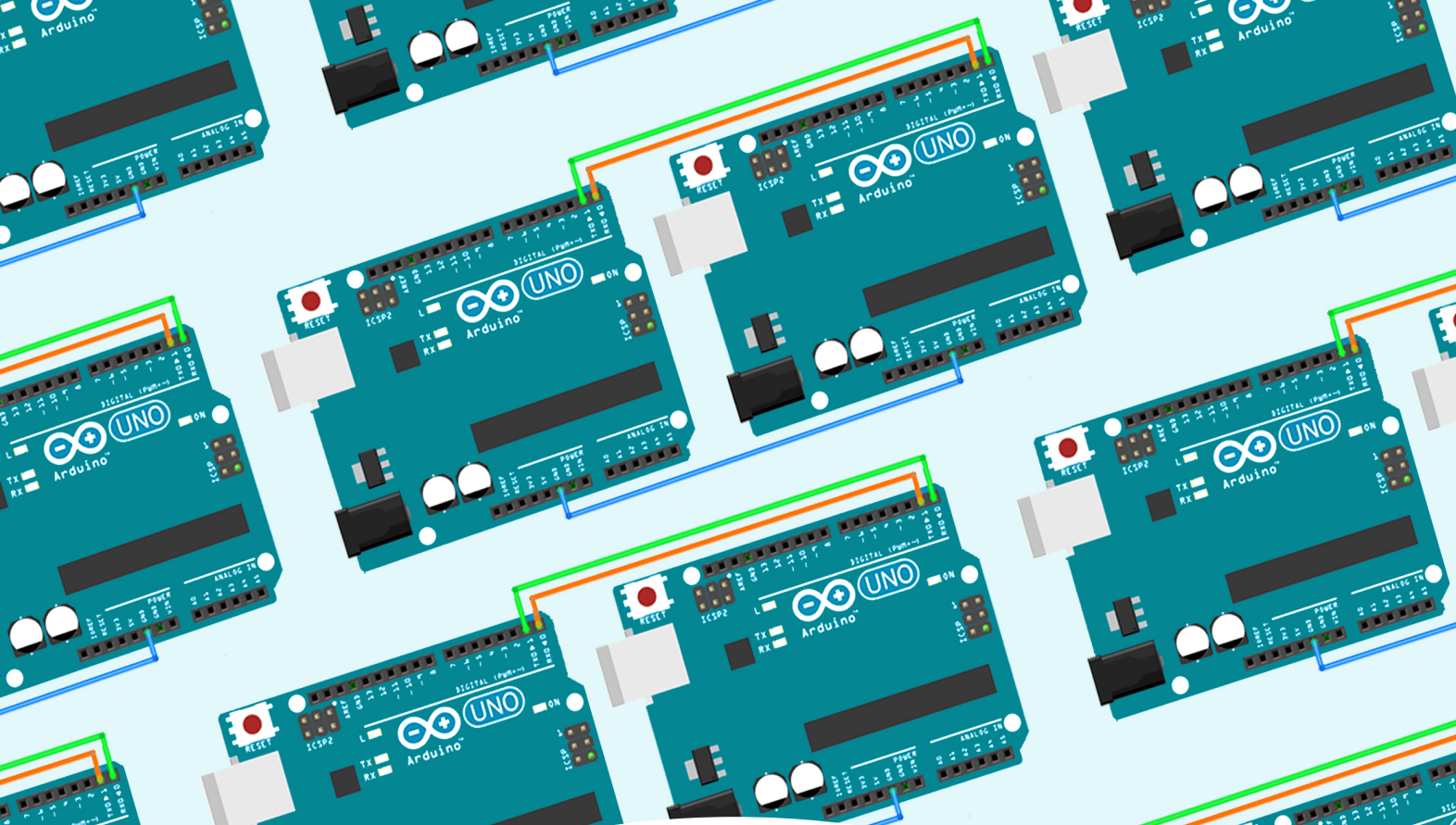


ARDUINO

ԱՐԴՈՒԻՆՈ / ARDUINO

ՆԱԽԱԳԾԵՐԻ ՁԵՌՆԱՐԿ

PROJECTS GUIDE

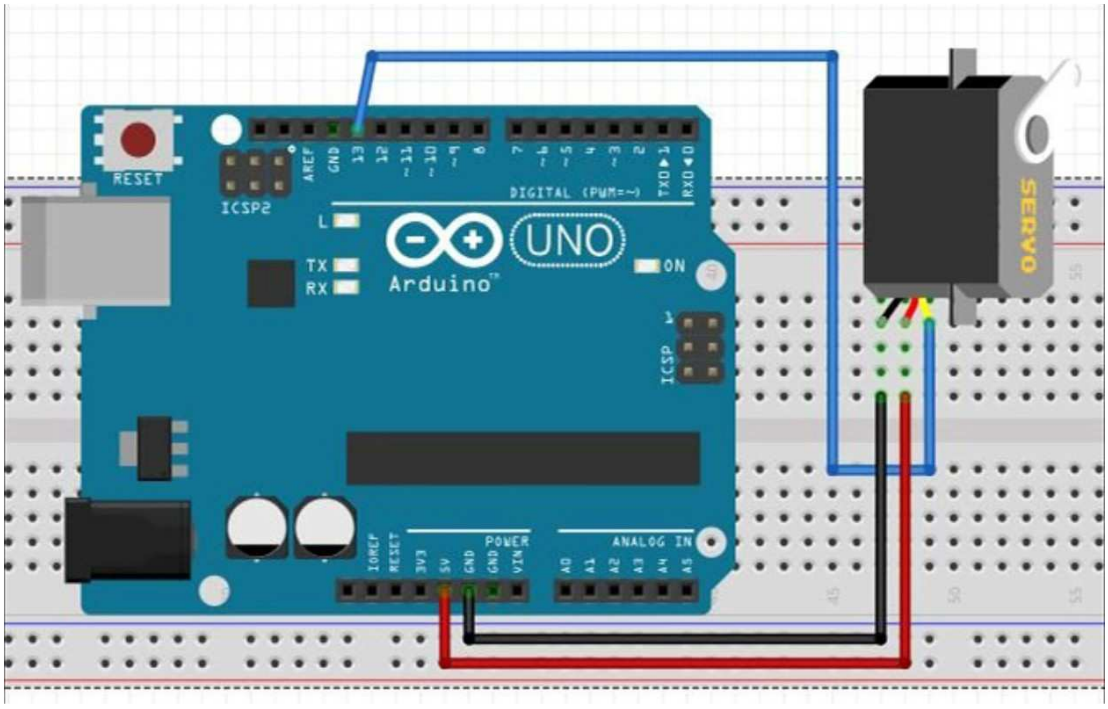
Նախագծերի ձեռնարկ

Arduino

Single-board microcontroller

«ԱՐԴՈՒԻՆՈ»

Միասալիկ միկրոկառավարման վահանակ



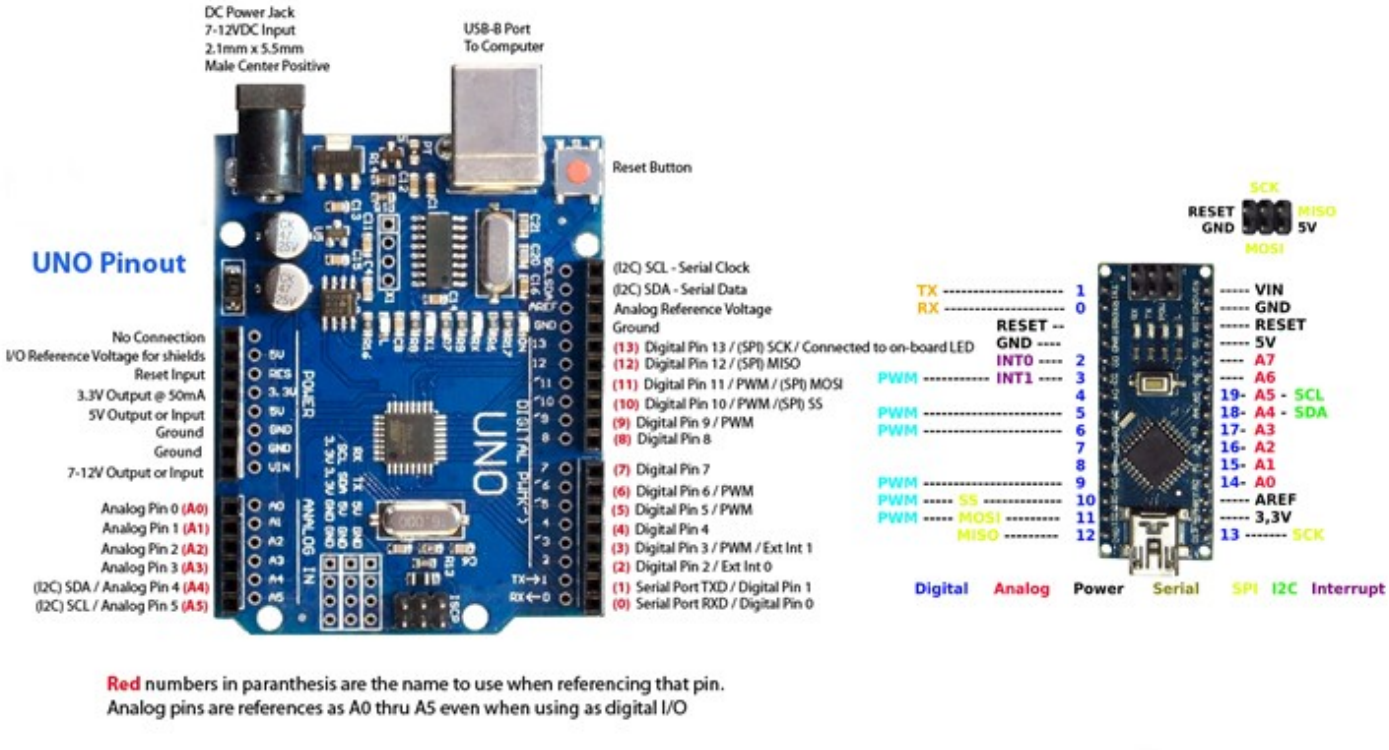
Հեղինակ՝ Արմեն Մխոյան

Բովանդակություն

1. Ներածություն

Սույն ձեռնարկը նախատեսված է ծանոթանալու Arduino կառավարման սալիկի կառուցվածքի, նրա հետ աշխատելու, **Arduino IDE** ծրագրավորման միջավայրի ուսումնասիրման և հիմնային գիտելիքներ ստանալու համար:

Arduino բառի տակ թաքնված է մի ամբողջ հնարավորությունների համակարգ: Այս համակարգում կան կառավարման սալիկների (միկրոկառավարիչներ- microcontrollers) մի շարք տեսակներ, որոնք ունեն իրենց սեփական պրոցեսորը և հիշողությունը: Նրա վրա տեղակայված են մի շարք կոնտակտներ, որոնց կարելի է միացնել լույսեր, շարժիչներ, տվիչներ, դռան մագնիսական փականներ և գրեթե այն ամենը ինչ աշխատում է էլեկտրականությամբ: Սույն ձեռնարկում կուսումնասիրենք **Arduino UNO** և **Arduino Nano** տպասալերը (նկ. 1), որոնց միջոցով կիրականացնենք նշված նախագծերը: Համապատասխան միացված սարքավորումների աշխատանքը կառավարվում է նրանցում նախապես տեղակայված ծրագրի միջոցով, որը ծրագրավորվում է **Arduino IDE** միջավայրում: Ծրագրավորումը կատարվում է C լեզվի հիման վրա: **Arduino IDE** ծրագիրը կարելի է ներբեռնել https://www.arduino.cc/en/Main/Software հղումով:



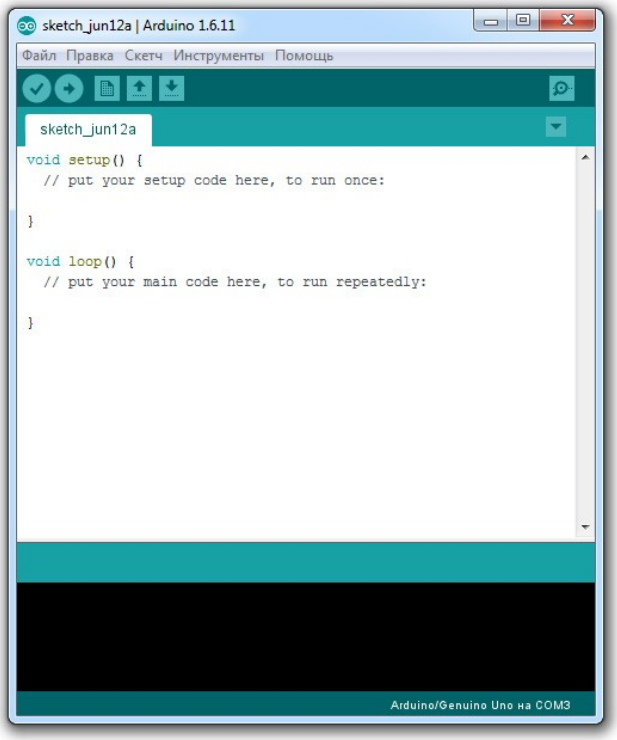
Նկ. 1

Տեխնիկական կառուցվածք. Arduino կառավարման սալիկների բոլոր տեսակներն ունեն հետևյալ անհրաժեշտ բաղադրամասերը, որոնց միջոցով հնարավոր է դառնում ծրագրավորումը և համատեղ միացումներ տարբեր կատարող սարքերին և տվիչներին.

