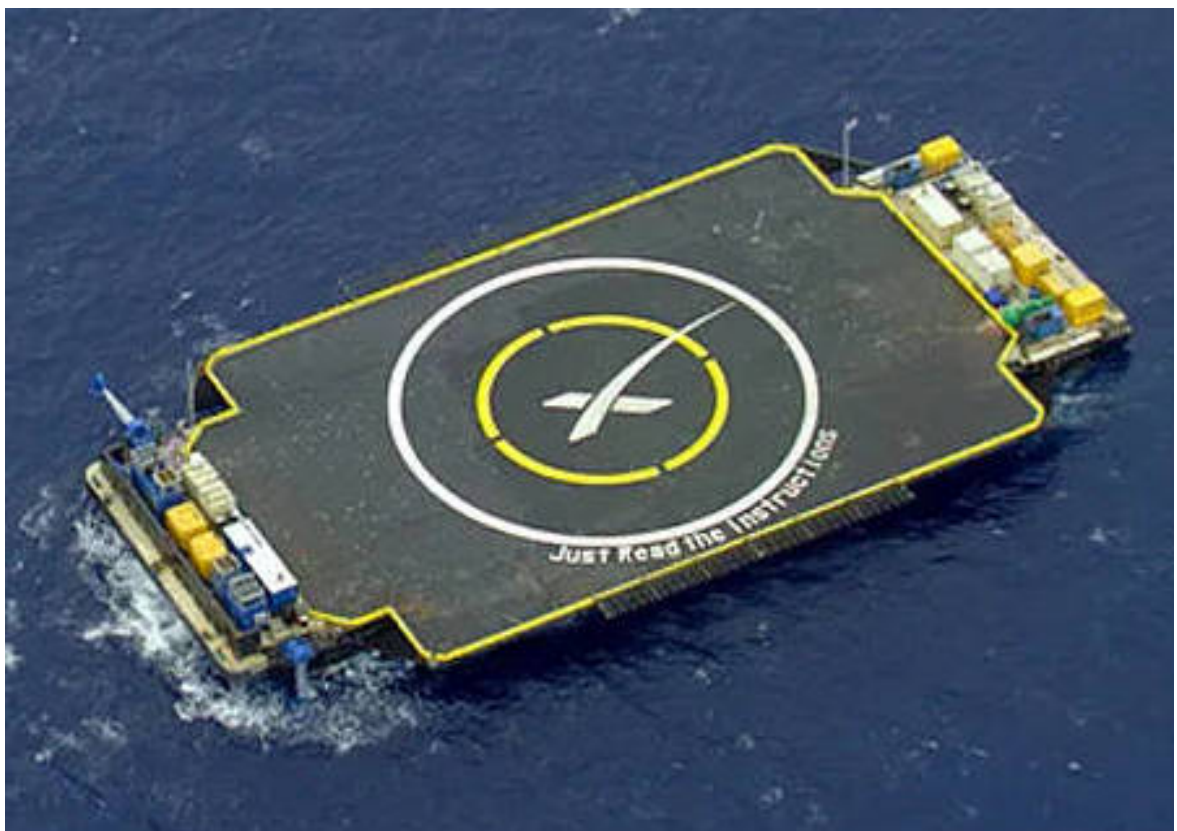


Latvijas Universitāte
Fizikas, matemātikas un optometrijas fakultāte

Arduino programmēšana



Saturs

1	No spīddiodes līdz simbolam	1
1.1	Arduino	1
1.2	Prmais piemērs: kompilējam un uzlādējam	2
1.3	Otrais piemērs: Mainām mirgošanas parametrus	3
1.4	Arduino programmas izpildes gaita	4
1.5	Vēlreiz programma Blink	5
1.6	Septiņu segmentu indikators	7
1.7	Pieslēdzam un pārbaudam	7
1.7.1	Izvadām simbolus uz indikatora	9
1.7.2	Parkārtojam programmu ērtai lietošanai	11
1.8	Programmas dalījums vairākos failos, Arduino IDE stils	12
	Literatūra	14
2	Bibliotēkas	17
2.1	Attāluma mērīšana izmantojot ultraskaņas attāluma sensoru	17
2.2	Servomotora vadība	20
	Literatūra	21

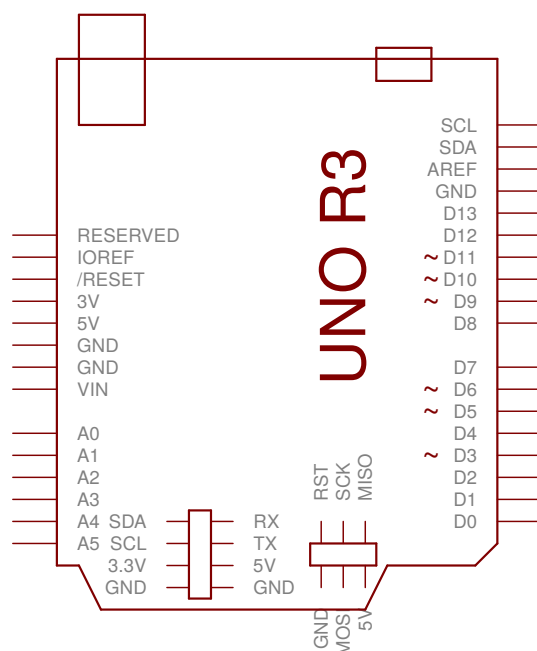
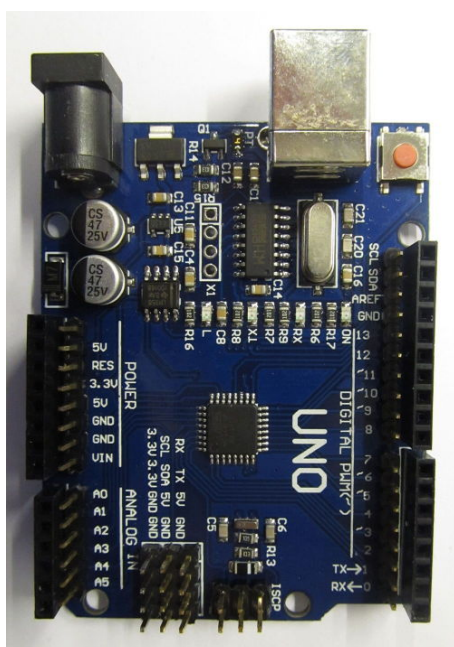
Nodaļa 1

No spīddiodes līdz simbolam

1.1 Arduino

Arduino [1] ir atvērtā pirmkoda elektronikas un programmatūras datorplatforma, kas izveidota lai ļautu lietotājam viegli savienot dažnedažādus sensorus ar neliela izmēra datoru un viegli uzrakstīt atbilstošo programmatūru. Piemēram, klasiskā Arduino Uno [2] izmēri ir 69×54 mm, tam ir 6 ieejas, kuras var nolasīt analogos signālus un 14 ciparu signāliem atbilstošas ieejas/izejas. Atvērtā pirmkoda elektronika nozīmē, ka visiem interesentiem ir brīvi pieejama Arduino Uno elektriskā shēma [3]. Savukārt programmatūru sākot no programmu izstrādes rīka Arduino IDE [4] līdz pat dažnedažādām bibliotēkām [5] var lietot bez maksas, kā arī ir brīvi pieejams visu šo programmu pirmkods. Atvērtā pirmkoda izmantošana ļauj Arduino platformu lietot kā uzzīņas līdzekli vai mācību materiālu elektronikā un programmēšanā, atvieglo Arduino savietojamu perifērijas iekārtu un programmatūras izstrādi. Bez tam, nepārkāpjot licences nosacījumus (parasti tā ir Creative Commons Attribution-ShareAlike 3.0 License [6]) šos materiālus var izmantot savu konstrukciju veidošanā. Tā, piemēram, ir radies Arduino Uno R3 *savietojamais* dators, kas redzams 1.1 zīmējumā un kuru izmantosim

Att. 1.1: Arduino Uno R3 *savietojamais* dators un tā apzīmējums kursa materiālos dotajās shēmās



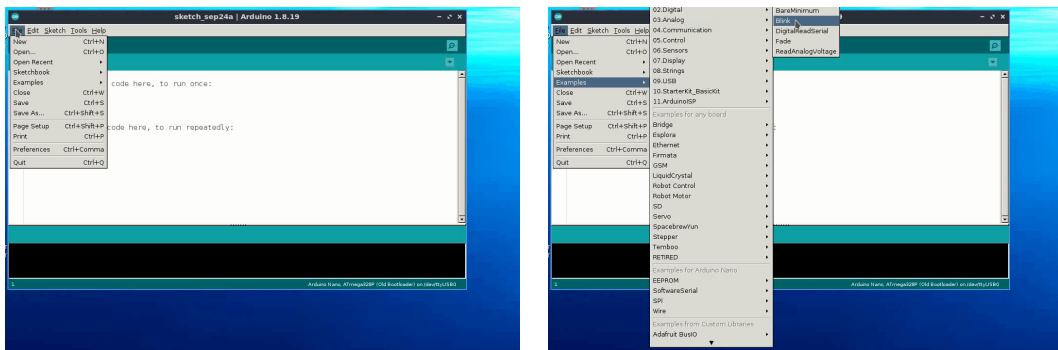
šajā kursā. Galvenā atšķirībā no Arduino Uno R3, kuras dēļ ir izvēlēts konkrēti šis dators, ir papildus tapīņu tipa spraudņu rindas.

