogu konfigurirati za ulaz (*INPUT*) ili izlaz (*OUTPUT*) informacija. Iako se ovaj opis odnosi na digitalne pinove, većina Atmeginih analognih nožica može se konfigurirati i koristiti na isti način.*

INPUT

*Arduino (Atmega) nožice su pretpostavljeno konfigurirane kao ulazne (*INPUT*), pa ih ne treba posebno konfigurirati ako se koriste za ulaz.*
*Za Arduino (Atmega) pinove konfiguirane nalogom pinMode() kao ulazne (*INPUT*), kaže se da su u stanju "visoke impendance". One imaju vrlo male zahtjeve na strujni krug koji vrši očitavanje, jer su im otpori ekvivalentni serijski vezanom otporniku od 100 megaoma ispred nožice. To ih čini prikladnima za očitanje raznih senzora (npr. osjetnika dodira, LED foto-diode, analognih senzora sa shemom poput RCTime i sl.).*
*To međutim također znači, da će ove nožice konfigurirane sa pinMode(pin,*INPUT*) koje nisu nikamo spojene (ili su na nikamo spojenoj žici) mijenjati stanje po slučajnim vrijednostima uzrokovanim "elektronskom bukom" iz okoline ili kapacitivnom spregom sa susjednim nožicama.*

PULLUP OTPORNIK SA ULAZNOM NOŽICOM

Često je korisno postaviti ulazni pin na poznato stanje ako nema ulaznog signala. To se može postići dodavanjem pullup otpornika vezanog na +5 V ili vezanjem ulazne nožice preko pulldown otpornika na masu (0 V). Otpornik od 10 kΩ je dobra vrijednost za pullup ili pulldown otpornik.

SVOJSTVA NOŽICE KONFIGURIRANE KAO INPUT_PULLUP

Atmega čip ima ugrađen pullup otpornik od 20k kome se može pristupiti postavom pinMode() na INPUT_PULLUP*. Takva postava obrče ponašanje* INPUT *režima, tako da* HIGH *označava isklju**č**en, a* LOW *uklju**č**en senzor vezan na nožicu.*
Vrijednost tog otpornika zavisi o korištenom mikrokontroleru. Na većini AVR ploča je od 20 do 50 kΩ, a na Arduinu Due između 50 i 150 kΩ. Točna vrijednost je na podatkovnoj tablici ploče.
Senzor na nožici konfiguriranoj sa INPUT_PULLUP *drugim krajem se spaja na masu (0 V). Otvoreno jednostavno tipkalo se na nožici čita kao* HIGH*, a pritisnuto, tj. zatvoreno kao* LOW*.*
Pullup otpornici isporučuju dovoljno struje za prigušeno svjetlo LED-diode na ulaznoj nožici.
*Pullup otpornike kontroliraju isti registri (interne pozicije u memoriji čipa), koji kontroliraju stanje nožice (*HIGH *ili* LOW*). Zato ulazni pin (*INPUT*) s uključenim pullup otporikom pokazuje* HIGH *ako pin prebacimo s pinMode() na* OUTPUT*. To funkcionira i u drugom smjeru, tj. izlazna nožica u stanju* HIGH *imat će priključen pullup otpornik ako se s pinMode() konfigurira na ulaznu.*

Prije Arduina 1.0.1 interni pull-up se mogao konfigurirati na sliedeći način:
pinMode(pin, INPUT*); // postavlja nožicu kao ulaznu*
digitalWrite(pin, HIGH*); // uključuje pullup otpornik*

Napomena: Digitalni pin 13 je teže koristiti kao ulazni nego druge, jer je na većini ploča na njega tvornički vezan na ploči ulemljen LED s otpornikom. U slučaju uključenja u čip ugrađenog 20k otpornika, taj će raspolagati sa samo 1,7 a ne 5 V, jer će on-board LED s otpornikom na ploči srušiti napon nožice te će ona uvijek pokazivati LOW*. Zato ako se pin 13 mora koristiti kao digitalni ulaz, pinMode() treba postaviti na* INPUT *i koristiti vanjski pullup otpornik.*

SVOJSTVA NOŽICA KONFIGURIRANIH ZA IZLAZ (OUTPUT) *vidi u opisu OUTPUT (str.14).*
Izlazne nožice mogu isporučiti struje do 40 mA priključenim elementima, a jače ih mogu uništiti.

PROGRAMIRANJE

(OPISI I SVOJSTVA FUNKCIJA, KOMANDI I VARIJABLI)

Arduino koristi prilagođenu varijantu računalnog jezika CC ++ , gdje su sadržane tri grupe programskih elemenata i naredbi:

- *strukturne naredbe (sadrže elemente Arduino C++ koda)*
- *vrijednosti (varijable i konstante)*
- *funkcije (kontroliraju razvojnu ploču i izračune)*

Pa, idemo redom po navedenim grupama i njihovim podgrupama. Klik na komandu iz popisa premješta Vas na opis elemenata nizanih približno abecednim redosljedom. U cilju lakšeg praćenja stranih tutoriala i izvornih opisa koristimo izvorne (engleske) nazive podgrupa. Tko se do sada zanimao za bilo kakvo elementarno programiranje, dio nabrojanih komandi prepoznat će i u ovom popisu (matematski i poredbeni operatori, strukturne komande i dr, sa uobičajenom ili nešto izmijenjenom sintaksom).

STRUKTURNE KOMANDE *- elementi Arduino CC++ koda)*

Sketch (skica)
loop()
setup()

Control Structure
break - prekid
continue - nastavi
do...while – izvrši ... ako je
else – u protivnom
for - od
goto - idi na
if - ako
if ... else – ako ... u protivnom
return – vrati (se na)
switch ... case – uključi u slučaju
while – ako je

Further Syntax
#define (define)
#include (include)
/ */ (block comment)*
// (single line comment)
; (semicolon)
{} (curly braces)

Arithmetic Operators
% (remainder) – ostatak)
** (multiplication) – množenje*
+ (addition) – pribrajanje
- (subtraction) – oduzimanje
/ (divizion) – dijeljenje
= (assigment operator) postavi

Comparison Operators
!= (not egal to) – nejednako
< (less than) – manje od
<= (less than or egal to)
== (egal to) – jednako
> (greather than) – veće od
>= (greather than or egal to)

Booleqan Operators
! (logical not) – logička negacija
&& (logical end) – logički dodat
|| (logical or) – logički "ili"

Pointer Access Operators
& (reference operator)
** (dereference operator)*

Bitwise operators
& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Compound operators
&= (compound bitwise and)
**= (compound multiplication)*
++ (increment) – porast
+= (compound addition)
-- (decrement) – smanjenje
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)
%= (compound modulo)

VARIABLE *(Arduino vrste podataka i konstanti)*

Constants
Floating Point Constants
Integer Constants
HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTN
true | false

Data Types
String()
array
bool
boolean
byte
char
double
float
int
long
short
string
unsigned char
unsigned int
unsigned long
void
word

Conversion
byte()
char()
float()
int()
long()
word()

Variable Scope & Qualifiers
const
variable scope
static
volatile

Utilities
PROGMEM
sizeof()

FUNKCIJE *(kontrola Arduino razvojne ploče i izračuni)*

Digital I/O
DigitalRead()
digital Write()
pinMode()

Analog I/O
analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family
analogReadResolution()
analogWriteResolution()

Random Numbers
random()
randomSeed()

Advaced I/O
noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time
delay()
delayMicroseconds()
micros()
milis()

Characters
isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Math
abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry
cos()
sin()
tan()

Inerrupts
interrupts()
noInterrupts()

Bits and Bytes
bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

Communications
Serial
stream

External Interrupts
attachInterrupt()
detachInterrupt()

USB
Keyboard
Mouse

Opisi komandi, vrste varijabli i funkcija u nastavku su navedeni su približno abecednim redosljedom. No prije toga, za početnike kratka napomena o načinu pisanja koda u Arduino C++ jeziku:
Kao i kod drugih jezika, kod se piše u programskim redcima – linijama. U načelu, svaka linija sadrži jednu naredbu, no to nije uvijek obavezno. Svaka naredba međutim (uz neke izuzetke) obavezno završava znakom ; Izostanak te završne oznake naredbe uzrokovao bi programsku grešku.
Često se počinje deklariranjem (može se reći imenovanjem) varijabli koje će se uključiti u kod, navođenjem vrste varijable i njenog imena iza toga, npr:

```
int vrijedSenz;     // gdje int deklarira cjelobrojnu varijablu (integer), koja se zove "vrijedSenz".
/* Imena varijabli su pri tome proizvoljna. Usput, desni dio gornjeg retka iza znakova // je
objašnjenje – uputa, primjedba programera. Takve tekstove zovemo komentarima, koji ne utječu
na izvođenje ili ponašanje programa, tj. procesor koji obrađuje kod ih ignorira. Komentar u jednom
retku (jednolinijski) se počinje znakom  //  i završava na kraju retka. Komentar u više redaka (blok
komentar) se označava ovako kao ovaj odsječak i obavezno mora imati početnu i završnu oznaku
kao ovaj odsječak. Izostanak završne oznake blok-komentara proizvest će greške u programu. */
```

Program (ili kôd, kako se češće kaže) sadrži pojedine cjeline – sekvence. One obično počinju nekom naredbom, možda i uvjetom (koji najčešće sadrži izraze s poredbenim ili drugim operatorima) a iza toga unutar vitičastih zagrada { } slijedi naredba ili niz naredbi. Svaka naredba kako smo rekli završava znakom ; a iza otvaranja vitičaste zagrade (i naredbe ili nizanja naredbi) mora slijediti zatvorena vitica, u protivnom program prijavljuje grešku.

Saberemo li rečeno, neka programska sevenca u načelu izgleda ovako:

```
deklariranje varijabli;
naredba ili funkcija;
{
naredba1;
naredba2;
. . . .
naredba n;
}
```

Često se usvaja način pisanja kao gore, međutim u načelu nije bitno da li su vitičaste zagrade pisane na početku ili na kraju retka, bitno je da sekvencu moraju okruživati obje (otvorena i zatvorena) i da svaka naredba treba završavati znakom ;
Postoje izuzetci, no oni će biti naznačeni u opisu naredbi kod kojih se javljaju.

Uz deklaracije, varijable i naredbe u kodu će se naći i funkcije i izrazi, sačinjeni uz pomoć navedenih raznovrsnih (aritmetskih, poredbenih i drugih) operatora, koji su opisani u nastavku. Treba ih dobro proučiti, jer se neki od njih (npr =) značajno razlikuju od uobičajene namjene u aritmetici, a mnogi će početnicima biti posve nepoznati.

Možda je najvažnija sposobnost računala (ili mikroprocesora) sposobnost "grananja programa", odnosno odlučivanja, tj. usmjeravanja tijeka izvođenja zavisno od nekog uvjeta, rezultata, ulaznog podatka i sl. Ova se sposobnost realizira uz pomoć naredbi if, if...else, while, do...while, (ako, ako…u protivnom, čini…dok) koje se svode na izvođenje jedne naredbe ili grupe naredbi ili programske grane ako je neki uvjet ispunjen, a druge ako to nije slučaj. No, o tome detaljnije u opisu tih naredbi.

Na kraju još napomena, da su u nastavku u naredbama i funkcijama, tj. sintaksi (načinu pisanja) masnim znakovima otisnuti oni elementi koji doslovno moraju biti prisutni u naredbi odnosno sintaksi, a običnim znacima onaj dio naredbe ili sintakse koji zavisi od konteksta koda (to su uglavnom parametri naredbi i funkcija), koji je dakle izmjenjiv od slučaja do slučaja. Uz ovaj pregled, lakše ćete pratiti nastavak i primjere kodova za Arduino. Takvi primjeri nisu navedeni, jer ih ima dovoljno uz sve verzije Arduina, kao i u dostupnim izvorima. Svakako ih treba proučiti. Ovaj sažetak s navedenim opisima treba Vam pomoći da ih lakše razumijete ili napišete svoj prvi kod.

SKUPNI PREGLED KOMANDI

Zaglavlje kolona nije uvijek striktno poštivano, neke kolone su po potrebi spojene zbog sažetijeg tabelarnog prikaza.

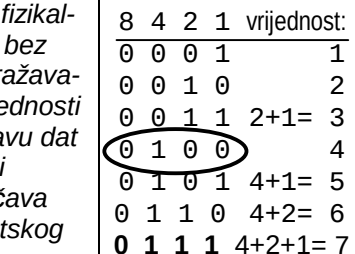
KOMANDA	OPIS	PRIMJEDBA	sintaksa	GRUPA podgrupa

Arithmetic Operators +
+
-
*
/
	Ovi aritmetski operatori (osim =) izračunavaju zbroj, razliku, umnožak ili rezultat dijeljenja dviju vrjednosti u uobičajenom matematskom smislu. Međutim, rezultat ovisi i o vrsti operanda, Tako primjerice dijeleći cjelobrojne operande 9/4 dobivamo (cjelobrojni) rezultat 2, različit od računskog	Tip varijable rezultata treba biti u stanju prihvatiti rezultat operacije. Ako su operandi različitog tipa, "širi" tip se koristi za kalkulaciju. Ako je jedan od operanada s pomič. zarezom ili double, koristit će se izračun s pomičnim zarezom	Primjeri sintakse:
			y = y + 3;
			x = x – 7;
			i = j * 6;
			r = r / 5;
				STRUCTURE
				Arithmetic operators

	• integer konstante su pretpostavljeno **int** tipa, pa neke kalkulacije s njima mogu završiti u overflow (npr. 60*1000 će dati negativan rezultat) • Koristi varijablu rezultata dovoljne velične za prihvat najvećeg mogućeg rezultata kalkulacije • istraži u kom trenutku tvoja varijabla može biti "probijena" i što se može desiti u drugom smjeru – npr. (0 – 1) ili (0 –32768) • za računice koje zahtijevaju frakcije, koristi float varijable, ali to nosi i nedostatke: šire veličine i sporo izračunavanje u procesoru • koristi cast operator, tj. (int)myFloat za konvertiranje jednog tipa varijable u drugi "u letu".			