- Atmel microcontroller,
- USB - միջավայր ծրագրավորման և տվյալների փոխանցման-ստացման համար,
- Լարման կարգավորիչ և հոսանքի կոնտակտներ,
- Մուտք-ելքի կոնտակտներ (pin-եր), ցուցանիշային լուսադիոդներ (Debug, Power, Rx, Tx),
- Վերագործարկման սեղմակ,
- (ICSP) Ներկառուցված հաջորդական ծրագրավորման միջավայր։

Atmel microcontroller-ը հանդիսանում է Arduino կառավարման սալիկի հիմնային բաղադրիչը։ Հիմնական տեսակներում, այդ թվում նաև Arduino Uno կառավարման սալիկում տեղադրված է ATmega-ն։ Հիմնականում նրա վրա տեղակայված է լինում նաև լուսադիոդ, որն անմիջապես միացված է 13-րդ կոնտակտին, որի միջոցով կարող ենք առանց հավելյալ սարքավորումների կատարել առաջին աշխատանքը՝ լույսի թարթում։

Ծրագրային կառուցվածք. Arduino IDE ծրագրի կառուցվածքն իր մեջ ներառում է 2 անհրաժեշտ ենթածրագրեր՝ **void setup() {}** և **void loop () {}** : Նոր ծրագիր (**sketch**) բացելիս կտեսնենք, որ դրանք արդեն ներառված են (նկ. 2)։ **void setup()** ենթածրագրի (**ֆունկցիա**) ձևավոր փակագծերի ներսում գրված հրամանները կատարվում են մեկ անգամ՝ առաջին անգամ ծրագիրը միացնելիս։ Այս բաժնում մասնավորապես կսահմանենք մեր նախագծերում մեզ անհրաժեշտ կոնտակտների (**PIN**-երի) տեսակը, որը կարող է լինել OUTPUT կամ INPUT տեսակի (ըստ համապատասխան նշանակության կիրառության)։ **void loop()** ենթածրագիրը ցիկլի կամ կրկնողության ծրագիր է, որը սկսում է իր աշխատանքը կատարել **void setup()** ենթածրագրի աշխատանքի կատարումից անմիջապես հետո այնքան ժամանակ, քանի դեռ Arduino տպասալը միացված է հոսանքի աղբյուրին։ Մենք կարող ենք ծրագրային մասերը ավելի կարգավորված գրելու համար ստեղծել համապատասխան ենթածրագրեր, որոնց կանչելով **void loop()** ենթածրագրում կարող ենք աշխատեցնել։ Մեր կողմից ստեղծված ենթածրագրի անունը պետք է փոփոխականի համապատասխան ստանդարտներին բավարարող տեսք ունենա և սկսվի **void** հրամանով։ Օր՝ **void Forward():** Իսկ **void loop()-**ում կանչելիս գրում ենք միայն անունը՝ **Forward()**:



Syntax (ծրագրի այլագիրը). Ցանկացած ծրագրավորման լեզու ունի իր գրելաձևի որոշակի պահանջներ: Այստեղ բավական է հիշել հետևյալ պահանջները.

- **/ / (միատող մեկնաբանություն)** – Այն հաճախ օգտագործում են ծրագրի ինչ որ հատվածում մեկնաբանություն ավելացնելու համար: Համապատասխան նիշերից հետո գրված ցանկացած հրամանները կամ տեքստը կանտեսվի ծրագրի աշխատանքի իրականացման ժամանակ:
- **/ *..... */ (բազմատող մեկնաբանություն) –** Ի տարբերություն նախորդ մեկնաբանության տեսակի այստեղ ծրագրի աշխատանքի իրականացման ժամանակ անտեսվում է սիմվոլների արանքում գրվածը, որը կարող է տեղակայվել նույնիսկ մի քանի տողերի վրա:
- **{ } (ձևավոր փակագծեր) –** Այն տեղադրվում է հրամանների հաջորդականության սկիզբը և վերջը առանձնացնելու համար: Օգտագործվում է ենթածրագրերում և ցիկլի օպերատորներում:
- **; (կետ-ստորակետ) –** Հրամաններն ավարտվում են այս սիմվոլով: Ամենատարածված սխալը ծրագրի աշխատանքի ժամանակ լինում է դրա բացակայության պատճառով:

Փոփոխականներ. Ցանկացած ծրագրավորման լեզվում կարևոր գործոն ունեն փոփոխականները: Այստեղ մենք անընդհատ առնչվելու ենք թվերի հետ (կարող են

լինել թե **PIN**-երի համարները, թե որոշակի պարամետրեր և այլն): Փոփոխականները մեզ կօգնեն նրանց հետ ավելի արդյունավետ աշխատել: Դիտարկենք հետևյալ տեսակները.

- **int (ամբողջ տիպ) –** հիմնական օգտագործվող տեսակը, որը զբաղվեցնում է 2 բայթ (16 բիթ) հիշողություն, իսկ արժեքների միջակայքը՝ **-32768 ÷ 32767**:
- **long (երկար տիպ) –** օգտագործվում է այն դեպքում, երբ **int** տիպի հնարավորությունները չեն բավականացնում: Զբաղեցնում է 4 բայթ (32 բիթ) հիշողություն, իսկ արժեքների միջակայքը՝ **-2147483648 ÷ 2147483647**:
- **float (իրական տիպ) –** իրական թվերում ամբողջ մասը կոտորակայինից առանձնացվում է **կետի(.)** միջոցով: Զբաղվեցնում է 4 բայթ (32 բիթ) հիշողություն, իսկ արժեքների միջակայքը՝ -3.4028235E+38
- **boolean (տրամաբանական տիպ) –** Այս տիպը զբաղեցնում է 1 բիթ հիշողություն և ընդունում է երկու արժեք՝ **true** կամ **false** :
- **char (սիմվոլային տիպ) –** Այս դեպքում սիմվոլները պահպանվում են օգտագործելով ACSII կոդավորումը (օրինակ <<A>> = 65): Զբաղեցնում է 1 բայթ (8 բիթ) հիշողություն: Տողային տիպն այստեղ համարվում է սիմվոլների զանգված:

Մաթեմատիկական գործողություններ . Մեզ անհրաժեշտ մաթեմատիկական գործողությունները հետևյալն են.

- **=** - կատարում է վերագրման գործողություն(օր. x=10*2 գործողության արդյունքում x փոփոխականին վերագրում է 20 արժեք),
- **%** **-** բաժանումից ստացված մնացորդի հաշվում (օր. 12%10 –ի դեպքում կլինի 2),
- **+** - գումարում,
- **-** - հանում,
- ***** - բազմապատկում,
- **/** - բաժանում:

Համեմատության օպերատորներ. Օգտագործվում են տրամաբանական համեմատություններ կատարելու համար.

- **==** - **հավասար է** պայման (օր. 12==10 դեպքում կստացվի false արժեք, իսկ 5==5 դեպքում կստացվի true արժեք),
- **!=** - **հավասար չէ** պայման (օր. 12!=10 դեպքում կստացվի true արժեք, իսկ 5!=5 դեպքում կստացվի false արժեք),
- **>** **- մեծ է** պայման,
- **>=** **- մեծ կամ հավասար է** պայման,

- < **- փոքր է** պայման,
- <= **- փոքր կամ հավասար է** պայման:

Պայմանի և ցիկլի օպերատորներ. Այստեղ նկարագրված են նախագծերում օգտագործվող տեսակները.

- **if (պայման 1) { }**

  else if (պայման 2) { }

  else { }

Այս դեպքում եթե **պայման 1**-ը ճիշտ է (true) կատարվում է առաջին ձևավոր փակագծերում եղած գործողությունը, հակառակ դեպքում ստուգվում է **պայման 2**-ը, եթե ճիշտ է, ապա կատարվում է երկրորդ ձևավոր փակագծերում կատարվող գործողությունը, հակառակ դեպքում՝ երրորդ ձևավոր փակագծի գործողությունը: Նույն տրամաբանությամբ 2-ից ավել պայմանների դեպքում:

- **for (int i=0; i<=255; i++) { }**

Այստեղ ձևավոր փակագծերում գրված գործողությունը կկատարվի սահմանված քանակով, որտեղ 1-ին պարամետրը i փոփոխականի սկզբնական արժեքն է, 2-րդը՝ վերջնական արժեքը, որի կատարումով կավարտվի ցիկլը, իսկ 3-րդը՝ i փոփոխականի փոփոխման քայլն է (այս դեպքում i++ հրամանը համարժեք է i=i+1): i փոփոխականի արժեքը կարող է ինչպես աճել, այնպես էլ նվազել (օր. i--), ինչպես նաև աճման կամ նվազման կարգը կարող է լինել այլ մաթեմատիկական գործողությունների տեսքով ներկայացված (օր. i=i*2):

Թվային ազդանշաններ.

- **pinMode (pin, Mode) –** Սահմանում է համապատասխան **pin**-ի տեսակը: **Mode**-ը կարող է ընդունել **INPUT** և **OUTPUT** արժեքներ:
- **digitalWrite (pin, value)** – եթե նախապես սահմանված **pin**-ը որպես **OUTPUT** տեսակ է հանդիսանում, ապա **value**-ի արժեքը կարող է լինել **HIGH** (տրամաբանական 1, +5Վ) կամ **LOW** (տրամաբանական 0 կամ GND):
- **digitalRead (pin) -** եթե նախապես սահմանված **pin**-ը որպես **INPUT** տեսակ է հանդիսանում, ապա այս հրամանը մուտքում կվերադարձնի **HIGH** կամ **LOW** ազդանշանի արժեք:

Անալոգային ազդանշաններ. Arduino-ն թվային սարք է համարվում, բայց կարող է աշխատել նաև անալոգային ազդանշանների հետ հետևյալ հրամանների միջոցով.

- **analogWrite (pin, value)** – Arduino-ի մի քանի **pin**-եր (օր. Arduino Uno- ի դեպքում 3,5,6,9,10,11 pin-երը) աշխատում են **PWM** (ԼԻՄ**-** լայնակի իմպուլսային մոդուլյացիա) ռեժիմում, որի ժամանակ մեծ արագությամբ տրամաբանական 1-եր և 0-ներ են փոխանցվում: Այս դեպքում Միջին լարման արժեքը կախված է միավոր ժամանակում տրամաբանական 0-ների և 1-երի քանակից և կարող է փոխվել 0 (0Վ)-ից մինչև 255 (+5Վ) միջակայքում:
- **analogRead (pin) -** այս հրամանով մենք աշխատում ենք անալոգային **INPUT** տեսակի կոնտակտների հետ (Arduino Uni-ի դեպքում դրանք 6-ն են՝ **A0-A5**), որի միջոցով համապատասխան կոնտակտում չափում ենք լարման արժեքը, որը կարող է ստանալ 0 (0Վ) –ից մինչև 1024 (+5Վ) միջակայքի թվային արժեքներ:

Երկու թվային միջակայքերում արժեքների համապատասխանեցում.

map - այս հրամանով կարող ենք ցանկացած միջակայքի ընթացիկ արժեքից ստանալ մեկ այլ միջակայքի ընթացիկ արժեք՝ այդ միջակայքերի նվազագույն և առավելագույն արժեքները համապատասխանեցնելով: Փակագծերում գրվում է 5 պարամետր, որոնցից 1-ինը որևէ միջակայքի ընթացիկ արժեքն է, որից պետք է ստանաք այլ միջակայքի ընթացիկ արժեք, 2-րդը՝ դրան համապատասխանող միջակայքի նվազագույն արժեքն է, 3-րդը՝ դրան համապատասխանող առավելագույն արժեքը, 4-րդը՝ փոխվելիք պարամետրի նվազագույն արժեքը, իսկ 5-րդը՝ դրա առավելագույն արժեքը: Հրամանի արդյունքում կստանանք փոխված ընթացիկ արժեքը, որը պետք է վերագրել որևէ փոփոխականի: Օր՝ ենթադրենք ունենք լուսադիոդ, որի արժեքները կնշանակենք L տառերով, և ունենք պոտենցոմետր, որի արժեքները կնշանակենք P տառերով: Անհրաժեշտ է պոտենցոմետրի ընթացիկ արժեքից կախված ստանալ լուսադիոդի համապատասխան ընթացիկ արժեք: Քանի որ պոտենցոմետրից ստացված արժեքների միջակայքը 0-1023 է, իսկ լուսադիոդին փոխանցվող արժեքների միջակայքը՝ 0-255, ապա պետք է համապատասխանություն դրվի միջակայքերի մեջ, որը կստանանք հետևյալ հրամանով՝

$$L_ը = \mathbf{map}(P_ը, P_{min}, P_{max}, L_{min}, L_{max}),$$

որտեղ $P_ը$-ը պոտենցոմետրի ընթացիկ արժեքն է, P_{min}-ը վերջինիս նվազագույն արժեքը (0), P_{max}-ը՝ առավելագույն արժեքը (1023), L_{min}- ը լուսադիոդի նվազագույն արժեքն է (0), L_{max}-ը՝ առավելագույն արժեքը, իսկ $L_ը$-ի մեջ կստանանք լուսադիոդի ընթացիկ արժեքը:

Անհրաժեշտ սարքավորումներ.