Programmatūra Arduino projektiem parasti tiek izstrādāta izmantojot Arduino IDE [4]. Tā vienlīdz labi strādā gan Windows, gan Macintosh OSX, gan arī dažādās LINUX sistēmās. Tiek izmantota programmēšanas valodas C++ variants [7]. Kursa ietvaros pēc iespējas centīsimies izvairīties no tiem brīnišķīgajiem risinājumiem, kurus piedāvā C++ un centīsimies gūt pamatprasmes darbā ar programmēšanas valodu C. Šīs valodas *patstāvīgai* padziļinātai apguvei iesaku izmantot sekojošas grāmatas [8, 9], kuras ir pieejamas LU FMF bibliotēkā.

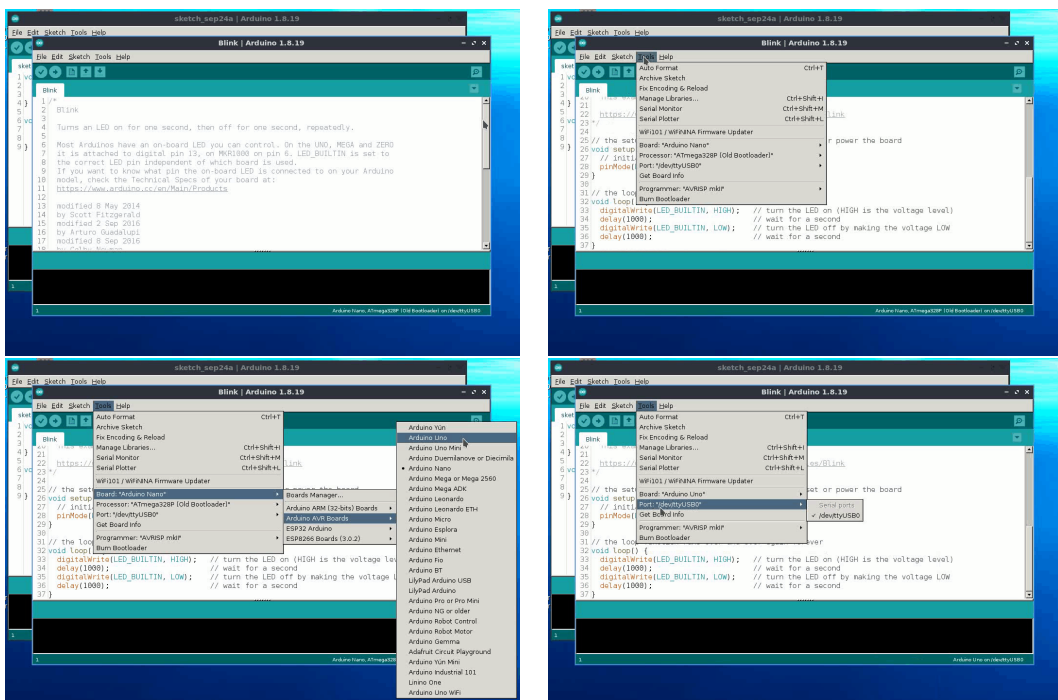
Kā alternatīvu programmu izstrādē var lietot s4a [10] kas ir vizuālās programmēšanas līdzekļa Scratch atvasinājums vai arī pašu ScratchX [11].

1.2 Prmais piemērs: kompilējam un uzlādējam

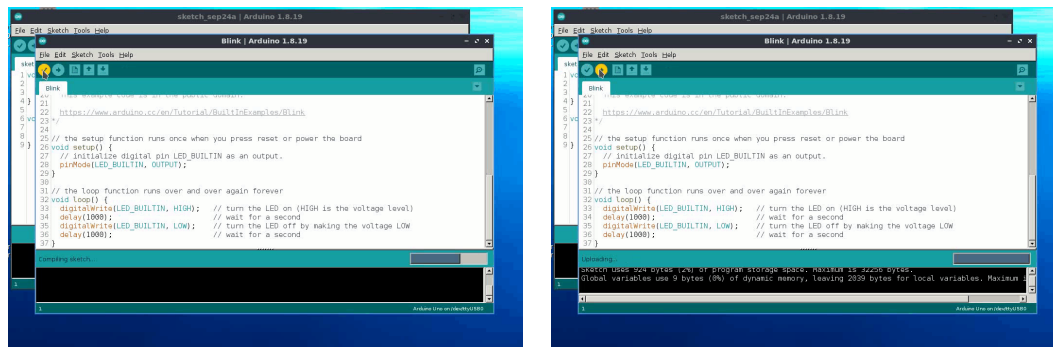
1. Atveram *Arduino IDE* un meklējam piemēru Blink



2. Atvērās! Pārslēdzam *Arduino IDE* darbam ar *Arduino UNO*, pārbaudam, ka tas ir pieslēgts



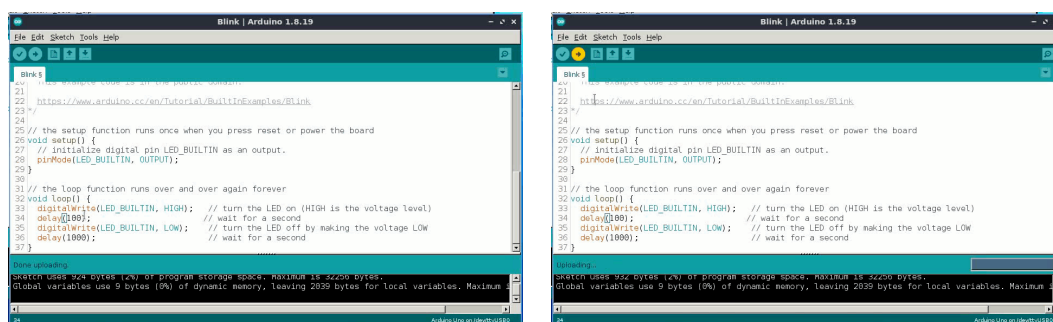
3. Pārveidojam šo piemēru no cilvēkam saprotama teksta *Arduino UNO* saprotamā binārā datu formā. Pēc tam ielādējam to *Arduino UNO*



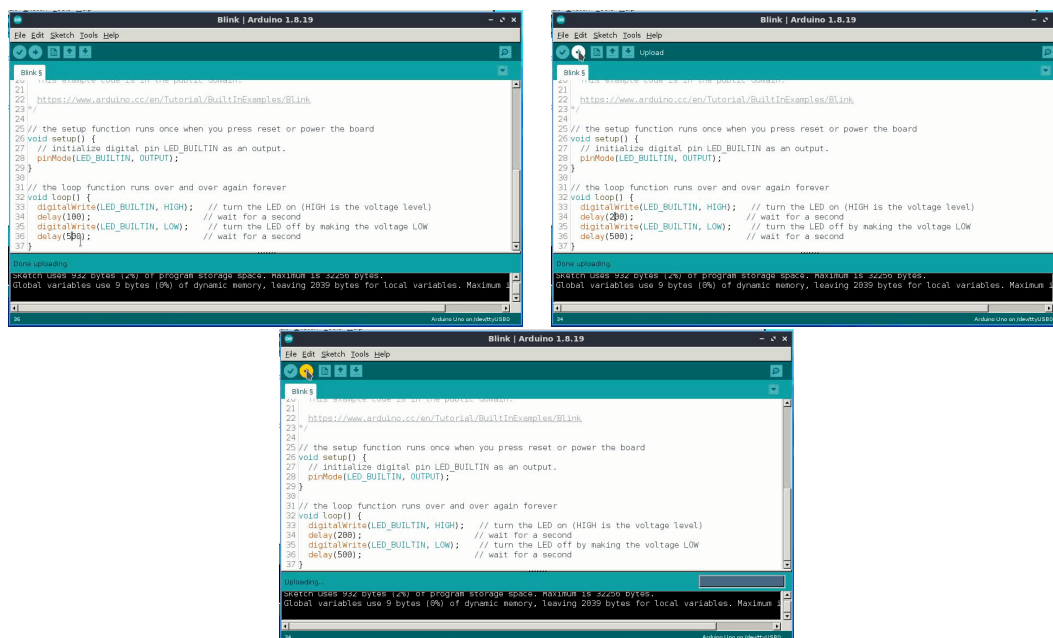
4. Saktāties un redzam, kā ieslēdas un izslēdas spīddiode uz *Arduino UNO*! Urrā, strādā!