= Assigment operator
(nesmije se zamijeniti sa ==) !
	Operator dodjele – varijabli ili izrazu ispred znaka jednakosti dodijejuje (postavlja na) vrijednost iza znaka jednakosti. Lijeva varijabla mora moći prihvatiti dodijeljenu vrijednost, u protivnom će postavljena vrijednost biti pogrešna. **Operator dodjele nema jednaku ulogu kao u algebri, tj. ne izračunava vrijednost lijeve strane izraza, nego varjabli ili izrazu s lijeva dodijeljuje (tj. postavlja je na) desnu vrijednost.**	Primjer:
		int ocitanje; //deklarira se (imenuje) varijabla "ocitanje"
		ocitanje = analogRead(0);
		/* (digitalizirana) vrijednost napona na analognoj nožici (pin) 0 sprema se kao vrijednost očitanja*/
		Objašnjenje iznad ujedno je primjer za način pisanja višeredne primjedbe između **/*** i ***/** u sklopu programa. Takva primjedba ne utječe na rezultat ili tijek izvršavanja programa, jer je program ignorira.
				STRUCTURE
				Arithmetic Operators

Za razumijevanje slijedećih, Bitwise operatora, treba znati pojam digitalnog brojevnog sustava. To je sustav sa samo dvije znamenke, i to **0** *i* **1***. U fizikalnom smislu u digitalnoj tehnici* **1** *označava stanje pod naponom u promatranoj točki nekog uređaja (u engleskom se bilježi i kao* HIGH*), a* **0** *stanje bez napona (*LOW*). Ta, digitalna "cifra" se zove* ***bit****, dakle razlikuje se tzv. jedinični bit (***1***) i nulti bit (***0***). Kao što se vrijednosti u dekadskom sustavu izražavaju zbrojem dekadskih jedinica, (jedinice, desetice, stotice. . .) tako se i u binarnom sustavu prikazuju zbrojem binarnih jedinica – bitova, čije su vrijednosti 1, 2, 4, 8 itd (2^n) koje zajedno čine kombinaciju koja se zove* ***bajt*** *(zaokruženo na skici). Primjer prikaza nekoliko vrijednosti u četverobitnom sustavu dat je na desnoj skici. Zbroj vrijednosti svih jediničnih bitova u bajtu prikazan je u desnoj koloni. U suvremenim računalima se koriste (16), 32 i 64 bitni bajtovi. Korištenjem npr. 8-bitnog sustava, bajtom se može definirati 255 alfaumeričkih znakova. U fizičkoj interpretaciji, jedinični bit načelno označava prisustvo indikativnog elementa uređaja (napona na nožici elektroničkog elementa, minijaturnog magnetskog ili laserskog traga na disku ili magnetskog polja u memoriji računala, itd), a nulti bit otsustvo indikativnog elementa. U Arduinu, bitovima u bajtu prikazuju se također stanja napona na nožicama mikroprocesora. Tako se primjerice stanje nožce pod naponom označava kao* HIGH *(visoko), a nožice bez napona kao* LOW *(nisko).*



Bitwise operators

& bitwise and

| bitwise or

^ bitwise xor

*u C++ jeziku Bitwise operatori kalkuliraju na nivou bitova. U digitalnom obliku brojevi se izražavaju uz pomoć jediničnih i nultih bitova **0** i **1** (vidi primjer.. Ovi operatori omogućuju rješavanje čestih zadataka u programiranju. Detalje o njima (engleski) možete naći ovdje. U Arduinu, & i | se često koriste za kontrolu stanja napona na nožicama mikrokontrolera postupkom Read-Modify-Write (Očitaj-promijeni-upiši). Svaka od osam nožica (0,1,2,3,4,5,6,7) može **imati napon** (stanje HIGH –visoko tj. 1) ili biti **bez napona** (stanje LOW – nisko tj. 0). Stanje svih osam nožica tzv. PORT–a, tj. "vratiju" se tako može izraziti 8-bitnim brojem u digitalnom obliku, pri čemu svaki bit označava stanje na jednoj od nožica. Operatorima & i | te Compound operatorima ++ (za porast) i -- (smanjenje) programom se mogu mijenjati pojedini ili svi bitovi očitanog ili postavljenog stanja i time zadati novo stanje PORT-a, tj. napona na svim nožicama jednim digit.brojem. Pamtimo da desni pin nosi oznaku 0, (oznake rastu s desna na lijevo)*

Pravilo za primjenu operatora & (bitwise and) je:
Ako su oba ulazna bita na istom mjestu u ulaznim operandima 1 onda je bit rezultata na tom mjestu 1, a u protivnom je 0
U Arduinu, cjelobrojne vrijednosti izražavaju se u 16 bitnom digitalnom obliku, pa se gornje pravilo primjenjuje na svaki od 16 bitova, npr:

```
int a = 92;   // binarno:  0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0   prvi operand
int b = 101;  // binarno:  0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1   drugi operand
                          ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓   Usporedbu po gornjem pravilu vršiti na svakom od 16 bin.mjesta
int c = a & b; // rezultat: 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1   Ovaj rezultat (1000101) u decimalnom sustavu ima vrijednost 68
```

Jednako tako za | pravilo je: Ako je bar jedan (ili su oba) ulazna bita na istom mjestu 1, rezultat na tom bin.mjestu je 1, a u protivnom je 0

Pravilo za primjenu XOR je: Ako su ulazni bitovi na istom binarnom mjestu isti, rezultirajući bit na tom mjestu je 0, a ako su različiti, rezultat je na tom mjestu 1.
Operator ^ se često koristi za invertiranje ciljanog bita iz 1 u 0 ili obratno, u cjelobrojnom izrazu. Primjerice ako je nalogom `DDRD=DDRD | B00100000;` *digital pin 5 (šesti od desna) postavljen kao izlaz (OUTPUT), nalog* `PORTD = PORTD ^ B00100000;` *promijenit će stanje samo pin-a 5*

Primjer:

```
int x = 12;  //binarno 1100
int y = 10;  //binarno 1010
int z = x^y; = 6  tj:  0110
```

STRUCTURE bitwise operators &, |, <

&& (logical AND)
|| (logical OR)
! (NOT)

BOOEAN OPERATORS (koriste se u if naredbi)
Vraća TRUE (istnito) ako su oba uvjeta ispunjena
vraća TRUE ako je jedan od uvjeta istinit
vraća TRUE ako izraz u zagradi nije istinit

Primjeri: (nesmije se zamijeniti && sa &, || sa | ili ! sa ~)

```
if(digitalRead(2)==HIGH && digitalRead(3)==HIGH) { // read switches}
if (x>0) || y>0) { . . . }     // vraća TRUE ako je x ili y veći od nule
if (!x) { . . .}  // vraća TRUE ako izraz sa x neistinit (npr ako je x=0)
```

STRUCTURE Boolean Operators

Compound operators

+= pribroji
-= oduzmi
*= množi sa
/= dijeli sa
%= ostatak

Ove (compound) operacije nad nekom variablom x vrše uobičajene matemaske operacije s drugom varijablom ili konstantom, kako je prikazano na sintaksi (vidi i primjere).
To je samo kraći način pisanja za desne izraze u predzadnjoj koloni. Varijabla x može biti bilo kog tipa, a rezultat varijabla bilo kog tipa ili konstanta.

Primjeri:

	nova vrijednost varijable:	
x = 2;	2	(2 je dodijeljena vrij)
x *= 3;	6	jer je 2*3=6
x -= 3;	3	jer je 6-3=3
x /= 3;	1	jer je 3/3=1
x %=2;	1	jer 1/2=0 a ostatak =1

compound operacija:	ekvivalentna je izrazu:
x+=y;	x= x+y;
x-=y;	x= x -y;
x*=y;	x= x*y;
x/=y;	x= x /y;
x%=y;	x= x%y;

STRUCTURE Compound operators

++ increment
-- decrement

povećava ili smanjuje varijablu za 1
(vidi sintaksu desno i primjere u predzadnjoj koloni)

```
x++   // povećava x za 1 i vraća staru vrijedn. x
++x   // povećava x za 1 i vraća novu vrijedn. x
x--   // smanjuje x za 1 i vraća staru vrijedn. x
--x   // smanjuje x za 1 i vraća novu vrijedn. x
```

```
x = 2;
y = ++x;  // x i y su sada 3
y = x--;  // x je sada 2,a y 3
```

STRUCTURE Compound operators

#define

*daje ime konstanti prilikom kompajliranja, pa se ušteđuje memorijski prostor za to u Arduino čipu. U nekim slučajevima zamjena konstante imenom može uzrokovati greške, pa se preferira koristiti ključnu riječ **const** mjesto **#define***

Iza #define linije ne stavlja se ;
***U protivnom program će prijaviti grešku** (// this is an error).*

#define imeKonstante vrijednost
Primjer:

```
#define ledPin 3
```

STRUCTURE Further Sintax

#include	*umeće vanjske librarie (među njima i one pisane za Arduino) u našu skicu programa (scetch).*	*Iza #include linije ne stavlja se ; U protivnom program će prijaviti kritičnu grešku, kao kod #define.*	*Primjer za sintaksu:* ***#include*** *<avr/pgmspace.h>*	*STRUCTURE Further Sintax*
%	*Ostatak pri dijeljenju cjelobrojnih vrijednosti. Operator je koristan pri održavanju neke varijable u zadanim granicama (npr. veličine matrice).*	*primjeri: x=9%6; rezultat je x=3 jer je 9:6=1 s ostatkom 3 x=10%5 nova vrjednost je x=0 jer je 10:5=2 a ostatak je 0*	*ostatak=djeljenik % djelitelj*	*STRUCTURE Arithmetic Operators*
*& (reference) * (dereference)*	*Pointer operatori spadaju u složeniji alat C jezika možda teži za početnike. Veliku većinu Arduino skica moguće je napisati bez njih, pa nisu objašnjeni ni u izvorniku po kome je pisan ovaj sažetak. Ipak, za manipuliranje nekim strukturama podataka mogu pojednostaviti kod, pa ih je u naprednijoj fazi dobro znati koristiti.*			*STRUCTURE Pointer access operators*
*/ * * /*	*Blok komentara (višelinijski komentar). Komentari su upute ili primjedbe programera koje program ignorira pri izvođenju. Ne troše prostor u čipu.*	*Ne zaboravite zaključiti, tj. označiti kraj višelinijskog komentara oznakom * /*	*/ * ovo je višelinijski komentar * /*	*STRUCTURE Further Sintax*
//	*Jednolinijski komentar sadrži uputu ili primjedbu i sl. programera pisanu u jednom retku. Program ignorira kometar pri izvođenju. Ne troši memoriju.*	*Komentar je sve iza // do kraja retka (nema završne oznake)*	***//*** *ovo je jednolinijski komentar*	*STRUCTURE Further Sintax*
;	*Točka-zarezom se zaključuje naredba. Ako naredba nije zaključena sa ; kompajler će prijaviti grešku u programu.*	*U slučaju prijave nerazjašnjene greške, prvo treba provjeriti nedostaje li koji ; ispred mjesta greške.*		*STRUCTURE Further Sintax*
{ }	*Vitičaste zagrade su važan simbol u C progr. jeziku Iza {i sadržaja programske iteracije mora slijediti} Nije važno da li su zagrade pisane na početku, kraju ili u novom retku. Sadržaj je pri tome u pravilu naredba ili grupa naredbi koje slijede iza fukcije kojom su inicirane.*	*Zaboravljena zaključna zagrada } može proizvesti programsku grešku koja se teško otkriva. Postoje i druge primjene vitičaste zagrade (mogu zamijeniti return u funkcijama, endif i next u petljama)*	*funkcija { Naredba ili grupa naredbi (svaka zaključena sa ;) }*	*STRUCTURE Further Sintax*
abs()	*Izračunava apsolutnu (tj. pozitivnu) vrijednost broja ili iraza u zagradi*	*Izraz u zagradi ne smije sadržavati druge funkcije, jer takav izraz može dati pogrešan rezultat*	*Primjer: umjesto abs(a++); koristi: abs(a); a++;*	*FUNCTIONS Math*
analogRead() *FUNCTIONS Digital I/O*	*Očitava vrijednost na specificiranoj analognoj nožici. Arduino razvojne ploče imaju 6-16 kanala sa 8 ili 16 nožica (pin) -zavisno o tipu ploče- i 10 bitni analogno-digitalni konverter. Mogu ulazne napone od 0 do 5 V pretvoriti u digitalne cjelobrojne vrijednosti od 0 do 1023. Slijedi da preciznost digitalnog iskazivanja vrijednosti iznosi 4.9 mV Ulazni raspon i preciznost mogu se mijenjati korištenjem* ***analogReference()*** *naloga*	*Ako je analogni ulaz "u zraku", tj. nije nikamo spojen, vrijednost funkcije analogReference() će skakutati zavisno o mnogim faktorima (npr. o drugim analognim ulazima, blizini ruke do ploče i dr) Sintaksa: analogRead(pin) a rezultat se prikazuje u obliku: int(vrijednost zmeđu 0 i 1023)*	***int analogPin = 3;*** *//potenciometar je spojen na . // pin 3, a vanjska LED dioda na masu i +5V* ***int val = 0;*** *// 0 je spremljena kao value read* ***void setup() b {*** ***serial.begin(****9600****);*** *// postavlja počet.serial* ***}*** ***void loop() {*** ***val = analogRead(analogPin);*** *// očitava pin* ***Serial.printin(val);*** *// prepravlja poč.vrijednost* ***{***	

<table>
<tr>
<td>analogReadResolution()</td>
<td>Naredba postavlja vrjednost (u bit-ima) rezultata naloga analogRead(). Zadana (default) rezolucija je 10 bitna (vraća vrijednosti između 0-1023) zbog kompatibilnosti unatrag sa AVR pločama.
analogReadResolution() je naime proširenje Arduino 2 i 10 naloga Analog API.Ove ploče imaju 12-bitne ADC sposobnosti, kojima se može pristupiti promjenom rezolucije rezultata s 10 na 12, pa će se onda oni prikazati između 0 i 4095.
Sintaksa:
analogReadResolution(rezolucija u bitovima između 1 i 32)
(Vidi napomenu desno) Primjer (engl.) vidi u izvorniku.</td>
<td colspan="2">Napomena:
Ako vrijednost analogReadResolution() postavimo na veću rezoluciju od dostupne, Arduino će koristiti najvišu razlučivost a višak bitova zamijenit će s nulama.
Primjerice, korištenje Due ili Zero s analogReadResolution (16) dati će približni 16-bitni broj s prvih 12 bita koji sadrže pravo ADC očitanje, a zadnja 4 bita ispunit će nulama.
Ako vrijednost analogReadResolution () postavimo na vrijednost nižu od mogućnosti razvojne ploče, dodatni najmanje bitni bitovi koji se čitaju iz ADC-a će se odbaciti.
Korištenjem 16 bitne, ili bilo koje razlučivosti koja je veća od stvarnih hardverskih mogućnosti, omogućuje se pisanje skica za automatsku obradu na budućim uređajima s ADC-om visoke razlučivosti kada oni postanu dostupni, bez ispravljanja koda.</td>
<td>FUNCTIONS
Zero,Due & MKR Family</td>
</tr>
<tr>
<td>analogReference()</td>
<td>Postavlja zadani (default) referentni napon analognog ulaza (to je najviši dopušteni napon ulaznog raspona) za neku od slijedećih opcija:
• DEFAULT: 5 ili 3.3 V zavisno o Arduino ploči
• INTERNAL: 1.1 ILI 2.56 V zavisno o Arduino ploči
• INTERNAL1V1: 1.1 V samo za Arduino Mega
• INTERNAL2V56: 2.56 V samo za Arduno Mega
• EXTERNAL: za napon na AREF nožici (0-5V)</td>
<td colspan="2">Sintaksa:
analogReference(opcija) (nalog ne vraća nikakav rezultat)

POZOR !
Prije korištenja AREF pin-a za vanjsku referencu (dakle i prije naloga analogRead()) analogReference() treba postaviti na EXTERNAL, u protivnom je moguće uništenje mikrokontrolera razvojne ploče. Ne koristi ništa ispod 0 ni preko 5V za vanjski ref.napon na AREF pinu

Alternativno, vanjski referentni napon se može na AREF pin vezati preko 5kΩ otpornika, što dopušta prekapčanje između vanjskog i internog ref. napona. No,taj otpornik će smanjiti napon koji će se uzeti kao referenca 32/(32+5) puta , jer se veže u red sa internim 32K otpornikom na AREF nožici.</td>
<td>FUNCTIONS
Analog I/O</td>
</tr>
<tr>
<td>analogWrite()</td>
<td>Upisuje analognu vrijednost (PWM val) na ciljanu nožicu. Može se koristiti za različito osvjetljenje LED diodom ili za pogon motora različitim brzinama. Nakon naloga analogWrite(), odabrani pin generira neprekinuti kvadratni signal (val) specificiranog radnog ciklusa do sljedećeg naloga analogWrite() ili digitalRead() ili digitalWrite() na istom pinu. Frekvencija PWM signala na većini pin-ova je približno 490 Hz, a za najčešće Arduino ploče vidi tabelu desno..
Ne ne treba naložiti pinMode() prije postave nožica za izlaz uz pomoć naloga analogWrite().