- **LED Լուսադիոդ -** Ունի 2 ելք, որոնցից երկարը (անոդ) միացվում է դրական պոտենցիալին (մեր նախագծերում **pin**-երին՝ դրանց աշխատանքի կառավարումն իրականացնելու համար), իսկ կարճ ելքը միանում է **GND**-ին: Սխալ միացման դեպքում այն չի աշխատի: Կարևոր է նշել, որ հոսանքի ուժը սահմանափակելու համար անհրաժեշտ է օգտագործել **դիմադրություն**:

- **Դիմադրություն (ռեզիստոր) –** Սահմանափակում է շղթայում անցնող հոսանքի ուժի արժեքը: Այն անհրաժեշտ է միացնել հոսանքի շղթային հաջորդաբար: Չափման միավորը՝ Օհմ:

- **RGB լուսադիոդ -** Ունի իր ներսում 3 գույնի լուսադիոդներ՝ կարմիր, կանաչ և կապույտ: Դրանց տարբեր պայծառությունների միացման և համադրության դեպքում հնարավոր է ստանալ RGB գունային համակարգի տարբեր գունային երանգներ: Լինում է 2 տեսակի՝ **Կատոդային** (-) տեսակի և **Անոդային** (+) տեսակի: Ունի 4 կոնտակտ, որոնցից երկար ոտիկը միանում է GND-ին կամ 5Վ-ին՝ կախված լուսադիոդի տեսակից, իսկ մյուս 3 ոտիկները համապատասխանաբար կարմիր, կանաչ և կապույտ լեդ-երի միացման կոնտակտներն են:

- **Պոտենցոմետր՝ 10 կՕհմ դիմադրությամբ -** Ֆունկցիոնալ առումով համարվում է դիմադրություն, որի արժեքը կարող է փոփոխվելով կառավարվել: կարելի է միացնել ինչպես հաստատուն, այնպես էլ փոփոխական և իմպուլսային հոսանքի շղթաներին: Պտտույտի չափը մեկ ամբողջականից փոքր է՝ 300 աստիճան: Միացման համար 3 կոնտակտներից եզրայինները միանում են GND և 5Վ-ին, իսկ մեջտեղի կոնտակտը միանում է անալոգային **INPUT-**ներից մեկին:

- **Շարժիչ (DC motor)** – Պտտվում է, երբ նրանում էլեկտրական հոսանք է անցնում: Նրա աշխատանքը կառավարելի դարձնելի համար **Arduino**-ին միացնելիս մենք կօգտագործենք շարժիչի աշխատանքի կարգավորիչ մոդուլ (**Motor driver**):

- **VEX Motor Controller -** Շարժիչի աշխատանքի կարգավորման համար գոյություն ունեն մի շարք մոդուլներ, որոնցից կօգտագործենք 2-ը: Նշվածով աշխատելիս մենք շարժիչը կմիացնենք և կաշխատեցնենք սերվո տեսակի շարժիչի համապատասխան գրադարանով և հրամաններով: Շարժիչի 2 կոնտակտները միանում են վերջինիս մի կողմի կոնտակներին, իսկ դրանից դուրս եկող 3 կոնտակներից սև գույնը միանում է GND-ին, նարնջագույնը՝ 5Վ-ին, և սպիտակը՝ համապատասխան **pin-**ին, որին պետք է ազդանշան ուղարկել:

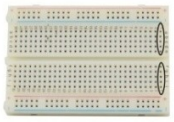
- **Motor Controller L9110 –** Այս մոդուլին կարելի է միացնել 2 շարժիչ: Մի կողմում եզրային ելքերը միանում են համապատասխան **pin-**երին, մեջտեղի ելքերը՝ հոսանքի աղբյուրին կամ **Arduino**-ի համապատասխան հոսանքի ստացման ելքերին, մյուս կողմից միացումը կատարվում է

համապատասխան շարժիչներին: Միացման սխեման կլինի համապատասխան նախագծում:

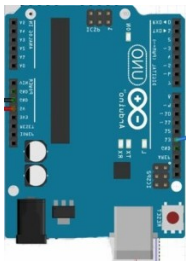
- **Սերվո շարժիչ (Servo motor) -** Ստացված էլեկտրական ազդանշանները վերածում է իր առանցքի շուրջ համապատասխան անկյան դիրքի գալու պտույտի: Նախագծում կօգտագործենք 0-180 աստիճան պտտման միջակայք ունեցող սերվո շարժիչ: Սարքը միացնելիս մենք **Arduino IDE** միջավայրում կօգտվենք **Servo.h** գրադարանից: Մանրամասները և միացման սխեման կլինի համապատասխան նախագծում:

Հեռաչափ (Ultrasonic) – Չափում է իրենից մինչև իր առջև տեղակայված արգելքի հեռավորությունը: Այն միացնելու համար մենք կօգտվենք **Ultrasonic.h** գրադարանից: Մանրամասները և միացման սխեման կլինի համապատասխան նախագծում:

Breadboard – Այն նախատեսված է միացումների հետ հեշտ աշխատելու համար: նախագծերի ժամանակ ներկայացված կլինի վերջինիս կոնտակտների կառուցվածքը և նրանից օգտվելու արդյունավետությունը:



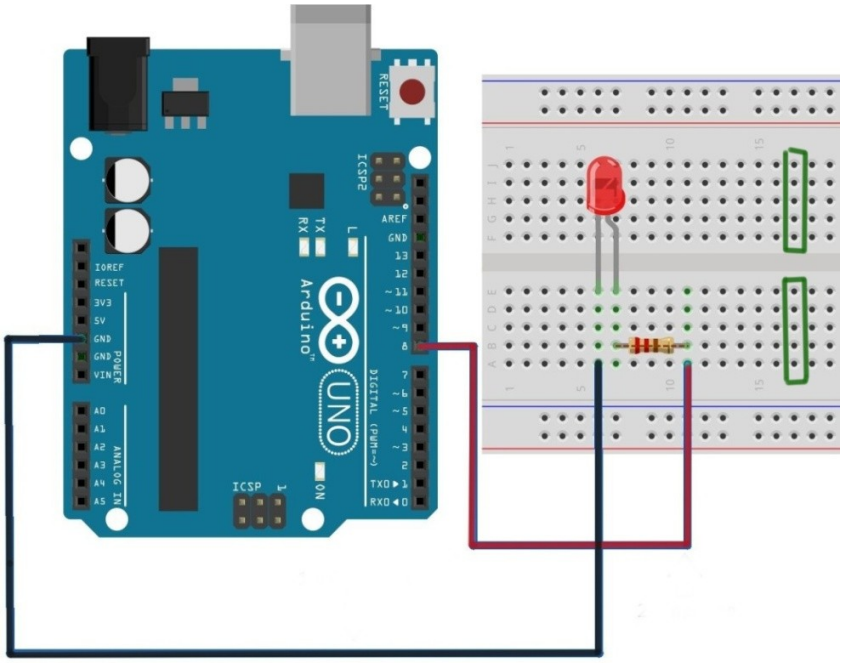
Arduino Uno կամ Arduino Nano – Եվ իհարկե մեզ անհրաժեշտ է Arduino Uno կամ Arduino Nano տպասալ (նկ 1.): Ինչպես տեսնում եք նկարում **UNO**-ն ունի համակարգչի **USB** բնիկին միանալու հնարավորություն (USB Port to computer), **DC Power Jack**-ի բնիկ, որից կարելի է միացնել 7-12Վ լարման մարտկոց, աջ կողմում կարմիր փակագծերում նշված թվերով սահմանված են համապատասխան թվային անուններով **pin-**երը (0 և 1 **pin** – երը մասնավորապես օգտագործում են **Bluetooth մոդուլ** միացնելիս, իսկ **PWM** տառերով նշված **pin** –երը հանդիսանում են **ԼԻՄ** տեսակի, որոնց մասին արդեն նշվել է): Ձախ կողմում կարմիր տառերով նշված A0- ից A5 **pin –** երը հանդիսանում են անալոգային: Նույն շարքը շարունակում են հոսանքի միացման կամ փոխանցման կոնտակտները (**VIN** – մինչև 12Վ լարման միացում կամ փոխանցում այլ սարքի): Ահա մեր նախագծերին անհրաժեշտ հիմնային գիտելիքները: Ավելի մանրամասն անդրադարձ կլինի նախագծերի ուսումնասիրման հատվածներում:



2. Նախագիծ 1 - Լուսադիոդի միացում

Այս նախագծում մեզ անհրաժեշտ են լուսադիոդներ, դիմադրություններ, Arduino Uno տպասալ և Breadboard:

Միացման սխեմա.



Նկ. 3

Լուսադիոդի միացման համար ինչպես նախապես հասկացանք երկար ելքը պետք է միանա **pin-**երից որևէ մեկին (տվյալ դեպքում **pin 8** - ին), իսկ մյուսը՝ **GND –** ին: **Breadboard-**ը մեզ կօգնի առանց լարերը միմյանց փաթաթելու կամ ամրացնելու հավաքել մեզ անհրաժեշտ սխեման: **Breadboard**-ի վրա **+** և **-** նշանների ուղղությամբ բոլոր կոնտակտները միացված են իրար (տվյալ նշանները պայմանականորեն են դրված), ինչպես նաև կանաչ ուղղանկյունների ներսում գտնվող բոլոր կոնտակտները միացված են իրար: Նույն տրամաբանությամբ այդ շարքի մյուս կոնտակտները նույնպես միացված են իրար: Նախքան միացմանն անցնելը հասկանանք դիմադրության արժեքի որոշումը և նրա տեղադրման նշանակությունը:

Ենթադրենք Arduino տպասալը միացված է +5Վ լարման: Այս դեպքում համապատասխանաբար **pin 8** – ի թվային ազդանշանը **HIGH** արժեքի դեպքում կլինի +5Վ: Իսկ լուսադիոդի թույլատրելի լարումը կազմում է +2Վ: Օգտվելով ֆիզիկայի գիտելիքներից փորձենք գտնել այնպիսի դիմադրություն, որն իր վրա կվերցնի մնացած +3Վ լարումը:

Ցանկացած ճարտարագետ - էլեկտրիկի համար ամենակարևոր բանաձևը **Օհմի** օրենքն է հանդիսանում, որը ներկայացնում է շղթայում հոսանքի ուժի (Ա), լարման (Վ) և դիմադրության (Օհմ) հարաբերակցությունը: Կարևոր է հասկանալ այս տերմինների ֆիզիկական նշանակությունը.

- Լարումն իրենից ներկայացնում է 2 կետերի միջև էլեկտրական պոտենցիալների տարբերությունը,
- Հոսանքը հոսում է բարձր էլեկտրական պոտենցիալ էներգիա ունեցող կետից, որպիսի նվազեցնի պոտենցիալ էներգիան: Համեմատական անցկացնելով կարելի է հոսանքի ուժը պատկերացնել որպես ջրի հոսք, իսկ լարումը՝ որպես ջրվեժի բարձրություն: Ջուրը (հոսանքը) հոսում է ավելի բարձր կետից (բարձր լարումից) դեպի ավելի ցածր կետը (ցածր լարում ունեցող կետ): Հոսանքը գետի ջրի նման միշտ կհոսի այն ճանապարհով, որտեղ ամենաքիչ դիմադրությանը կհանդիպի:
- Համեմատության տեսանկյունից դիմադրությունը նման է խողովակի հաստությանը, որքան խողովակը նեղ է, այնքան դիմադրությունը մեծ է և միավոր ժամանակում ավելի քիչ քանակով ջուր (հոսանք) կարող է անցնել և հակառակը:

Օհմի օրենքը արտահայտվում է հետևյալ բանաձևի միջոցով՝

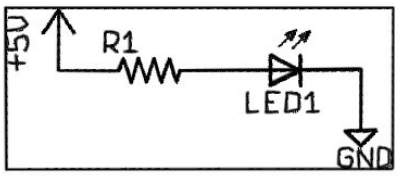
$$\mathbf{U=IR},$$

որտեղ U-ն լարումն է (Վ), I-ն՝ հոսանքի ուժը (Ա), իսկ R-ը՝ դիմադրությունը (Օհմ):

Լուսադիոդը բնութագրվում է որոշակի առավելագույն թույլատրելի լարման և հոսանքի ուժի արժեքով: Որքան մեծ քանակության հոսանք անցնի նրանով (չգերազանցելով թույլատրելի արժեքը), այնքան ավելի մեծ կլինի լույսի պայծառությունը: Ամենատարածված լուսադիոդների համար թույլատրելի հոսանքի ուժի արժեքը կազմում է 20 մԱ = 0.02 Ա:

1000 միլիԱմպեր = 1 Ամպեր

Դիտարկենք նկ. 4-ում պատկերված միացման սխեման և կիրառելով Օհմի օրենքը շղթայի համար որոշենք R1 դիմադրության արժեքը:



Նկ. 4

Ենթադրենք, որ LED1 լուսադիոդը ստանդարտ տեսակի է՝ վերը նշված բնութագրերով (0.02Ա առավելագույն թույլատրելի հոսանքի ուժ, 2Վ

առավելագույն թույլատրելի լարում): 5Վ լարումը պետք է բաշխվի LED1 լուսադիոդի և R1 դիմադրության միջև: Քանի որ լուսադիոդի թույլատրելի արժեքը 2Վ է, ապա մնացած 3Վ լարումը պետք է ուղղված լինի դիմադրությանը: Օհմի օրենքից օգտվելով կարող ենք հաշվել՝

$$R1=U/I=3/0.02=150\ \text{Օհմ}$$

Այս կերպ 150 Օհմ դիմադրություն օգտագործելիս նրանում և լուսադիոդով կանցնի 0.02 Ա հոսանք: Մեծացնելով դիմադրությունը հոսանքի ուժի արժեքը կնվազի: 220 Օհմ դիմադրության դեպքում լուսադիոդը բավականին պայծառություն կարողանում է ապահովել (կօգտագործենք 220 Օհմ դիմադրությունը, քանի որ այն համեմատաբար ավելի տարածված է):

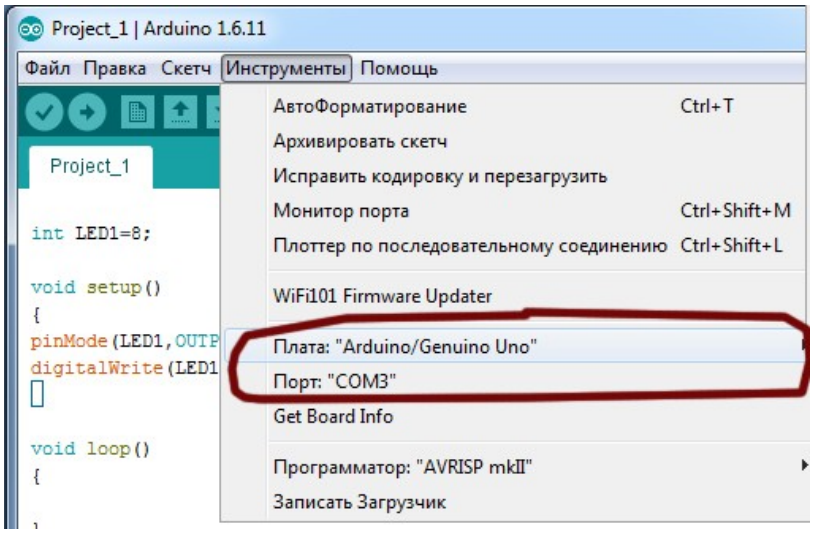
Ծրագրավորում.

```
int LED1=8;                    //լուսադիոդի կոնտակտի համարը (pin 8)

void setup()
{
pinMode(LED1,OUTPUT);          // սահմանում ենք լուսադիոդի կոնտակտի տեսակը՝ որպես OUTPUT
digitalWrite(LED1,HIGH);       // լուսադիոդի ելքում HIGH արժեք ենք սահմանում (+5Վ)
}

void loop()
{
                               // Ցիկլային ենթածրագրում ոչինչ չենք գրում
}
```

Նկ. 5

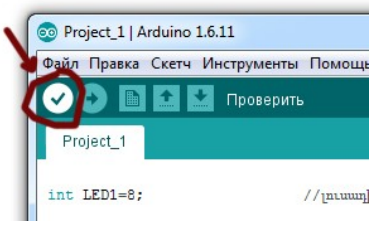
Մուտք գործելով **Arduino IDE** ծրագիր ներմուծում ենք նկ. 5 –ում ներկայացված ծրագրային կոդը: Առաջին տողում հայտարարել ենք փոփոխական LED1 անունով, որը մեզ մոտ կհանդիսանա որպես լուսադիոդի միացման կոնտակտի համարը (մեր դեպքում **pin 8**), **void setup()** ենթածրագրում ինչպես նախապես արդեն ծանոթացել ենք սահմանում ենք ծրագրի սկզբում մեկ անգամ կատարվող գործողություններ, մասնավորապես **pin**-երի տեսակների սահմանումներ: Այն կատարվում է **pinMode** հրամանով, որտեղ մեր օրինակում LED1 պարամետրը 8 արժեքն ունի (համապատասխանաբար կատարում ենք **pin 8**-ի սահմանում), իսկ երկրորդ պարամետրը սահմանում է, թե ինչ տեսակի պետք է այն լինի: Մեր օրինակում կարող ենք լուսադիոդի միացման հրամանը տեղադրել հենց **void setup() -** ում, քանի որ մեզ անհրաժեշտ է պարզապես միացնել այն: **void loop()** ենթածրագիրն այս դեպքում դատարկ կմնա: հաջորդ օրինակում, երբ կկատարենք լուսադիոդի միացում և անջատում, արդեն հրամանները կտեղակայենք **void loop()** ենթածրագրում: Այս ծրագիրը **Arduino Uno** տպասալի պրոցեսոր ներբեռնելու համար միացնում ենք վերջինս համակարգչին՝ USB լարի միջոցով, ծրագրի ներսում ընտրում ենք Arduino-ի

տեսակը, որն օգտագործելու ենք և համապատասխան USB պորտի համարը, որին միացրել ենք (նկ. 6):

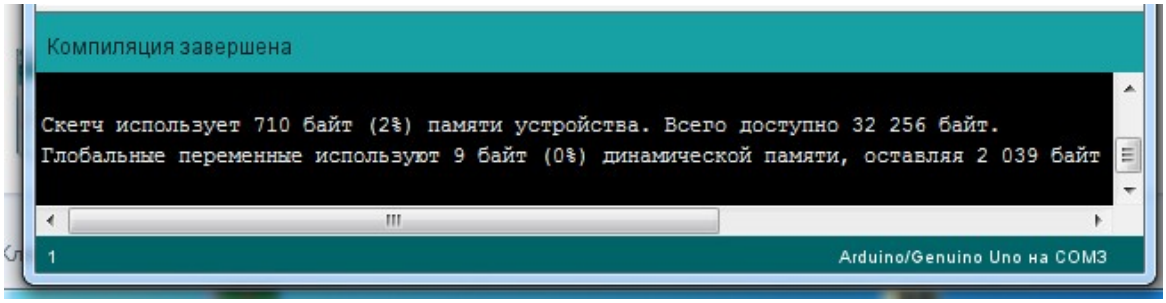


Նկ. 6

Ցանկացած ծրագիր, նախքան այն ներբեռնելը Arduino տպասալի պրոցեսսոր կարող ենք ստուգել կոդային սխալների առկայությունը (նկ. 7):

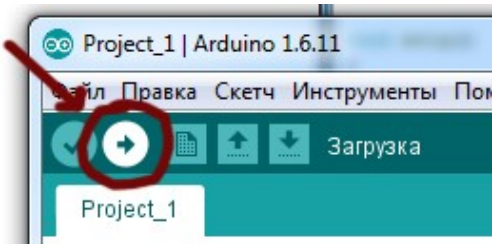


Սխալների բացակայության դեպքում ներքևի հատվածում կհայտնվի հետևյալ հաղորդագրությունը (նկ 8).



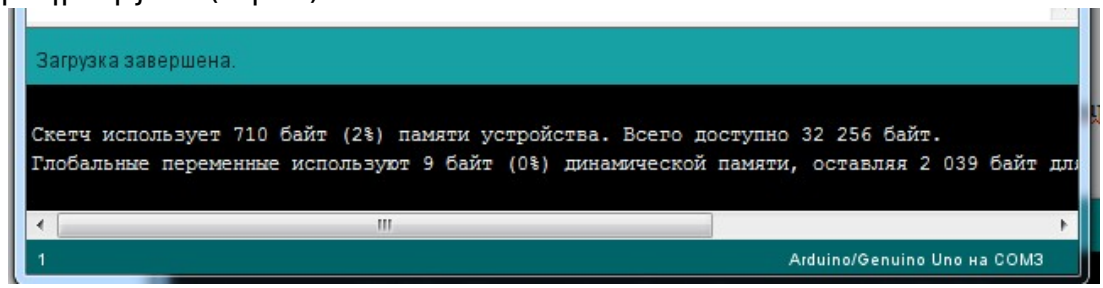
Նկ. 8

Ծրագիրը ներբեռնելու համար արդեն կսեղմենք սլաքի նշանը (նկ 9)։



Նկ. 9

Ծրագրի ներբեռնումն ավարտվելուց հետո ներքևի հատվածում կտեսնեք նմանատիպ հաղորդագրություն (նկ 10).



Նկ. 10

Վերը նշված քայլերը ճիշտ կատարելու դեպքում լուսադիոդի լույսը կվառվի։

Որպիսի լուսադիոդի լույսը վառվելուց հետո կարողանանք նաև այն անջատել ծրագրում կատարենք հետևյալ փոփոխությունները (նկ. 11)՝

```
int LED1=8;                 //լուսադիոդի կոնտակտի համարը (pin 8)

void setup()
{
pinMode(LED1,OUTPUT);       // սահմանում ենք լուսադիոդի կոնտակտի տեսակը՝ որպես OUTPUT
}

void loop()
{
 digitalWrite(LED1,HIGH);    // լուսադիոդի ելքում HIGH արժեք ենք սահմանում (+5Վ)
 delay(1000);                // սպասել 1000 միլիվայրկյան (1 վայրկյան)
 digitalWrite(LED1,LOW);     // լուսադիոդի ելքում LOW արժեք ենք սահմանում (0Վ)
 delay(1000);                // սպասել 1000 միլիվայրկյան (1 վայրկյան)
}
```

Նկ. 11

Ինչպես տեսնում ենք այստեղ ավելացել է նոր հրաման՝ delay(1000), որն օգտագործում ենք դրան նախորդող հրամանի կատարման տևողությունը կառավարելու համար: Փակագծերում գրված թիվը չափվում է միլիվայրկյաններով: Այս օրինակում կստացվի ամեն գործողությունը կատարելուց հետո սպասել 1 վայրկյան, ապա անցնել հաջորդին: Այս ծրագրի արդյունքը կլինի լուսադիոդի անընդհատ միացում և անջատում՝ կատարելով դրանցից յուրաքանչյուրը 1 վայրկյան:

Դիտարկենք լուսադիոդի միացումը **ԼԻՄ** (**PWM**) **pin**-երից որևէ մեկով: Միացնենք **3**-րդ **pin**-ին: Այս դեպքում հնարավորություն ունենք լուսադիոդի պայծառությունը փոփոխելու թվային արժեքներով: Ծրագիրը կստանա հետևյալ տեսքը (նկ. 12)՝

```
int LED1=3;                    //լուսադիոդի կոնտակտի համարը (pin 3)

void setup()
{
pinMode(LED1,OUTPUT);          // սահմանում ենք լուսադիոդի կոնտակտի տեսակը՝ որպես OUTPUT
}

void loop()
{
 analogWrite(LED1,255);         // լուսադիոդի ելքում 255 արժեք ենք սահմանում (+5Վ)
 delay(1000);                   // սպասել 1000 միլիվայրկյան (1 վայրկյան)
 analogWrite(LED1,0);           // լուսադիոդի ելքում 0 արժեք ենք սահմանում (0Վ)
 delay(1000);                   // սպասել 1000 միլիվայրկյան (1 վայրկյան)
}
```

Նկ. 12

Այստեղ **digitalWrite** հրամանը փոխարինվում է **analogWrite** հրամանով, որի 2-րդ պարամետրը ընդունում է 0(0Վ)-ից 255(+5Վ) միջակայքի արժեք: Այդ արժեքից կախված լուսադիոդն իր նվազագույն և առավելագույն արժեքների սահմաններում համապատասխան պայծառությամբ կսկսի վառվել:

Դիտարկենք մեկ այլ ծրագիր, որի արդյունքում լուսադիոդի լույսը անջատված վիճակից կսկսի միանալ ու հասնել իր առավելագույն արժեքին և հակառակը: Այս դեպքում մենք կօգտագործենք **for** ցիկլի օպերատորը, որին նախապես անդրադարձել ենք: Այս դեպքում ծրագրի առաջին մասում **for** ցիկլը կկատարի աճման գործընթաց՝ **for (int i=0; i<=255; i++),** որի փոփոխման քայլը կլինի 1 միավոր, իսկ երկրորդ մասում կկատարի նվազման գործընթաց՝ **for (int i=255; i>=0; i--),** որի ժամանակ նվազման քայլը նորից կլինի 1 միավոր:

Այս դեպքում **analogWrite** հրամանի երկրորդ պարամետրը կդառնա **for** ցիկլի ինդեքսը՝ **i**-ն, որի փոփոխման ընթացքում լուսադիոդի լույսի պայծառությունը կսկսի փոփոխվել (նկ. 13):

```
int LED1=3;                         // լուսադիոդի կոնտակտի համարը (pin 3)

void setup()
{
pinMode(LED1,OUTPUT);               // սահմանում ենք լուսադիոդի կոնտակտի տեսակը՝ որպես OUTPUT
}

void loop()
{
  for (int i=0; i<=255; i++)        // աճման ցիկլ
  {
    analogWrite(LED1,i);            // լուսադիոդի ելքում i արժեք ենք սահմանում (+5Վ)
    delay(10);                      // սպասել 10 միլիվայրկյան
  }
 for (int i=255; i>=0; i--)         // նվազման ցիկլ
  {
    analogWrite(LED1,i);            // լուսադիոդի ելքում i արժեք ենք սահմանում (+5Վ)
    delay(10);                      // սպասել 10 միլիվայրկյան
  }
}
```

Նկ. 13

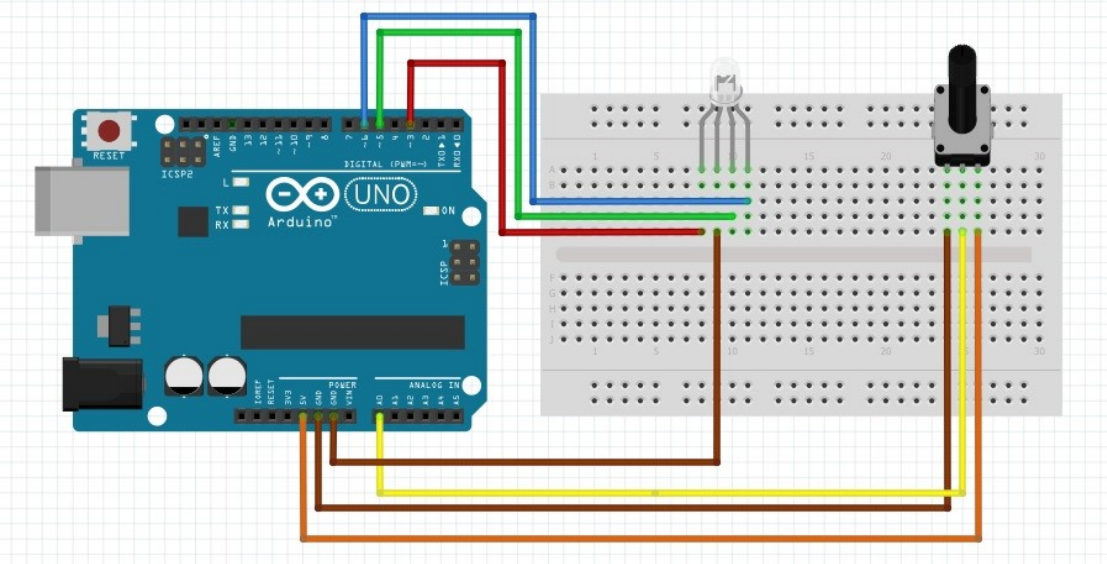
Որպիսի **i** փոփոխականի արժեքն ակնթարթորեն նվազագույնից առավելագույն և հակառակը չհասնի օգտագործում ենք delay(10) հրամանը՝ 10 միլիվայրկյան ընդմիջումներ սահմանելով: Նույն կերպ կարելի է միացնել մեկից ավել լուսադիոդներ՝ ամեն մեկին համապատասխանաբար տարբեր **pin-**երի միացնելով, որպիսի դրանցից ամեն մեկի լույսի պայծառությունը կարողանանք ծրագրային մասով կառավարելի դարձնել:

Առաջադրանք. Ստեղծել լուսամփոփի մոդել՝ 3 լուսադիոդ (կարմիր, դեղին, կանաչ) մեքենայի համար և 2 լուսադիոդ (կարմիր, կանաչ) հետիոտնի համար: Դրանց աշխատանքը կարգավորել հետևյալ սկզբունքով՝ մեքենայի լուսամփոփի կարմիր լույսը հետիոտնի լուսամփոփի կանաչ լույսի հետ միաժամանակ միանում է 5 վայրկյան որից հետո հետիոտնի լուսամփոփը մնում է նույն կարգավիճակում, իսկ մեքենայի լուսամփոփի կարմիր լույսը փոխվում է դեղինով: Այդ վիճակում կես վայրկյան մնալով մեքենայի լուսամփոփի դեղին լույսը փոխվում է կանաչով, իսկ հետիոտնի լուսամփոփի կանաչը՝ կարմիրով: Այդ վիճակում 5 վայրկյան մնալով նույն գործողությունները կատարել նաև հետադարձ հաջորդականությամբ:

3. Նախագիծ 2 - Պոտենցոմետր և RGB լուսադիոդ

Այս նախագծի համար մեզ անհրաժեշտ է Պոտենցոմետր 10ԿOհմ դիմադրությամբ, RGB տեսակի լուսադիոդ, Arduino Uno միկրոկոնտրոլլեր և Breadboard։

Միացման սխեմա.



Նկ. 14

Նախ ծանոթանանք պոտենցոմետրի աշխատանքին և միացմանը, ինչպես նաև փորձենք նրանից ստացվող ազդանշանը կարդալ անալոգային **INPUT**-ներից որևէ մեկում։ Եզրային կոնտակտներից մեկը կմիացնենք 5Վ-ին, իսկ մյուսը՝ GND-ին։ Մեջտեղի կոնտակտը ազդանշանի փոխանցման համար կմիացնենք **A0**-ին (նկ. 14)։ Այս օրինակում կփորձենք ազդանշանի արժեքը կարդալ **analogRead** հրամանով և այն կարտածենք **Serial Monitor**-ի վրա, որը կարող ենք բացել **Arduino IDE** ծրագրավորման միջավայրի աջ վերևի հատվածում համապատասխան նշանը սեղմելով։ (Նախագիծ 6-ում նորից անդրադարձ՝ նկ. 28)։

Արտածելու համար ծանոթանանք նոր հրամանների հետ.

Serial.begin(9600); - այս հրամանով **void setup** ենթածրագրում կսահմանենք տվյալների փոխանցման և ստացման արագությունը, որը չափվում է բիթ/վայրկյան կամ **baud** միավորով։

Serial.println(); - USB բնիկի միջոցով միկրոկոնտրոլլերի **A0**-ում ստացված ազդանշանի արժեքը կարտածենք հետևյալ հրամանի միջոցով։ **println**

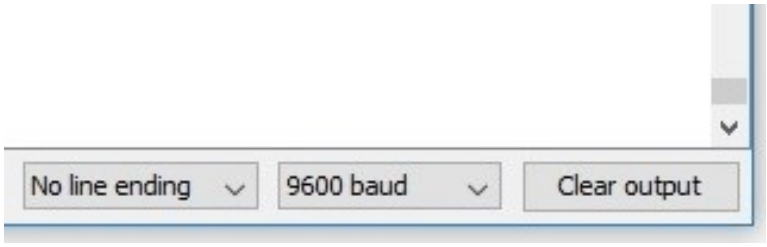
հրամանի դեպքում ամեն հաջորդ արժեքը կարտածի հաջորդ տողում, իսկ եթե այն փոխարինենք **print** հրամանով, ապա արժեքները կարտածվեն իրար հետևից։

Ծրագրավորում.

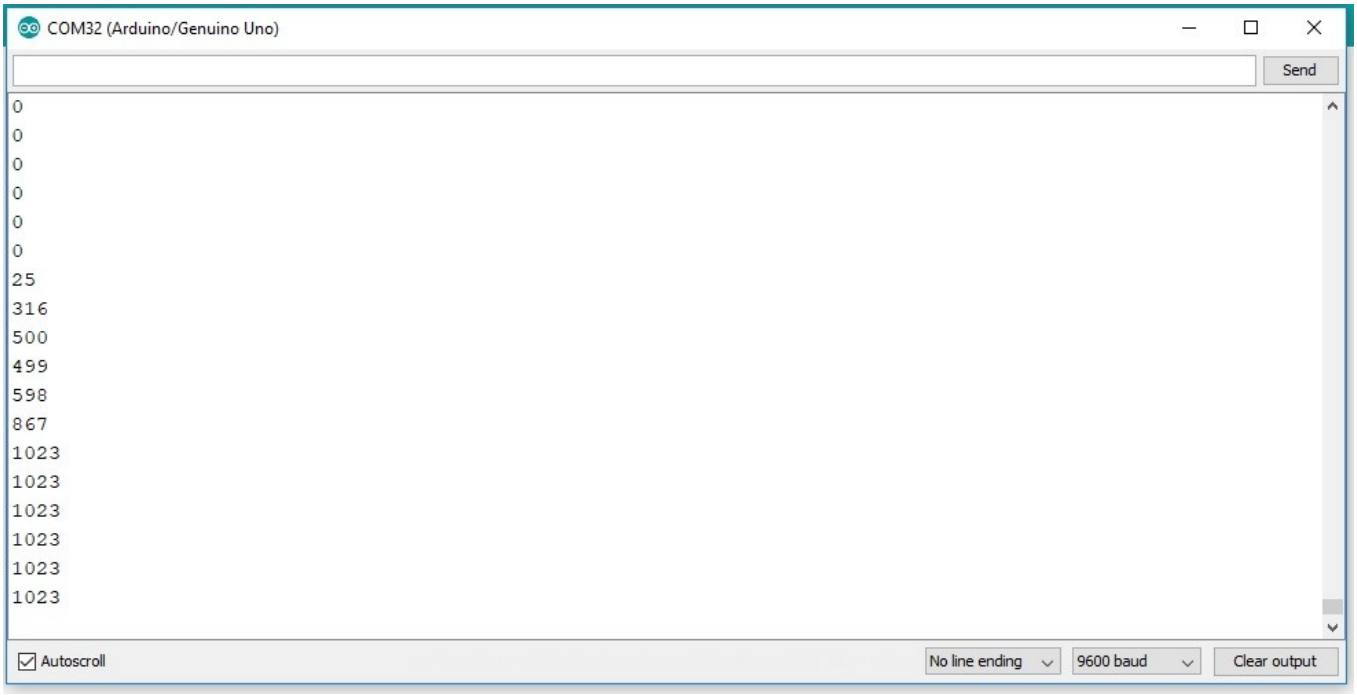
```
void setup() {

   Serial.begin(9600);  // սահմանում ենք տվյալների փոխանցման և ստացման արագությունը

}
void loop() {

 int P=analogRead(A0);     //   ստեղծում ենք ամբողջ տեսակի փոփոխական,
                           //   որի մեջ պահում ենք A0-ից ստացված ազդանշանի արժեքը

       Serial.println(P); //  արտածում ենք այդ փոփոխականի արժեքը
       delay(100);        //  սպասել 100 միլի վրկ.
}
```

Նկ. 15

Այստեղ ինչպես նշեցինք ավելանում է տվյալների փոխանցման և ստացման արագությունը սահմանող հրաման, որի փակագծերում գրված թիվն արտահայտվում է բիթ/վայրկյան-ով։ **Serial Monitor**-ը բացելուց հետո ներքևի աջ անկյունում կտեսնենք ստացման արագության թվային արժեքը։ Այն պետք է համապատասխանության մեջ դրվի մեր կողմից ծրագրային կոդում սահմանված արագության թվի հետ՝ տվյալ դեպքում 9600 baud (նկ. 16)։ Այնուհետև ստեղծում ենք **int** տեսակի փոփոխական, որի մեջ պահում ենք պոտենցոմետրից փոխանցած ազդանշանի արժեքը, իսկ **Serial.println** հրամանով կարտածենք ստացված արժեքը ամեն 100 միլիվայրկյանը մեկ (նկ. 17)։

