- Vanemlektor M.J. kurdab kolleegidele:
- “2016 aastal räägin tudengitele mis on bitt – nemad ikka ei saa aru !”
- “2017 aastal räägin tudengitele mis on bitt – nemad ikka ei saa aru !”
- ……
- “2022 aastal räägin tudengitele mis on bitt – mina hakkan vaikselt aru saama juba aga nemad ikka ei saa aru”

# EEM3010 SISSEJUHATUS MEHHATROONIKASSE

Sügis 2022

Sissejuhatus programmeerimisse 2

Martin Jaanus NRG-308
martin.jaanus@ttu.ee 620 2110, 56 91 31 93

Õppetöö : http://isc.ttu.ee
Õppematerjalid : http://isc.ttu.ee/martin

# Teemad

Programmeerimisest madala taseme keeltes (C)

- Kodeerimine (andmetüübid), matemaatika
- Arduino programmeerimise eripära

- Slaididel on palju näiteid (eelkõige “vigastest” programmidest)

  Proovige need iseseisvalt järgi !

Programmeerimisoskust ei saa õpetada – see tuleb läbi kogemuste !

# C keel

- 1958 kõrgkeel ALGOL (tänapäevaste kõrgkeelte eelkäija)
- 1969 Bell Laboratories , keel B
- Programeerimiskeel, mis oleks platvormist sõltumatu aga samas kiire.
- Kõrgkeeled ei suuda garanteerida ajakriitilistes kohtades kiirust ning kompaktsust
- 1973 programmeerimiskeel C (B täiustus), standardiseeriti 1983
- C++ 1989 (objektorienteeritud programmeerimine)
- C# 2002 (.net framework)
- Väga palju sugulaskeeli (sh php, java )
- Peavad olema üsna head eelteadmised riistvarast.
- Vahendid võivad tunduda primitiivsed, aga tegelikult on väga võimekad.

https://en.wikipedia.org/wiki/ALGOL

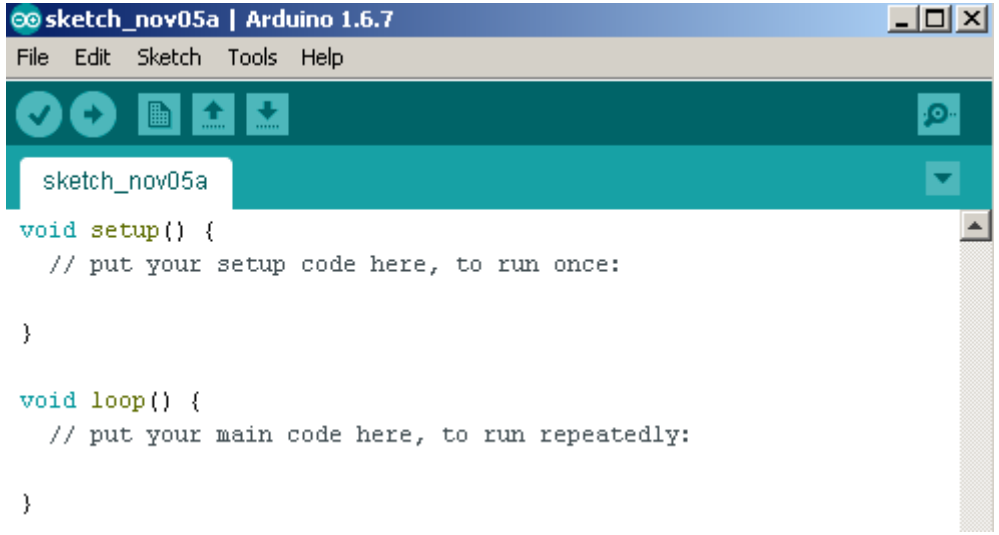
# Arduino programmeerimise eripärad

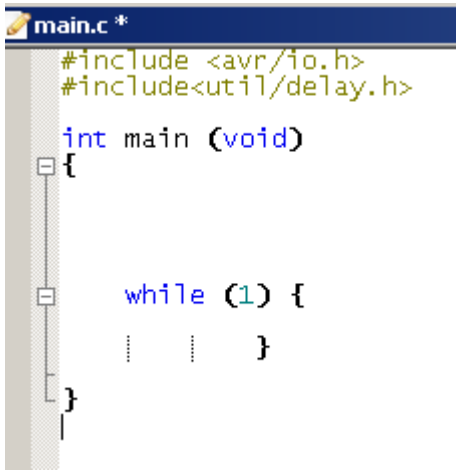
- Arduino arenduskeskkond kasutab olemuselt C++ kompilaatorit, ehk kehtivad C/C++ keele reeglid.
- Lisaks eelnevale on olemas „mugavus“täiendused ning abifunktsioonid, mida C keele maailmas ei tunta. Lisatud definitsioonid liidetakse programmi kompileerimisel. Kui on plaanis tulevikus hakata reaalses C/C++ keeles programmeerima, tasuks  nendega äraharjumist vältida !
- Reeglina värvitud need keskkonnas oranžiks (aga ei pruugi)

# Arduino programm (sketš)

Arduino koodis peavad olema ilmtingimata funktsioonid setup () , ehk kood täidetakse ühe korra ning loop () , kus koodi täidetakse tsüklis . Vastasel juhul tuleb kompileerimisel veateade !