Funkcija analogWrite() nema nikakve veze s analognim pinovima ili funkcijom analogRead().</td>
<td>Frekvencija kvadratnog vala PWM

Razvojna ploča | Frekvencija Hz | Nožice
UNO1 i slične | 980 | 5 i 6
Leonardo | 980 | 9,10,11
ATmega168 i 328 | 3, 5, 6, 9, 10, 11
Mega | 2-13 i 44-46
starije ATmega8 | 9, 10, 11
Arduino Due2-13 + DAC0 i DAC1

(DAC su digitalno-analogni pretvornici koji djeluju kao pravi analogni izlazi).</td>
<td>Sintaksa:
analogWrite(pin,vrijednost)
Vrjednost podrazumijeva radni ciklus između 0 i 255. Pri tome je:
0 isključeno
255 stalno uključeno
Napomena: PWM izlaz na pin-u 5 i 6 ima duži ciklus od očekivanog [jer koristi isti tajmer sa milis() i delay()], pa vrijednost 0 možda neće u potpunosti isključiti izlaz na tim nožicama</td>
<td>FUNCTIONS
Analog I / O</td>
</tr>
</table>

<table>
<tr>
<td>analogWrite()
nastavak

PRIMJER</td>
<td colspan="3">Primjer: Postavlja izlaz na LED-diodu proporcionalno vrijednosti očitanoj sa potenciometra
int ledPin = 9; // priključuje LED na digitalni pin 9
int analogPin = 3; // potenciometar veže na analogni pin 3
void setup()
{
pinMode(ledPin, OUTPUT); // postavlja nožicu LED-dode kao izlaz
}
void loop()
val = analogRead(analogPin); // očitava pin sa analognog ulaza
analogWrite(ledPin, val / 4); // očitanje ide od 0-1023, a analogWrite() od 0-255
}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>analogWriteResolution()</td>
<td>Proširenje Analog API za Arduino Due i Zero
Postavlja rezoluciju analogWrite() funkcije, uz zadanu (default) 8 bitnu vrijednost (tj. između 0 i 255) za omogućenje kompatibilnosti unatrag sa pločama temeljenim na AVR. Hardverske sposbnosti za za Due i Zero ploče vidi desno.
Postavom write rezolucije na 12 (Due) odnosno 10 (kod Zero) omogućuje se puno iskorištenje ugrađenog digitalno-analognog konvertera (DAC) tih razvojnih ploča. I za analogWriteResolution() važe napomene date uz analogReadResolution().</td>
<td>Hardverske sposobnosti ploče Due:
12 nožica, zadano 8-bitni PWM (kao ploče temeljene na AVR), mogu se postaviti na 12-bitnu rezoluciju i:
2 nožice sa 12-bitnim DAC (digital-analognim konverterom)
Zero: 10 nožca, zadano 8-bitni PWM (kao ploče temeljene na AVR), mogu se postaviti na 12-bitnu rezoluciju, i:
1 nožica s 10-bitnim DAC konverterom.</td>
<td>Sinaksa:
analogWriteResolution(bitovi)
Bitnost u zagradi (od 1-32) se odnosi na rezoluciju korištenu u funkciji analogWrite(). U slučaju postave previsoke ili preniske rezolucije, Arduino će koristiti raspoloživu, uz korekciju bitnosti kao kod analogReadResolution()</td>
<td>FUNCTIONS
Zero,Due & MKR Family</td>
</tr>
<tr>
<td>array</td>
<td>Array je skup ili niz varijabli, koje se dohvaćaju uz pomoć broja koga zovemo indeks (broj u uglatoj zagradi). U C-jeziku oni mogu biti komplicirani, ali korištenje jednostavnih nizova relativno je jednostavno.
Skup može deklarirati, tj. kreirati bilo koja od slijedećih naredbi:
int myInst[6]; // neobavezno
int myPins[] = {2,4,8,3,6};
int mySensVals[6] = {2,4,-8,3,2};
char message[6] = "hello";
Array se može kreirati i bez inicijaliziranja myInst</td>
<td colspan="2">int myPins nalaže kreiranje skupova bez zadavanja veličine, pa kompajler sam kreira odgovarajući niz na temelju brojanja elemenata. Inicijaliziranje s dimenzioniranjem niza omogućuje int mySensVals[]
Pri tome za skupove tipa char treba dodati početni element za tzv. null-character, kao što je učinjeno kod zadnje naredbe lijevo. "hello" sadrži 5 znakova, a naredba nosi indeks 6 u uglatim zagradama.
Pristup array-u: Prvi element niza treba biti nulto indeksiran, npr: mySensVals[0] == 2, mySensVals[1] == 4, itd. Tako primjerice u skupu s deset elemenata posljednji element ima indeks 9. Primjerice nalog int myArray[10]={9,3,2,4,3,2,7,8,9,11}; nije valjan i sadržat će slučajne vrijednosti, čitane ili pisane na pogrešna memorijska mjesta, što može srušiti sistem ili uzrokovati programske greške koje se teško otkrivaju. C-kompajler naime za razliku od BASIC-a ili JAVE ne provjerava jeli pristup skupu unutar deklariranih granica, pa treba oprezno deklarirati..</td>
<td>VARIABLE
Data Types</td>
</tr>
<tr>
<td>attachInterrupt()
Digitalne nožice sa sposobnosću prekida</td>
<td>Pojedine nožice (pin) mikrokontrolera, mogu se namijeniti za generiranje prekida, koji su korisni za automatizaciju radnji za rješenje tajminga u programima za mikrokontrolere (npr.kod čitanja okretnih kodera ili praćenja unosa korisnika). Desno su navedeni takvi pin-ovi za razne ploče. Sve varijable koje se mijenjaju unutar attach-funkcija treba deklarirati kao volatile, jer delay() ne radi, a rezultati milis() funkcije neće rasti unutar attach-funkcija (vidi niže sekciju o ISR)</td>
<td colspan="2">Razvojna ploča digit.pinovi
Uno, Nano, Mini i druge 328 ploče 2,3
Mega (uklj.2560 i ADK) 2,3,18,19,20,21
Micro,Leonardo i druge 32u4-bazirane ploče 0,1,2,3,7
Zero svi digitalni pinovi osim 4
Due svi digitalni pinovi</td>
<td>FUNCTIONS
External Interrupts</td>
</tr>
</table>

<table>
<tr>
<td>attachInterrupt()

nastavak</td>
<td>Primjeri korištenja prekida (interrupt): Bilo bi jako zahtjevno napisati kod za bilo kakvu akciju u slučaju prihvata impulsa sa rotirajućeg davača, ili senzora zvuka koji pokušava uhvatiti klik ili foto-prekidača koji primjerice nastoji uhvatiti pad novčića, jer program stalno treba provjeravati senzor, da ne bi propustio neki ulazni impuls. Prekid Interrupt-nalogom u takvim situacijama oslobađa procesor za druge akcije, bez propuštanja ulaznih impulsa.
O rutinama servisa za prekid (Interrupt Service Routines) - ISR
ISR su posebne vrste funkcija, koje imaju određena jedinstvena ograničenja, koja druge funkcije nemaju. Nemogu imati parametre i ne vraćaju nikakav rezultat. Općenito, ISR treba biti što kraći. Ako program (sketch) koristi više ISR-ova, samo se jedan može izvršavati istovremeno, a drugi se aktiviraju kad prvi završi, sukladno sadržanim prioritetima. Nalozi koji zahtijevaju prekid rada kao milis(), delay(), micros(), neće raditi unutar ISR-a. (delayMicroseconds() će međutim raditi normalno, jer ne koristi brojač). Za prijenos podataka između ISR-a i glavnog programa obično se koriste globalne varijable. Da bismo bili sigurni da su te varijable uredno ažurirane, treba ih prijaviti kao volatile. Primjer vidi na slijedećoj stranici.</td>
<td colspan="2">Sintaksa:
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
. ne preporuča se
attachInterrupt(interrupt,ISR,mode); ne preporuča se
attachInterrupt(pin, ISR, mode); (samo Due i Zero).
Parametri:
interrupt broj prekida
ISR nema parametara
mode definira kad se prekid okida:
- LOW – kad je pin bez napona
- CHANGE – kad pin mijenja vrijednost
- RISING – kad pin dobije napon
- FALLING - kad pin izgubi napon
- HIGH - kad je pin pod naponom (Due i Zero)</td>
<td>FUNCTIONS
External Interrupts</td>
</tr>
<tr>
<td>attachInterrupt()
Digitalni pin-ovi sa sposobnošću prekida
. (nastavak)</td>
<td colspan="2">attachInterrupt() – nastavak
Primjer:
int pin = 13;
volatile int state = LOW;
void setup() {
pinMode(pin, OUTPUT);
attachInterrupt(digitalPinToInterrupt(pin), blink, CHANGE);
}
void loop() {
digitalWrite(pin, state);
}
void blink() {
state = !state;</td>
<td>Brojevi prekida
Nožice sposobne za interrupt i njihove oznake razlikuju se kod raznih Arduino ploča. Zbog toga u pravilu treba koristiti digitlPinToInterrupt(pin) umjesto direktnog unosa interrupt-broja u attachInterrupt(), zbog kompatibilnost u slučaju izvođenja koda na drugoj ploči.
Ipak, stariji kodovi često koriste direktne interrupt-brojeve a ti su kod čestih ploča na pinu:
Ploča: int.0 int.1 int.2 int.3 int.4 int.5
Uno,Ethernet 2 3
Mega2560 2 3 21 20 19 18
32u4-bazirani 3 2 0 1 7
Due možeš navesti pin-number na svim nožicama
Zero također sve osim 4 možeš navesti u attachInterrupt</td>
<td>FUNCTIONS
External Interrupts</td>
</tr>
<tr>
<td>bit()</td>
<td>Funkcija izračunava vrijednost specificiranog bit-a (bit 0 is 1, bit 1 is 2, bit 2 is 4, itd.).</td>
<td>Parametar:
n: je bit čija se vrjednost izračunava</td>
<td>Sintaksa:
bit(n)</td>
<td>FUNCTIONS
Bits an Bytes</td>
</tr>
<tr>
<td>bitClear()</td>
<td>Funcija postavlja bit numeričke vrijednosti na 0</td>
<td>x: numerička varijabla koja se "anulira"
n: definira koji se bit "anulira" počevši od 0 za najniže označen (s desna)</td>
<td>bitClear(x,n)</td>
<td>FUNCTIONS
Bits an Bytes</td>
</tr>
<tr>
<td>bitRead()</td>
<td>Funkcija čita bit od broja</td>
<td>x: broj čiji se bit očitava
n: definira koji se bit očitava počevši od 0 za najniže označen (s desna)</td>
<td>bitRead(x,n)</td>
<td>FUNCTIONS
Bits an Bytes</td>
</tr>
</table>

bitSet()	*Funkcija postavlja (upisuje 1 na . .) bit numeričke varijable*	*x: broj čiji se bit setira* *n: definira koji se bit postavlja počevši od* **0** *za najniži (s desna)*	*bitSet(x,n)*	*FUNCTIONS* *Bits an Bytes*
bitWrite()	*Funkcija upisuje bit numeričke varijable*	*x: broj čiji se bit upisuje (pamti)* *n: definira koji se bit upisuje počevši od* **0** *za najniži (s desna)*	*bitWrite(x,n,b)*	*FUNCTIONS* *Bits an Bytes*
highByte()	*Ekstraktira najviši (krajnji lijevi) bajt riječi (ili drugi niži bajt od opsežnijeg tipa podataka)*	*x: vrjednost bilo koje vrste*	*highByte(x)*	*FUNCTIONS* *Bits an Bytes*
lowByte()	*Ekstraktira najniži (krajnji desni) bajt varijable (npr. riječi).*	*x: vrjednost bilo koje vrste*	*lowByte(x)*	*FUNCTIONS* *Bits an Bytes*
word()	*Pretvara vrjednost u* word *vrstu podataka ili kreira* word *iz dva bajta*	*x: vrjednost bilo koje vrste* *h: najviši (lijevi) bajt od word* *l: najniži (desni) bajt od word*	*word(x)* *word(h,l)*	*FUNCTIONS* *Bits an Bytes*
bool	*varijabla sadrži TRUE (istina) ili FALSE (neistina). Boolean varijabla zauzima jedan bajt u memoriji.*	*Primjer:* ***int LEDpin =** 5**;** // LED-svjetleća dioda je spojena na pin 5* ***int switchPin =** 13**;** // trenutni prekidač je na13,a druga strana na masi* ***bool running = false;** // ili nestandardna verzija: **boolean running = false;*** ***void setup() {*** ***pinMode(**LEDpin, **OUTPUT);*** ***pinMode(**switchPin, **INPUT);*** ***digitalWrite(**switchPin, HIGH**);** // uključuje otpornik za pokretanje* ***}*** ***void loop() {*** ***if (digitalRead(switchPin) == LOW)*** ***{** // prekidač je pritisnut - pullup drži pin pod naponom (high) normalno* ***delay(**100**);** // kašnjenje isključenja prekidača* ***running = !running;** // prekapča running varijablu* ***digitalWrite(LEDpin, running)** // signalizira putem LED-diode* ***}*** *}*		*VARIABLE* *Data Types*
boolean	*Nestandardna varijanta bool tipa varijable. Preporuča se korištenje standardne varijable bool.*			
break	*Break služi za izlaz iz petlji iniciranih nalozima for, while ili do…while, ignorirajući uvjete definirane u petlji. Također izlazi iz switch…case naloga.*	*Uobičajeno je unutar petlje definiran neki uvjet (fukcijom if) iza koje može slijediti nalog break;*	*. . . .* *if (uvjet){nalog ako je uvjet ispunjen;* *break;}* *. . . ostatak petlje*	*STRUCTURE* *Control Structure*
byte	*Sprema 8-bitnu vrijednost, u rasponu 0-255*	*Primjer:* **byte** b **= B**10010**;** // "B" je binarni formatizer (B10010 je decimalno 18)		*VARIABLE* *Data Types*
byte()	*konvertira vrijednost u bajtni tip podataka*	*Parametri: x: vrijednost bilo kog tipa* *. bajtovima*	*Sintaksa:* *byte(x)*	*VARIABLE* *Conversion*

char	Vrsta podatka koja zauzima 1 bajt memorije za spremanje vrijednosti znakova. Slovne vrijednosti se upisuju unutar apostrofa (npr: ´A´), a višeslovni stringovi između navodnika ("ABC"). No, i slovne vrijednosti se pamte u brojčanom obliku, koristeći tzv. **ASCII tabelu** brojčanih oznaka za pojedina slova, pri ćemu je nad tim oznakama dopušteno vršiti i arimetičke operacije. Detaljnije vdi: www.arduino.cc/en/Serial/Println.html	Primjer: Oba izraza ispod su ekvivalentna, jer je decimalna oznaka slova A po ASCII tabeli 65 char myChar = 'A'; char myChar = '65'; Također, dopustiv je i izraz: 'A'+1, a rezultat će biti 66, što u ASCII tabeli odgovara slovu B	Detaljno o ASCII standardnim i proširenim kodovma za razna područja Europe, te pretvorbi između pojednih oblika vidi: http://www.orbus.be/pdt/pdt0026.htm	VARIABLE Dana Types
char()	Pretvara vrijednost u char tip podataka	Parametar x je vrijednost bilo kog tipa	Sintaksa: char(x)	VARIABLE Conversion

const	Ključna riječ, koja označava konstantu, odnosno varijabli bilo koje vrste daje "Read only" svojstvo (samo za čitanje), dakle iako se varijabla iza const može koristiti kao svaka druga varijabla (npr. u aritmetičkim operacijama), ne može joj se mijenjati vrijednost. Shodno tome, npr. pokušaj dodjele vrijednosti operatorom = (assign) proizvest će grešku prevodtelja (compiller error) – vidi primjer u predzadnjoj koloni.	Konstante definirane sa const poštuju pravila koja važe za varijable scoping, koje upravljaju drugim varijablama, pa se zbog zamki koje prate nalog #define, preferira korištenje const umjesto #define, koja je neprikladna i za defniranje nizova (arrays). Općenito, preferira se korištenje const umjesto #define.	const float pi = 3.14;
			float x;
			// ....
			x = pi * 2; // prikladno za korištenje const u računanju
			
			pi = 7; // ilegalan pokušaj dodjele (tj mijenjanja) vrijednosti konstante
				VARIABLES
				Variable Scope & Qualifiers

constrain()	Uklapa broj u granice zadanog opsega. Ako je broj manji ili veći od granica raspona, postavlja mu vrijednost na donju ili gornju graničnu vrijednost Primjer: sansVal = constrain(sensVal, 10, 150); // svodi očitanje senzora u zadani raspon 10 -150	Parametri: x: broj bilo kog tipa . a: donja granica opsega . b: gornja granica opsega	Sintaksa: constrain(x, a, b)	FUNKTIONS math
continue	nalog preskače ostatak trenutne iteracije ako uvjet nje ispunjen, u petljama (for, while, do...while) i nastavlja (ispitujući uvjete petlje) sa slijedećim iteracijama.	Uobičajeno unutar petlje stoji neki uvjet (definiran fukcijom if) iza koga slijedi nalog continue;	. . . . if (uvjet) {nalog ako je uvjet ispunjen; continue;} . . . ostatak petlje	STRUCTURE Control Structure
sin() cos() tan()	izračunava trigonometrijske funkcije. Rezultat za sin() i cos() će biti vrijednost između –1 i 1, a za tan() između – beskonačno i + beskonačno. ctg() = 1/tan() (kotangens je 1/tangens kuta)	**Kut u zagradi treba zadati u radijanima** (u **float** tipu vrijednosti) Rezultat se prikazuje kao broj s dvostukom preciznošću (**double**)		FUNKTIONS Trigonomethry