1.3 Otrais piemērs: Mainām mirgošanas parametrus

1. Redaktorā 34 rindiņā mainām no 1000 uz 100. Kompilējam un novērojam



2. Redaktorā 36 rindiņā mainām no 1000 uz 500 un 34 rindiņā no 100 uz 200. Kompilējam un novērojam



3. *Arduino UNO*! Urrā, tas nav nekas tik briesmīgs!

1.4 Arduino programmas izpildes gaita

Datora programma ir binārais kods, kas atrodas tā atmiņā un kuru tā procesors "saprot" un spēj izpildīt. Nedaudz vienkāršojot, Arduino gadījumā to var sadalīt vairākās daļās:

1. Daļa, kuru uzraksta lietotājs konkrētā projekta vajadzībām;
2. Ja Arduino pieslēgtas kādi sensori vai citas iekārtas, tad ar tiem parasti mijiedarbojas izmantojot iepriekš izstrādātu programmatūru - bibliotēkas [5]. Tās bez izmaiņām lieto visos projektos, kur tiek izmantota atbilstošā iekārta un programmas binārajā kodā to iekļauj tikai tad, kad vajag. Līdzīgi rīkojas izmantojot sarežģītus un specifiskus algoritmus. Tātad šī daļa mainās atkarībā no projekta, bet, atšķirībā no pirmās daļas, netiek mainīta konkrētā projekta izstrādes laikā.
3. Pārējā daļa, kas kopīga visiem Arduino projektiem

Nedaudz ielūkosimies pēdējā daļā. Lai veiktu kaut vai pašu vienkāršāko operāciju - ieslēgtu mirdzdiodes, procesoram jāveic vairākas operācijas. Lai zinātu kādas tieši, lieliski jāorientējas gan procesora dokumentācijā, kuras apjoms bieži vien pārsniedz 400 lapaspuses [12], gan arī paša Arduino elektriskajā shēmā [3]. Pat pieredzējušam programmētājam tas nav viegli! Bez tam, dažādas Arduino variācijās tiek izmantoti dažādi procesori

- Uno, Nano, Mini izmanto jau iepriekš minēto Atmel ATmega328 procesoru [12].
- Due izmanto principiāli atšķirīgo Atmel SAM3X8E ARM Cortex-M3 CPU;
- Intel Galileo izmanto Intel Quark SoC X1000 Application Processor

Kā redzams, pat procesora dokumentācijas lieliska pārzināšana nepalīdz, jo procesori mēdz būt dažādi! Tāpēc, lai lietotajam būtu vieglāk, ir lietderīgi mirdzdiodes ieslēgšanai izveidot atbilstošu *apakšprogrammu*, *funkciju*, realizēt to katram procesoram un iekārtai kā nepieciešams, bet saukt to vienādi visos gadījumos. Šo te no Arduino modeļa atkarīgo programmatūras daļu kopā ar iepriekš minēto pirmo un otro daļām ar Arduino IDE [4] palīdzību sagatavo procesoram saprotamā veidā un aukšuplādē tajā. Savukārt programmatūras daļa kas rūpējas par to, ko procesoram darīt pēc barošanas ieslēgšanas (auksts restarts), pēc RESET pogas nospiešanas (siltais restarts), kā arī saņem aukšuplādēto kodu un pēc tam to palaiž, vienmēr atrodas Arduino energoneatkarīgajā atmiņā. Arī aukšuplādētā programma nonāk energoneatkarīgajā atmiņā.

Pirms turpinām, nedaudz par *apakšprogrammām* jeb *funkcijām*. Pieņemsim, ka programmā vairākkārt nepieciešams atrast sin x vērtības. Aprēķināsim to izmantojot pakāpju rindas pirmos trīs locekļus

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots = x \left[1 - \frac{x^2}{6} \left(1 - \frac{x^2}{20} \right) \right] + \dots \quad (1.1)$$

Funkcija, kas veic šos aprēķinus ir sekojoša

Listing 1.1: Sin3.ino

```
1 float sin3( float x )
2 {
3   return x*(1.-x*x/6.*(1.-x*x/20.));
4 }
```

Var teikt ka funkcija ir vairākkārtīgai lietošanai *gramatiski pareizi* noformēta programmas daļa, kas ideālā gadījumā ir neatkarīga no pārējās programmas:

- funkcijai ir vārds. Šinī gadījumā `sin3`
- funkcijai var nodot parametrus. Šinī gadījumā *skaitli ar peldošo komatu* `x`. Šādus parametrus sauc par funkcijas argumentiem.
- funkcija var *atgriezt* savas darbības rezultātu. Šinī gadījumā pēc formulas (1.1) aprēķināto $\sin x$ vērtību
- Pats programmas teksts ievietots figūriekavās `{ }`.
- Atbilstoši C/C++, katra programmas rindiņa noslēdzas ar semikolu `;`
- Atgriežamo vērtību raksta pēc vārdiņa `return`.
- Atbilstoši C++, vienu funkciju no otras atpazīst pēc funkcijas vārda, tās argumentu skaita un to tipiem.

Pati vienkāršāka programma, kuru var uzrakstīt ar Arduino IDE palīdzību un kura nedara neko no lietotāja viedokļa, ir šāda

Listing 1.2: Empty.ino

```
1 void setup() {  
2   // put your setup code here, to run once:  
3  
4 }  
5  
6 void loop() {  
7   // put your main code here, to run repeatedly:  
8  
9 }
```

Nokompilēta tā aizņem 444 baitu. Šis lielums var mainīties atkarībā no Arduino iekārtas kā arī no izmantotās Arduino IDE versijas. Kā viss notiek pēc tam, kad programma ir nonākusi Arduino atmiņā, pēc barošanas ieslēgšanas vai RESET pogas nospiešanas?

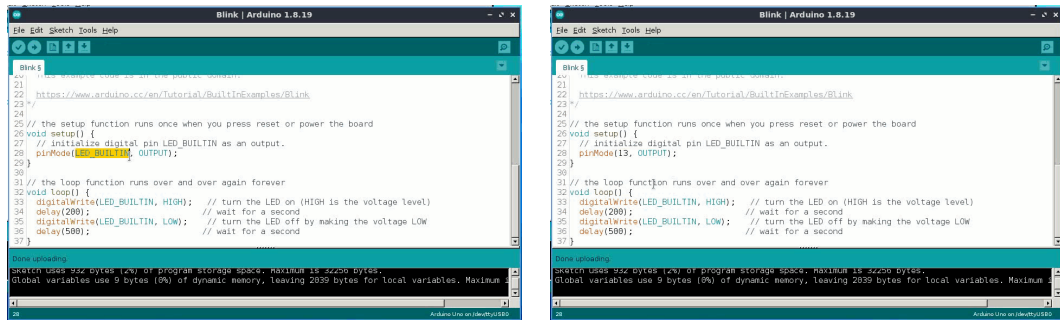
1. Veic operācijas darbības uzsākšanai
2. Izpilda funkciju `setup()` no lietotāja programmas
3. Turpina sagatavošanās operācijas
4. Izpilda funkciju `loop()` no lietotāja programmas
5. Veic virkni dažādu servisa operāciju
6. Atgriežas uz punktu 4

Uzdevums: Izmantojot nodarbības laikā gūtās zināšanas, padomājiet kā varētu nomērīt 1, 3 un 5 posma darbības laikus!