Arduino „keele“ kirjeldus : https://www.arduino.cc/reference/en/

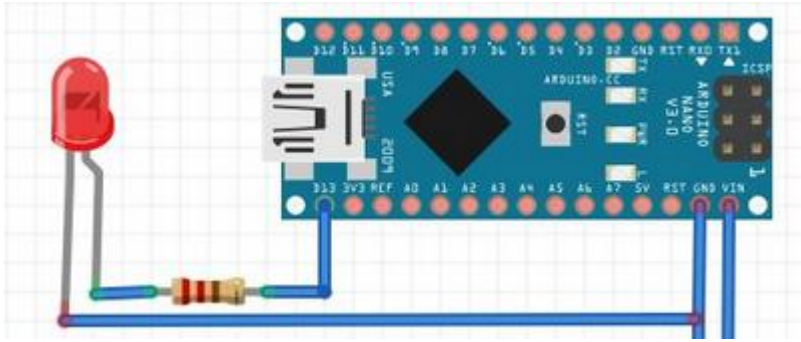



C keelne „tühi“ programm

# Arduinio koodinäide

- Vilgutame valgusdioodi



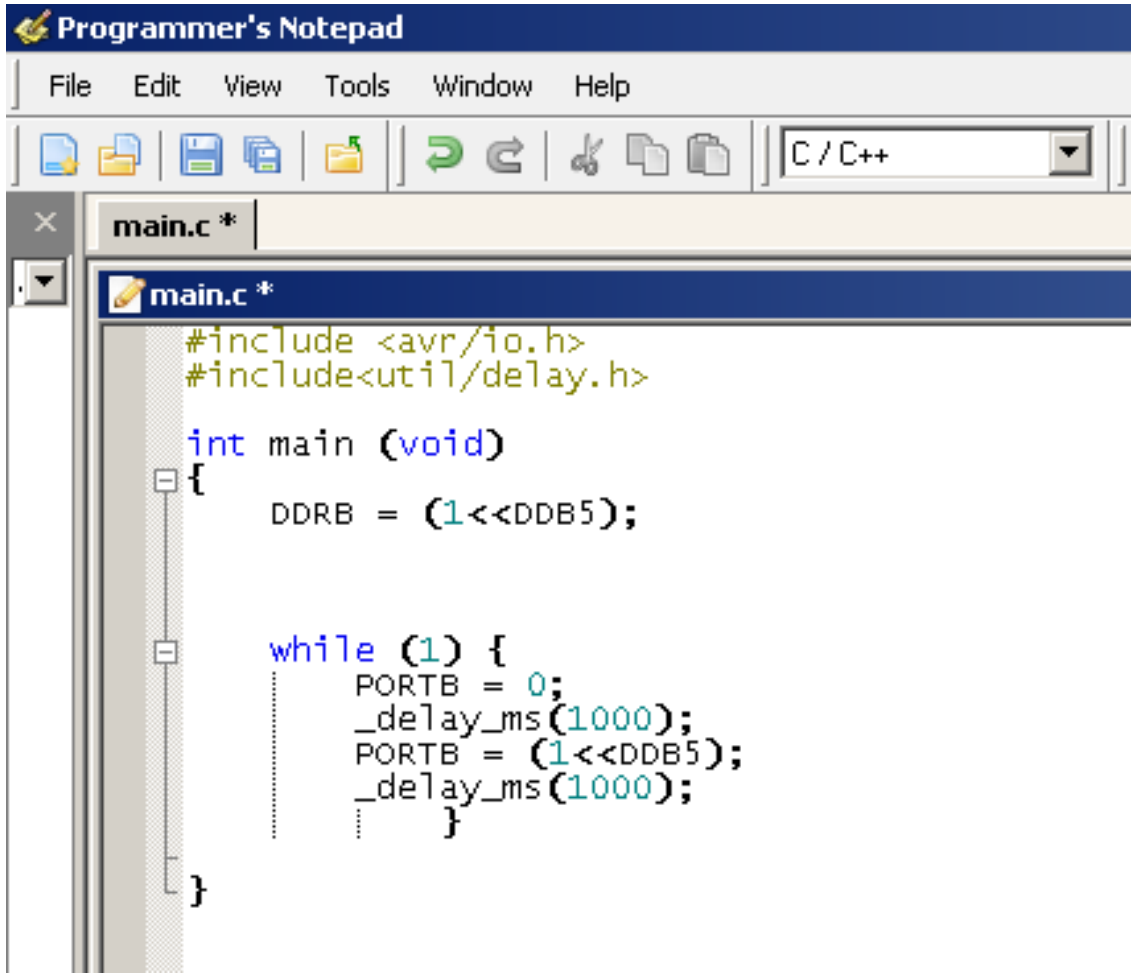
Blink | Arduino 1.6.7

File Edit Sketch Tools Help

Blink §

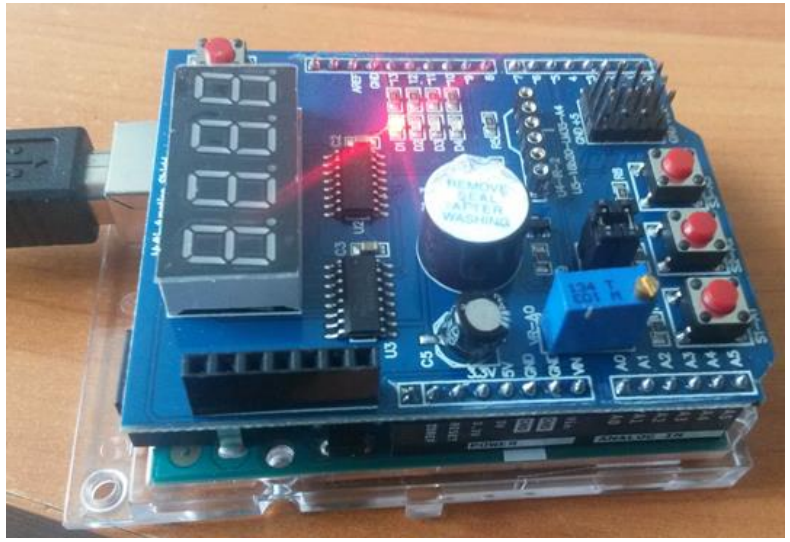
```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

# C koodinäide, sama mikrokontroller



# Assembleri koodinäide, sama mikrokontroller

- Näide tehtud õppeaine „Mikroprotsessorsüsteemid“ jaoks
- See on vaid funktsooni väljakutsumine ja kohustuslik programmi „kest“



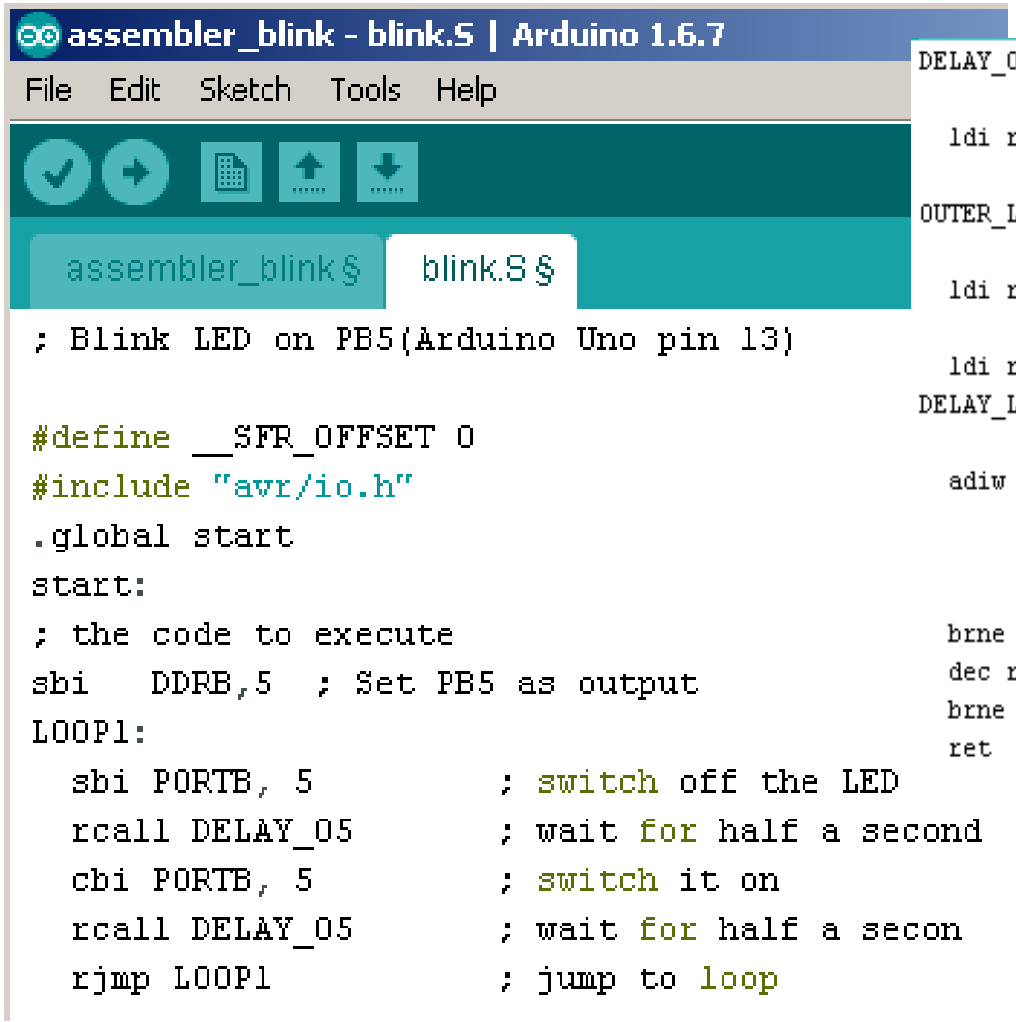
assembler_blink | Arduino 1.6.7

File Edit Sketch Tools Help

assembler_blink§ blink.S

```
extern "C" {
  // function prototypes
  void start();
;
}

void setup() {
  start();

}
void loop() {

}
```

# Assembleri koodinäide, sama mikrokontroller




```
; Blink LED on PB5(Arduino Uno pin 13)

#define __SFR_OFFSET 0
#include "avr/io.h"
.global start
start:
; the code to execute
sbi   DDRB,5   ; Set PB5 as output
LOOP1:
  sbi PORTB, 5          ; switch off the LED
  rcall DELAY_05        ; wait for half a second
  cbi PORTB, 5          ; switch it on
  rcall DELAY_05        ; wait for half a secon
  rjmp LOOP1            ; jump to loop
```

```
DELAY_05:               ; the subroutine:

  ldi r16, 31           ; load r16 with 31

OUTER_LOOP:             ; outer loop label

  ldi r24,0    ; load registers r24:r25 with 1021, our new
                        ; init value
  ldi r25, 0          ; the loop label
DELAY_LOOP:             ; "add immediate to word": r24:r25 are
                        ; incremented
  adiw r24, 1           ; if no overflow ("branch if not equal"), go

                        ; back to "delay_loop"