<table>
<tr>
<td>delay()</td>
<td>Pauzira program za vrijeme upisano u zagrade u milisekundama (tip podatka: unsigned long)
Lako je programirati trepčuću LED diodu sa delay() funkcijom, a ona se koristi i za kratke zadatke, kao npr. za otklanjanje smetnji. Funkcija međutim ima i nedostatke: Ni jedno čitanje senzora, matematski izračuni ili manipulacija pin-ovima, ne može se vršiti za trajanja delay() odgode, tj većina aktivnosti se zaustavlja. Stoga iskusniji programeri izbjegavaju koristiti delay() za događaje duže od 10 ms. Alternativu za kontrolu vremena omogućuje funkcija milis() – vidi i primjer ispod nje. Ipak, delay() ne sprječava prekide (interrupts) na Atmegi, jer se komunikacija na RX pinu bilježi, PWM (analogWrte) vrijednosti i stanja pin-a se održavaju, pa interrupts() uredno radi po očekivanju.</td>
<td>Sintaksa: delay(ms)
Primjer:
int ledPin = 13; //LED vezana na digital-pin 13
void setup()
{
pinMode(ledPin, OUTPUT); // postavlja nožicu kao izlaznu
}
void loop()
{
digitalWrite(ledPin, HIGH); // pali LED-diodu
delay(1000); // čekanje 1 sekundu
digitalWrite(ledPin, LOW); // gasi LED-diodu
delay(1000); // čekanje 1 sekundu
}</td>
<td>FUNKTIONS
Time</td>
</tr>
<tr>
<td>delayMicroseconds()</td>
<td>Pauzira program za vrijeme upisano u zagrade u mikrosekundama (milionti dio sekunde)
Sintaksa; delayMicroseconds(µs)
Ttip podatka u zagradi treba biti: unsigned int (pozitivan cjelobrojni oblik – detaljnije vidi na linku).
Trenutno je najduže trajanje pauze po ovoj funkciji 16383. što se može promijeniti u budućim verzijama Arduina. Za duže pauze može se koristiti delay()
Nije garantirano precizno kašnjenje za vrijeme ispod 3 µs
Kod Arduna iznad modela 0018 ne sprječava interrupts()</td>
<td>Primjer: generiranje niza ipulsa s periodom od cca 100 mikrosekundi.
int outPin = 8; // postavlja pin 8 kao digital pin
void setup()
{
pinMode(outPin, OUTPUT); // postavlja pin 8 kao ilazni pin }
void loop()
{
digitalWrite(outPin, HIGH); // aktivira nožicu
delayMicroseconds(50); // pauziranje 50 mikrosekundi
digitalWrite(outPin, LOW); // deaktivira nožicu
delayMicroseconds(50); // pauziranje 50 mikrosekundi
}</td>
<td>FUNTIONS
Tme</td>
</tr>
<tr>
<td>detachInterrupt()
za detalje vidi i attachinterrupt()</td>
<td>Isključuje aktivni prekid (interrupt).
Parametri :
• prekid: broj (oznaka) prekida koji se želi isključiti. Detaljnije vidi: attachinterrupt()
• pin: broj (oznaka) pina koji se isključuje (Arduino Due)</td>
<td>Sintaksa:
detachinterrupt(prekid)
detachInterrupt(digitalPinToInterrupt(pin));
detachinterrupt(pin) (samo Arduino Due i Zero)</td>
<td>FUNKTIONS
External Interrupts</td>
</tr>
<tr>
<td>digitalWrite()</td>
<td colspan="2">Izlazni (OUTPUT) digitalni pin postavlja u stanje HIGH (5 ili 3,3 V zavisno o tipu ploče) ili LOW (0 V, tj. potencijal mase). Nožicu konfigurirati kao izlaznu nalogom pinMode(). Na Ulaznom digital pinu nalog uključuje (HIGH) ili isključuje (LOW) interni pullup-otpor na nožici. Za uključenje pullup-otpornika preporuča se postava pinMode(INPUT-PULLUP)
Sintaksa:
digitalWrite(pin, HIGH ili LOW)
pin je broj (oznaka) pin-a</td>
<td>FUNCTIONS
Dgital I/O</td>
</tr>
</table>

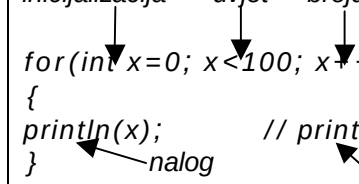
digitalRead()	***Oč**itava stanje (*HIGH *ili* LOW*) sa specificirane digitalne nožice.* *Primjer desno postavlja pin 13 na vrijednost pin-a 7, koji je deklariran kao ulazni. (Sekvenca upravlja LED-diodom zavisno o stanju tipkala)*
	Sintaksa:
	Napomene:
	• *Ako je ugrađena, LED dioda je često priključena na nožicu 13 i uobičajeno nosi naziv ledPin*
	• *Ako nožica nije spojena ni na što, **digitalRead()** može pokazivati HIGH ili LOW po slučajnom redosljedu.*
	• *Analogne ulazne nožice ne mogu se konfigurirati kao digialne (to su A0, A1, . . . itd)*
		*Sintaksa: digitalRead(pin) // pin je broj (oznaka) pin-a u **int** (cjelobrojnom) obliku.*
		Primjer:
		`int ledPin = 13; // LED prikjučen na digital pin 13`
		`int inPin = 7; // tipkalo priključemo na digital pin 7`
		`int val = 0; // varijabla koja se pamti kao očitanje`
		`void setup(){`
		`pinMode(ledPin, OUTPUT); // digital pin 13 postavlja za izlaz`
		`pinMode(inPin, INPUT); // digital pin 7 postavlja kao ulazni`
		`}`
		`void loop(){`
		`val = digitalRead(inPin); // očitava ulaznu nožicu`
		`digitalWrite(ledPin, val); // potavlja LED na vrijednost tipkala`
		`}`
				FUNCTIONS
				Dgital I/O

do...while	*služi za ponavljanje rutine u petlji poput naredbe while, ali je uvjet za ponavljanje na kraju petlje, pa će se ona uvijek izvesti, najmanje jedan puta. Primjer ponavlja radnje iza do dok je x ispod 100*	***do{*** *Primjer* ***delay(**50**);** // čeka stabilizaciju senzora* *x **= readSensors();** //čita senzor* ***}while (**x < 100**):***	*do* *{ //blok naredbi* *}**while** (ispitivanje uvjeta);*	*STRUCTURE* *Control Structure*
double *(vidi i float)*	*su brojevi s pomičnim zarezom (floating point) ali dvostruke preciznosti. Na Uno i drugim na Atmegi temeljenim razvojnim pločama zauzimaju 4 bajta. Na Arduinu Due, **double** brojevi su 8-bajtni, tj. koriste 64 bitnu preciznost.*	*Napomena:* *Korisnici koji prenose kod s varijablama dvostrue preciznosti (double) iz drugih izvora, trebali bi provjeriti da li se izvorna preciznost može aplicirati na Arduino razvojnim pločama temeljenim na Atmega miroprocesoru.*		*VARIABLES* *Constants*
else *(vidi također i if...else)*	*U strukturi if...else odluke, else omgućuje provedbu naloga koji se trebaju izvršiti ako uvjet u zagradi if () nije ispunjen. Među njih se može uključiti i nova if...else naredba, i tako program strukturirati u više nivoa odlučivanja.*	*Svaka provjera uvjeta vodi do slijedeće, ako je uvjet ispunjen, u protivnom, izvršit će se nalozi u njenoj else sekvenci. Else sekvence su uvijek neobavezne*	*if (uvjet1)* *{nalozi ispunjen uvjet1;}* *else if(uvjet2)* *{nalozi ispunjen uvj.1 i 2**;**}* *else* *{nalozi neispunjen uvj.1**;**}*	*STRUCTURE* *Control Structure*

float
 	
Sintaksa:
float** imeVarij **=** vrjed**;
 	
Zadanu "vijed"nost treba dati u float obliku
	je tip varijable-decimalnog broja s tzv. "pomičnim zarezom" (floating-point), moguć u granicama od -3.4028235E+38 do 3.4028235E+38. Pamte se sa 32 bita, tj. u 4 osam-bitna bajta.
	Često se koriste za aproksimaciju analognih vrijednosti, jer omogućuju veću rezoluciju od cjelobrojnih vrjednosti (int).
	*Ako se pišu bez decimalne to**č**ke, Arduino **ć**e ih shvatiti kao int, odnosno prevest će ih u cjelobrojne (integer) – vidi primjer desno.*
	Valja voditi računa o navedenome, da se izbjegnu pogrešne računice u Arduino-programiranju.
		*Imaju samo 6–7 decimalnih znakova (zbirno ispred i iza zareza). Za razliku od drugih platformi, gdje **double** – tip brojeva može ići do 15 znamenkaste preciznosti, kod Arduina **double** i **float** imaju jednaku preciznost.*
		Floating-point brojevi nisu egzaktni i mogu pokazati čudne rezultate pri usporedbi. Takoprmjerice 6.0/3.0 može biti različit od 2.0.
		Obrađuju se znatno sporije od cjelobrojnih, pa ih treba izbjegavati npr. ako se u kakvoj petlji izračunava kakva vrijednost, kritična za tajming.
		Primjeri:
		`x=1; // vrijednosti x i y su pretpostvljeno cjelobrojne (integer)`
		`y=x/2; // Rezultat je 0 jer cjelobrojni iznosi nemogu imati decimala`
		`z = (float)x/2.0; // dat će rezultat 0.5 samo ako se pisalo 2.0 a ne 2`
				VARIABLES
				Data Types

<table>
<tr>
<td>float()</td>
<td>Varijablu bilo kog tipa pretvara u float tip podatka (varjablu - decimalni broj s pomičnim zarezom)</td>
<td>Parametar:
x: je varijabla bilo kog tipa</td>
<td>Sintaksa:
float(x)</td>
<td>VARIABLES
Conversion</td>
</tr>
<tr>
<td>Floating Point Constants</td>
<td>Konstante s pomičnim zarezom se za vrijeme prevođenja (kompajliranja) koda prevode u vrijednosti na koje upućuje izraz kojim su zadane. Mogu biti zadane u decimalnom ili eksponencijalnom obliku. E i e su jednako valjani simboli za eksponencijalni oblik.
Poput cjelobrojnih konstanti (integer constants), i ove s pomičnim zarezom doprinose boljoj čitljivosti koda.</td>
<td colspan="2">Primjeri:
floating-point konstanta predstavlja izraz: čija je vrijednost:
10.0 10 10
2.34E5 2.35*10^5 234000
67e-12 67.0*1/10^12 .000000000067

Sintaksa:
n = .005;</td>
<td>VARIABLES
Constants</td>
</tr>
<tr>
<td>for petlja

(vidi i while)</td>
<td>naredba opetuje naloge unutar vitičastih zagrada iza naredbe. Za odbrojavanje i zaustavljanje petlje uobičajeno se koristi (uzlazni, rjeđe silazni) brojač. Naredba je korisna za sva ponavljajuća izvođenja, a česta je uporaba u kombinaciji sa nizovima za operacije nad grupom podataka ili pin-ova.
for petlje se u C jeziku koriste puno fleksibilnije nego u nekim drugim programskim jezicima. Bilo koji ili svi elementi zaglavlja (inicijalizacja, uvjet, brojač) mogu nedostajati (osim točka-zareza). Uz to, za te elemente može biti korištena svaka valjana C-naredba sa u C-u valjanim tipom podataka, uključivo float. Ovakvi nesvakidašnji tipovi for naloga mogu riješiti neke rijetke programske probleme</td>
<td>inicijalizacija uvjet brojač
 
for(int x=0; x<100; x++)
{
println(x); // printa 0 – 99
}
nalog komentar</td>
<td>Sintaksa:

for (inicijalizacija; uvjet; brojač)
{ nalozi;
}</td>
<td>STRUCTURE
Control
Structure</td>
</tr>
<tr>
<td>goto</td>
<td>(idi na:) – premješta izvođenje na označenu (label) točku programa (navedenu iza goto). Razumnom uporabom goto naloga može se pojednostaviti kod programa.</td>
<td>Neki programeri zaziru od uporabe goto naredbe, jer nepažljiva upo-raba može stvoriti neodređen tijek programa, koji se teško ispravlja.</td>
<td>{goto label;}
label: je proizvoljno ime -oznaka progr.sekvence</td>
<td>STRUCTURE
Control
Structure</td>
</tr>
</table>

<table>
<tr>
<td>HIGH | LOW</td>
<td colspan="3">Općenito jedinični bit (1) označava prisustvo, a nulti bit (0) odsustvo indikativnog elementa u digitalnom sustavu, pa se između ostaloga to primijenjuje i na stanje napona na nožicama elektroničkih elemenata. Tako se prisustvo napona bilježi kao (1) ili HIGH (visoko), a stanje bez napona sa (0) ili LOW (nisko). Ta stanja mogu biti očitana nalogom digitalRead() ako je nožica (pin) nalogom pinMode() definirana kao izlazna (OUTPUT), ili se ta stanja mogu zadati, odnosno postavljati nalogom digitalWrite(), ako je nožica nalogom pinMODE() definirana kao ulazna (INPUT). Razmotrimo najprije mogućnosti stanja HIGH:

Pojam HIGH primijenjen na nožice (pin) nešto se razlikuje, zavisno o tome da li je nožica ulazno (INPUT) ili izlazno (OUTPUT) konfigurirana.
• Arduino (npr. model Atmega) će na ulazno (INPUT) konfiguriranoj nožici po nalogu digitallRead() očtati, tj. izvijestiti stanje HIGH, ako je napon na nožici veći od 3V na 5-voltnoj ploči, ili veći od 2V na 3.3-voltnoj razvojnoj ploči. U protivnom, očitanje će biti LOW.
Međutim, ulaznoj nožici se nalogom digitalWrite() može nametnuti stanje HIGH, neovisno o stanju koje nameće vanjski strujni krug. Nalog naime uključuje interni "potezni" otpornik 20k koji će podići stanje nožice i ako je izvorno bilo nisko (LOW).
• Ako izlazno (OUTPUT) konfiguriranu nožicu nalogom digitalWrite() postavimo na HIGH, napon na nožici će biti nazivni, tj. 5V na 5-voltnoj, odnosno 3.3V na 3,3V ploči. Ovo stanje može pokrenuti strujni tok, npr. "upaliti" LED-diodu, priključenu s nožice preko serijskog otpornika na masu. Suprotno tome, nametnuto LOW stanje, uzrokovat će 0V na nožici (za obje verzije razvojne ploče), te npr. upaliti će LED-diodu priključenu preko serijskg otpornika na +5V, odnosno +3.3V zavisno o verziji razvojne ploče.</td>
<td>VARIABLES
Constants</td>
</tr>
<tr>
<td>if (conditional)
(vidi i if...else)
= =
!=
<, >
<=, >=</td>
<td>if (uvjet) ispituje je li uvjet u zagradi ispunjen. Ako nije ispunjen, nalozi koji slijede se preskaču.
x = = y varijable su jednake
x != y varijable su nejednake
x < y i x > y x je manje od i x je veće od y
x <= y i x >= y manje ili jednako i veće ili jednako</td>
<td>Ako nisu pisane vitičaste zagrade, uvjetovan je samo prvi nalog iza if
== se ne smije se zamijeniti sa =

ovo su poredbeni operatori
(Comparison Operators)</td>
<td>if (uvjet) {//nalozi;}
if (uvjet) //nalog;</td>
<td>coditional
(Contr.Structure)

STRUCTURE
Comparison
Operators</td>
</tr>
<tr>
<td>if ... else
(vidi također i detalje za else)</td>
<td>if () provjerava uvjet u zagradi, pa ako je ispunjen, izvršava se slijed naredbi iza if naloga. Uvjet u zagradi mora sadržati neki od, ili više poredbenih operatora (manje, veće, jednako, nejednako, veće ili jednako, manje ili jednako) Jednostavan if nalog ne mora sadržavati dio else.</td>
<td>Pažnja! Ne smije se umjesto == poredbeog operatora za provjeru <u>da li je</u> neki x jednak zadanoj vrijednosti, koristiti =, koji x-u <u>dodijeljuje</u> zadanu vrijednost</td>
<td>if (uvjet)
{nanizati linije naredbi;
}
else (neobavezno)
{naredbe ako uvjet nije ispunjen;}</td>
<td>STRUCTURE
Control
Structure</td>
</tr>
<tr>
<td>INPUT_PULLUP</td>
<td colspan="3">Arduinov Atmega mikrokontroler ima ugrađeni pull-up otpornik (interno spojen na napajanje). Ako preferirate njegovo korištenje umjesto vanjskog pull-up otpornika, možete koristiti argument INPUT_PULLUP u pinMode() nalogu. Primjer takve primjene može se vidjti u: http://www.arduino.cc/en/Tutorial/InputPullupSerial
ULAZNE NOŽICE KONFIGURIRANE SA INPUT ILI INPUT_PULLUP MOŽE UNIŠTITI PRIKLJUČENJE NA NEGATIVNI NAPON (NAPON MANJI OD NAPONA MASE), ILI VEĆI NAPON OD NAZIVNOG NAPONA PLOČE (5 ili 3 V)!</td>
<td>VARIABLES
Constants</td>
</tr>
</table>

***INPUT	OUTPUT***	*Digitalne nožice (digital pin) se mogu koristiti kao ulazne (*INPUT*), *INPUT_PULLUP *i kao izlazne (*OUTPUT*). Konfiguriranjem nožice nalogom pinMode(), mijenjaju se njena električna svojstva kako slijedi:* *Ulazne (*INPUT*) nožice* *Za Arduino (Atmega) pinove konfiguirane nalogom pinMode() kao ulazne (*INPUT*), kaže se da su u stanju "visoke impendance". One imaju vrlo male zahtjeve na strujni krug koji vrši očitavanje, jer su im otpori ekvivalentni serijski vezanom otporniku od 100 megaoma ispred nožice. To ih čini prikladnima za **oč**itanje raznih senzora.* PULL-UP *i* PULL-DOWN *Ako bi se međutim, na ulaznoj nožici očitavao otvoreni prekidač, nožica će "plutati" i može davati nepredvidljiva očitanja. Zbog oiguranja ispravnog očitanja, koristi se tzv. pull-up ili pull-down otpornik, koji povlači otvoreni prekidač u poznato stanje. Otpornik od 10 kΩ dovoljno je mali za pouzdano izbjegavanje "plutanja", a istodobno dovoljno je velik da ograniči struju kad je sklopka zatvorena. Više o tome nudi uputstvo DigitalReadSerial (engl.)* *Korištenje pull-down otpornika rezultira stanjem* LOW *na ulaznoj nožici pri otvorenom, a* HIGH *pri zatvorenom prekidaču.* *Obrnuto, pull-up otpornik osigurava stanje ulazne nožice* HIGH *pri otvorenom, a* LOW *pri zatvorenom prekidaču.*			*VARIABLES* *Constants*
OUTPUT	*Nalogom pinMode() izlazno konfigurirane nožice (*OUTPUT*) u stanju su niske impendance, što znači da mogu osigurati jače struje za vanjske strujne krugove. Atmegine nožice mogu isporučiti ili apsorbirati struju do 40 mA, što je dovoljno npr. za pogon LED-dioda. Zahtjevnija trošila (npr. motori, releji, zavojnice i sl.) moraju se napajati preko transistora ili drugih sučelja – elektroničkih krugova. IZLAZNE NOŽICE MOŽE UNIŠTITI KRATKI SPOJ NA UZEMLJENJE ILI POZITIVNI IZVOR SNAGE, odnosno na previsok napon ! Često će to ožicu pretvoriti u "mrtvi pin", sa očuvanim ostalim funkcijama procesora. Zbog toga je dobro izlazne nožice na druge uređaje spajati preko otpornika od 470 Ω ili 1 kΩ, ako nema potrebe za isporukom maksimalno dopustive struje.*			

int	*Cjelobrojni oblik (integer) je primarni tip podataka za pamćenje brojeva. Arduino Uno (i druge na Atmegi temeljene razvojne ploče) int podatke spremaju kao 16-bitne vrijednosti, (odnosno kao dva 8-bitna bajta), u području od – 2147483648 do 2147483648, tj. između –2^31 i (2^31)-1*
	Negativni int brojevi se spremaju tehnikom koja se zove 2's complement math*. Najviši bit, ponekad nazivan "sign bit" (bit predznaka), naznačuje broj kao negativan. Ostatak bita se invertira i dodaje 1. Arduino barata s negativnim brojevima na očekivani način. Ipak, moguće su neočekivane komplikacije pri radu sa* ***bitshift right*** *(**>>**)operatorom.*
	Vidi također Data Type unsigned Int
		*Napomena**:** "Kapacitet" 16-bitnog zapisa je max. cjelobrojni raspon veličine 2^16 = 65536, tj. od –32768 do +32767 uključivo 0. Ako varijable premaše maksimalni kapacitet, "prebacit" će se na suprotnu granicu (u oba smjera). Evo primjera (lijevo) za 16-bitni **int**:*
		x = -32768;
		x = x –1 //prekoračen je negatvni maksimum
		. pasje vrijednost x prebacila na sup-
		. rotnu granicu i sad je 32767 ili:
		x = 32767:
		x = x+1; //prekoračen je pozitvni maksimum
		. pa se vrijednost prebacila na sup-
		. rotnu granicu i sad je: -32768
			Primjer:
			int ledPin = 13;
			
			Sintaksa:
			int imevarijable = vrijednost;
				VARIABLES
				Data Types

int()	*Pretvara bilo koju vrstu vrijednosti u cjelobrojnu*	*Parametar x je vrijednost bilokog tipa*	*int(x)*	*VARIABLES* *Conversion*

Integer Constants	*Cjelobrojne konstante su brojevi, direktno korišteni u skici (programu), npr.* 123. *Pretpostavljeno, tretiraju se kao cjelobrojne s bazom 10 na kakve smo naviknuti, ali to se može promijeniti korištenjem tzv. formatizera (B,0 i 0x) (vidi desno) ili* U *i* L *modifikatora u nastavku: Mjesto oblika cjelobrojnih konstanti s bazom 10 (dekadski brojevi), specijalni modifikatori omogućuju oblik unošenja konstante i s drugim bazama: u ili U iza broja (npr: 33u) označava* ***unsigned*** *format .l ili L iza broja (100000L) unosi broj u* ***long*** *formatu .**uL** ili **UL** (npr: 32767ul) unosi konstantu kao* unsigned long *(za pojašnjenje vidi Napomenu desno)*	*Baza Primjer Formatizer je komentar* *10 (decimalna) 123 nema formatizera* *2 (binarna) B1111011 B radi samo s 8-bitnim vrijednostima (0-255)* *8 (oktalna) 0173 0 prihvatljivi su znakovi 0 – 7* *16 (heksadecimalna) 0x7B 0x prihvatjivi znakovi: 0-9, A-F, a-f* *Napomena: unsigned format podrazumijeva format broja kome je mogući raspon smješten samo na pozitivnoj strani, tj. kod 16-bitnog broja od **0** do (2^16)-1, tj. **65535**, umjeso od –32768 do +32767. "Unsigned" je po tome "bez predznaka" budući nemože biti negatvan.* ***long** je tip broja s proširenim opsegom, spremanog s 32 a ne 16 bita.* *Za detalje vidi opise tipova podataka **unsigned** **Int** i **long***		*VARIABLES Constants*
interrupts()	*Ponavlja nalog za pauziranje iza prekida pauza sa nointerrupts(). Interrupts() dopušta neke važne radnje u pozadini, koje su pretpostavljeno uključene za vrijeme pauziranja. Neke funkcije ne rade dok je pauziranje prekinuto, pa npr. ulazne komunikacje mogu biti ignorirane. Pauze mogu poremetiti vremenske tijekove koda, pa ih valja prekinuti za izvođenja kritičnih dijelova koda.*	*Primjer:* *vod setup() { }* *void loop()* *{* *nointerrupts();* *// kritični vremensko-osjetljivi dio koda ovdje* *interrupts();* *// preostali dio koda smjestiti ovdje* *}*		*FUNCTIONS Interrupts*
isAlpha()	*Provjerava je li znak u zagradi slovni (alpha)*	*Rezultat provjere je True ili False*	*Sintaksa: isApha(znak)*	*FUNCTIONS Characters*
isAlphaNumeric()	*Provjerava je li znak u zagradi alfanumerički*	*Rezultat provjere je True ili False*	*isAphaNumeric(znak)*	
isAscii()	*Provjerava je li znak u zagradi ASCII tipa*	*Rezultat provjere je True ili False*	*isAscii(znak)*	
isControl()	*Provjerava je li znak u zagradi kontrolni znak*	*Rezultat provjere je True ili False*	*isControl(znak)*	
isDigit()	*Provjerava je li znak u zagradi broj (digit)*	*Rezultat provjere je True ili False*	*isDigit(znak)*	
isGraph()	*Provjerava može li se znak u zagradi printati*	*Rezultat provjere je True ili False*	*isGraph(znak)*	
isHexadecimal Digit()	*Provjerava je li u zagradi valjan heksadecimalni broj*	*Rezultat provjere je True ili False*	***isHexadecimalDigit**(znak)*	

isLowerCase()	*Provjerava je li znak u zagradi malo slovo*	*Rezultat provjere je True ili False*	*isLowerCase(znak)*	
isPrintable()	*Provjerava može li se znak u zagradi printati*	*Rezultat provjere je True ili False*	*isPrintable(znak)*	
isPunct()	*Provjerava je li znak u zagradi interpunkcija*	*Rezultat provjere je True ili False*	*isPunkt(znak)*	
isSpace()	*Provjerava je li znak u zagradi praznina (blenk)*	*Rezultat provjere je True ili False*	*isSpace(znak)*	
isUpperCase()	*Provjerava je li znak u zagradi veliko slovo*	*Rezultat provjere je True ili False*	*isUpperCase(znak)*	
isWhiteSpace()	*Provjerava je li u zagradi bijeli prostor*	*Rezultat provjere je True ili False*	*is**WhiteSpace** (znak)*	
Keyboard and Mouse libraries	*Ove osnovne biblioteke omogućuju razvojnim pločama Zero i Due, te na 32u4 temeljenim, da se prikazuju kao izvorni miš ili tipkovnica s pločom povezanog računala. Treba ih oprezno koristiti. Funkcije poput Mouse.Move() i Keyboard.print() premjestit će kursor, ili poslati pritiske na tipke priključenom računalu. Preporuča se korištenje kontrolnog sustava za uključivanje tih funkcija (fizički prekidač ili osigurati regiranje samo na ulaz pod kontrolom). Najbolje je prvo testirati izlaz nalogom Serial.print(), da bi znali koje su vrijednosti prijavljene Vidi primjere ovdje.*			USB (32u4 based boards and Due/Zero only)
Keyboard *(vidi također Mouse)*	*Funkcija omogućuje pločama Leonardo, Micro i Due slanje pritiska na tipku priključenom računalu. Neke ASCII znakove (posebno koji nisu predviđeni za ispis) nije moguće slati računalu. Podržano je korištenje modifikatorskih tipki (koje mijenjaju ponašanje druge istovremeno pritisnute tipke poput Ctrl, Alt i dr.). Podržane su slijedeće tipkovničke radnje:* *Keyboard.begin () tipkovnica počinje* *Keyboard.end () tipkovnica završava* *Keyboard.press () pritisnuto* *Keyboard.print () tiskanje* *Keyboard.println () utisnuto* *Keyboard.release () otpušteno* *Keyboard.releaseAll () sve otpušteno* *Keyboard.write () upis tipke*	*Primjeri:* *KeyboardAndMouseControl: Pokazuje naloge miša i tipkovnice u programu.* *KeyboardMessage: Šalje tekstualni niz kada se pritisne gumb.* *KeyboardLogout: odjavljuje trenutnog korisnika "vrućom tipkom"* *KeyboardSerial: Čita bajt iz serijskog porta i šalje pritisak na tipku.* *KeyboardReprogram: otvara novi prozor u Arduino IDE i reprogra-mira ploču jednostavnim trepćućim programom* *ButtonMouseControl: upravlja kretanjem kursora s 5 tipki.* *JoystickMouseControl: upravlja kretajem kursora računala s palicom kad se pritisne gumb.*		**FUNCTIONS** USB (32u4 based boards and Due/Zero only)
LED_BUILTN	*Većina Arduino razvojnih ploča ima nožicu spojenu na ugrađenu LED-diodu (on-board LED) u seriji s otpornikom. Konstanta* LED_BUILTIN *je broj te nožice. Na većini razvojnih ploča, ta je dioda priključena na digital pin 13.*			*VARIABLES Constants*