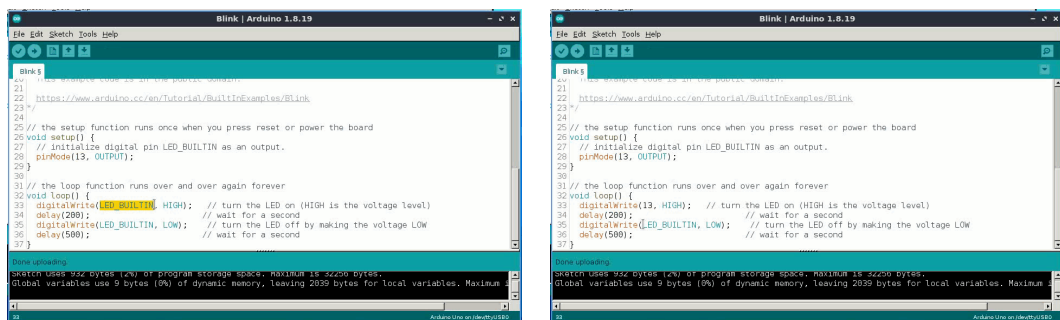
1.5 Vēlreiz programma Blink

Arduino programmēšanu parasti sāk ar programmu Blink, kas ieslēdz un izslēdz iebūvēto spīddiodi. *Arduino UNO* spīddiode pievienota 13 digitālajam izvadam. Izdarīsim šīs izmaiņas

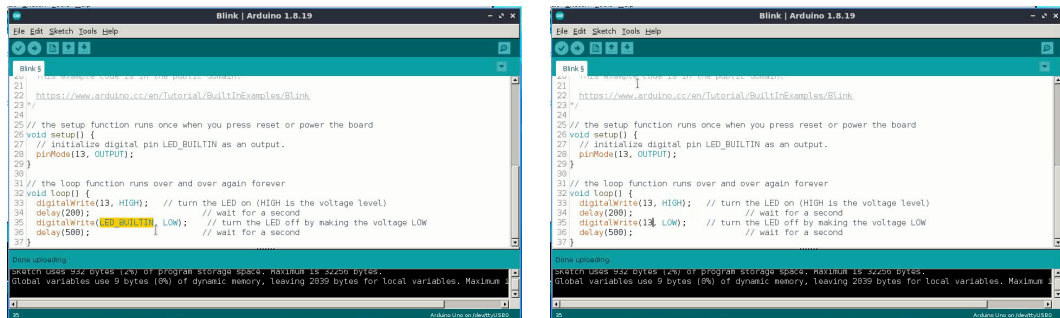
1. Redaktorā 28 rindiņā mainām LED_BUILTIN par 13



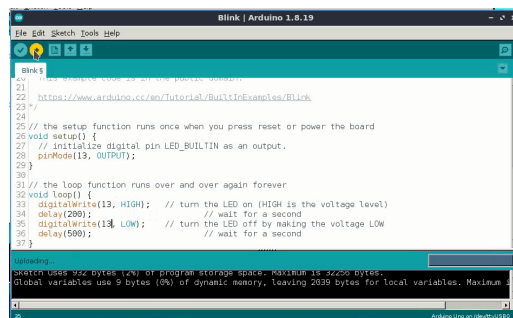
2. Redaktorā 33 rindiņā mainām LED_BUILTIN par 13



3. Redaktorā 35 rindiņā mainām LED_BUILTIN par 13



4. Kompilējam un redzam, ka nekas nemainās!



Tālāk skatīsimies šo pašu programmas kodu listinga veidā. Lai ietaupītu vietu, izdēsu visu programmas galvu (lai man piedod programmas autori). Komentārus pārrakstīju latviski un sanāca tikai 10 rindiņas.

Listing 1.3: Blink.ino

```

1 void setup() {
2   pinMode(13, OUTPUT);    // Didgitalo liniju 13 ieslēdz par izeju
3 }
4
5 void loop() {
6   digitalWrite(13, HIGH); // ieslēdz mirdzdiodi (HIGH ir sprieguma līmenis)
7   delay(1000);            // gaida vienu sekundi
8   digitalWrite(13, LOW);  // izslēdz mirdzdiodi (LOW ir sprieguma līmenis)
9   delay(1000);            // gaida vienu sekundi
10 }
```

Analizējam šo kodu! Te izmantotas trīs *funkcijas*

`pinMode(pin, mode)` - konfigurē norādīto digitālo līniju vai nu par ieeju vai arī par izeju, [13]:

pin: digitālās līnijas numurs

mode: ja OUTPUT, tad izeja, ja INPUT vai INPUT_PULLUP tad ieeja.

`digitalWrite(pin, value)` - norādīto pinu uzstāda zemā vai augstā līmenī, [14]

pin: digitālās līnijas numurs

value: spriegums digitālās līnijas izejā, LOW vai HIGH

`delay(ms)` - aptur programmas izpildi uz noteiktu laiku [15];

ms: laiks milisekundēs no 0 līdz $2^{32} - 1$. Tātad ilgākais laiks ir apmēram 50 diennaktis!

1.6 Septiņu segmentu indikators

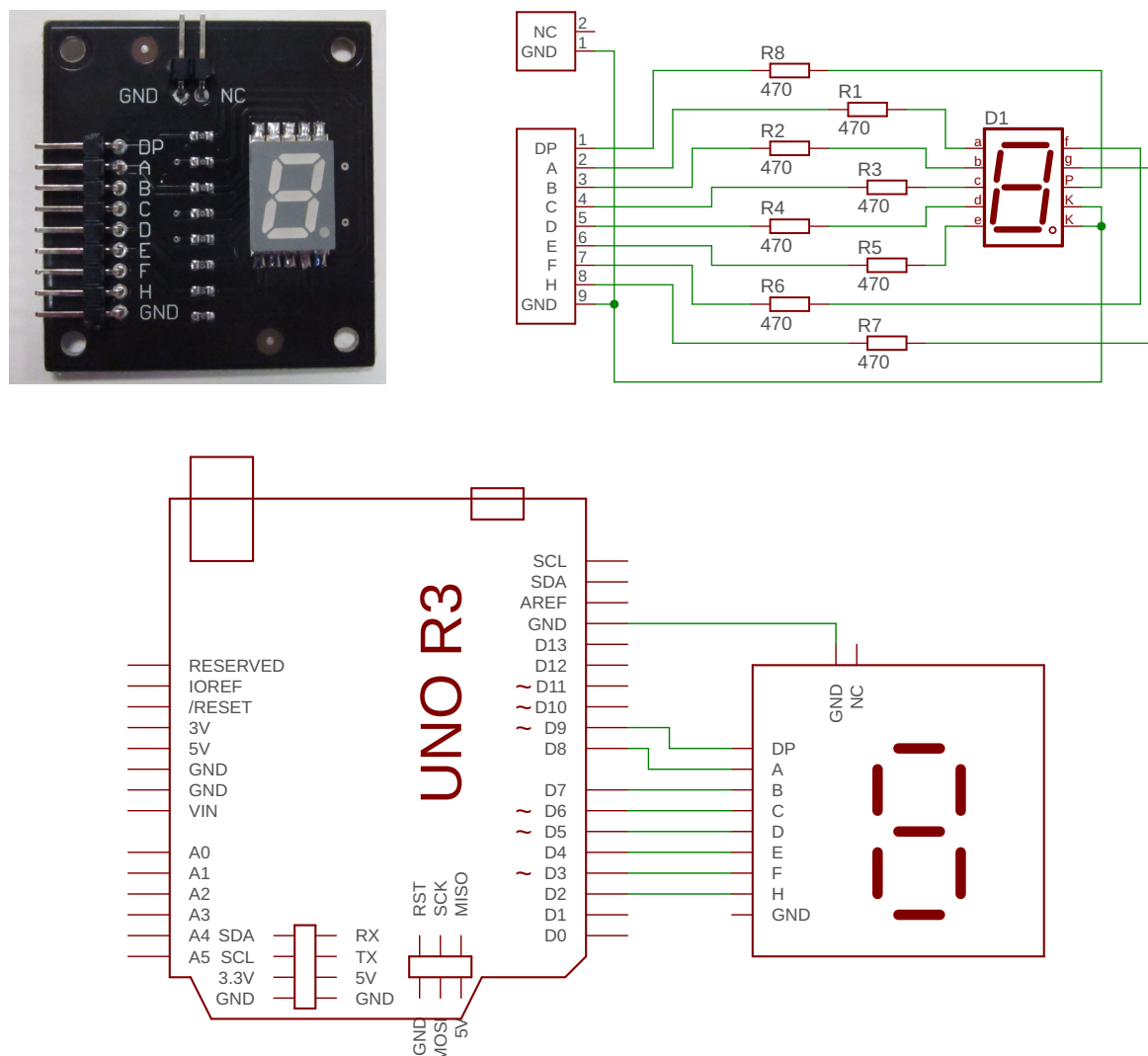
1.7 Pieslēdzam un pārbaudam

Ciparu un dažu burtu attēlošanai lieto 7 segmentu indikatorus [16]. Mūsdienās tas vienkārši ir 8 mirdzdiodes, kas izvietotas kā redzams zīmējumā 1.2. Simboli veidojas pareizi ieslēdzot un izslēdzot *īstās* mirdzdiodes.