  brne DELAY_LOOP
  dec r16               ; decrement r16
  brne OUTER_LOOP       ; and loop if outer loop not finished
  ret                   ; return from subroutine
```

# Arduino (C/C++) keele süntaks

- Kompilaator ei loe tühikuid ja reavahesid, koodi paigutus võib olla suvaline , siiski programmikoodi kirjutamisel on olemas tavad, mis hõlbustavad koodist arusaamist.
- Google → obfuscating code

https://www.quora.com/What-is-the-most-complicated-C-code

# Arduino (C/C++) keele süntaks

- Tehakse vahet väiksetel ja suurtähtedel ! Identifikaatorid ei tohi kokku langeda reserveeritud sõnadega (vt. Arduino reference, neid on rohkem kui C keeles, keskkond värvib need ära)
- Näiteks x=2; ja X=2; on erinevad laused.
- Lihtlause lõpeb semikooloniga ;
- näit if (a>b) x=2;
- Liitlause pannakse looksulgude vahele {}, koosneb lihtlausetest.
- Näit if (a>b) {x=2; y=3;}
- Kommentaarid, kui ühel real, eraldatakse // , toimib kuni reavahetuseni. Pikemad kommentaarid algavad /* ja lõppevad */ . Kõik, mis nende vahel, kompilaator ignoreerib.

# Arduino (C/C++) keele süntaks

- Muutujad peavad olema defineeritud
- Saab defineerida alguses, siis eraldatakse kohe mälu.
- Saab deklareerida ka lauses, funktsioonis, siis eraldatakse mälu, aga võetakse mälu pärast selle täitmist.

- Näit unsigned short MinuMuutuja;
- Näit float MinuMuutuja,SinuMuutuja,x,y;

- Muutuja defineerimisel eraldatakse sellele mälu, milles võib olla suvaline sisu ! Et algväärtustada muutujat, saab teha näiteks nii :
- Näit unsigned short MinuMuutuja=0;

# Kuidas arvutisse info on kodeeritud ?

- Peaaegu kõik tänapäevased arvutid töötavad kahendsüsteemis
- Kõige väiksem infoühik on bitt. (0 - 1, tõde-vale jms)

- Andmetüübiks bool (Boole algebra järgi)  C-s puudub, alates C++
- Kõik väärtused mis on 0 on „false“, muul juhul on „true“
- bool x=false; bool y=true;
-
- Kuigi bool on vaid 1 bitt , võtab see mälu ära 1 baidi (8 bitti )

- Kõrgkeeles programmeerija ei pea teadma, kuidas arvuti töötab, insenerid, kes on seotud sardsüsteemide väljatöötamisega võiks seda teada ! Neil endil on lihtsam.

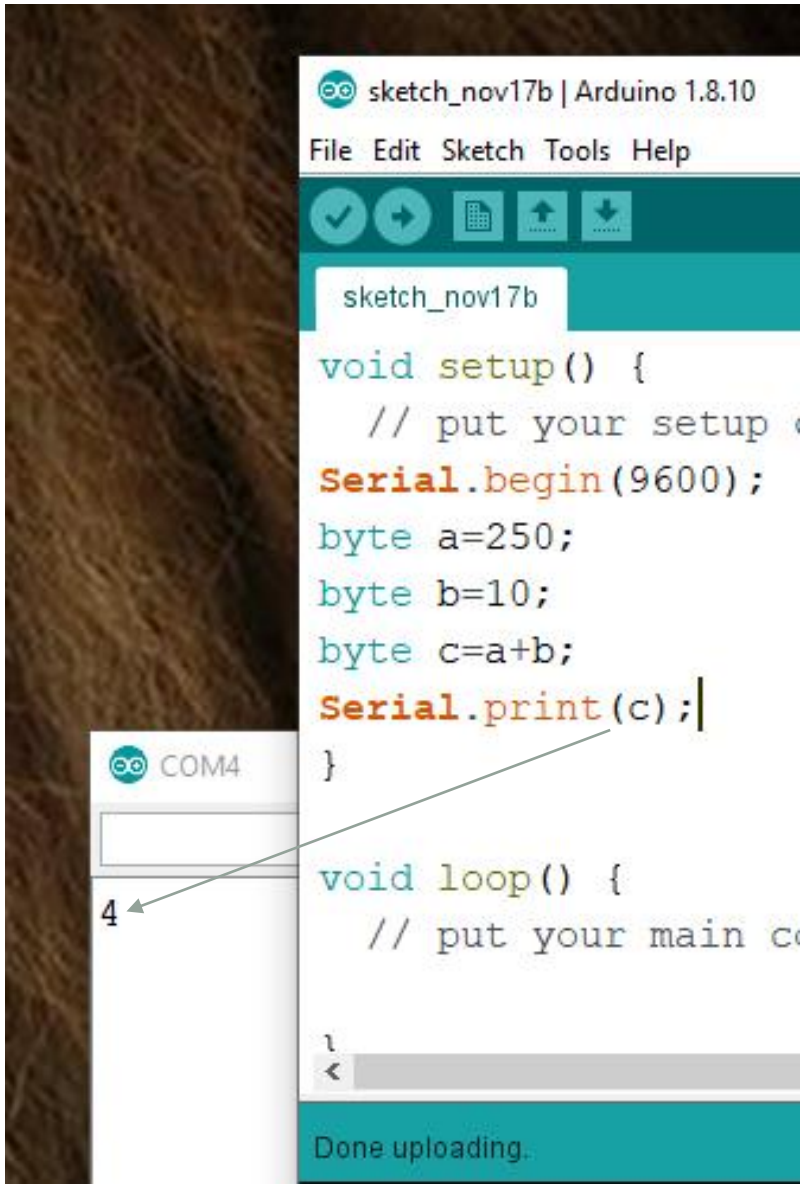
# Positiivsed täisarvud

- Vajalik bittide arv on vähemalt $int\ (log_2(n)) + 1$, kus n on kujutatav arv , näiteks arvu 14 jaoks on vaja nelja bitti :

$2^3\ 2^2\ 2^1\ 2^0$

- 14 (dec)= 1 1 1 0
- Bitijada pikkused on arvutis standardiseeritud:
- 8 bitti (bait,byte ), 16 bitti (sõna, word), 32 bitti topeltsõna (double word).
- Võimalike väärtuste arv $2^n$, aga kuna loendatakse nullist, on maksimum $2^n$-1
- Kui on kasutusel 8 bitti, ehk 1 bait andmeid, saab esitada $2^8 = 256$ väärtust , ehk väärtused 0-255
- Arv 14(dec) =00001110b

- Andmetüüp – byte (Arduino) , unsigned char (C) - 0-255
- Andmetüüp – unsigned int 0-65535 , kaks baiti
- Andmetüüp – unsigned long int 0 – 4294967295 , neli baiti
- On võimalikud ka pikemad tüübid.



# Negatiivsed (märgiga) täisarvud

- Kuigi eraldi märgibitt ning arvubitid tundub loogiline, on see halb
- Esitatakse täiendkoodis (Two's complement ).
- Täiendkood: Sama posiivne arv , vahetame nullid ühtedeks ning ühed nullideks ja liidame juurde 1 (sisuliselt teeme nullkoha nihutamise)
- Arv +14(dec)= 00001110b
- Arv -14(dec)=  11110001b+00000001b=11110010b

- Kontroll  00001110b
- +1111 0010b
- ----------
- (1)00000000b

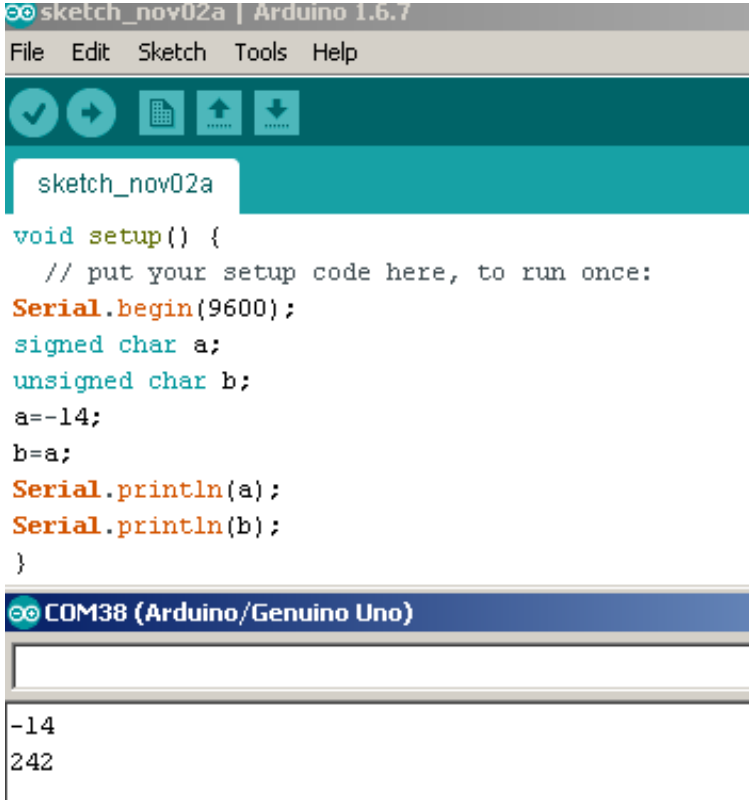
Ülekanne (Carry), kui ei võeta spetsiaalselt midagi ette, läheb see kaduma !

NB !

Arv -14 võib olla aga ka positiivne arv 242 !!! Arvuti ei tee sellel vahet !, vahet peab tegema programmeerija (ja vahest ka kompilaator).

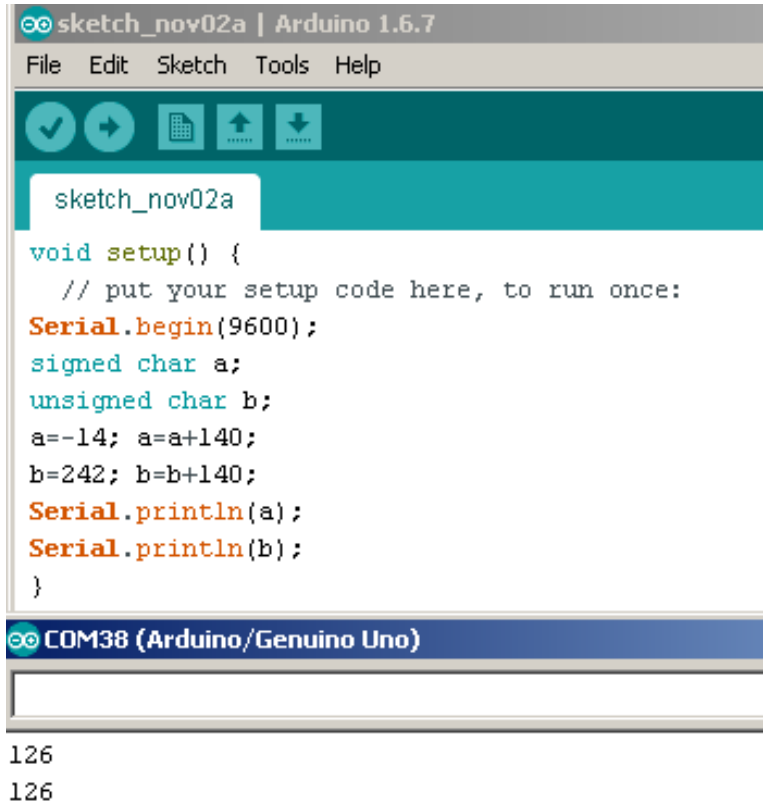
# Negatiivsed (märgiga) täisarvud

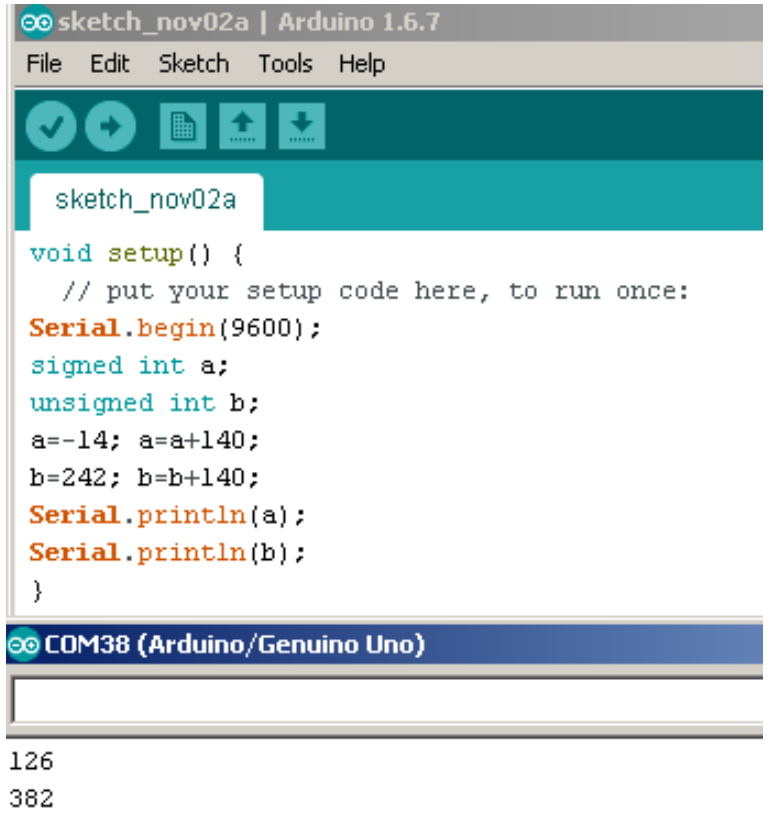
- Andmetüüp signed char  -128...127 (C, Arduino )
- Andmetüüp int (Arduino) signed short int (C)
- -37268...37267
- Andmetüüp long (Arduino) signed long int (C)
- -2147483648.... 2147483647



# Kui ületame piire

- ...ei anna madalataseme keeled veateadet
- Arvuti teeb seda, mida tal kästakse teha !



# Negatiivsed (märgiga) täisarvud

- Kuigi teatud „probleemidest“ päästaks pikemad admetüübid, on neil ka suur puudus – nende kasutamine vajab rohkem mälu ning võtab rohkem protsessori aega !
- See võib olla väga oluline kui mälumaht ning kiirus on piiratud  või ajakriitiline (mikrokontrollerid)

- Kui on selge, et lühema andmetüübiga saab hakkama, siis tuleks kasutada seda.


Attiny13A andmelehest

– Up to 20 MIPS Througput at 20 MHz
• High Endurance Non-volatile Memory segments
– 1K Bytes of In-System Self-programmable Flash prc
– 64 Bytes EEPROM
– 64 Bytes Internal SRAM
– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
– Data retention: 20 Years at 85°C/100 Years at 25°C (s
– Programming Lock for Self-Programming Flash & E
• Peripheral Features

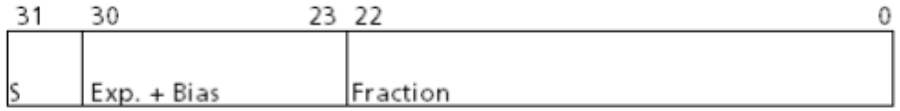
http://ww1.microchip.com/downloads/en/DeviceDoc/doc8126.pdf

# Ujukomaarvud

- Ujukomaarv on paigutatav arvuti mällu kolme komponendi, märgi, arvu tüve ehk mantissi ja astendaja ehk eksponendiga. $märk * tüvi * k^{astendaja}$
- Erinevad meetodid, levinuim IEEE 754 formaat,
- $(-1)^{s}*(1.+M)*2^{E-Bias}$
- Normaliseeritud arvu esimene bitt on alati 1, seda ei salvestata
- Täpsus $1.18 \times 10^{-38}$ kuni $3.4 \times 10^{38}$ , säilitab 6 tüvenumbrit
- , 32 bitti pikk
- Topelttäpsus $2.23 \times 10^{-308}$ to $1.8 \times 10^{308}$ 12 tüvenumbrit, 64 bitti pikk

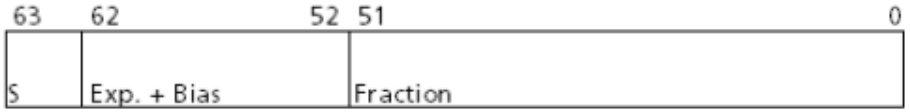
# Ujukomaarvud

- Single float