<table>
<tr><td>long</td><td colspan="2">je varijabla za spremanje na proširenom prostru za pamćenje. Sprema se na 4 osam-bitna bajta, dakle zauzim 32 bita u memoriji. Opseg mogućih vrijednosti je između – 2 147 483 648 i 2 147 483 647.
Primjer:
long speedOffLight = 186000L; // Objašnjenje sufiksa "L" vidi u opisu int().</td><td>Sintaksa:
long imeVarijable = vrijedn;</td><td>VARIABLES
Data Types</td></tr>
<tr><td>long()</td><td>Pretvara varijablu bilo kog tipa u long</td><td>Vraća varijablu u long obliku.</td><td>long(varijabla)</td><td>VARIABLES
Conversion</td></tr>
<tr><td>loop()</td><td>Petlja –omogućuje ponavljanje ciljanog slijeda naredbi, nanizanih iza definiranje uvjeta komandom if (uvjet) ako je uvjet ispunjen, a u protivnom izvodi se nalog iza retka else</td><td>Loop se primijenjuje nakon definiranja početnih vrijednosti funkcijom setup</td><td>void loop()
{ if (uvjet)
naredbe ako je uvjet ispunjen
else
naredbe ako uvjet nije ispunjen
}</td><td>STRUCTURE
sketch</td></tr>
<tr><td>map()</td><td>Mijenja raspon vrijednosti iz izvornog u drugačiji raspon (npr iz fromLow u toLow, iz fromHigh u toHigh ili iz in-between u in-between).
Ne ograničava vrijenosti unutar raspona, jer vrijednosti van raspona mogu biti korisne. Po želji, za ograničenje raspona koristi se funkcija constrain() prije ili iza map().</td><td>Funkcija može i obrnuti raspon, npr:
y = map(x, 1, 50, 50, 1);
a također radi i s negativnim brojevima
npr: y = map(x, 1, 50, 50, -100);
Funkcija računa cjelobrojno, tj. ostaci djeljenja se ispuštaju,nema zaokruženja.
Pimjer desno mapira analognu vrjednost u 8-bitnu</td><td>void setup() {}
void loop()
{
int val = analogRead(0);
val = map(val,0,1023, 0,255);
analogWrite(9, val);
}</td><td>FUNCTIONS
Math</td></tr>
<tr><td rowspan="2">max()</td><td>Vraća veću od dviju vrijednosti bilo kog tipa.
Iako zvuči nelogično, max() se često koristi za ograničenje donjeg kraja raspona, a min() gornjeg kraja.</td><td>Primjer: dodijeljuje očitanje senzora većem broju od tog očitanja i broja 20
sensVal = max(sensVal, 20);</td><td>Sintaksa:
max(x, y)</td><td rowspan="2">FUNCTIONS
Math</td></tr>
<tr><td>Pozor: Korištenje drugih funkcija unutar zagrada može proizvesti pogrešan rezultat (vidi desno =>)</td><td colspan="2">max(a--, 0); // ovo vodi k pogrešnom rezultatu
max(a,0);
a--; // umjesto gornjega izdvoji funkciju -- van zagrada</td></tr>
<tr><td>micros()</td><td>Vraća broj mikrosekunda od pokretanja Arduino programa. Vrijednost se vraća na nulu (ponovo kreće) nakon približno 70 minuta. Na 16 MHz-noj Arduino ploči (npr. Duemilanove i Nano) ta funkcja ima rezoluciju od 4 mikrosekunde (tj. vrijednost je uvijek višekratnik od 4). Na 8 MHz-nim pločama (npr. LilyPad) rezolucija je 8 mikrosek</td><td colspan="2">Primjer:
unsigned long time;
void setup(){
Serial.begin(9600);
}
void loop(){
Serial.print("Time: ");
time = micros(); // ispisuje vrijeme od početka programa
Serial.println(time);
delay(1000) // čekanje sekundu da se ne ispisuje ogromna količina ispisa
}</td><td>FUNCTIONS
Time</td></tr>
</table>

<table>
<tr>
<td>milis()</td>
<td>Vraća vrijeme u milisekundama (unsigned long) od pokretanja programa na Arduino razvojnoj ploči.

S obzirom da se rezultat iskazuje u obliku unsigned long, nastat će logička greška (logic errors) ako programer pokuša raditi izračune sa kraćim tipom podataka (npr. int) Čak i long tip podatka može proizvesti takvu grešku, jer je njegova najveća moguća vrijednost upola manja od unsigned long koji je sav u pozitivnom području.
(vidi opise tipova podataka long i unsigned long)</td>
<td colspan="2">Primjer:
unsigned long time;
void setup(){
Serial.begin(9600);
}
void loop(){
Serial.print("Time: ");
time = millis(); //ispisuje vrijeme od startanja programa
Serial.println(time); // čeka sekundu (da smanji količinu podataka)
delay(1000);
}</td>
<td>FUNCTIONS
Time</td>
</tr>
<tr>
<td rowspan="2">min()</td>
<td>Vraća veću od dviju vrijednosti bilo kog tipa.
Iako zvuči nelogično, min() se često koristi za ograničenje gornjeg kraja raspona, a max() donjeg kraja.</td>
<td>Primjer: dodijeljuje očitanje senzora manjem broju od tog očitanja i broja 100
sensVal = max(sensVal, 100);
// ujedno graniči da rezultat nikad ne premaši 100</td>
<td>Sintaksa:
min(x, y);</td>
<td rowspan="2">FUNCTIONS
Math</td>
</tr>
<tr>
<td>Pozor: Korištenje drugih funkcija unutar zagrada može proizvesti pogrešan rezultat (vidi desno =>)</td>
<td colspan="2">min(a++, 100); // ovo vodi k pogrešnom rezultatu
max(a,100);
a++; // umjesto gornjega izdvoji funkciju ++ van zagrada</td>
</tr>
<tr>
<td>Mouse
(Vidi također Keyboard)</td>
<td colspan="2">Funkcija omogućuje pločama Leonardo, Micro i Due kontrolu kretanja kursora na priključenom računalu. Promjena položaja ursora, uvijek je relativna u odnosu na prethodni položaj.
Mouse.Move() i Keyboard.print() treba oprezno koristiti. Te funkcije će premjestiti kursor, ili poslati pritiske na tipke priključenom računalu. Preporuča se korištenje kontrolnog sustava za uključivanje tih funkcija (fizički prekidač ili osigurati regiranje samo na ulaz koji je pod kontrolom). Najbolje je prvo testirati izlaz nalogom Serial.print(), da bi znali koje su vrijednosti prijavljene. Neke načine rukovanja time vidi na Arduino uputama (engleski):
http://www.arduino.cc/en/Tutorial/KeyboardAndMouseControl, ButtonMouseControl, i JoystickMouseControl.</td>
<td>Podržane su slijedeće radnje:
Mouse.begin () miš počinje
Mouse.click () klik
Mouse.end () miš završava
Mouse.move () pomak miša
Mouse.press () pritisak mišje ipke
Mouse.release () otpuštanje mišje tipke
Mouse.isPressed () mišja tipka stisnuta</td>
<td>FUNCTIONS
USB (32u4 based boards and Due/Zero only)</td>
</tr>
<tr>
<td>nointerrupts()</td>
<td>Obustavlja pauziranje, koje se može nastaviti nalogom interrupts(). Pauze mogu poremetiti vremenske tijekove koda, pa ih valja prekinuti za izvođenja kritičnih dijelova koda. Neke funkcije ne rade dok je pauziranje prekinuto, pa npr. ulazne komunikacje mogu biti ignorirane.</td>
<td colspan="2">void setup() {}
void loop()
{
noInterrupts();
// kritične, vremenski osjetljive kodove smjestiti ovdje
interrupts();
// ostale kodove nizati ovdje
{</td>
<td>FUNCTIONS
Interrupts</td>
</tr>
<tr>
<td>noTone()
(vidi i tone())</td>
<td>prekida generiranje pravokutnih valova aktiviranih nalogom tone(). Nema učinka ako ton nije generiran.</td>
<td>Napomena: Za reproduciranje razne visine tonova na više pin-ova, mora se pozvati noTone() na jednom pinu, prije iniciranja tone() na drugom.</td>
<td>Sintaksa:
noTone(pin)</td>
<td>FUNCTIONS
Advanced I/O</td>
</tr>
</table>

pinMode()	*konfigurira nožicu u zagradi kao ulaznu ili izlaznu (funkcionalnosti nožica vidi u opisu digital pins) Od Arduino 1.0.1 moguće je uključiti ugrađene pullup otpornike nalogom* ***pinMode(***INPUT_PULLUP***)****, dok konfiguracija* INPUT *izričito isključuje interni pullup.* *Analogne nožice (A0, A1 . . . itd.) se ne mogu konfigurirati kao digitalne.*	*Primjer:* ***int ledPin =*** *13****;*** *// LED je na digital pinu 13* ***void setup() {*** ***pinMode(****ledPin, OUTPUT****);*** *. // postavlja digital pin kao izlazni* ***}*** ***void loop() {*** ***digitalWrite(****ledPin, HIGH****);*** *. // uključuje LED-diodu* ***delay(****1000****);*** *// čeka sekundu* ***digitalWrite(****ledPin, LOW****);*** *// gasi LED* ***delay(****1000****);*** *// čekanje 1 sekundu* ***}***	*pinmode(pin, mode);* *pin je broj (oznaka) nožice* *mode je:* INPUT *(za ulazni pin),* OUTPUT *(izlazna nožica)* INPUT_PULLUP *(uključuje ugrađeni otpornik za interno napajanje slabijih trošila, poput LED-dode)*	*FUNCTIONS* *Digita I/O*
pow()	*izračunava vrijednost potencije, može i s razlomljenim eksponentom. Prikladna je za generiranje eksponencijalnog niza vrijednosti ili krivulja.*	*Baza i eksponent se zadaju kao float* *Rezultat se vraća u obliku double*	*Sintaksa:* *pow(baza, eksponent)*	*FUNCTIONS* *Math*
PROGMEM	*sprema podatke u fleš-(programsku) memoriju umjesto u SRAM, gdje se inače spremaju (za Arduino ploče dostupne su naime razne vrste memorije). Ta ključna riječ je varijabilni modifikator, koga treba koristiti samo s tipovima podataka definiranim u biblioteci* ***pgmsoace.h*** *dostupnoj samo u AVR arhitekturi. Zato na vrhu skice (programa) prvo treba uključiti:* ***#include <avr /pgmspace.h>*** *Treba reći, da u razim verzijama Arduina PROGMEM negdje radi, a negdje ne.* *Preporuka je slijedeća:* ***const*** *dataType imeVariable****[] PROGMEM = {};*** *// koristi ovaj oblik* ***const PROGMEM*** *dataType imeVariable****[] = {};*** *// ili ovaj oblik* ***const*** *dataType* ***PROGMEM*** *imeVariable****[] = {};*** *// a ne ovaj* *Detaljnije i primjere vidi u engleskom izvorniku opisa PROGMEM*	*Sintaksa:* ***const*** dataType imeVarijable ***[] PROGMEM =*** *{data0, data1, data3 ...****};*** dataType *je bilo koja vrsta varijable* ime varijable *je naziv za ciljani niz podataka* *Matrice* *Česta je potreba raditi s puno teksta, npr. kod projekata sa LCD zaslonom za postavljanje nizova teksta, što je ustvari dvodimenzionalna matrica. Te velike strukture prikladno je spremati u programsku memoriju.*	*VARIABLES* *Utilityes*
pulseIn()	*mjeri trajanje impulsa (*HIGH *ili* LOW*) na nožici. Na primjer, ako je vrijednost* HIGH*,* ***pulsein()*** *čeka da pin dosegne stanje* HIGH*, starta mjerenje vremena, čeka stanje pina* LOW *i zaustavlja mjerenje vremena. Potom vraća duljinu impulsa u mikrosekundama - ili* 0 *ako nije primljen puni impuls unutar zadanog vremena (timeout). Ovo mjerenje je određeno empirijski i može griješit u kraćim impulsima. Radi u području od 10 mikroseknd do 3 minute. Funkcija radi samo ako je aktivian* ***interrupts()****. Ako je funkcija pozvana kad je pin već visok (*HIGH*), čekat će i izmjeriti trajanje sljedećeg impulsa.*	*Primjer:* *int pin = 7;* *unsigned long duration;* *void setup() {* *pinMode(pin,* INPUT*);* *}* *void loop()* *{* *duration = pulseIn(pin,* HIGH*);* *}*	*Sintksa:* *pulsein(pin, value) ili:* ***pulsein(****pin****,*** *value****,*** *timeout****)*** pin*: oznaka nožice (****int****)* value*: tip impulsa (*HIGH *ili* LOW*)* timeout *(opcionalno): vrijeme čekanja da se impuls kompletira (mikrosekunda).* *Rezultat je unsigned long tipa*	*FUNCTIONS* *Advanced I/O*
random()	*generira pseudo-slučajne brojeve.* *Vraća slučajni broj u granicama opsega* min *i* max-1 *u* ***long*** *obliku (broja povećane preciznosti).*		*Sintaksa:* *random(max) ili:* *random(min, max)* *max: gornja granica opsega* *min: donja granica (opcija)*	*FUNCTIONS* *Random Numbers*

<table>
<tr>
<td>randomSeed()</td>
<td>inicjalizira generator pseudo-slučajnih brojeva na proizvoljnoj točki u slučajnom nizu, koji je uvijek isti, iako je vrlo dug i slučajan.</td>
<td>Ako je važno da se niz vrijednosti generiranih sa random() razlikuju na kasnijim izvršenjima skice, koristite randomSeed() za inicijaliziranje generatora slučajnih brojeva s prilično slučajnim unosom, kao što je analogRead() na nepovezanom pinu.

Nasuprot tome, ponekad može biti korisno imati pseudo-slučajne sekvence koje se točno ponavljaju. To se može postići pozivanjem randomSeed() sa fiksnim brojem prije pokretanja slučajnog niza.</td>
<td>Primjer:
long randNumber;
void setup(){
Serial.begin(9600);
randomSeed(analogRead(0));
}
void loop(){
randNumber = random(300);
Serial.println(randNumber);
delay(50);
}</td>
<td>FUNCTIONS
Random Numbers</td>
</tr>
<tr>
<td>return</td>
<td>Ključna riječ return obustavlja prethodeću funkciju i (po potrebi) vraća zadanu vrijednost</td>
<td>Obje sintakse desno su valjane. Pod "vrijednost" se podrazumijeva bilo koja varijaba ili tip konstante.</td>
<td>{return;} ili
{return vrijednost;}</td>
<td>STRUCTURE
Control Structure</td>
</tr>
<tr>
<td>serial</td>
<td colspan="2">Koristi se za komunikaciju između Arduino ploče i računala ili drugih uređaja. Sve Arduino ploče imaju barem jedan serijski priključak: Serial (poznat i kao UART ili USART). Taj komunicira na digitalnim pinovima 0 (RX) i 1 (TX), kao i sa računalom putem USB-a. Stoga, ako koristite ove funkcije, ne možete koristiti pinove 0 i 1 za digitalni ulaz ili izlaz. Za komunikaciju s Arduino pločom može se koristiti ugrađeni serijski monitor Arduino okruženja. Kliknite gumb serijskog monitora na alatnoj traci i odaberite istu brzinu prijenosa (baud rate) koja se koristi u pozivu za begin().
Arduino Mega ima tri dodatna serijska porta: Serial1 na pinovima 19 (RX) i 18 (TX), Serial2 na pinovima 17 (RX) i 16 (TX) i Serial3 na pinovima 15 (RX) i 14 (TX). Da bi se ove nožice koristile za komunikaciju s osobnim računalom, trebat će dodatni USB-to-serial adapter, jer one nisu spojene na Mega USB-to-serial adapter. Da bi ih koristili za komunikaciju s vanjskim TTL serijskim uređajem, trebamo spojiti TX pin na RX pin našeg uređaja, RX na TX pin našeg uređaja, a masu Mega na masu uređaja. (Ne spajajte ove pinove izravno na RS232 serijski port; jer oni rade na +/- 12V i mogu oštetiti Arduino ploču.)
Arduino Due ima tri dodatna 3.3V TTL serijska porta: Serial1 na pinovima 19 (RX) i 18 (TX); Serial2 na pinovima 17 (RX) i 16 (TX) i Serial3 na pinovima 15 (RX) i 14 (TX). Pinovi 0 i 1 su također povezani s odgovarajućim pinovima ATmega16U2 USB-to-TTL Serial čipa, koji je spojen na USB debug port. Osim toga, na čipu SAM3X, SerialUSB, nalazi se izvorni USB-serijski port.
Ploča Arduino Leonardo koristi Serial1 za komunikaciju preko TTL (5V) serial-a na pinovima 0 (RX) i 1 (TX). Serial je rezerviran za USB CDC komunikaciju. Više informacija vidi na stranici Leonardo getting_started i na stranici hardware_page .</td>
<td>Funkcije:
if (Serial)
available()
availableForWrite()
begin()
end()
find()
findUntil()
flush()
parseFloat()
parseInt()
peek()
print()
println()
read()
readBytes()
readBytesUntil()
readString()
readStringUntil()
setTimeout()
write()
serialEvent()