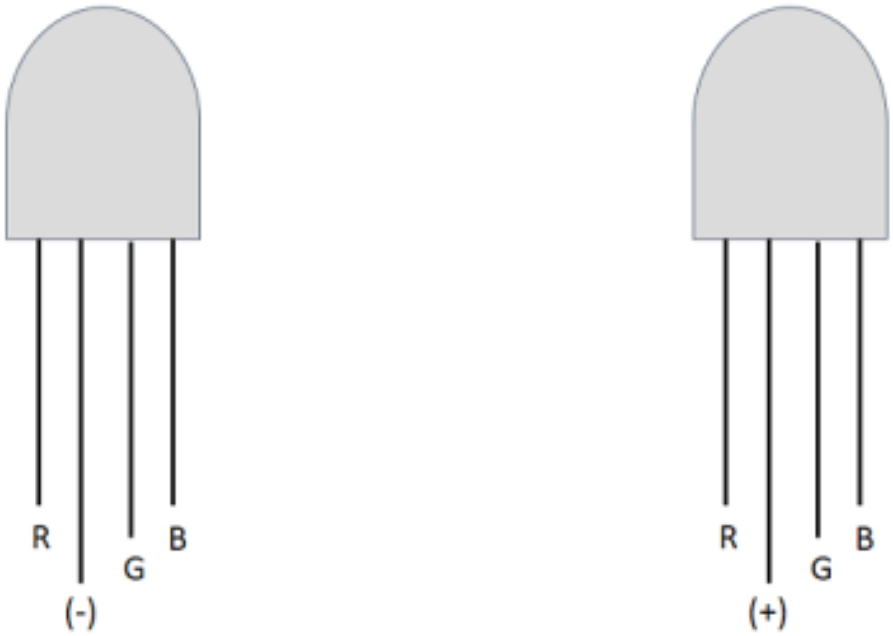


Նկ. 16



Նկ. 17

Առանձին դիտարկենք նաև RGB լուսադիոդի աշխատանքը, այնուհետև այն կհամադրենք պոտենցոմետրի աշխատանքի հետ: Ինչպես ծանոթացանք վերևում այն ունի 2 տեսակ՝ անոդային և կատոդային: Կախված, թե որ տեսակի հետ ենք մենք աշխատելու ազդանշանի թվային արժեքները տարբեր կլինեն: Նախ արտաքինից տարբերակելու համար GND տեսակն ավելի բյուրեղային թափանցիկ տեսք ունի: Անոդային տեսակի երկար ոտիկը միանում է 5Վ-ին, իսկ կատոդայինի դեպքում՝ GND-ին: Մյուս 3 ոտիկները համապատասխանաբար կարմիր, կանաչ և կապույտ լույսերի միացման համար են: Այս դեպքում երկար ոտիկը կհանդիսանա երեքի համար ընդհանուր կոնտակտ: Անոդայինի համար այն կհանդիսանա ընդհանուր (+), այսինքն 255 արժեք ստացող կոնտակտ, իսկ կատոդայինի համար՝ ընդհանուր (-), այսինքն 0 արժեք ստացող կոնտակտ (նկ. 19): Նշված 3 գույնի լույսերից ցանկացածը միացնելու համար անհրաժեշտ է այնպիսի թվային ազդանշան ուղարկել, որի տարբերությունը իրեն համապատասխան տեսակի լուսադիոդի (կատոդային կամ անոդային) երկար ոտիկի արժեքի հետ 0-ից տարբեր թիվ կլինի: Այսինքն եթե մենք աշխատում ենք անոդային տեսակի լուսադիոդի հետ, ապա լույսերից ցանկացածը միացնելու համար նրան պետք է տալ 0-ից 254 թվային արժեք: 255 արժեքն արդեն կանջատի լույսը, քանի որ տարբերությունը կլինի 0: Իսկ Կատոդայինի դեպքում պետք է ազդանշանը լինի 1-ից 255 միջակայքի արժեք, 0-ի դեպքում տարբերությունը նույն 0-ն կլինի: Կախված փոխանցվող թվային արժեքից այդ լույսի պայծառությունը տարբեր կլինի: Ինչքան տարբերությունը մեծ լինի, այնքան ավելի պայծառ կվառվի:



Նկ. 18

Դիտարկենք կատոդային լուսադիոդի միացումը՝այսինքն երկար ոտիկը կմիացնենք **GND**-ին, իսկ մյուս 3-ը համապատասխանաբար **3,5,6 pin**-երին, քանի որ դրանք **ԼԻՄ** տեսակի են: Փորձենք ըստ հերթականությամբ միացնել և անջատել նշված երեք գույնի լույսերը՝ 1 վայրկյան սպասումներով: Ազդանշանի արժեքը 255 կլինի, որպիսի առավելագույն պայծառությամբ այն միացնենք: Ամեն լույսի միացման հետ մյուս 2 լույսերը կանջատենք: Ծրագրի կոդը կունենա հետևյալ տեսքը.

```
   /*սահմանում ենք հաստատուններ, որոնց կօգտագործենք RGB լուսադիոդի համապատասխանաբար
   կարմիր, կանաչ և կապույտ լույսերը միացնելու համար*/

#define R 3   // R անունով հաստատուն, որին կվերագրենք 3 արժեք՝ 3-րդ pin-ի համար
#define G 5   // G անունով հաստատուն, որին կվերագրենք 5 արժեք՝ 5-րդ pin-ի համար
#define B 6   // B անունով հաստատուն, որին կվերագրենք 6 արժեք՝ 6-րդ pin-ի համար

void setup() {
  pinMode(R, OUTPUT);      // R-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
  pinMode(G, OUTPUT);      // G-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
  pinMode(B, OUTPUT);      // B-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
}
void loop() {

     analogWrite(R,255);
     analogWrite(G,0);
     analogWrite(B,0);
        delay(1000);
     analogWrite(R,0);
     analogWrite(G,255);
     analogWrite(B,0);
        delay(1000);
     analogWrite(R,0);
     analogWrite(G,0);
     analogWrite(B,255);
        delay(1000);

}
```

Նկ. 19

#define հրամանով սահմանում ենք հաստատուններ, որոնց արժեքը ծրագրի կոդի որևէ հատվածում չի կարող փոփոխման ենթարկվել:

Ինքնուրույն կարող եք փորձել միաժամանակ 3 գույներին տարբեր արժեքներ փոխանցելով RGB գունային համակարգի գույների համադրություններ ստանալ:

Իսկ հիմա պոտենցոմետրի արժեքից կախված փորձենք լուսադիոդի գույներից մեկը փոփոխել։ Այստեղ արդեն կօգտագործենք վերը նշված **map** հրամանը, որի միջոցով պոտենցոմետրի ընթացիկ արժեքը 0-1023 միջակայքից կփոխակերպենք 0-255 միջակայքի ընթացիկ արժեքի, որն էլ կփոխանցենք որպես լույսերից մեկի միացման ազդանշան։ Ծրագրի կոդը կունենա հետևյալ տեսքը.

```
/*սահմանում ենք հաստատուններ, որոնց կօգտագործենք RGB լուսադիոդի համապատասխանաբար
կարմիր, կանաչ և կապույտ լույսերը միացնելու համար*/

#define R 3   // R անունով հաստատուն, որին կվերագրենք 3 արժեք՝ 3-րդ pin-ի համար
#define G 5   // G անունով հաստատուն, որին կվերագրենք 5 արժեք՝ 5-րդ pin-ի համար
#define B 6   // B անունով հաստատուն, որին կվերագրենք 6 արժեք՝ 6-րդ pin-ի համար

int P1,L1;     // սահմանում ենք P1 և L1 անուններով int տեսակի փոփոխականներ
void setup() {
  pinMode(R, OUTPUT);       // R-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
  pinMode(G, OUTPUT);       // G-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
  pinMode(B, OUTPUT);       // B-ի համապատասխան pin-ին սահմանում ենք OUTPUT տեսակ
}
void loop() {

    P1=analogRead(A0);          // P1-ի մեջ պահում ենք A0-ից ստացված ազդանշանի արժեքը
    L1=map(P1,0,1023,0,255);  // L1-ի մեջ կստանանք փոխված միջակայքի P1-ին համապատասխան արժեք
      analogWrite(R,L1);        // R-ի համապատասխան pin-ին ուղարկենք L1 արժեքով ազդանշան
}
```

Նկ. 20

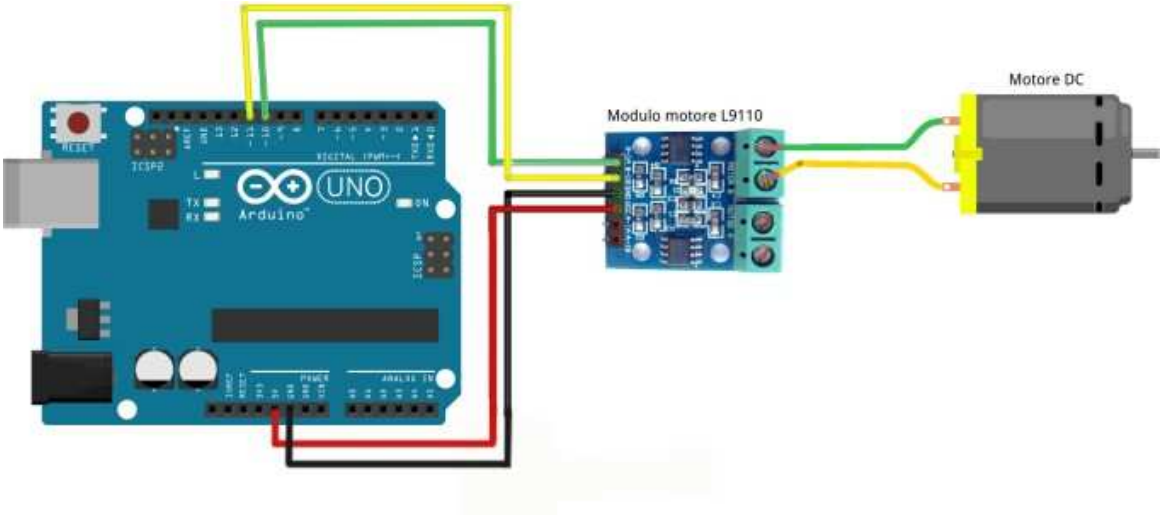
Այստեղ P1 փոփոխականը պոտենցոմետրի ընթացիկ արժեքն է ստանում, իսկ L1 փոփոխականը 0-255 միջակայքում դրան համապատասխան ընթացիկ արժեք, որն էլ որպես ազդանշան փոխանցում է լույսերից մեկին՝ տվյալ դեպքում կարմիր լույսին համապատասխան **pin**-ին։

Առաջադրանք. Միացնել միաժամանակ 3 պոտենցոմետր, ստանալ դրանցից ամեն մեկի արժեքները տարբեր անալոգային **INPUT**-ներում, այնուհետև ամեն պոտենցոմետրի ընթացիկ արժեքին համապատասխան ստանալ 0-255 միջակայքի ընթացիկ արժեքներ, որոնք կփոխանցվեն որպես կարմիր, կանաչ և կապույտ լույսերի ազդանշանի արժեքներ։ Այդ նախագծի միջոցով կկարողանաք պոտենցոմետրերի պտույտներով տարբեր գունային երանգներ ստանալ RGB գունային համակարգում։

4. Նախագիծ 3 - Շարժիչի միացում L9110 մոդուլով

Այս նախագծում մեզ անհրաժեշտ է DC շարժիչ, շարժիչի աշխատանքի կարգավորման մոդուլ (Motor contorller), որը մեր օրինակում կլինի L9110 մոդելը և Arduino Uno միկրոկոնտրոլլեր:

Միացման սխեմա.



Նկ. 21

Այս օրինակում կօգտագործենք 1 շարժիչ, սակայն տվյալ մոդուլը հնարավորություն է տալիս միաժամանակ միացնել և կառավարել 2 շարժիչներ: Շարժիչի աշխատանքի համար նրա կոնտակներից մեկը պետք է ստանա 0Վ լարում, իսկ մյուսը 0Վ-ից տարբեր (իր նվազագույն անհրաժեշտ լարումից բարձր արժեք): Պտտման ուղղությունը փոխելու համար կոնտակտների արժեքները տեղերով պետք է փոխել, իսկ աշխատանքը դադարեցնելու համար պետք է 2 կոնտակտներին միաժամանակ 0 արժեք փոխանցել: Ինչպես տեսնում ենք (նկ. 21) շարժիչի միացման համար մենք օգտագործել ենք 10 և 11 համարի **pin**-երը: Համապատասխան մոդուլի վրա մի կողմում կան 6 կոնտակտներ, որոնցից զույգ եզրայինները շարժիչների միացման համար են, իսկ մեջտեղի 2-ը **GND** և **VCC** կոնտակտներն են (**VCC-**ն կարող է միանալ Arduino-ի **5V**, **3.3V**, **VIN** կոնտակտներին, կամ առանձին մինչև 12Վ լարման մարտկոցի): Այս նախագծում միացված է **5V** կոնտակտին: Մոդուլի մյուս կողմում կա 4 կոնտակտ, որոնք միանում են հենց շարժիչներին: Վերը նշված սխեմայի համար ստեղծենք **Arduino IDE** միջավայրում աշխատանքային ծրագիր, որը շարժիչն անընդհատ կպտտի 1 վայրկյան, այնուհետև 1 վայրկյան դադար կտա (նկ. 22): Այստեղ **M1** փոփոխականը շարժիչի 1-ին կոնտակտի համարն է(**pin 10**), իսկ **M2**-ը՝ 2-րդ

կոնտակտի համարն է (**pin 11**): Այստեղ նույնպես կօգտագործենք **analogWrite** հրամանը, որպիսի կարողանանք շարժիչի արագության արժեքը փոփոխենք:

Ծրագրավորում.

```
int M1=10;                    // շարժիչի 1-ին կոնտակտի համարը (pin 10)
int M2=11;                    // շարժիչի 2-րդ կոնտակտի համարը (pin 111)

void setup()
{
  pinMode(M1,OUTPUT);           // սահմանում շարժիչի 1-ին կոնտակտի տեսակը՝ որպես OUTPUT
  pinMode(M2,OUTPUT);           // սահմանում շարժիչի 2-րդ կոնտակտի տեսակը՝ որպես OUTPUT
}

void loop()
{
  analogWrite(M1,0);          // շարժիչի 1-ին կոնտակտին փոխանցում ենք 0 արժեք
  analogWrite(M2,255);        // շարժիչի 2-րդ կոնտակտին փոխանցում ենք 255 արժեք
  delay(1000);
  analogWrite(M1,0);          // շարժիչի 1-ին կոնտակտին փոխանցում ենք 0 արժեք
  analogWrite(M2,0);          // շարժիչի 2-րդ կոնտակտին փոխանցում ենք 0 արժեք
  delay(1000);
}
```