Darbam ar Arduino izveidots indikatora panelis, kura elektriskā shēma un fotogrāfija dota 1.3 zīmējuma augšā. Šā paša zīmējuma apakšā parādīts, kā to pieslēgt Arduino. Veiciet šādus **uzdevumus**, savai programmai par pamatu ņemot Blink programmu 1.3



Att. 1.2: Septiņu segmentu indikators, segmentu kodējums



Att. 1.3: Septiņu segmentu indikatora panelis, tā shēma un savienojums ar Arduino.

Ņemam jau iepriekš lietoto programmu *Blink* no P.1.3 un nomainām tās tekstā digitālo izeju 13 uz 9.

Listing 1.4: Blink2.ino

```
1 void setup() {
2   pinMode(9, OUTPUT);    // Didgitalo liniju 9 ieslēdz par izeju
3 }
4
5 void loop() {
6   digitalWrite(9, HIGH); // ieslēdz mirdzdiodi (HIGH ir sprieguma limenis)
7   delay(1000);           // gaida vienu sekundi
8   digitalWrite(9, LOW);  // izslēdz mirdzdiodi (LOW ir sprieguma limenis)
9   delay(1000);          // gaida vienu sekundi
10 }
```

Kompilējam un novērojam! Tagad būtu interesanti izmēģināt visus segmentus lietojot ciparu izejas 8, 7, 6, 5, 4, 3, un 2. Tā kā cipari jāmaina 3 vietās, tad būtu lietderīgi izdomāt kautkā tā, lai ietaupītu šo darbu un izslēgtu muļķīgas kļūdas, kad daļu ciparu nomaina un daļu aizmirst nomainīt. Šim nolūkam izmantojam *mainīgo*, kurā noglabā ciparu izejas

numuru, kas ir vesels skaitlis. Skatamies kodu. Mainīgais `pinLed` definēts *ārpus* abām funkcijām tāpēc, lai tas būtu pieejams *abām* funkcijām.

Listing 1.5: Blink3.ino

```
1 int pinLed = 8;
2
3 void setup() {
4   pinMode(pinLed, OUTPUT);
5 }
6
7 void loop() {
8   digitalWrite(pinLed, HIGH);
9   delay(1000);
10  digitalWrite(pinLed, LOW);
11  delay(1000);
12 }
```

Tā kā `pinLed` programmas izpildes laikā nemainās, tad liederīgi to deklarēt kā `const`. Ieviesīsim vēl vienu mainīgo `halfPeriod`, kas būs vienāds ar mirgošanas pusperiodu. Pie kam, pēc katra perioda to palielinām.

Listing 1.6: Blink4.ino

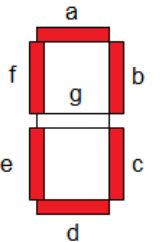
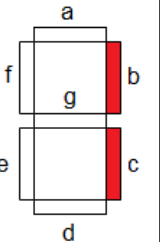
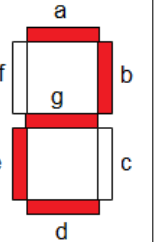
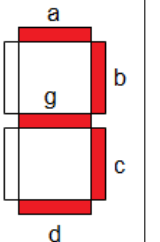
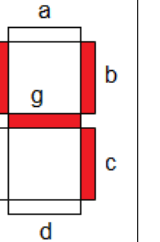
```
1 const int pinLed = 9;
2
3 void setup() {
4   pinMode(pinLed, OUTPUT);
5 }
6
7 void loop() {
8   static int halfPeriod = 100;
9
10  digitalWrite(pinLed, HIGH);
11  delay(halfPeriod);
12  digitalWrite(pinLed, LOW);
13  delay(halfPeriod);
14
15  halfPeriod = 1.05*halfPeriod;
16 }
```

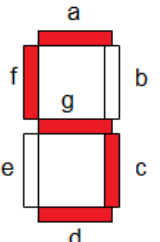
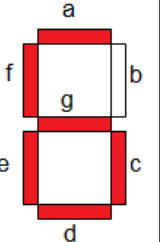
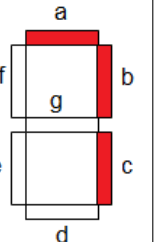
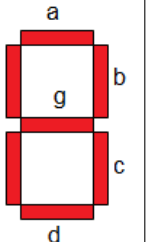
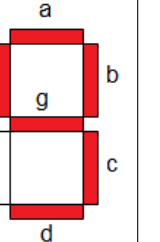
Šeit mainīgo `halfPeriod` definējam kā `static`, jo savādāk katru reizi izsaucot `loop`, tam atkal piešķirs vērtību 100.

- Izmēģinām programmu P.1.6 un novērojam.
- Mainām `halfPeriod` sākuma vērtību rindā 8 un novērojam.
- Mainām koeficientu 1.05 rindā 15 un novērojam.
- Kas jādara, lai mirgošanas periods samazinātos?
- Izdzēšam atslēgas vārdu `static` rindā 8 un novērojam.

1.7.1 Izvadām simbolus uz indikatora

Talāk mēģināsim izveidot simbolus uz indikatora. Vispirms aizpildām tabulu 1.1 katram simbolam no 0 līdz 9 norādot kuri no segmentiem ir jāieslēdz (HIGH) un kuri ir jāizslēdz (LOW).

						
Izeja	Segments					
8	A	HIGH	LOW			
7	B	HIGH	HIGH			
6	C	HIGH	HIGH			
5	D	HIGH	LOW			
4	E	HIGH	LOW			
5	F	HIGH	LOW			
2	G	LOW	LOW			

						
8	A					
7	B					
6	C					
5	D					
4	E					
5	F					
2	G					

Tabula 1.1: Segmentu izvietojums indikatorā. Simbolu veidošanas tabula

Uzrakstīsim programmu, kas pēc kārtas ar sekundes intervālu rāda simbolus no 9 līdz 0. Kā to izdarīt? Vispirms lietderīgi nodefinēt visus segmentu pinus. Tā būs ērtāk strādāt ar iepriekš aizpildīto tabulu.

Listing 1.7: Seg7Smart.ino

```

1 const int pinA = 8;
2 const int pinB = 7;
3 const int pinC = 6;
4 const int pinD = 5;
5 const int pinE = 4;
6 const int pinF = 3;
7 const int pinG = 2;
```

Tālāk Funkcijā setup() par izejām nokonfigurē 7 ciparu līnijas, kuras nodefinējām iepriekšējā fragmentā.

```

9 void setup() {
10   pinMode(pinA, OUTPUT);
11   pinMode(pinB, OUTPUT);
12   pinMode(pinC, OUTPUT);
13   pinMode(pinD, OUTPUT);
```

```

14   pinMode(pinE, OUTPUT);
15   pinMode(pinF, OUTPUT);
16   pinMode(pinG, OUTPUT);
17 }

```

Talāk, funkciju loop() sākam ar ciparu 9

```

19 void loop() {
20   digitalWrite(pinA, HIGH); // 9
21   digitalWrite(pinB, HIGH);
22   digitalWrite(pinC, HIGH);
23   digitalWrite(pinD, HIGH);
24   digitalWrite(pinE, LOW);
25   digitalWrite(pinF, HIGH);
26   digitalWrite(pinG, HIGH);
27   delay( 1000 );

```

un beidzam ar ciparu 0

```

101  digitalWrite(pinA, HIGH); // 0
102  digitalWrite(pinB, HIGH);
103  digitalWrite(pinC, HIGH);
104  digitalWrite(pinD, HIGH);
105  digitalWrite(pinE, HIGH);
106  digitalWrite(pinF, HIGH);
107  digitalWrite(pinG, LOW);
108  delay( 1000 );
109 }

```

Izpildot šo uzdevumu, esam ieguvuši programmu samērā garu programmu. Pievērsiet uzmanību iekavu izvietojumam, programmas teksta izlīdzinājumam, tukšumiem!

1.7.2 Parkārtojam programmu ērtai lietošanai

Ko darīt, ja vēlamies attēlot ciparus citā kārtībā? Ir lietderīgi izveidot savu funkciju, kurai kā parametru nodod attēlojamo ciparu. Par pamatu izmantojam iepriekšējo programmu. Attēlosim jauno funkciju un atkal sākam ar ciparu 9.