```
0  000  0000  0  000 0000 0000 0000 0000 0000 = 0.0
0  011  1111  1  000 0000 0000 0000 0000 0000 = 1.0
1  011  1111  1  011 0000 0000 0000 0000 0000 = -1.375
1  111  1111  1  111 1111 1111 1111 1111 1111 = NaN
```

- Double float

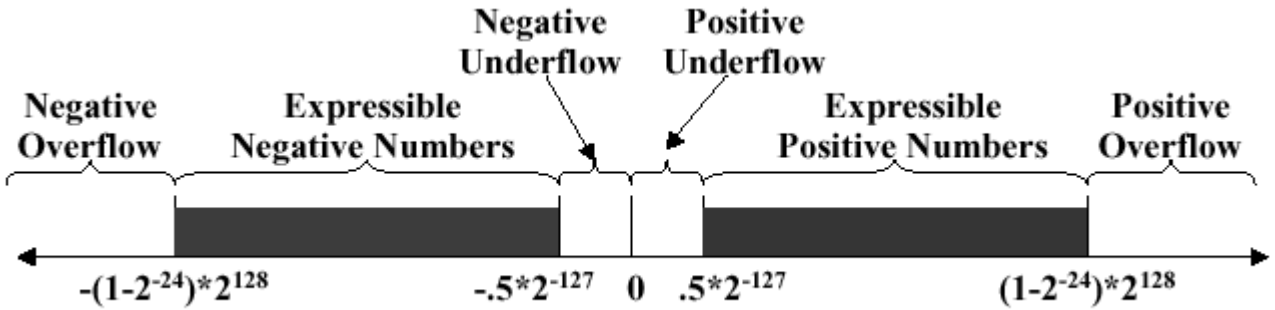