Նկ. 22

Որպիսի պտույտի ուղղությունը փոխենք, **void loop()** ենթածրագրում կավելացնենք նաև հակադարձ արժեքներով հրամանները: Արդյունքում ծրագիրը կունենա հետևյալ տեսքը, իսկ շարժիչը անընդհատ կպտտվի մի ուղղությամբ 1 վայրկյան, դադար կունենա 1 վայրկյան և կպտտվի հակառակ ուղղությամբ նորից 1 վայրկյան (նկ. 23):

```
void loop()
{
  analogWrite(M1,0);
  analogWrite(M2,255);
   delay(1000);
  analogWrite(M1,0);
  analogWrite(M2,0);
   delay(1000);
  analogWrite(M1,255);
  analogWrite(M2,0);
   delay(1000);
  analogWrite(M1,0);
  analogWrite(M2,0);
   delay(1000);
}
```

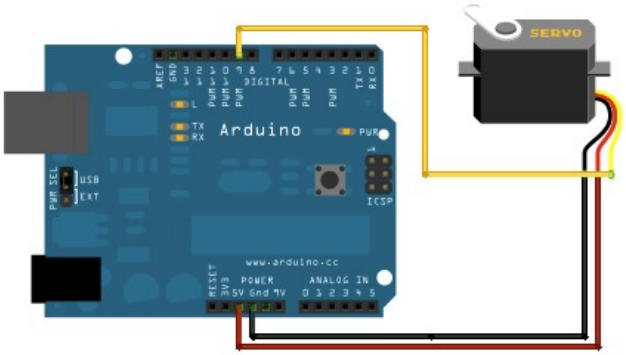
Նկ. 23

Առաջադրանք. Միացնել միաժամանակ 2 շարժիչ, ծրագրավորել վերջիններիս աշխատանքը հետևյալ կերպ. 2 շարժիչները պտտել նույն ուղղությամբ 5 վայրկյան, դրանցից մեկը պտտել հակառակ ուղղությամբ 4 վայրկյան, այնուհետև հաջորդը պտտել հակառակ ուղղությամբ 3 վայրկյան և դադար՝ 2 վայրկյան:

5. Նախագիծ 4 - Servo շարժիչի միացում

Այս նախագծում մեզ անհրաժեշտ է Սերվո շարժիչ և Arduino Uno միկրոկոնտրոլլեր:

Միացման սխեմա.



Նկ. 24

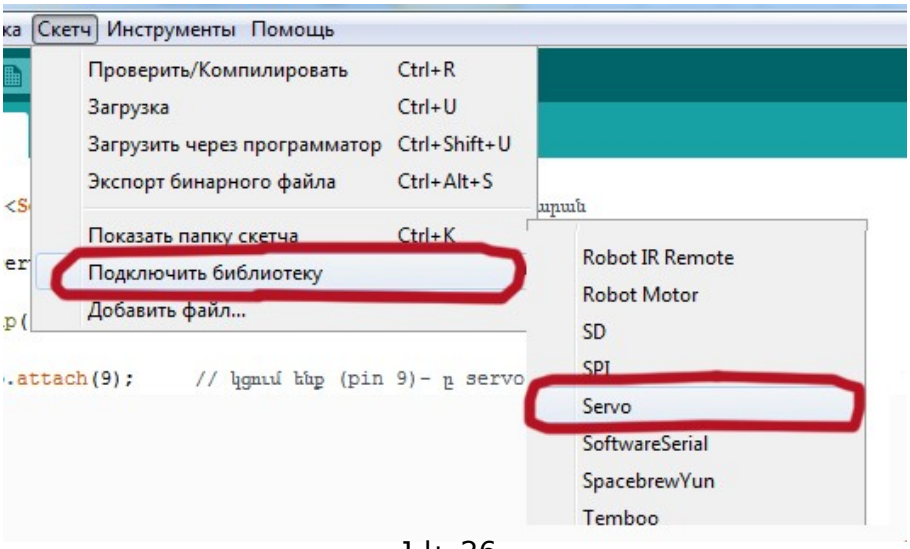
Այս նախագծում կփորձենք **Սերվո** շարժիչի ֆիքսված անկյունային դիրքերի փոփոխություններ կատարել: Այն նախատեսված է 0-ից 180 աստիճանի միջակայքում աշխատելու համար: Ունի 3 կոնտակտ, որոնցից սև կամ շագանակագույնը միանում է **GND** կոնտակտին, կարմիր գույնը՝ 5V կոնտակտին, իսկ դեղինը՝ **ԼԻՄ** տեսակի **pin**-երից որևէ մեկին: Այս օրինակում կմիացնենք **pin 9 –** ին: Ի տարբերություն նախորդ նախագծերի այստեղ **Arduino IDE** միջավայրում կօգտագործենք նաև գրադարան հասկացությունը (նկ. 25):

Ծրագրավորում.

```
#include <Servo.h>          // ավելացնում ենք servo.h գրադարան

Servo myservo;              // ստեղծում ենք myservo անունով servo տեսակի օբյեկտ
                            // որի միջոցով կկառավարենք վերջինիս անկյունային դիրքերը
void setup()
{
  myservo.attach(9);        // կցում ենք (pin 9)- ը servo շարժիչին՝ օգտագործելով myservo օբյեկտը
}

void loop()
{
   myservo.write(0);        // servo շարժիչի անկյունային դիրքը համապատասխանեցնում ենք 0 աստիճանի
    delay(2000);            // սպասել 2 վայրկյան
   myservo.write(180);      // servo շարժիչի անկյունային դիրքը համապատասխանեցնում ենք 180 աստիճանի
    delay(2000);            // սպասել 2 վայրկյան
}
```

Նկ. 25

#include <Servo.h> հրամանը կարելի է ինչպես գրել, այնպես էլ ավելացնել գրադարան բաժնից (նկ. 26):



Նկ. 26

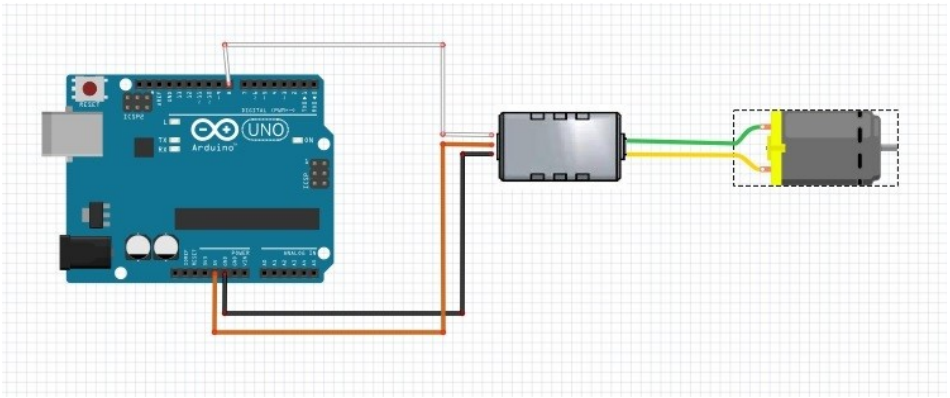
Գրադարանի ներմուծումը մեզ հնարավորություն կտա օգտվել իր մեջ պարունակվող օբյեկտներից: Նկարագրությունների հատվածում **Servo myservo** հրամանի միջոցով ստեղծում ենք **Servo** տիպի **myservo** օբյեկտը (համեմատելով նախորդ օրինակներին՝ մենք սահմանում էինք օրինակ **int** տիպի **M1** փոփոխականը): **Սերվո** շարժիչի հետ աշխատելիս նրան միացող համապատասխան **pin**-ի սահմանումը **void setup** բաժնում ստանում է այլ տեսք՝ **myservo.attach(9)** : Այս դեպքում ազդանշան փոխանցելու համար կօգտագործենք **9**-րդ **pin**-ը: Ազդանշանի փոխանցման համար **analogWrite** հրամանի փոխարեն օգտագործում ենք **myservo.write()** հրամանը, որի փակագծերում գրված պարամետրը ցույց է տալիս, թե ինչ անկյունային դիրք պետք է ստանա **Սերվո** շարժիչը (մեր օրինակում **0-180** միջակայքի): Հետևյալ ծրագրի արդյունքում այն անընդհատ կտեղակայվի **0** դիրքի վրա 2 վայրկյան, այնուհետև կտեղակայվի **180** դիրքի վրա նորից 2 վայրկյան:

Առաջադրանք. Օգտագործելով լուսադիոդի միացման օրինակում դիտարկած **for** ցիկլը գրել ծրագիր, որի արդյունքում **Սերվո** շարժիչը պետք `տեղակայվի **0** դիրքում, այնուհետև պետք է տեղափոխվի դեպի **90** դիրք՝ ամեն 1 աստիճան անկյունները փոխելիս 10 միլիվայրկյան ընդմիջումներով, սպասի 2 վայրկյան, այնուհետև նույն տրամաբանությամբ տեղափոխվի դեպի **180** դիրք և հակառակը: Կատարել նաև այլ տարբեր փորձարկումներ:

6. Նախագիծ 5 - Շարժիչի միացում VEX մոդուլով

Այս նախագծում մեզ անհրաժեշտ է DC շարժիչ, շարժիչի աշխատանքի կարգավորման մոդուլ (**VEX**) և Arduino Uno միկրոկոնտրոլլեր:

Միացման սխեմա.



Նկ. 27

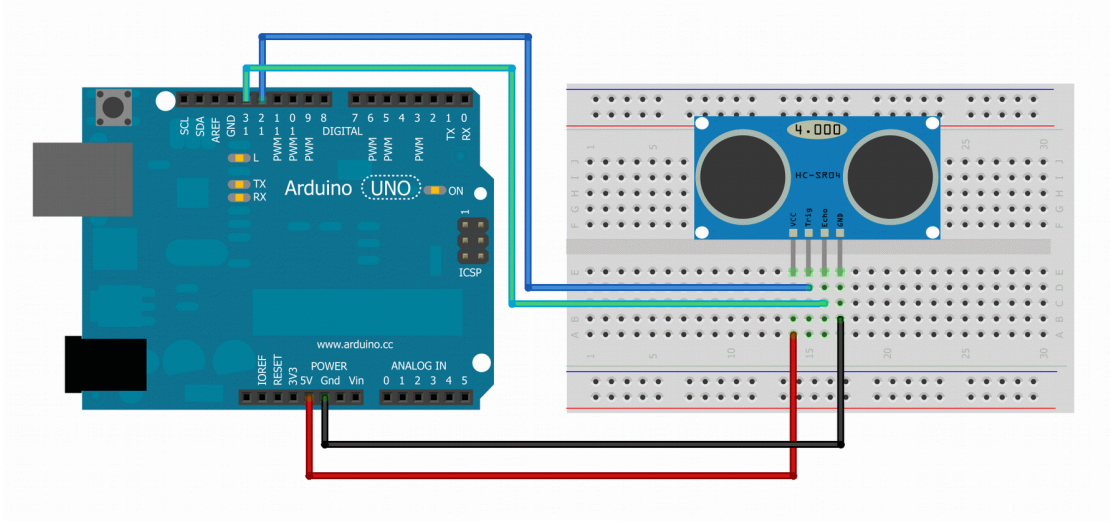
Այս մոդուլով աշխատելիս մենք կօգտագործենք Սերվո շարժիչի աշխատանքի ծրագրավորման հրամանները: Միացումը կլինի հետևյալ սկզբունքով՝ **Vex controller -** ի մի կողմի երկու կոնտակտները կմիանան շարժիչի կոնտակտներին, իսկ 3 կոնտակտներից սպիտակը կմիանա **pin**-ին, նարնջագույնը՝ 5Վ-ին, իսկ սևը՝ GND-ին: Ինչպես նախորդ օրինակում, այստեղ նույնպես կօգտագործենք **#include <Servo.h>** գրադարանը, կստեղծենք **Servo** տեսակի օբյեկտ, իսկ շարժիչի աշխատանքի հրամանով փոխանցվող թվային արժեքները կաշխատեն հետևյալ սկզբունքով՝ 90 թվային արժեքը դադարի վիճակը կլինի, 90-ից փոքր թվային արժեքների դեպքում մի ուղղությամբ կպտտվի շարժիչը, իսկ 90-ից մեծ թվային արժեքների դեպքում՝ հակառակ ուղղությամբ: Նախորդ նախագծի ծրագրային կոդը կարող ենք օգտագործել նաև այս նախագծի համար:

Առաջադրանք. Տարբեր թվային արժեքներ տեղադրելով գտնել 90-ից փոքր արժեքների նվազագույնը և 90-ից մեծ արժեքների առավելագույնը՝ այսինքն սահմանային արժեքները: Օգտագործելով պոտենցոմետրի միացման օրինակը՝ պոտենցոմետրի միջանկյալ դիրքը համապատասխանեցնել շարժիչի 90 թվային արժեքի հետ, և դեպի աջ ու ձախ պտտելու դեպքում համապատասխանաբար շարժիչը աջ ու ձախ ուղղությամբ սկսի պտտվել՝ ստանալով պոտենցոմետրի ընթացիկ արժեքին համապատասխան ընթացիկ արագության արժեք շարժիչի համար:

7. Նախագիծ 6 - Հեռաչափի (Ultrasonic) միացում

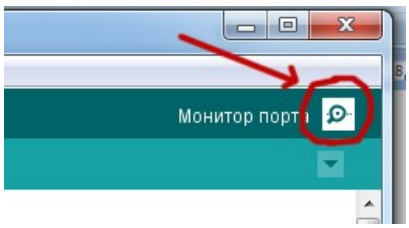
Այս նախագծում մեզ անհրաժեշտ է Հեռաչափ (Ultrasonic), Breadboard և Arduino Uno միկրոկոնտրոլլեր:

Միացման սխեմա.