Listing 1.8: Seg7Func.ino

```

19 void print( int cip ) {
20   if( cip == 9 ) {
21     digitalWrite(pinA, HIGH); // 9
22     digitalWrite(pinB, HIGH);
23     digitalWrite(pinC, HIGH);
24     digitalWrite(pinD, HIGH);
25     digitalWrite(pinE, LOW);
26     digitalWrite(pinF, HIGH);
27     digitalWrite(pinG, HIGH);
28   } else if( cip == 8 ) {

```

Kas jauns un iepriekš neredzēts ie šajā fragmentā? Vispirms, mūsu funkcijai ir parametrs cip, kas vesels skaitlis, par ko norāda īpašības vārds int tā priekšā. Otrā lieta ir zarošanās. Rindiņā 20 mēs redzam parbaudi vai parametra cip ir vienāda ar 9. Lai nesajauktu ar piešķiršanas operāciju = salīdzināšanai operāciju ir vienāds apzīmē ar ==. Tāpēc rindiņa 20 lasāma šādi: *ja taisnība, ka parametrs cip ir vienāda ar 9, tad izpildām programmas koda bloku, kas sākas ar { un beidzas ar }*. Šinī gadījuma tas ir programmas kods, kas zīmē ciparu 9. Savukārt, *ja nav taisnība, tad pilda bloku pēc else*. Mēs šeit esam piestiprējuši klāt nākošo if, kas pārbauda vai cip ir vienāda ar 8. Tā turpinām uz priekšu, līdz par pēdējam simbolam.

```
92 } else if( cip == 0 ) {  
93     digitalWrite(pinA, HIGH); // 0  
94     digitalWrite(pinB, HIGH);  
95     digitalWrite(pinC, HIGH);  
96     digitalWrite(pinD, HIGH);  
97     digitalWrite(pinE, HIGH);  
98     digitalWrite(pinF, HIGH);  
99     digitalWrite(pinG, LOW);  
100 } else {  
101     digitalWrite(pinA, LOW);  
102     digitalWrite(pinB, LOW);  
103     digitalWrite(pinC, LOW);  
104     digitalWrite(pinD, LOW);  
105     digitalWrite(pinE, LOW);  
106     digitalWrite(pinF, LOW);  
107     digitalWrite(pinG, LOW);  
108 }  
109 }
```

Te pēc pēdējā else nodēšam visus indikatora segmentus. Tas nostrādā tajā gadījumā, ja cip nav nedz 9, nedz 8, ...nedz arī 0. Tālāk piemērs kā to izmantot. Piemēram izvadām uz indikatora katru trešo ciparu.

```
111 void loop() {  
112     for(int i=0; i<10; i=i+3) {  
113         print( i );  
114         delay( 1000 );  
115     }  
116 }
```

1.8 Programmas dalījums vairākos failos, Arduino IDE stils

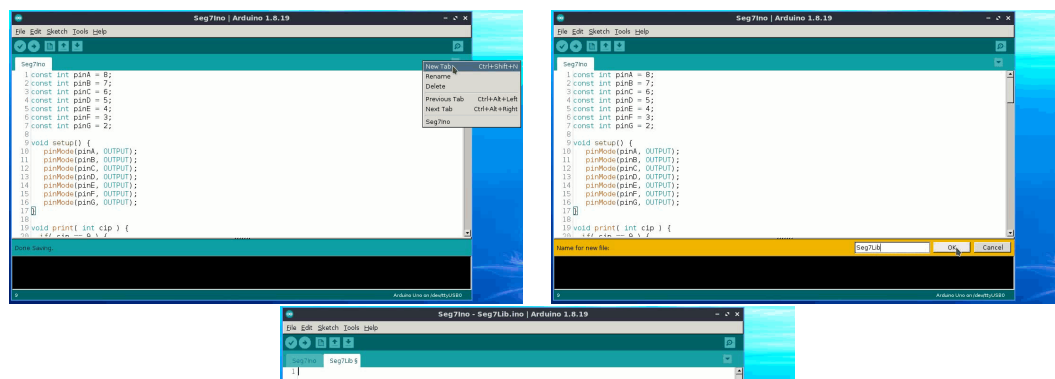
Lai mūsu iepriekšējo programmu P.1.8 varētu viegli lietot arī citos projektos, sadalīsim to vairākos failos divās daļās: īsā galvenajā daļā un otrajā daļā, uz kuru pārvietosim ciparu zīmēšanas funkcijas. Šeit aprakstīts ir Arduino IDE stils kā to darīt, kas nedaudz atšķiras no tā, kā to dara klasiskajā C/C++. Darba kārtība ir attēlota zīmējumā 1.4 un ir sekojoša:

1. Augšējā labajā stūrī atrodam iesimīti, kas atver menu ar *New Tab*. Var lietot arī taustiņu kombināciju Ctrl+Shift+N.
2. Lejā parādās jauna faila radīšanas sadaļa. Tur rakstām Seg7Lib. Tur būs mūsu bibliotēka.
3. Rezultātā mūsu projektā ir divi faili.

Faili Seg7Lib.ino pagaidām tukšs. Uz to pārnēsīsim lielāko daļu no programmas teksta. Sākam ar izeju definīcijām

Listing 1.9: Seg7Lib.ino

```
1 const int pinA = 8;  
2 const int pinB = 7;  
3 const int pinC = 6;  
4 const int pinD = 5;  
5 const int pinE = 4;  
6 const int pinF = 3;  
7 const int pinG = 2;
```

Att. 1.4: Projekts Arduino IDE vidē

Tālāk pārnesam funkciju `void setup()` pārsaucot to par `void setup_seg7()`

Listing 1.10: Seg7Lib.ino

```

9 void setup_seg7() {
10   pinMode(pinA, OUTPUT);
11   pinMode(pinB, OUTPUT);
12   pinMode(pinC, OUTPUT);
13   pinMode(pinD, OUTPUT);
14   pinMode(pinE, OUTPUT);
15   pinMode(pinF, OUTPUT);
16   pinMode(pinG, OUTPUT);
17 }
```

Funkciju `void print(int cip)` pārveidojam atbīrvojoties no `if() {} else if() else {}` ķēdes. Un pārsaucam to par `void print_seg7(int cip)`

Listing 1.11: Seg7Lib.ino

```

19 void print_seg7( int cip ) {
20   switch( cip ) {
21     case 9:
22     digitalWrite(pinA, HIGH); // 9
23     digitalWrite(pinB, HIGH);
24     digitalWrite(pinC, HIGH);
25     digitalWrite(pinD, HIGH);
26     digitalWrite(pinE, LOW);
27     digitalWrite(pinF, HIGH);
28     digitalWrite(pinG, HIGH);
29     break;

102   case 0:
103     digitalWrite(pinA, HIGH); // 0
104     digitalWrite(pinB, HIGH);
105     digitalWrite(pinC, HIGH);
106     digitalWrite(pinD, HIGH);
107     digitalWrite(pinE, HIGH);
108     digitalWrite(pinF, HIGH);
109     digitalWrite(pinG, LOW);
110     break;
111   default:
112     digitalWrite(pinA, LOW);
113     digitalWrite(pinB, LOW);
114     digitalWrite(pinC, LOW);
```

```
115     digitalWrite(pinD, LOW);  
116     digitalWrite(pinE, LOW);  
117     digitalWrite(pinF, LOW);  
118     digitalWrite(pinG, LOW);  
119 }  
120 }
```

Galvenajā programmas daļā jāizveido jauna `setup()` funkcija, `loop()` funkcija arī jāuzlabo viena vietā. Atrodi to!

Listing 1.12: Seg7Ino.ino

```
1 void setup() {  
2     setup_seg7();  
3 }  
4  
5 void loop() {  
6     for(int i=0; i<10; i=i+3) {  
7         print_seg7(i);  
8         delay(1000);  
9     }  
10 }
```

Kā redzam, galvenā programmas daļa pavisam īsa un ļoti labi parskatāma. Bet otro daļu atklādo un tad par to var aizmirst.

Literatūra

- [1] Arduino home. <https://www.arduino.cc/>. [Online; accessed 2022.09.25].
- [2] Arduino UNO. <https://docs.arduino.cc/hardware/uno-rev3>. [Online; accessed 2022.09.25].
- [3] Arduino UNO reference design. <https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf>. [Online; accessed 2022.09.25].
- [4] Arduino Software (IDE). <https://www.arduino.cc/en/software/>. [Online; accessed 2022.09.25].
- [5] Arduino Libraries. <https://www.arduino.cc/reference/en/libraries/>. [Online; accessed 2022.09.25].
- [6] Creative commons attribution-sharealike 3.0 license. <https://creativecommons.org/licenses/by-sa/3.0/>. [Online; accessed 2022.09.25].
- [7] Arduino Language Reference. <https://www.arduino.cc/reference/en/>. [Online; accessed 2022.09.25].
- [8] Stephen Prata. *C Primer Plus*. Sams, fifth edition, 2004.
- [9] Greg Perry. *Absolute beginner's guide to C*. Sams, 1994.
- [10] S4A home. <http://s4a.cat/>. [Online; accessed 2022.09.25].
- [11] Arduino UNO. <https://scratchx.org>. [Online; accessed 2022.09.25].
- [12] Atmel ATmega328P. <https://www.microchip.com/en-us/product/ATmega328P>. [Online; accessed 2016.09.18].