```
0   000 0000 0000 0000 0000 ... 0000 0000 0000 = 0.0
0   011 1111 1111 0000 0000 ... 0000 0000 0000 = 1.0
1   011 1111 1110 0110 0000 ... 0000 0000 0000 = -0.6875
1   111 1111 1111 1111 1111 ... 1111 1111 1111 = NaN
```

https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/FloatingPointDataTypes.html

# Ujukomaarvud

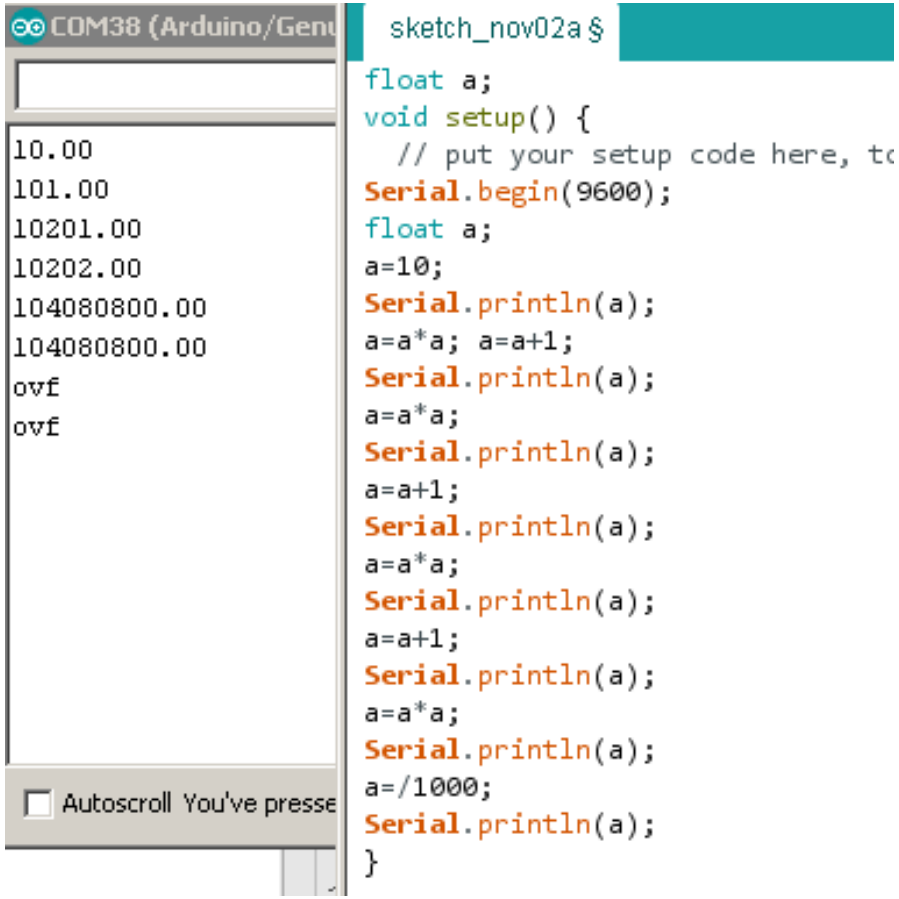
- Mikrokontrolleri puhul kasutage vaid siis kui muudmoodi ei saa (kui ei ole protsessoril ujukoma arvutuse tuge, tehakse tehted tarkvaraliselt, aga see võtab aega ! )
- Probleemid : ületäitumine, alatäitumine
- Probleeme tekitavad ka erinevas suurusjärgus olevate arvude liitmine-lahutamine




https://www.cise.ufl.edu/~mssz/CompOrg/CDA-arith.html

# Ujukomaarvud

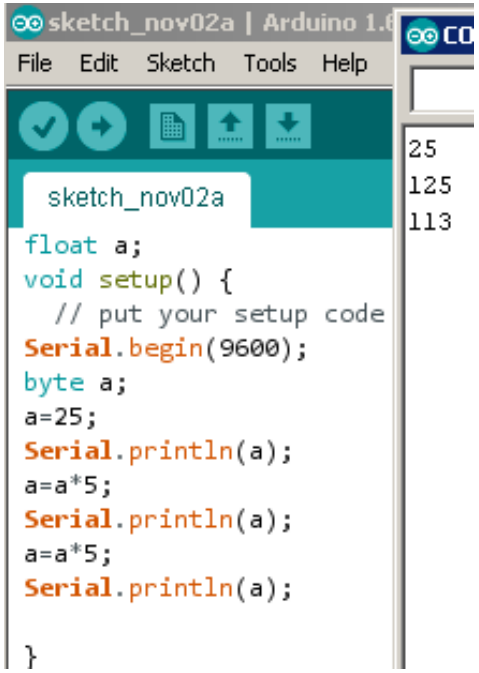
- Kui näiteks teha FOR tsükkel , inkremendiga 1, siis ei pruugi see lõpetada !

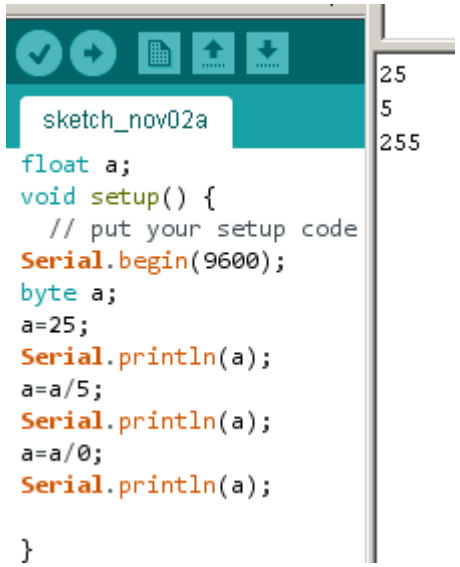


# Matemaatika

- Mikrokontrollerite eripära – kõik tehted ei ole riistvaraliselt toetatud.
- Korrutamine-jagamine võidakse samuti  teha tarkvaraliselt (näiteks TinyAVR ei oska korrutada-jagada) –aeglane (asendatakse liitmis-lahutamistsüklitega)
- Ületäitumiste eest ei hoiatata -

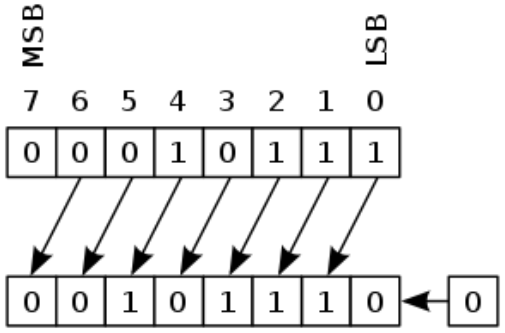
Programm töötab vigse tulemusega edasi. (tegelikult riistvaralise tehte puhul protsessoris staatuse lippude väärtust muudetakse). C keel neid ignoreerib.  Saab tekitada katkestuse.



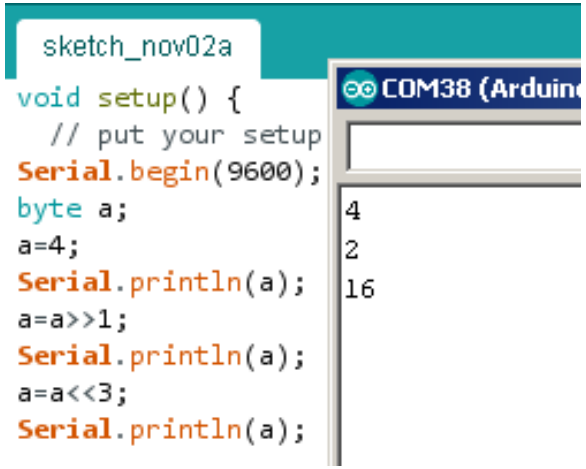



# Nihe

- Korrutamine-jagamine on ebamugavad tehted mikrokontrolleritele, välja arvatud kordajaga 2
- Kümnendsüsteemis nihe vastab 10ga korrutamisele-jagamisele.
- Nihe vasakule suurendab väärtust kaks korda, paremale vähendab kaks korda.
- Aritmeetiline nihe – märgibitti ei nihutata
- Loogiline nihe , asemele kirjutatakse nullid
- Ringnihe – asemele kirjutatakse välja langev(ad) bit(id)

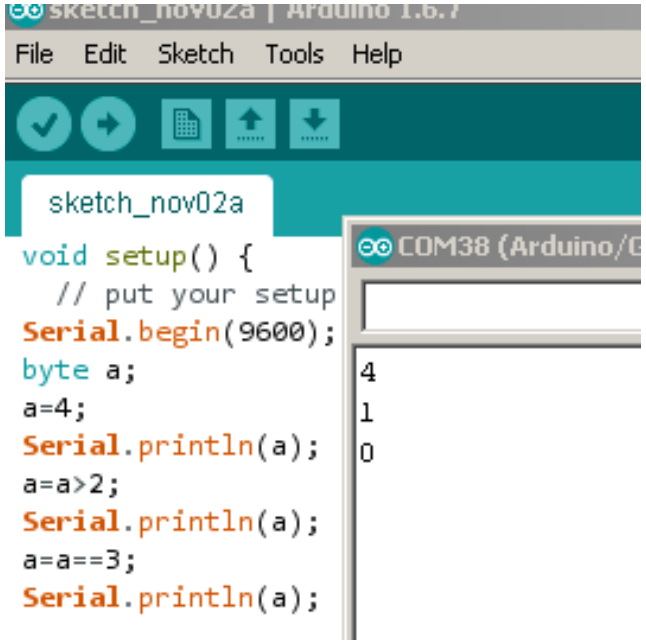



Kui võimalik , asendatakse korrutamine-jagamine nihutamisega. Õige algoritmi ja kordajate valik



# Võrdlustehted

- == (võrdne) != (ei ole võrdne) < (väiksem kui) > (suurem kui)
- <= (väiksem võrdne kui)
- >= (suurem võrdne kui)
- Tehte tulemus on signed int, väärtusega 1, kui väide on tõene või 0 kui väide on vale
- Mitte segi ajada = ja == !
- = on omistamine !



# Omistamine ,inkrement ja dekrement

- Omistamine (=) on sama harilik tehe nagu teised. Avaldis võib sisaldada mitu omistamistehet.

- Omistamine on võimalik vaid siis kui omistamismärgist vasakule jääb kindla mäluväljaga muutuja (ei tohi näiteks 2=a;)
- Saab teha tänu sellele üsna omapäraseid lahendusi:
- c=a+b;  d=e+c; aga sama on ka d=e+(c=a+b);
- Kui muutujad on eri tüüpi ,kaasnevad tehetega tüübiteisendused , mis võivad omakorda tekitada segadust.
- Kui muutujate väärtuseid on vaja muuta vaid ühe võrra, saab avadlise kirjutada lihtsamalt
- a=a+1; asemel a++;
- a=a-1; asemel a--;
- Need käsud on toetatud ka riistvaraliselt.