Primjeri
ReadASCIIString
ASCII Table
Dimmer
Graph
Physical Pixel
Virtual Color Mixer
Serial Call Response
Serial Call Response ASCII</td>
<td>FUCTIONS
Comunication</td>
</tr>
</table>

setup()	*Funkcija Setup () se aktivira samo jednom, i to prilikom startanja skice za definiranje počenih varijabli (npr: pinMode, Serial i dr.)*	*Nakon aktiviranja funkcije Setup, budi se ili resetira Arduino razvojna ploča*	*void setup () {zadati početne vrijedn; } slijedi ostatak programa*	*STRUCTURE sketch*
shiftIn()	*Pomiče bajt podataka bit po bit. Počinje od najvrjednijeg (tj. krajnje lijevog) ili od najmanje-vrjednog (desnog) bita. Za svaki bit, clock-pin (nožica sata) počinje visoko (HIGH),* ***sljedeći bit se čita iz podatkovnog retka****, a zatim clock-pin pada nisko (LOW).* *Ako smo sučeljeni s uređajem koji sinkroniziran po rastućim rubovima, moramo osigurati da pin sata (clock-pin) bude* LOW *prije prvog poziva* ***shiftIn()****, npr. uz pomoć naloga* ***digitalWrite (****clockPin****,*** LOW***)****.* *Napomena: ovo je softverska implementacija; Arduino također nudi SPI biblioteku koja koristi hardversku implementaciju koja je brža, ali radi samo na određenim nožicama.*		*Sintaksa:* *bajt incoming = shiftIn (*dataPin, clockPin, bitOrder*)* dataPin*: pin s koga se prenosi podatak bit po bit (int).* clockPin*: pin na koji se prenosi očitanje sa dataPin.* bitOrder*: redosljed pomaka bitova* MSBFIRST *ili* LSBFIRST*, tj. od najvrijednijeg (lijevog) ili od najmanje vrijednog (desnog) bita.*	*FUNCTIONS* *Advanced I/O*
shiftOut()	*Pomiče bajt podataka bit po bit. Počinje od bita najviše (tj. krajnje lijeve) ili najmanje bitne vrijednosti (tj. desne).* ***Svaki bit se upisuje naizmjence na pin podataka****, nakon čega pinom sata (clock-pin) prolazi impuls (bude* HIGH*, pa* LOW*) čime poazuje da je bit dostupan.* *Ako smo u interakciji s uređajem koji je sinkroniziran po rubnom porastu, moramo osigurati da pin sata (clock-pin) bude* LOW *prije prvog poziva* ***shiftOut()****, npr. uz pomoć naloga* ***digitalWrite (****clockPin****,*** LOW***)****.* *Ovo je softverska implementacija; Arduino također nudi SPI biblioteku koja koristi hardversku implementaciju koja je brža, ali radi samo na određenim nožicama* *Napomena: dataPin i clockPin moraju nalogom* ***pinMode()*** *već biti konfgurirani za* OUTPUT.	***shiftOut(****dataPin, clockPin, bitOrder, value****)*** dataPin*: pin na koji se upisuje svaki bit (*int*)* clockPin*: pin s koga se prebacuje bit po pit (int) kad je* dataPin *postavljen na ispravnu vrijednost.* bitOrder*: redosljed pomaka bitova* MSBFIRST *ili* LSBFIRST*, tj. od najvrijednijeg (lijevog) ili od najmanje vrijednog (desnog) bita.* value*: podaci za izmještanje u bajtovima (*byte*).*	*FUNCTIONS* *Advanced I/O*
	shiftOut *se trenutno zapisuje za izlaz 1 bajta (8 bita) što zahtijeva operaciju u dva koraka za izlazne vrijednosti veće od 255.* *// izvedite slijedeće za MSBFIRST serial:* ***int*** *data = 500****;*** ***shiftOut(****dataPin****,*** clock***,*** MSBFIRST***, (****data ***>>*** *8****));*** *// prebacuje highbyte* ***shiftOut (****dataPin****,*** clock***,*** MSBFIRST***,*** data***);*** *// prebacuje lowbyte* *// ili iizvedite slijedeće za LSBFIRST seral:* *data* ***=*** *500****;*** ***shiftOut (****dataPin****,*** clock***,*** LSBFIRST***,*** data***);*** *// prebacuje lowbyte* ***shiftOut (****dataPin****,*** clock***,*** LSBFIRST***,*** data >> 8***);*** *// prebacuje highbyte*	*Ispušteni su opsežnji primjeri iz izvornog teksta:* ***https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/*** *(link kopiraj u adresnu traku web-preglednika)*	

short	*je 16-bitni tip podataka* *Na svim Arduino pločama (temeljenim na ATMega i ARM mikroprocesorima)* ***short*** *tip podataka sadrži dvije 8-bitne vrijednosti, tj. zauzima 16 bita memorije. To daje raspon od -32768 do 32767 (minimalna moguća vrijednost je 2 ^ 15, a maksimalna (2 ^ 15) – 1, uključivo 0).*	*Primjer:* `short ledPin = 13;`	*Sintaksa:* `short` imeVarijable = val imeVarijable: *naziv varijable* val: *dodijeljena vrijednost*	*VARIABLES* *Data Types*
sin()	*vidi* `cos()` *gdje su sve trigonometrijske funkcije*			*VARIABLES* *Data Types*

sizeof()	*operator vraća broj bajtova (bytes) u tipu varijable ili broj bajtova nekog niza vrijednosti ili matrice*
	Operator ***sizeof()*** *je koristan za rad s nizovima (npr. stringova i dr.) gdje je prikladno mijenjati veličinu niza bez izmjene drugih dijelova programa.*
	Program iz primjera desno ispisuje tekstualni niz znak po znak. Pokušaj promijeiti tekstualnu frazu.
	Uvažite da ***sizeof()*** *vraća ukupan broj bajtova, pa bi za veće tipove varijabli poput* `int`, `for` *petja izgledala ovako:*
	`for (i = 0; i < (sizeof(myInts)/sizeof(int)) - 1; i++) {`
	`// učini nešto sa myInts[i]`
	`}`
	Napominje se da ispravno oblikovan niz završava sa simbolom NULL, *koji ima vrjednost ASCII* 0 .
		`char myStr[] = "this is a test";`
		`int i;`
		`void setup(){`
		`Serial.begin(9600);`
		`}`
		`void loop() {`
		`for (i = 0; i < sizeof(myStr) - 1; i ++){`
		`Serial.print(i, DEC);`
		`Serial.print(" = ");`
		`Serial.write(myStr[i]);`
		`Serial.println();`
		`}`
		`delay(5000);` `// to usporava program`
		`}`
			Sintaksa:
			`sizeof(`varijabla`)`
			varijabla: *varijabla ili niz vrijednosti bilokog tipa*
				VARIABLES
				Utilities

sq()	*kvadrira broj (množi ga sa samim sobom kao* x^2*)*	*x je broj bilokog tipa*		*FUNCTIONS* *Math*
sqrt()	*vraća kvadratni korijen broja (kao* $\sqrt{x}$ *ili* $x^{1/2}$ *)* *rezultat je* ***double*** *tipa (broj dvostruke preciznosti)*	*x je broj bilokog tipa*		*FUNCTIONS* *Math*
stream	*Stream definira funkcije čitanja u Arduinu* *To je osnovna klasa za znakove i binarne tokove. Ne zove se izravno, već se poziva kad god se koristi funkcija koja se na nju oslanja.* *Kad koristimo bilo koju jezgrenu funkcionalnost koja koristi* ***read()*** *ili sličnu metodu, možemo sigurno pretpostaviti da se poziva na* ***Stream*** *klasu. Za funkcije kao što je* ***print(), Stream*** *nasljeđuje od klase ispisa* ***print()****.* *Neke biblioteke koje se oslanjajuna Stream uz funkcije desno uključuju:* Serial, *Wire*, *Ethernet Client*, *Ethernet Server*, SD	*Funkcije (koje uključuju Srteam):* • `available()` • `read()` • `flush()` • `find()` • `findUntil()` • `peek()` • `readBytes()` • `readBytesUntil()` • `readString()` `readStringUntil()` • `parseInt()` • `parsefloat()` • `setTimeout()` *Riječ je o naprednijim funkcijama pa njihov opis nije uključen u ovaj prijevod. Dostupne su na linkovma*		*FNCTONS* *Comunications*

static	Statična ključna riječ koristi se za stvaranje varijabli koje su vidljive samo jednoj funkciji. Međutim, za razliku od lokalnih varijabli koje se stvaraju i uništavaju svaki put kad se neka funkcija pozove, statične varijable ostaju "žive" čuvajući svoje podatke između poziva funkcija.
	
	Varijable koje su deklarirane kao statičke će biti kreirane i inicijalizirane samo kada se "njihova" funkcija pozove.
		Primjer:
		
		/ * RandomWalk
		* Paul Badger 2007
		* RandomWalk šeta gore-dolje slučajno između dvije
		* krajnje točke. Maksimalnim pomakom u jednoj petlji upravlja
		* parametar "step-up".
		* Statička varijabla se pomiče gore i dolje za slučajni iznos.
		Ova tehnika je također poznata kao "pink noise" i "drunken walk".
		* /
		
		#define randomWalkLowRange -20
		#define randomWalkHighRange 20
		int stepsize;
		
		int thisTime;
		int total;
		
		void setup ()
		{
		Serial.begin (9600);
		}
		void loop ()
		{ // tetst randomWalk funkcije
		stepsize = 5;
		thisTime = randomWalk (step up);
		Serial.println(thisTime);
		delay(10);
		}
		int randomWalk (int moveSize){
		static int place; // varijabla za pohranu vrijednosti u slučajnom hodu
		. // - deklariran static tako se pohranjuju vrijednosti
		// između poziva funkcija, ali niti jedna druga funk-
		. // cija ne može promijeniti njezinu vrijednost
		
		place = place + (random(-moveSize, moveSize + 1));
		
		if(place < randomWalkLowRange){
		// provjerava granicu opsega randomWalk-a
		place = place + (randomWalkLowRange - place);
		// reflektira broj u pozitivnom smjeru
		}
		elself(place > randomWalkHighRange) {
		place = place - (mjesto - randomWalkHighRange);
		// reflektira broj natrag u negativnom smjeru
		}
		return place; }
			VARIABLES
			Variable
			Scope &
			Qualifiers

string (char)

VARIABLES
Data Types

Tekstualni nizovi *se mogu prikazati na dva načina. Možemo koristiti* ***String*** *tip podataka većih mogućnosti i većih memorijskih zahtjeva (slijedeći opis), koji je dio jezgre od verzije 0019, ili možemo napraviti string iz niza tipa* ***char*** *i null-terminirati ga. Ovaj se opis odnosi na potonju metodu. U njoj, niz mora završavati tzv. nul-znakom (*NULL*)*

Sve slijedeće deklaracije su valjane za deklariranje stringa:

```
char Str1[15];                  // Objava niza znakova bez inicijalizacije
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};   // Objava s dodatnim znakom - kompajler
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'}; // će dodati nul-znak kao u trećem retku
char Str4[ ] = "arduino";       // inicijaliziranje konstante niza u navod-
                                // nicma – kompajler će znati broj znakova
char Str5[8] = "arduino";       // eksplicitna inicijalizacija (konstanta+vel)
char Str6[15] = "arduino";      // Inicijalizac. s dodatim prostorom za veći niz
```

Null terminacija

Općenito stringovi završavaju s tzv. nul-znakom (ASCII kod **0***), koji kazuju gdje treba prestati sa čitanjem stringa iz memorije, primjerice prilikom ispisa sa* ***serial.print()****. String dakle treba prostor i za znak više od broja znakova u tekstu. Ako ga ne dodamo kao* ***/0*** *prilikom deklariranja, bit će automatski dodan, jer će prevodilac (kompajler) string automatski završiti nul-znakom. Ipak, deklaraciju poput Str2 u primjeru gdje je eksplicitno deklariran premali broj znakova treba izbjegavati, jer to može uzrokovati probleme.*

U deklaraciji string se piše u dvostrukim, a znak po znak u jednostrukim navodnicima
Dugi string pisan u više redova: u svakom redu se odlomak stringa stavlja u navodnike.

Niz stringova (two-dimensional array)

Pri radu s dugim tekstovima (npr. u projektu sa LCD displejom) često je prikladno tekst razbiti u niz stringova. Kako je i sam string takav niz, time zapravo stvaramo dvodmenzionalni niz.
*U tom slučaju, svaki naziv pojedinog niza shvaća se kao "****pointer****", a čitav tekst će predstavljati niz* ***poenter****-a, što se na početku deklaracije naznačava zvijezdicom uz char*. Evo pirmjera:*

```
char* myStrings[]={"To je string 1", " To je string 2",
" To je string 3", " To je string 4", " To je string 5"};

void setup() {
Serial.begin(9600);
}
void loop() {
for (int i = 0; i < 6; i++){
  Serial.println(myStrings[i]);
  delay(500);
  }
}
```

String (object)

VARIABLES
Data Types

Napmena:

"Obični" znakovni nizovi nazivaju se nizovi sa malim s, a instance klase ***String*** *nazivaju se i nizovi sa velikim* ***S****. Imajte na umu da se konstantni nizovi, specificirani u "navodnicima", tretiraju kao* ***char*** *polja, (char* ***arrays****) a ne instance klase* ***String****.*

Klasa String*, dio jezgre od verzije 0019, omogućuje nam da koristimo i manipuliramoe nizove teksta na složenije načine nego što to čine "obični" znakovni nizovi. Mogu se povezati nizovi, dodati, potražiti i zamijeniti podnizovi i još puno toga. Zahtijevaju više memorije od jednostavnog znakovnog niza, ali su i korisniji.*

Primjeri:

- ***StringConstructors***
- ***StringAdditionOperator***
- ***StringIndexOf***
- ***StringAppendOperator***
- ***StringLengthTrim***
- ***StringCaseChanges***
- ***StringReplace***
- ***StringRemove***
- ***StringCharacters***
- ***StringStartsWithEndsWith***
- ***StringComparisonOperators***
- ***StringSubstring***

Funkcije:

- ***String()*** ,
- ***charAt()***
- ***compareTo()***

Funkcije – nastavak

- ***concat()***
- ***c_str()***
- ***endsWith()***
- ***equals()***
- ***equalsIgnoreCase()***
- ***getBytes()***
- ***indexOf()***
- ***lastIndexOf()***
- ***length()***
- ***remove()***
- ***replace()***
- ***reserve()***
- ***setCharAt()***
- ***startsWith()***
- ***substring()***
- ***toCharArray()***

- ***toInt()***
- ***toFloat()***
- ***toLowerCase()***
- ***toUpperCase()***
- ***trim()***

Operatori:

- ***[] (element access)***
- ***+ (concatenation)***
- ***+= (append)***
- ***== (comparison)***
- ***> (greater than)***
- ***>= (greater than or equal to)***
- ***< (less than)***
- ***<= (less than or equal to)***
- ***!= (different from)***

<table>
<tr>
<td>String()