- [13] Arduino Reference. pinMode(). <https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/>. [Online; accessed 2022.09.25].
- [14] Arduino Reference. digitalWrite(). <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/>. [Online; accessed 2022.09.25].
- [15] Arduino Reference. delay(). <https://www.arduino.cc/reference/en/language/functions/time/delay/>. [Online; accessed 2022.09.25].
- [16] Seven-segment display. https://en.wikipedia.org/wiki/Seven-segment_display. [Online; accessed 2022.09.25].

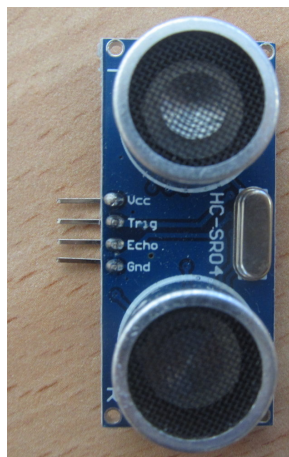
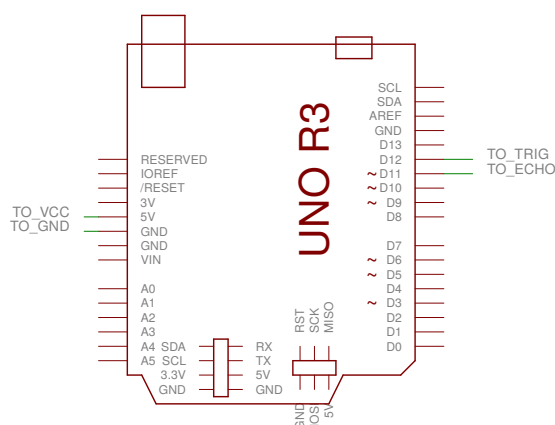
Nodaļa 2

Bibliotēkas

2.1 Attāluma mērīšana izmantojot ultraskaņas attāluma sensoru

Viens no izplatītākajiem un lētākajiem veidiem kā mērīt attālumu ir izmantot ultraskaņas attāluma sensoru HC-SR04 [1]. Ar tā palīdzību mēra laiku, kas nepieciešams lai ultraskaņas impulss veiktu attālumu no sensora līdz objektam un pēc tam atpakaļ līdz sensoram. Pēc tam, izmantojot skaņas ātrumu gaisā, izrēķina attālumu līdz objektam. Visas nepieciešamās lietas veic bibliotēka NewPing [2]. Tālāk apskatīsim šīs bibliotēkas pielietojuma standarta piemēru.

Sensors redzams 2.1 zīmējumā, skatāties uz tā ieeju/izeju marķējumiem. Vispirms ieejas Gnd un Vcc pieslēdzam atbilstošajām mikrodatora izejām. Savukārt ieeju Trig un Echo izeju varam pieslēgt pie jebkurām no mikrodatora ciparu ieejām/izejām (D0 un D1 bez īpašas vajadzības nevajadzētu izmantot, jo tad lai aukšuplādētu programmu, sensors uz laiku būs jāatslēdz). Tikai sava izvēle ir jādara zināma programmai. Parasti gan rīkojas otrādi, apskatās programmā kur kas jāpieslēdz un tā arī dara. Tāpēc skatāties 2.1 programmā 3 un 4 rindiņu. Tur teksts, ja tulkojam mainīgo nosaukumus, ka Trig pieslēdzam pie 12 ciparu ieejas/izejas, bet Echo izeju pie 11 ciparu ieejas/izejas. Pie viena 5 rindiņā uzstādām maksimālo attālumu centimetros, kuru vēlamies mērīt. Mēs protams varējām 3-7 rindiņu vietā rakstīt uzreiz NewPing sonar(12,11,200);, tikai tad būtu jāmeklē dokumentācijā kur



Att. 2.1: Ultraskaņas attāluma sensora HC-SR04 pieslēgšana Arduino savietojamam mikrodatoram.

isti kas jāslēdz un ko nozīmē mistiskais skaitlis 200. Programmētāji parasti ļoti nevēlas rakstīt komentārus savām programmām, tāpēc piemērā lietotais stils ļauj nedaudz paslinkot un tajā paša laikā ietaupīt milzīgi daudz laika, ja vajadzēs programmu lietot pēc mēneša vai pat vairākiem gadiem. Interesanti, ka šādu mainīgo izveide programmas darbības efektivitāti nesamazina. Mulsinošais `const` pasaka kompilatoram, ka mēs šo mainīgo vērtības nemainīsim. Tāpēc šādi mainīgie vietu datora atmiņā neaizņem, jo kompilators uzreiz ievieto to vērtības vajadzīgajās vietās.

Listing 2.1: UltraSonic.ino

```
1 #include <NewPing.h>
2
3 const int pinTrig = 12;
4 const int pinEcho = 11;
5 const int maxDist = 250;
6
7 NewPing sonar(pinTrig, pinEcho, maxDist);
8
9 void setup()
10 {
11     Serial.begin( 9600 );
12 }
13
14 void loop()
15 {
16     delay(250);
17     Serial.print("Ping: ");
18     Serial.print(sonar.ping_cm());
19     Serial.println("cm");
20 }
```

Programma 2.1 ierosina ultraskaņas signāla noraidīšanu, gaida tā pienākšanu atpakaļ un izmantojot nomērīto laiku aprēķina attālumu līdz objektam un to visu panāk izmantojot bibliotēkas.

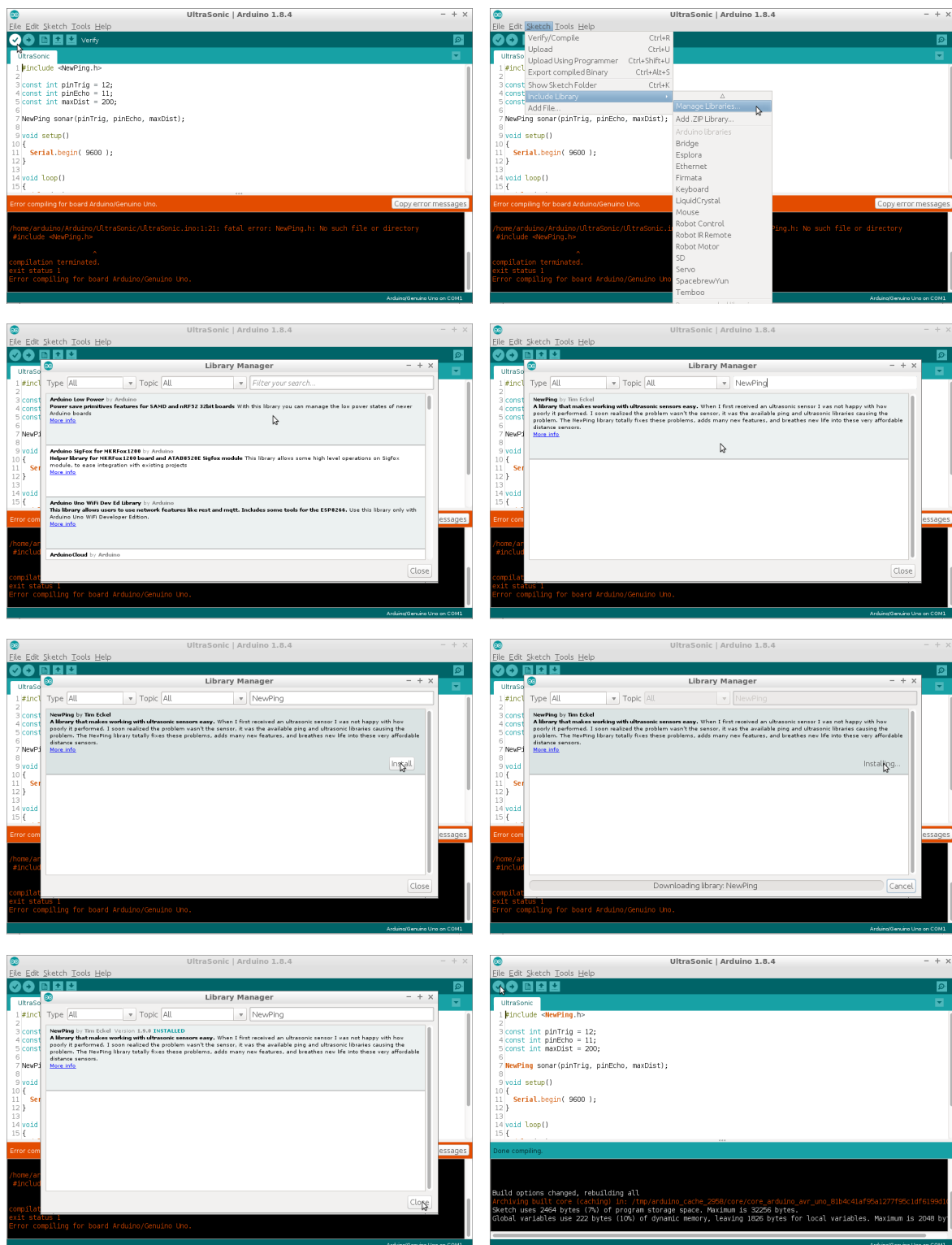
Kad mēģinām šo projektu kopmilēt, tad saņem kļūdas paziņojumu: *fatal error NewPing.h: No such file or directory*. Tas nozīmē, ka jūsu sistēmā pagaidām vēl nav instalēta NewPing bibliotēka. Kā to izdarīt, parādīts attēlā 2.2.