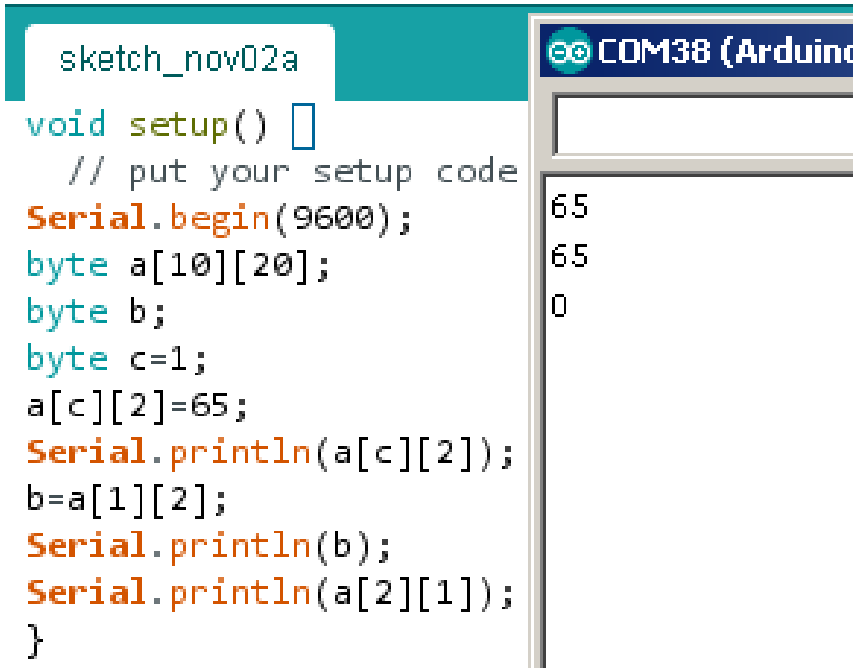
# Tehete järjekord

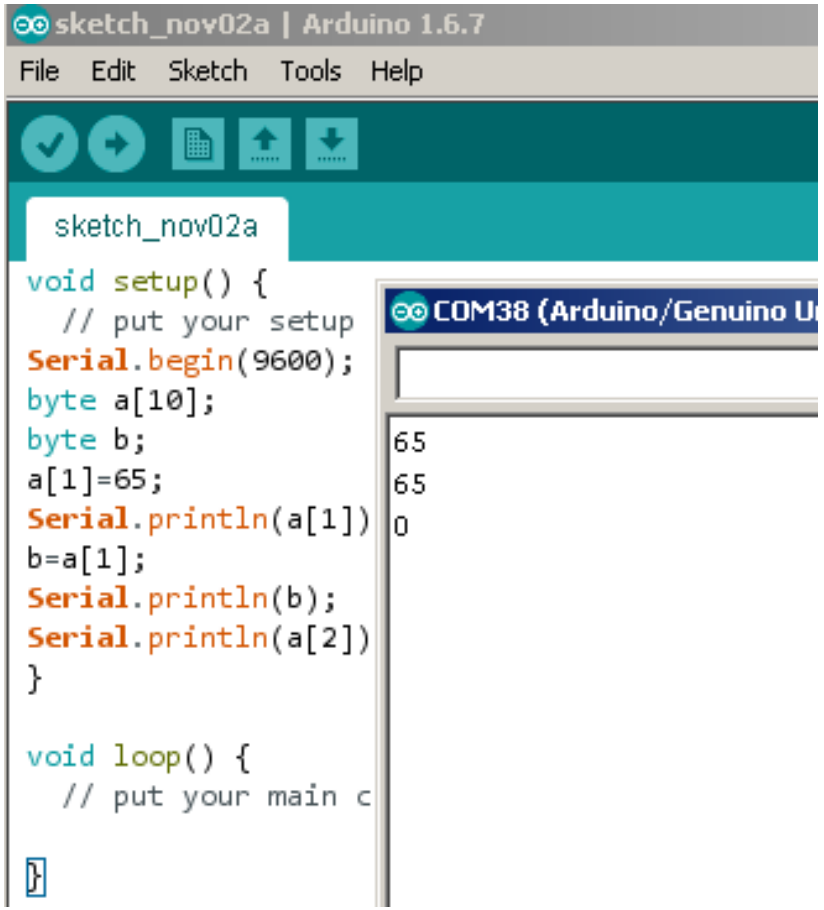
- Igas programmeerimiskeeles on oluline sellele mõistetav tehete järjekord
- C/C++ oma, kehtib ka Arduino keskkonnas
- https://www.viva64.com/en/t/0064/

| Precedence | Operator | Description |
|---|---|---|
| 1 | :: | Scope resolution |
| 2 | ++ -- | Suffix/postfix increment and decrement |
| | *type*() *type*{} | Function-style type cast |
| | () | Function call |
| | [] | Array subscripting |
| | . | Element selection by reference |
| | -> | Element selection through pointer |
| 3 | ++ -- | Prefix increment and decrement |
| | + - | Unary plus and minus |
| | ! ~ | Logical NOT and bitwise NOT |
| | (*type*) | C-style type cast |
| | * | Indirection (dereference) |
| | & | Address-of |
| | sizeof | Size-of |
| | new, new[] | Dynamic memory allocation |
| | delete, delete[] | Dynamic memory deallocation |
| 4 | .* ->* | Pointer to member |
| 5 | * / % | Multiplication, division, and remainder |
| 6 | + - | Addition and subtraction |
| 7 | << >> | Bitwise left shift and right shift |
| 8 | < <= | For relational operators < and ≤ respectively |
| | > >= | For relational operators > and ≥ respectively |
| 9 | == != | For relational = and ≠ respectively |
| 10 | & | Bitwise AND |
| 11 | ^ | Bitwise XOR (exclusive or) |
| 12 | \| | Bitwise OR (inclusive or) |
| 13 | && | Logical AND |
| 14 | \|\| | Logical OR |
| 15 | ?: | Ternary conditional |
| | = | Direct assignment (provided by default for C++ classes) |
| | += -= | Assignment by sum and difference |
| | *= /= %= | Assignment by product, quotient, and remainder |
| | <<= >>= | Assignment by bitwise left shift and right shift |
| | &= ^= \|= | Assignment by bitwise AND, XOR, and OR |
| 16 | throw | Throw operator (for exceptions) |
| 17 | , | Comma |

# Massiiv  (andmete kogum)

- Mugav on koondada sarnaseid elemente
- muutuja tüüp , nimi[elementide arv]
- Võib olla ka mitme dimensiooniga
- Näit byte a[10][20];

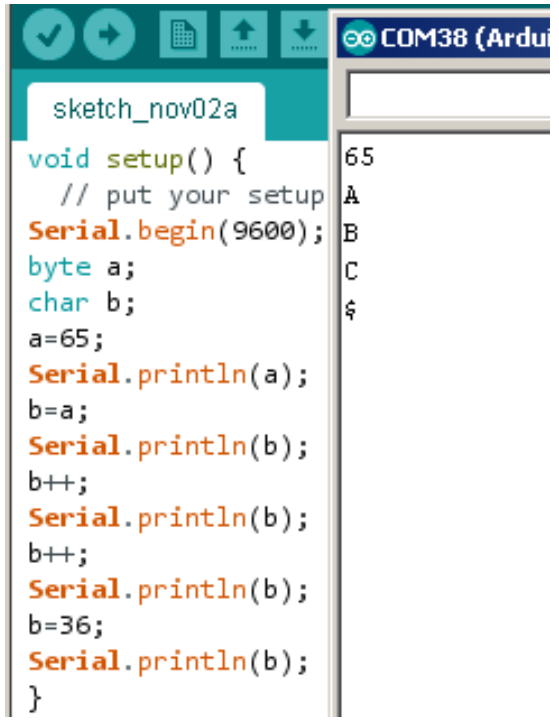


# Tekst (sümbolid)

- Numbriline väärtus vastab tähemärgile – tekstikodeering.
- Levinuim ASCII (American Standard Code for Information Interchange) kodeering 1963. aastast, põhineb inglise tähestikul .
- 7 bitiline kodeering , esimesed 28 märki on tehnilised.

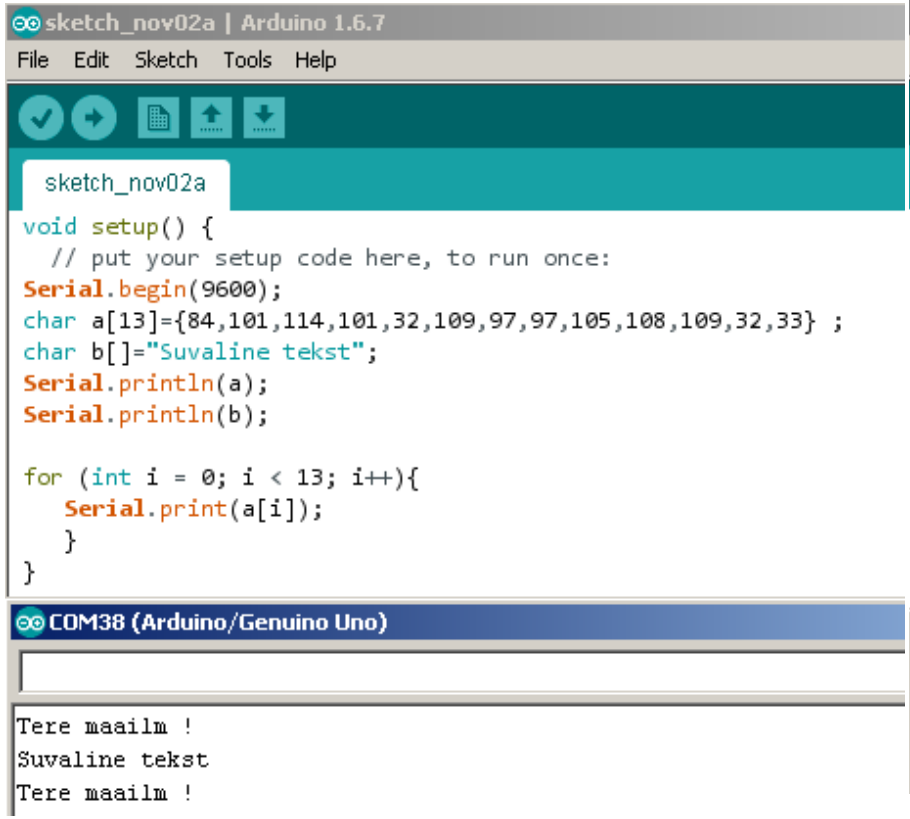
- Olemuselt on tähemärk ühebaidine arv, millega saab teha kõiki tehteid.
- Arduino kompilaator teeb vahet, kas tüüp on byte või char ja vaid sedagi vaid „print“ funktsioonis , C keeles on asi teisiti
- Arv kuvatakse märgina alles vastuvõtuseadmes)

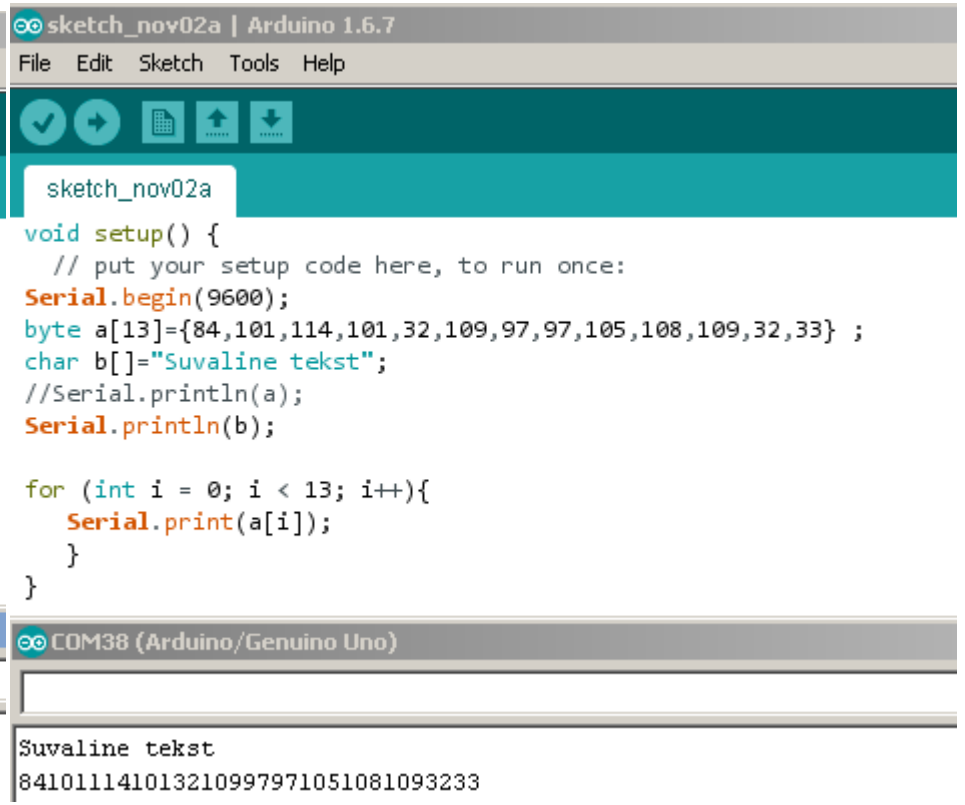



https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html

# Tekst (stringid)

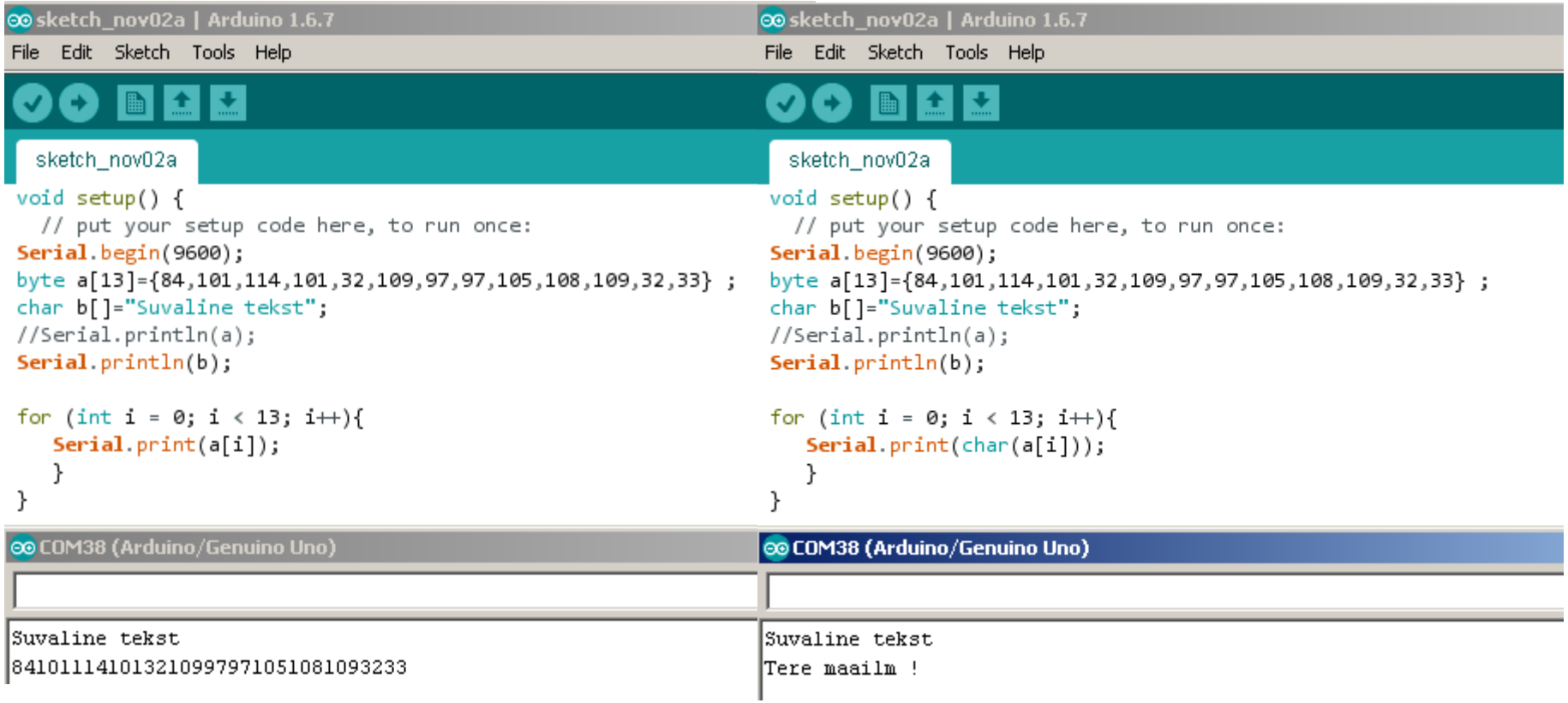
- String on märkide jada,  olemuselt ühebaidiste arvude massiiv.
- Saab teha kõiki tehteid.