VARIABLES
Data Types</td>
<td>Konstruira instancu klase String. Postoji više verzija koje konstrui-raju nizove iz različitih tipova podataka (tj. formatiraju ih kao nizove znakova), uključujući:
• konstantni niz znakova, u dvostrukim navodnicima (tj. char znak)
• jedan konstantan znak, u pojedinačnim navodnicima
• drugi primjer objekta String
• konstantni int broj ili long int broj
• konstantni int broj ili long int broj, koristeći određenu bazu
• int broj ili long int broj
• int broj ili long int broj, koristeći određenu bazu
• float ili double, koristeći zadana decimalna mjesta

Konstruiranje niza iz broja rezultira nizom koji sadrži ASCII prikaz tog broja. Pretpostavljena je baza deset, pa je:
String thisString = String(13); // rezultira String-om "13"
Međutim, možete koristiti i druge baze. Na primjer:
String thisString = String (13, HEX);
daje string "D", koji je heksadecimalni prikaz decimalne vrijednosti 13. Ili ako vam je draže binarno,
String thisString = String(13, BIN);
daje String "1101" koji je binarni oblik niza 13</td>
<td>Sintaksa:
String(vrjednost)
String(vrjednost, baza)
String(vrjednost, decimalnaMjesta)
Vrjednost koju želimo formatirati kao String može biti:
string, char, byte, int, long, unsigned int, unsigned long, float, double
baza (opcionalno) je željeni broj decmala (samo od float ili double broja)
Primjeri:
String stringOne = "Hello String"; // stvara String-konstantu
String stringOne = String('a'); // pretvara constant char u String
String stringTwo = String("This is a string"); // pretv.constant string u String object
String stringOne = String(stringTwo + " with more"); // spajanje dva niza
String stringOne = String(13); // koristi constant integer
String stringOne = String(analogRead(0), DEC); // koristi int i bazu
String stringOne = String(45, HEX); // koristi int i bazu (hexadecimal)
String stringOne = String(255, BIN); // koristi int i baze (binary)
String stringOne = String(millis(), DEC); // koristi long i bazu (decimalnu)
String stringOne = String(5.698, 3); // koristi float i decimalna mjesta
Vidi i: http://www.arduino.cc/en/Tutorial/StringConstructors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>switch ... case</td>
<td>Naredba switch uspoređuje vrijednost varijable u zagradi sa vrijednostima (label) navedenim u case nalogu. Ako je vriednost varijable jednaka vrijednosti u nekom od case naloga, izvršit će se kod sadržan u tom case nalogu. Na taj način mogu se izvršavati različiti kodovi, zavisno od različitih uvjeta, slično if nalogu.
"Var" je varijabla čija se vrijednost uspoređuje sa konstantama (label1, label2 i label3). Dopušteni tipovi podataka za "var" i "label" su int i char.</td>
<td>Ključna rijeć break (u pravilu sadržana na kraju svakog case naloga) osigurava izlaz iz switch naloga. Bez break, swich bi nepotrebno nastavljao izvođenje slijedećih sekvenci do kraja swich naredbe, ili do nekog break naloga.</td>
<td>swich (var){
case label1;
nalozi1
break;
case label2;
nalozi2
break;
default:
nalozi3
break; }</td>
<td>STRUCTURE
Control
Structure</td>
</tr>
<tr>
<td>tan()</td>
<td>vidi cos() gdje su sve trigonometrijske funkcije</td>
<td></td>
<td></td>
<td>VARIABLES
Data Types</td>
</tr>
</table>

<table>
<tr><td>tone()</td><td>Generira kvadratni val određene frekvencije (uz 50% radnog ciklusa) na pinu. Može se odrediti trajanje, inače se val nastavlja sve dok se ne pozove noTone(). Pin može biti spojen na piezo zujalicu ili drugi zvučnik za reprodukciju tonova.

Istodobno se može generirati samo jedan ton. Ako se ton već reproducira na drugom pin-u, poziv tone() neće imati učinka. Ako se zvuk reproducira na istom pinu, poziv će automatski postaviti frekvenciju.</td><td>Korištenje funkcije tone() prenosi se na izlaz PWM na pinovima 3 i 11 (na pločama osim verzije Mega).

Nije moguće generirati tonove niže od 31 Hz. Za tehničke detalje, pogledajte bilješke Bretta Hagmana.

NAPOMENA: ako želite reproducirati različite visine na više pinova, morate naložiti noTone() na jednom pinu prije nego pozovete tone() na sljedećem pinu</td><td>Sintaksa:
tone(pin, frekvencija) ili:
tone(pin, frekvencija, trajanje)
Parametri:
pin: nožica na kojoj želimo ton
frekvencija: Hz, unsigned long</td><td>FUNCTIONS
Advanced I/O</td></tr>
<tr><td>true | false</td><td colspan="3">Konstante koje iskazuju istinitost izraza ili tvrdnje
true (istina) se često definira kao 1, a false kao 0, no true ima širu definiciju. Svaki cjelobrjni broj (int) osim 0 može značiti istinu u ogičkm smislu. Dakle primjerice –1, 2 i –200 se također prihvaćaju kao true, također u Booleovm smislu.
Pamtimo da se true i false pišu malim slovma, za razliku od HIGH, LOW, INPUT, OUTPUT i dr.</td><td>VARIABLES
Constants</td></tr>
<tr><td>unsigned char</td><td colspan="3">Tip podataka koji zauzima 1 bajt (byte) memorije, kao i byte tip. Moguće vjednosti su od 0 do 255.
Zbog konsistencije Arduino stila programiranja preferira se korištenje byte tipa podataka.</td><td rowspan="3">VARIABLES
Data Types</td></tr>
<tr><td>unsigned int</td><td>Na Uno i drugim ATMEGA baziranim pločama, unsigned int (pozitivni cijeli brojevi) su isti kao i int po tome što spremaju vrijednost na 2 bajta, ali umjesto spremanja negativnih brojeva, unsigned int sprema samo pozitivne vrijednosti, dajući mogući raspon od 0 do 65 535, odnosno do (2 ^ 16) - 1).
Naziv "unsigned" odnosi se na nedostatak predznaka (tzv. "sign bit") kod ovog tipa vrjednosti, koji može biti samo pozitivan.

Due pohranjuje vrijednost na 4 8-bitna bajta (32-bita), u rasponu od 0 do 4 294 967 295, odnosno (2 ^ 32 - 1).

Sintaksa: unsigned int imeVarijable = vrijednost;

Primjer: unsigned int ledPin = 13;</td><td colspan="2">Razlika je između unsigned int i int u tome,što je ukupno raspoživi raspon vrijednosti od 65 535 decimalnih jednica (koliko omogućuje 16-bitni zapis) kod int oblika raspoređen od –32 768 do +32 767, dok je kod unsigned int taj raspon u cjelosti na pozitivnoj strani, tj. od 0 do 65 535.
Razlika nadalje leži u načinu interpretiranja najvišeg bita ("sign bitt", tj. bit predznaka). U Arduino int tipu, ako je najviši (tj. lijevi) bit "1", broj se interpretira kao negativan, a ostalih 15 bitova se interpretira sa tzv. 2' komplementarnim računom (2'complement math). Više o tome na: http://en.wikipedia.org/wiki/2%27s_complement .

POZOR ! Ako varijabla (kod oba tipa) premaši granicu mogućeg opsega vrijednosti, prebacit će se na suprotnu granicu. Na primjer:

unsigned int x
x = 0;
x = x+1 // x sada ima vrijednost 65 535, jer je prešao donju granicu</td></tr>
<tr><td>unsigned long</td><td colspan="2">long varijable imaju povećan prostor za spremanje podataka od 4 8-bitna bajta tj. ukupno 32 bita, pa je mogući raspon vrijednosti 2^32-1 dekadskih jedinica, što iznosi 4 294 967 295, kod unsigned tipa u cijelosti u pozitivnom području uključivo 0.</td><td>Sintaksa:
unsigned long var = val;
val: vrijednost data varijabli var .</td></tr>
</table>

<table>
<tr>
<td>variable scope</td>
<td>Za razliku od prijašnjih jezika poput Basic-a, gdje su sve varijable bile globalne, u programskom jeziku C, čiju verziju Arduino koristi, varijable imaju svojstvo zvano scope.
Dok su globalne varijable vidljive svim funkcijama u programu, lokalne varijable vide samo one funkcije u kojima su deklarirane. U Arduinovom okruženju, svaka varijabla deklarirana izvan funkcije (npr. setup(), loop(), itd.) je globalna.
Kad programi postaju veći i složeniji, lokalne varijable su dobar način da varijabli ima pristup samo ona funkcija koja ju je deklarirala. Time se sprječavaju programske greške, kad jedna funkcija nehotice modificira "tuđe" varijable.
Također, ponekad je dobro deklarirati i inicijalizirati varijablu unutar petlje for, jer to stvara varijablu kojoj se može pristupiti samo unutar zagrada za tu petlju.</td>
<td colspan="2">Primjer:
int gPWMval; // ovu varijablu će vidjeti sve fnkcije
void setup()
{
// ...
void loop()
{
int i; // "i" je vidljiva samo unutar "loop"
float f; // "f" je vidljiva samo unutar "loop"
// ...
for (int j = 0; j < 100; j++) {
// j se može dohvaćati samo unutar for-loop zagrada
}
}</td>
<td>Variable Scope & Qualifiers</td>
</tr>
<tr>
<td>void</td>
<td>Void (spada u grupu varijbli) je ključna riječ koja se koristi isključivo u deklariranju funkcija. Iza nje se očekuje funkcija.</td>
<td>Ako se iza void navede ista funkcija iz koje je pozvana, odgovor će biti "no information"</td>
<td>void setup()
{ . . postava poč.vrijednosti
}
void loop()
{ itd.</td>
<td>VARIABLE Data Types</td>
</tr>
<tr>
<td>volatile</td>
<td>volatile je ključna riječ poznata kao kvalifikator varijable, (variable qualifier) a obično se koristi prije tipa podataka varijable za izmijenu načina na koji kompajler a onda i program tretiraju varijablu.
Deklariranje varijable kao volatile je direktiva za prevodilac. Kompajler je softver koji prevodi vaš C / C ++ kod u strojni kôd, koji predstavja stvarne upute za Atmega čip u Arduinu.
Konkretno, usmjerava kompajler da učita varijablu iz RAM-a (Random Access Memory), a ne iz registra pohrane, što je privremeno mjesto memorije gdje se pohranjuju i manipuliraju programske varijable. Pod određenim uvjetima, vrijednost za varijablu pohranjenu u registrima može biti netočna.
U deklaraciji varijable treba primijeniti volatile kad god se njezina vrijednost može promijeniti nečim izvan kontrole sekcije koda u kojoj se pojavljuje, kao što je istodobno izvršena nit. U Arduinu, jedino mjesto na kojem se to može dogoditi je u dijelovima koda koji su po nterrupt service routine. vezani s prekidima (interrupts), koji se nazivaju prekidom servisne rutine (interrupt service routine).
Primjer desno prebacue LED kada interrupt pin mijenja stanje</td>
<td colspan="2">int pin = 13;
volatile int state = LOW;
void setup()
{
pinMode(pin, OUTPUT);
attachInterrupt(0, blink, CHANGE);
}
void loop()
{
digitalWrite(pin, state);
}
void blink()
{
state = !state;
}</td>
<td>VARIABLES Variable Scope & Qualifiers</td>
</tr>
<tr>
<td>while</td>
<td>Naredba while () pokreće petlju, koja se ponavlja nepekidno, sve dok je izraz u zagradi istinit. On se može promijeniti samo vanjskom intervencijom šadržanom u kodu (npr. provjerom nekog senzora) ili uključenom promjenljivom varijablom i sl.</td>
<td>var=0; Primjer:
while(var<200){
naredbe - izvest će se 200 puta -
var++;
}</td>
<td>while (izraz){
naredbe koje se izvode dok je izraz istinit
}</td>
<td>STRUCTURE Control Structure</td>
</tr>
</table>

word	Na Uno i drugim ATMega pločama word sprema 16-bitne unsigned brojeve. Na Due i Zero međutim brojeve sprema u 32-bitnom unsigned (samo pozitivne) obliku.	Primjer: word w = 10000;	VARIABLES Data Types

KORISNI LINKOVI (*engleski*)

Manuals and Curriculum
Arduino StackExchange
Board Setup and Configuration

Development Tools
Arduino on other Chips

Interfacing With Hardware

- Output
- Input
- User Interface
- Storage
- Communication
- Power supplies

General

Interfacing with Software
User Code Library

- Snippets and Sketches
- Libraries
- Tutorials

Suggestions & Bugs
Electronics Technique
Sources for Electronic Parts
Related Hardware and Initiatives
Arduino People/Groups & Sites
Exhibition
Project Ideas
Languages

Participate

- Formatting guidelines
- All recent changes
- PmWiki
- WikiSandBox training
- Basic Editing

Documentation index

Reference Home

Korekcije, sugestije i novu dokumentaciju postati na Forum.
Tekst ***Arduino reference*** *je objavljen pod licencom*:
Creative Commons Attribution-ShareAlike 3.0 License.
Primjeri kodova u tutorijalu licencirani su kao ***public domain***.

TABELA ASCII KODOVA (IZVOD ZA VELIKA I MALA SLOVA HRVATSKE ABECEDE)

(American Standard Code for Information Interchange, tj.
Američki standardni kod za razmjenu informacija.

Tabela može biti iskazana u decmalnom obliku brojeva (žuto), ali i oktalnom ili heksadecimalnom brojevnom sustavu ili u 8 bajtnom binarnom obliku (binarni kod)
Vidi se da je svaki znak izražen jednim 8-bitnim bajtom.
Kompletna tabela sadrži znatno veći broj znakova, ukjučivo zagrade, posebne znakove, kao i znakove koji se ne printaju.

Znak	Broj	Binarni kod	Kraći zapis		Znak	Broj	Binarni kod	Kraći zapis
razmak	32	00100000	20		X	88	01011000	58
!	33	00100001	21		Y	89	01011001	59
,	44	00101100	2C		Z	90	01011010	5A
.	46	00101110	2E		a	97	01100001	61
0	48	00110000	30		b	98	01100010	62
1	49	00110001	31		c	99	01100011	63
2	50	00110010	32		d	100	01100100	64
3	51	00110011	33		e	101	01100101	65
4	52	00110100	34		f	102	01100110	66
5	53	00110101	35		g	103	01100111	67
6	54	00110110	36		h	104	01101000	68
7	55	00110111	37		i	105	01101001	69
8	56	00111000	38		j	106	01101010	6A
9	57	00111001	39		k	107	01101011	6B
;	59	00111011	3B		l	108	01101100	6C
?	63	00111111	3F		m	109	01101101	6D
A	65	01000001	41		n	110	01101110	6E
B	66	01000010	42		o	111	01101111	6F
C	67	01000011	43		p	112	01110000	70
D	68	01000100	44		q	113	01110001	71
E	69	01000101	45		r	114	01110010	72
F	70	01000110	46		s	115	01110011	73
G	71	01000111	47		t	116	01110100	74
H	72	01001000	48		u	117	01110101	75
I	73	01001001	49		v	118	01110110	76
J	74	01001010	4A		w	119	01110111	77
K	75	01001011	4B		x	120	01111000	78
L	76	01001100	4C		y	121	01111001	79
M	77	01001101	4D		z	122	01111010	7A
N	78	01001110	4E		Š	138	10001010	8A
O	79	01001111	4F		Ž	142	10001110	8E
P	80	01010000	50		š	154	10011010	9A
Q	81	01010001	51		ž	158	10011110	9E
R	82	01010010	52		Ć	198	11000110	C6
S	83	01010011	53		Č	200	11001000	C8
T	84	01010100	54		Đ	208	11010000	D0
U	85	01010101	55		ć	230	11100110	E6
V	86	01010110	56		č	232	11101000	E8
W	87	01010111	57		đ	240	11110000	F0