1. Izvēlamies Sketch→Include Library→ Manage Libraries
2. Include Library meklēšanas logā rakstām NewPing
3. Ar peli uzklikšķinām uz pelēkā fona
4. Klikšķinām uz Install. Sagaidām kamēr gatavs, veram Manage Libraries logu ciet.
5. Kompilējam programmu vēlreiz, un viss kārtībā.

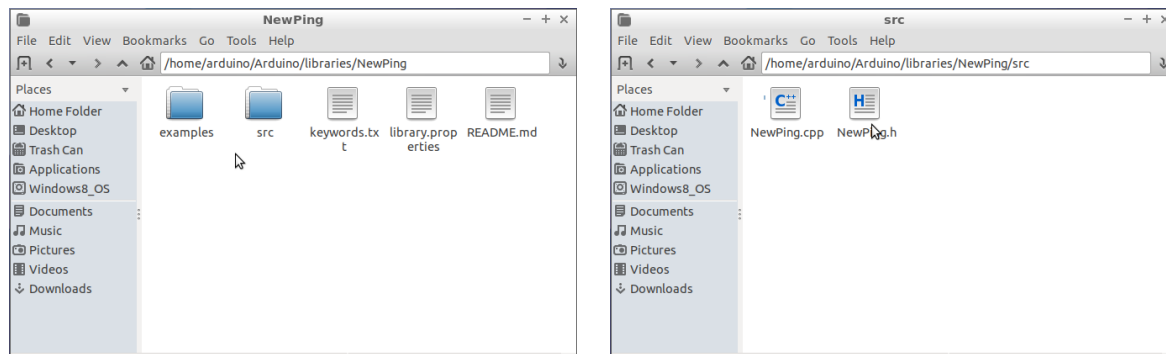
Kas tad notiek bibliotēkas instalēšanas procesa laikā? Izrādās, ka katalogā Arduino/libraries ir parādījies katalogs NewPing, kā redzam 2.3 zīmējumā. Tas savukārt satur divus katalogus `src` un `examples`. Pirmajā ir divi faili `NewPing.h` un `NewPing.cpp` kas arī ir NewPing bibliotēka. Savukārt katalogā `src` ir šīs bibliotēkas lietošanas piemēri.

Failus `NewPing.h` un `NewPing.cpp` varam paskatīties, bet ja nekas nav skaidrs, nevajag uztraukties. Pilnīgi pietiek zināt tikai trīs lietas:

- Programmas sākumā jāraksta `#include <NewPing.h>`. Šajā failā ir aprakstīti objekti un funkcijas, kurai mums būs nepieciešami šo bibliotēku lietojot.



Att. 2.2: NewPing bibliotēkas instalēšanas soli.



Att. 2.3: NewPing bibliotēkas instalēšanas rezultāts.

- Jādefinē objekts, kas strādās ar sensoru `NewPing sonar(pinTrig , pinEcho , maxDist)`. Argumentos ir to divu ciparu ieejas/izejas numuri, pie kurām pieslēgts sensors ka arī sensora maksimālais darbības attālums.
- Kad nepieciešams attālums līdz objektam, tad vienkārši izsaucam funkciju `sonar.ping_cm()`.

Tiem, kas vēlas zināt vairāk, varam sākt lasīt ar Arduino Language Reference [3]. Tālāk ieteicams lasīt labu C un pēc tam C++ grāmatu. Ir kur augt.

2.2 Servomotora vadība

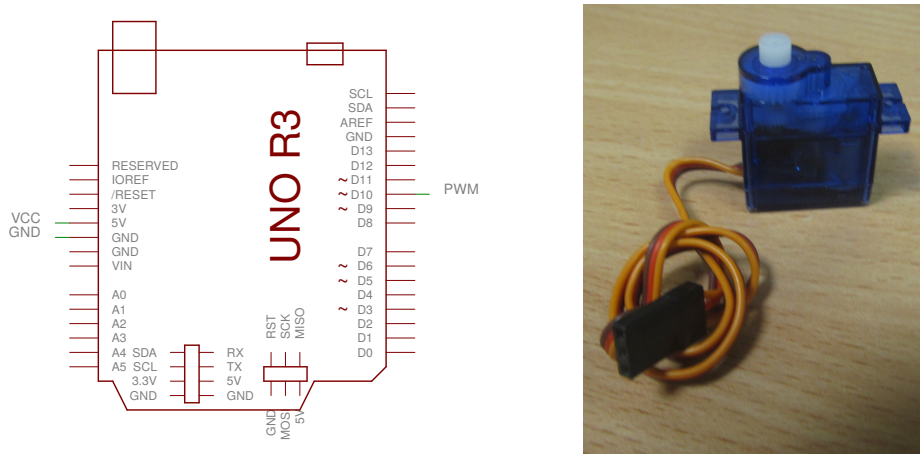
Servomotors ir kompakta iekārta mehānismu vadībai, kas ļauj to iestādīt jebkurā stabīlā pozīcijā. Tas nozīmē, ka servomotors pats sevi kontrolē, cik tas ir pagriezts. Servomotoram pievada barošanas spriegumu kā arī pseidoanalogo (PWM modulētu) signālu, kam proporcionāls ir tā pagrieziena leņķis. Pieslēguma shēma dota 2.4 zīmējumā, savukārt programma adaptēta no [4]. Tajā izmantota Servo bibliotēka [5]. Kā atpazīt vadus? Kā parasti elektronikā dara, vads ar pašu *augstāko*, tumšāko krāsu, ir GND. Nākošais, tāpat vidējais, ir barošanas līnija VCC. Trešais, tāpat otrs malējais, ir vadības signāls, kuru pieslēdz kādai no PWM spējīgai izejai. Tās ir tās, kuru numuriem blakus ir ~ simbols, kā tas redzams shēmā 2.4 zīmējumā.

Listing 2.2: Sweep.ino

```

1 #include <Servo.h>
2
3 const int pinServo = 10;
4 Servo myservo;
5
6 void setup()
7 {
8   myservo.attach( pinServo );
9 }
10
11 void loop()
12 {
13   int pos;
14
15   for(pos = 0; pos < 180; pos += 1) {
16     myservo.write( pos );
17     delay( 15 );
18   }

```

Att. 2.4: Servomotora pievienošana Arduino savietojamam mikrokontroleram.

```

19
20   for(pos = 180; pos>=1; pos-=1) {
21       myservo.write( pos );
22       delay( 15 );
23   }
24 }
```

Literatūra

- [1] Ultrasonic Ranging Module HC-SR04. https://github.com/sparkfun/HC-SR04_UltrasonicSensor. [Online; accessed 2022.09.25].
- [2] NewPing Library for Arduino. <https://www.arduino.cc/reference/en/libraries/newping/>. [Online; accessed 2022.09.25].
- [3] Arduino Language Reference. <https://www.arduino.cc/reference/en/>. [Online; accessed 2022.09.25].
- [4] Arduino Examples. Sweep. <https://www.arduino.cc/en/Tutorial/LibraryExamples/Sweep/>. [Online; accessed 2022.09.25].
- [5] Arduino Reference. Servo library. <https://www.arduino.cc/reference/en/libraries/servo/>. [Online; accessed 2022.09.25].