# Tekst

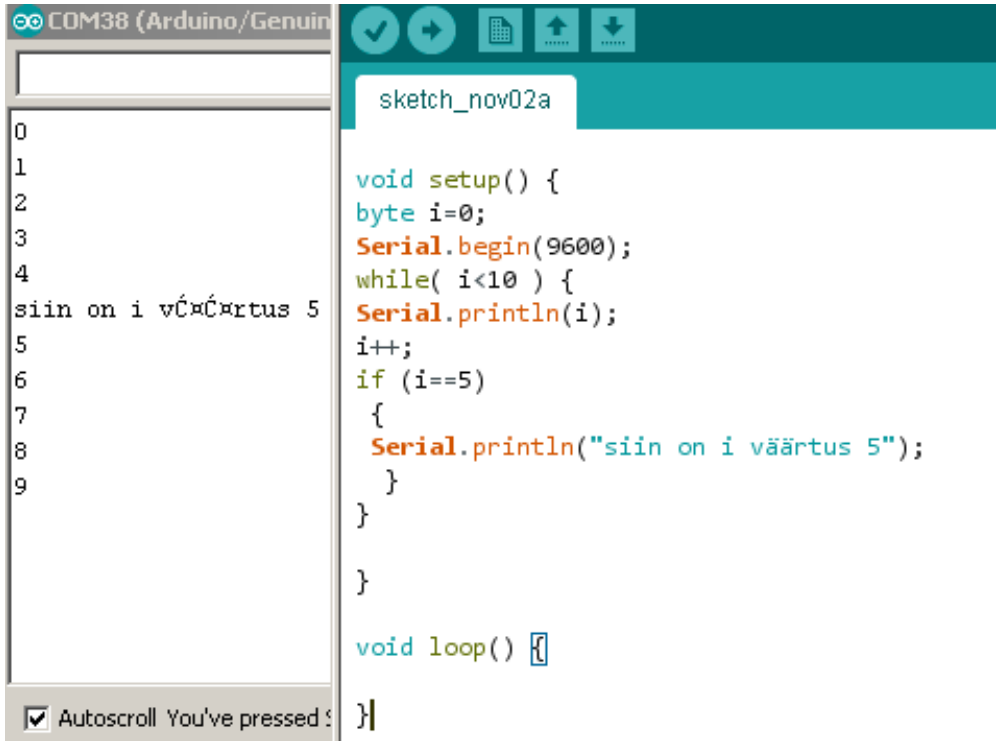
- Näited



# Programmi juhtimine (valiklause)

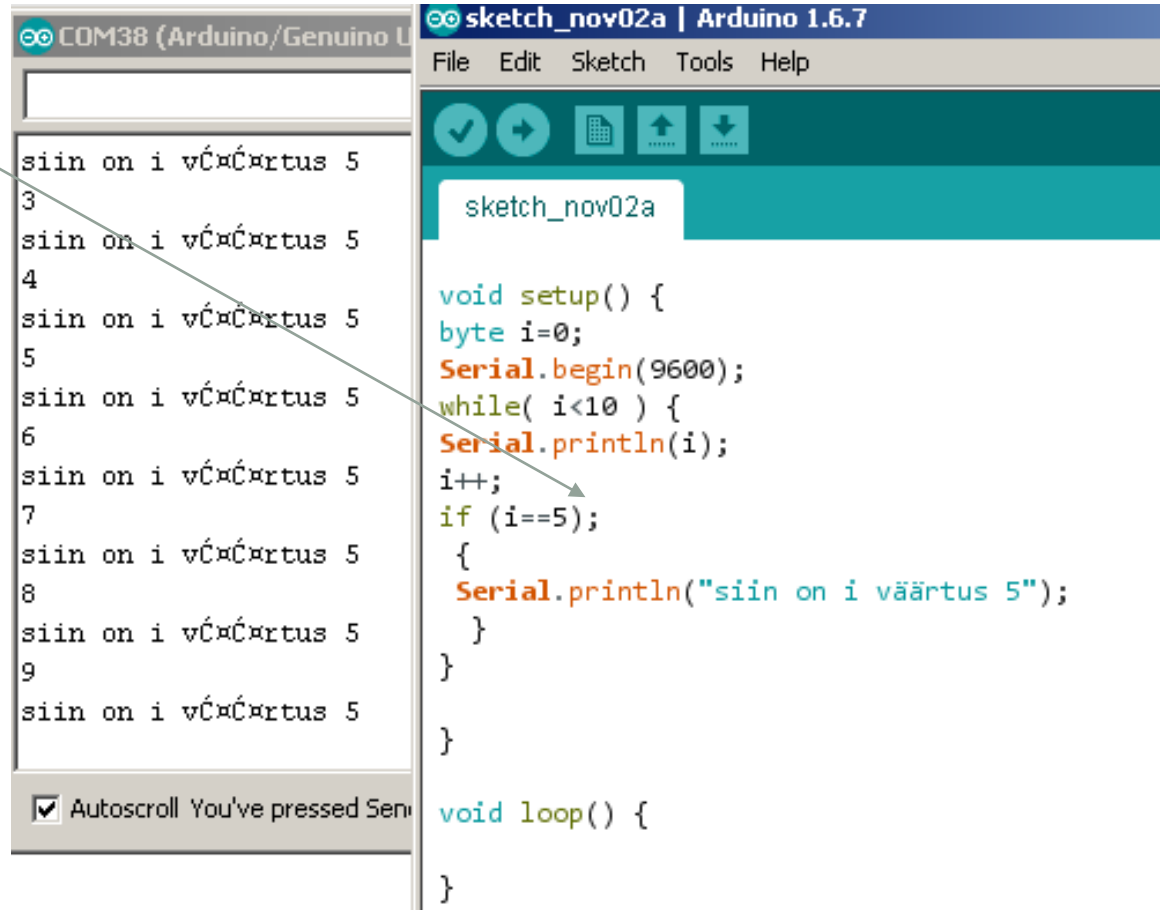
- if (tingimus)
  lause1
  else
  lause 2

Kui lauset 2 ei ole vaja võib else ära jätta.



# Programmi juhtimine  (valiklause)

- Pärast tingimust semikoolonit panna ei tohi , vastasel juhul täidetakse tühi lause ! C keele reeglite poolest on asi korras .



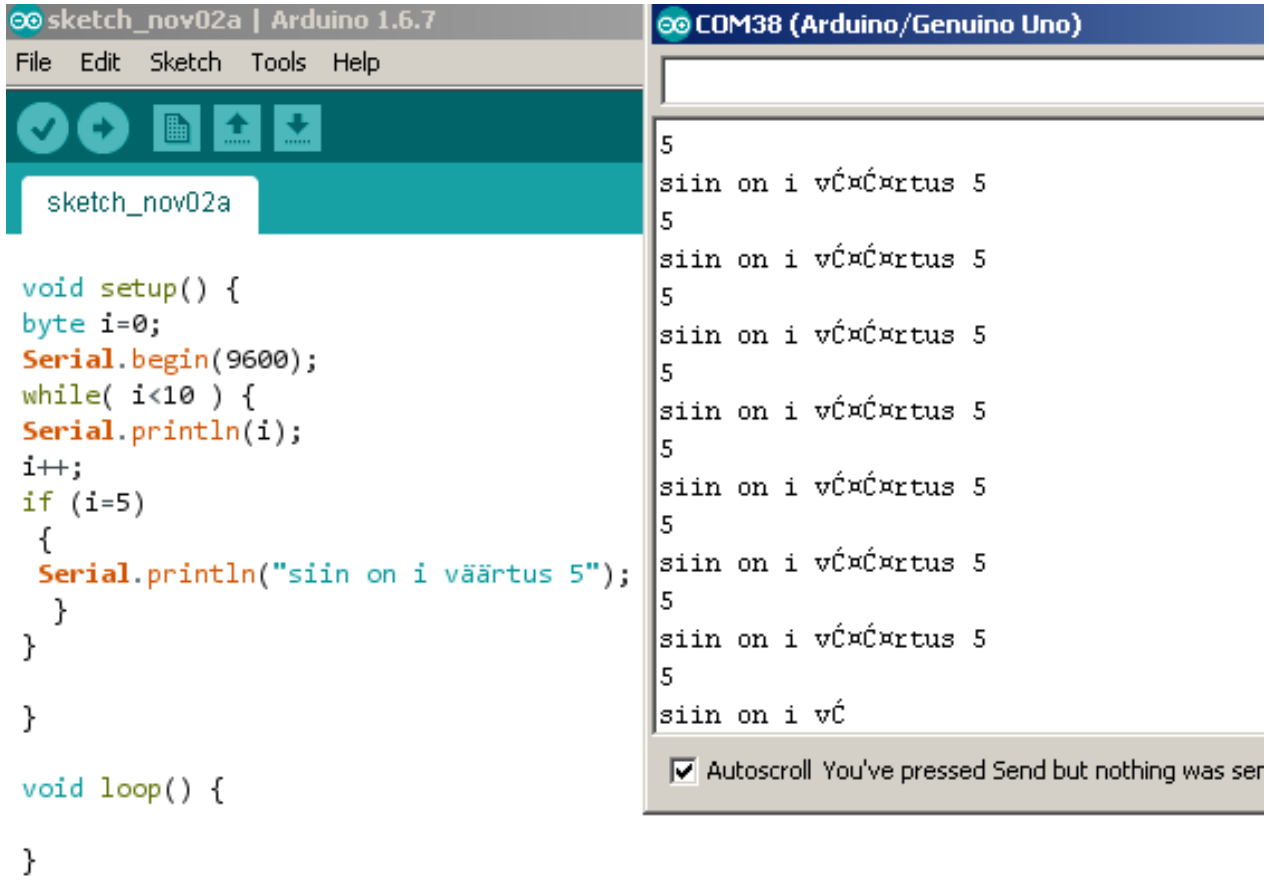

Ma teen kõik õigesti aga.....

Arvuti ei täida meie soove vaid meie käske !

# Programmi juhtimine  (valiklause)

- Teine tüüpiline loogikaviga (ka kogenud inimestel):  == asemele pannakse =
- Arvuti teeb, mis kästakse, i=5 ja tehte tulemus on tõene...

Vähegi pikemast koodist (eriti kui selle teinud on võõras  - “aga miks ei tööta, tegin taas kõik õigesti “) seda üles leida on väga raske !

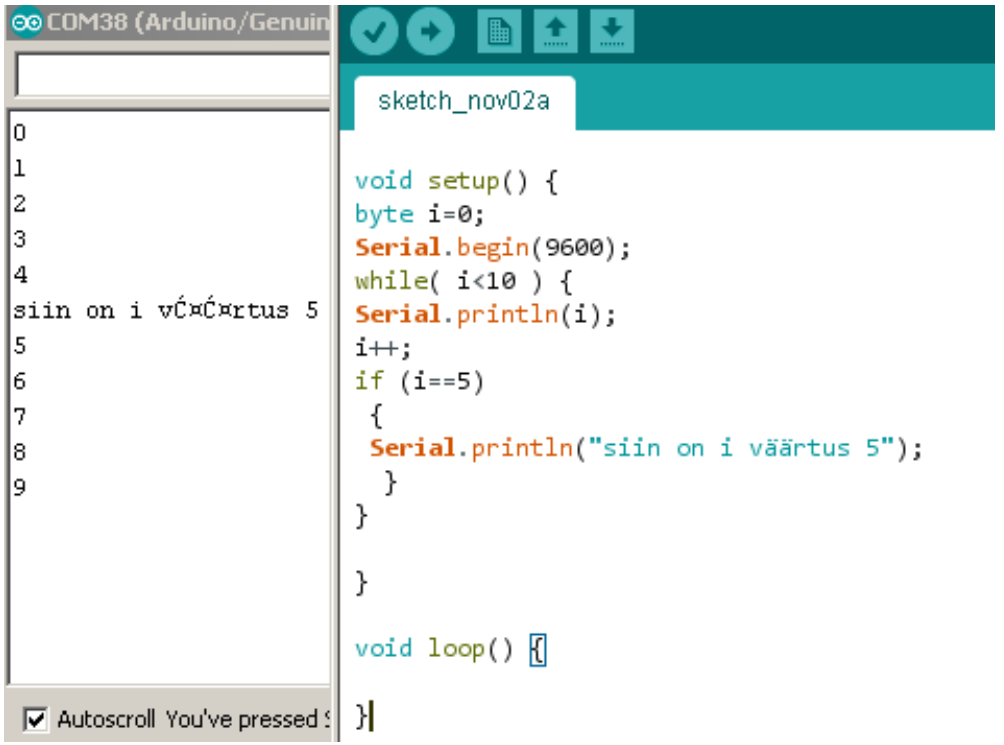


# Programmi juhtimine (kordus)

- while (tingimus)  lause
- Või do  lause  while (tingimus)
- Tingimuste korral samad probleemkohad, mis if lauses !
- Tsüklit saab katkestada

käsuga break;

Break katkestab vaid selle tsükli, mille sisse see kirjutatud on !



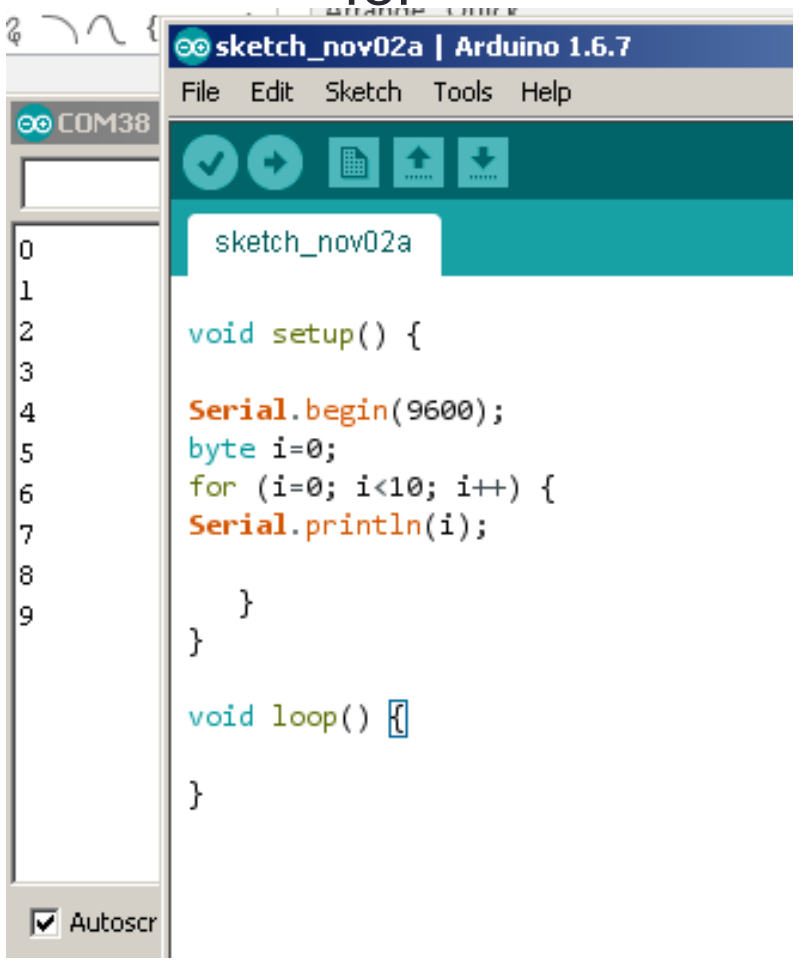
# Programmi juhtimine (loendus)

- for (avaldis1;avaldis2;avaldis3)  lause
- Laused võivad ka puududa kuid semikoolonid peavad olema. Avaldis 3 järele semikoolonit ei panda !
- Kõigepealt täidetakse ettevalmistatav avaldis1, siis täidetakse lauset niikaua kuni avaldis2 annab mittenullise tulemuse,  siis täidetakse avaldis3.
- Olemuselt teeb sama asja „while“
- avaldis1 while (avaldis2) {lause avaldis3}

# Programmi juhtimine (loendus)

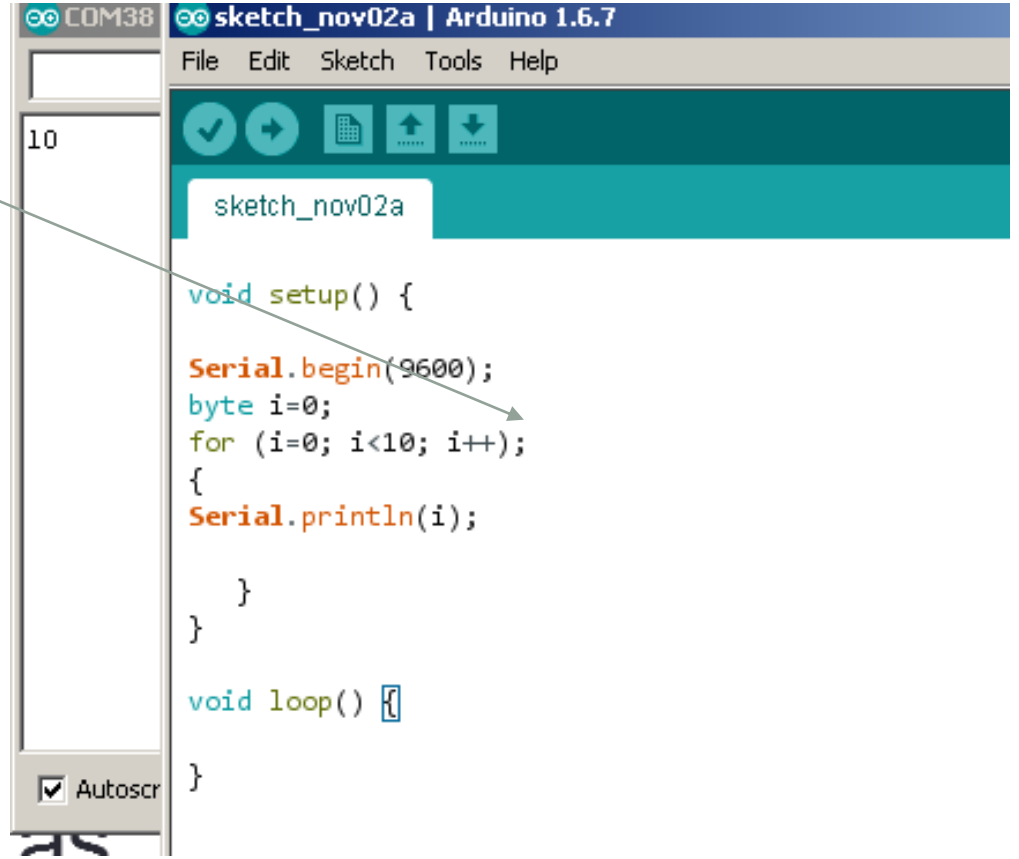
- Tulemus on sama.
- while


for



# Programmi juhtimine (loendus)

- Pärast for tsükli kirjeldust  semiloolonit panna ei tohi , vastasel juhul täidetakse vaid tühi lause ! C keele reeglite poolest on asi korras .
- Pärast tsükli täitmist on i väärtus 10, mis on korrektne , aga arvatavasti programmeerijale ebameeldiv .



# Definitsioonid

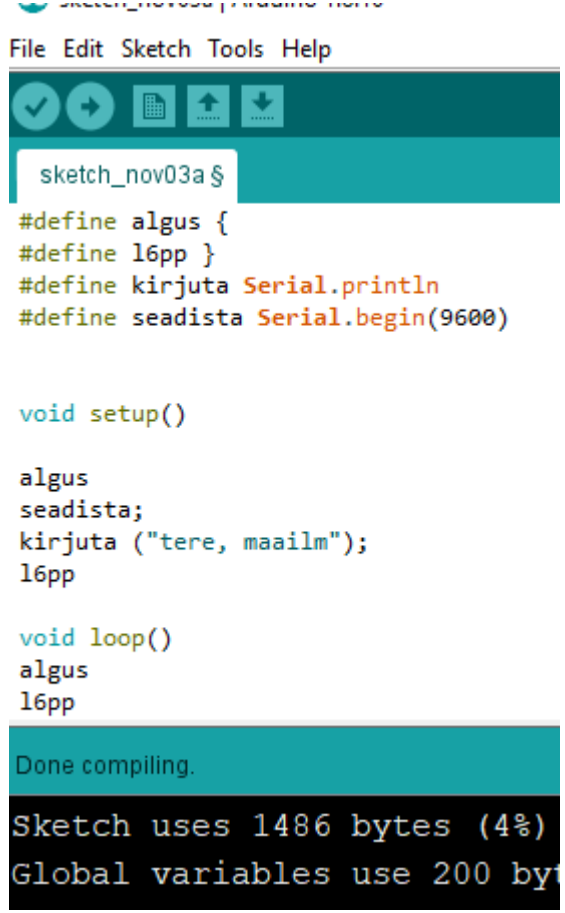
- Saab lihtsustada programmeerija tööd, saab teha koodi loetavamaks, muudetavamaks, erineva riistvara jaoks.

#define kartuleid 5 // igale poole kus on tekst “kartuleid” pannakse 5

#define porgandeid 12

#define doktorikraadiga_programmeerija Martin

- byte tulemus=kartuleid+porgandeid;
- byte tulemus=5+12; //sama mis eelmine rida
- Mugav, kui on vaja kasutada programmi mitmes kohas , lähteandmeid muutes.
- Definitsioon ei ole muutuja !
- Ehk kartuleid=kartuleid+3 ;
- on vigane lause (omistamistehe !)




Saab ka sedasi – Üks võimalikke viise, kuidas koodist arusaamine võõrastel äärmiselt ebamugavaks teha.

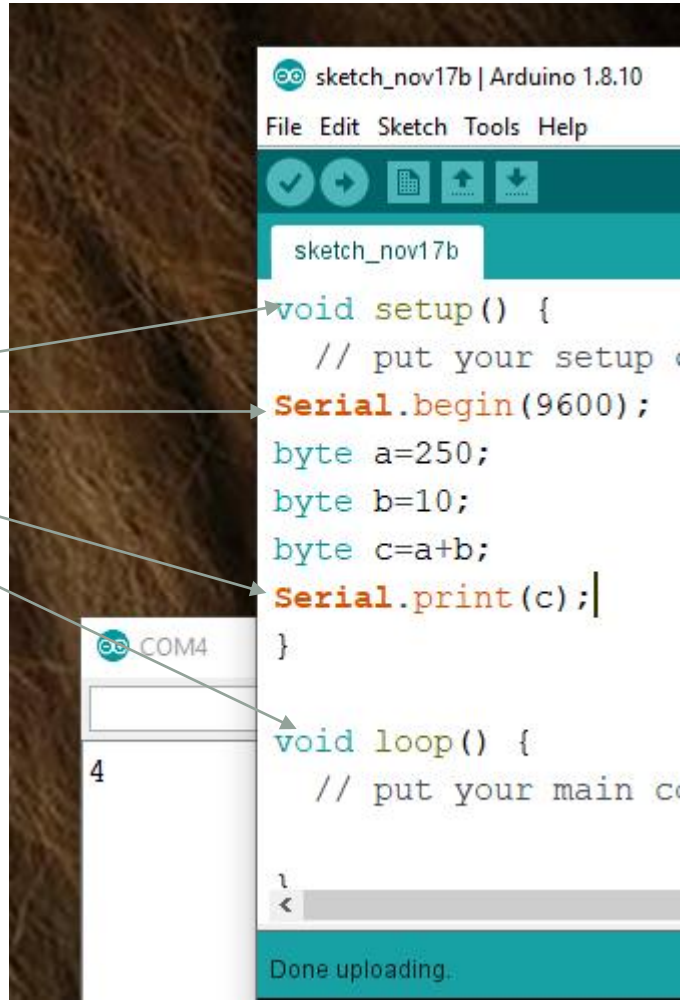
Arduino “keele mugavused” on ka suures osas definitsioonid.

# Funktsioon

- Oma olemuselt lausete kogum, millele on antud indentifikaator (nimi) ning sisendparameetrid
- Näide:

```
int MinuFuktsioon ( int sisend1, int sisend 2)
{
Int c=a+b;
return c;
}
```

funktsioonid




Tagastatav väärtus määratakse käsuga return ja saab olla vaid üks.

Kui funktsioon midagi ei tagasta (või pole sisendit), pannakse tüübiks void

C reeglitele vastav funktsioon void teemidagi (void) { }

Enamus Arduino „lisasid“ on teekidesse kirjutatud valmisfunktsioonid (oranži värvi) .....pinMode...digitalWrite....analogRead ....

# Näide - analogRead

- Kasutaja kirjutab a=analogRead(A0);
- Aga kutsutakse välja tegelikult see:
- C:\Program Files (x86)\Arduino\hardware\arduino\avr\cores\arduino\wiring_analog.c
- See on vaid koodi algus.

```
38  int analogRead(uint8_t pin)
39  {
40      uint8_t low, high;
41
42  #if defined(analogPinToChannel)
43  #if defined(__AVR_ATmega32U4__)
44      if (pin >= 18) pin -= 18; // allow for channel or pin numbers
45  #endif
46      pin = analogPinToChannel(pin);
47  #elif defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
48      if (pin >= 54) pin -= 54; // allow for channel or pin numbers
49  #elif defined(__AVR_ATmega32U4__)
50      if (pin >= 18) pin -= 18; // allow for channel or pin numbers
51  #elif defined(__AVR_ATmega1284__) || defined(__AVR_ATmega1284P__) ||
52      if (pin >= 24) pin -= 24; // allow for channel or pin numbers
53  #else
54      if (pin >= 14) pin -= 14; // allow for channel or pin numbers
55  #endif
56
57  #if defined(ADCSRB) && defined(MUX5)
58      // the MUX5 bit of ADCSRB selects whether we're reading from char
59      // 0 to 7 (MUX5 low) or 8 to 15 (MUX5 high).
60      ADCSRB = (ADCSRB & ~(1 << MUX5)) | (((pin >> 3) & 0x01) << MUX5);
61  #endif
```