Նկ. 28

Հեռաչափի միացման համար դիտարկենք նրա կոնտակտները, որոնք 4-ն են՝ **VCC** (միանում է **5V** կոնտակտին), **Trig** (կմիացնենք **pin 12** կոնտակտին), **Echo** (կմիացնենք **pin 13** կոնտակտին) և **GND** (համապատասխանաբար միանում է **GND**-ին): Հեռաչափի հետ աշխատելու համար մենք նորից կօգտվենք գրադարանից: Մեր դիտարկած նախորդ բոլոր օրինակներում ազդանշանը փոխանցվում էր **Arduino** տպասալից դեպի համապատասխան սարքավորումներ: Այս նախագիծը կտարբերվի նաև այդ տեսանկյունից: Քանի որ **հեռաչափը** հանդիսանում է որպես զգայարան, հետևաբար մենք պետք է ինֆորմացիա ստանանք նրանից: Այստեղ նորից կօգտվենք **serial** հասկացությունից, որը թույլ կտա կապ հաստատել համակարգչի **port**-ի հետ և չափված արդյունքը առանց որևէ այլ սարքավորում միացնելու կկարողանանք տեսնել **port**-ի էկրանին, որը կբացվի **Arduino IDE** միջավայրի վերևի աջ անկյունում գտնվող նշանը սեղմելիս (նկ. 28):



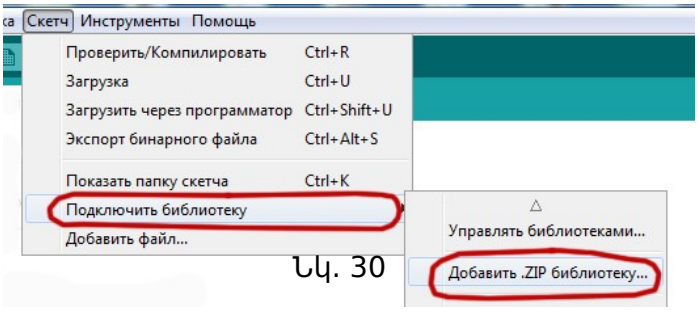
Նկ. 28

Ծրագրավորում.

```
#include "Ultrasonic.h"          // Ultrasonic.h գրադարանի ներմուծում

Ultrasonic ultrasonic(12, 13);   // Ultrasonic տիպի ultrasonic անունով օբյեկտի ստեղծում
                                 // ունի 2 պարամետր՝ միացող pin-երի համարները (Trig - 12, Echo - 13)

void setup()
{

  Serial.begin(9600);            // կապ ենք հաստատում համակարգչի հետ
                                 // 9600 թիվը տվյալների փոխանցման արագությունն է (բիթ/վ)
}

void loop()
{
  float dist_cm = ultrasonic.Ranging(CM);    // Հեռավորության արժեքի ստացում

  Serial.println(dist_cm);                   // Հեռավորության արժեքի արտածում port-ի էկրանին

}
```

Նկ. 29

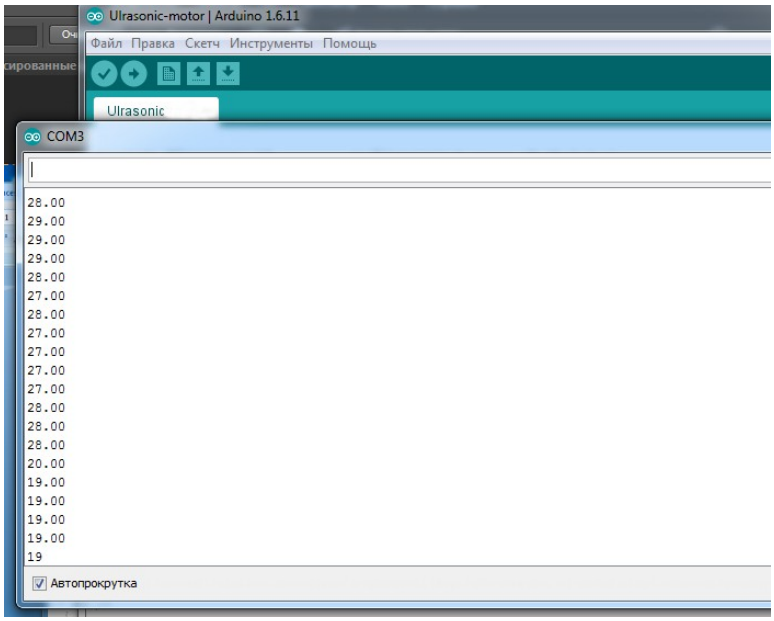
#include "Ultrasonic.h" հրամանով ներմուծում ենք **Ultrasonic.h** գրադարանը, որը **Arduino IDE** գրադարանների ցանկում բացակայում է: Այն կարելի է ներբեռնել և տեղադրել անցնելով հետևյալ հղումով http://techoblog.ucoz.net/publ/arduino_s_nulja_ultrazvukovoj_modul_hc_sr04/1-1-0-2 , որտեղ կգտնեք **ultrasonic-HC-SR04.zip** նիշքը: Ներբեռնելով այն ավելացրեք գրադարան բաժնում (նկ. 30):



Նկ. 30

Ինչպես նախորդ օրինակում **Servo** տիպի **myservo** օբյեկտ ստեղծեցինք, այստեղ նույն կերպ կստեղծենք **Ultrasonic** տիպի **ultrasonic** անվանումով օբյեկտ (անվանումը կարող եք ինքներդ սահմանել), որն ունի 2 պարամետր՝ նախորոք սահմանված **pin**-եր, որոնց էլ միանում են **Trig** և **Echo** կոնտակտները: Մեր օրինակում դրանք **12** և **13** համարի **pin**-երն են: Ինչպես նշվեց **Serial** հրամանի միջոցով կարող ենք կապ հաստատել համակարգչի հետ: Այն օգտագործվում է նաև **bluetooth** մոդուլ և այլ սարքավորումների հետ կապ հաստատելիս: Ինչպես արդեն գիտենք **Serial.begin(9600)** հրամանով սահմանում ենք համակարգչի հետ կապ հաստատելիս ինֆորմացիայի փոխանցման արագությունը, որը հենց փակագծերում գրված

թիվն է՝ **9600 բիթ/վ**: **void loop()** ենթածրագրում ունենք 2 գործողություն: Ստեղծում ենք **float (իրական)** տիպի **dist_cm** անունով փոփոխական, որին վերագրում ենք **հեռաչափից** դեպի արգելք եղած հեռավորության արժեքը: **ultrasonic.Ranging(CM)** հրամանը վերադարձնում է հենց այդ արժեքը: **Serial.println(dist_cm)** հրամանի միջոցով **port**-ի էկրանին անընդհատ կսկսվի արտածվել **dist_cm** փոփոխականի արժեքը, որը մեզ մոտ հենց **հեռաչափից** մինչև արգելք եղած տարածությունն է՝ արտահայտված սանտիմետրերով: Ծրագիրը ներբեռնելով և **port**-ի էկրանը բացելով կտեսնենք նմանատիպ արդյունք (նկ. 31):



Նկ. 31

Հեռաչափի օգնությամբ կարելի է շատ հետաքրքիր նախագծեր իրականացնել, որոնց թվին են պատկանում Ռադարի, փախչող և լաբիրինթ հաղթահարող ռոբոտների ստեղծումը:

Համադրելով նախորդ նախագծերից որևե մեկը, փորձենք **հեռաչափի** ցուցմունքի արժեքից կախված տարբեր ազդանշաններ ուղարկել օրինակ լուսադիոդին: Միացնելով լուսադիոդը արդեն իսկ փորձված տարբերակով (ենթադրենք **pin 3**-ին), ստեղծենք ծրագիր, որը **հեռաչափից** ստացված ազդանշանի հիման վրա կմիացնի և կանջատի այն: Այստեղ կօգտագործենք **if - else** պայմանը: Գրելաձևը հետևյալն է՝ **if (պայման) { գործողություն1 },** այսինքն եթե փակագծերում գրված պայմանը ճշմարիտ է և վերադարձնում է **true** արժեքը, ապա կատարիր ձևավոր փակագծերում եղած **գործողություն1**-ը: Երբ ունենք նաև հակառակ պայմանը օգտագործում ենք **else {գործողություն2}** հրամանը, որը նշանակում է եթե նախորդ պայմանը ճշմարիտ չէ և վերադարձնում է **false** արժեքը, ապա կատարիր **գործողություն2-**ը: Մեր օրինակը կստանա հետևյալ տեսքը, որտեղ համապատասխան գործողությունները լուսադիոդի միացումը և անջատումն են (նկ. 32)՝

Ծրագրավորում.

```
#include "Ultrasonic.h"          // Ultrasonic.h գրադարանի ներմուծում

Ultrasonic ultrasonic(12, 13);   // Ultrasonic տիպի ultrasonic անունով օբյեկտի ստեղծում
                                 // ունի 2 պարամետր՝ միացող pin-երի համարները (Trig - 12, Echo - 13)
int LED=3;

void setup()
{

  Serial.begin(9600);            // կապ ենք հաստատում համակարգչի հետ
                                 // 9600 թիվը տվյալների փոխանցման արագությունն է (բիթ/վ)
}

void loop()
{
  float dist_cm = ultrasonic.Ranging(CM);    // Հեռավորության արժեքի ստացում

  Serial.println(dist_cm);                   // Հեռավորության արժեքի արտածում port-ի էկրանին


 if (dist_cm<20)                             // Եթե dist_cm<20 վերադարձնում է true արժեք
  {
    analogWrite(LED, 255);                   // Կատարվում է լուսադիոդի միացում
}
  else                                       // հակառակ դեպքում վերադարձնում է false արժեք
  {
    analogWrite(LED, 0);                     // Կատարվում է լուսադիոդի անջատում
    delay(100);
  }
}
```

Նկ. 32

Այսպիսով 20սմ հեռավորությունից ավելի մոտ արգելք տեղադրելով **հեռաչափի** դիմաց լուսադիոդի լույսը կվառվի, հակառակ դեպքում կանջատվի:

Առաջադրանք. Նմանատիպ ծրագիր գրել նաև DC շարժիչի և Սերվո շարժիչի համար: Սահմանված հեռավորությունից մոտ արգելքի առկայության պարագայում DC շարժիչը սկսի պտտվել, հակառակ դեպքում դադարի պտտվել: Սերվո շարժիչի պարագայում կարելի է միջակայքերի բաժանում կատարել, համապատասխան հեռավորության միջակայքում Սերվո շարժիչը համապատասխան անկյան դիրք ընդունի:

8. Նախագծերի համադրում

Ծանոթանալով 4 նախագծերի միացման սխեմաներին և ծրագրավորման ընթացքին կարող եք հեշտությամբ միավորել դրանց և ստանալ բազմաթիվ հետաքրքիր նախագծեր: Օրինակ կարելի է ստեղծել ռոբոտի մոդել, որը կկարողանա շարժվել DC շարժիչների միջոցով, որի վրա տեղադրված կլինի Սերվո շարժիչը, իսկ վերջինիս պտտման գլխիկի վրա՝ հեռաչափը: Կարելի է ծրագրավորել, որ Սերվո շարժիչն անընդհատ համաչափ փոխի իր դիրքը 0- ից դեպի 180 և հակառակը: Տեղադրել պայման, որ եթե հեռաչափը ֆիքսի արգելք որոշակի սահմանված հեռավորությունից փոքր արժեքի դեպքում DC շարժիչները սկսեն նախապես ծրագրավորված ուղղություններով պտտվել և տեղաշարժել ռոբոտին: Այս նույն տրամաբանությամբ է ստեղծված <<փախչող>> ռոբոտի մոդելը: Կարելի է Սերվո շարժիչն ամրացնել դռան կամ որևէ փականի վրա այնպես, որ նրա 0 և 180 դիրքերը բացեն և փակեն այն, այնուհետև հեռաչափի ուղարկած ազդանշանի արժեքին համապատասխան փականը բացվի կամ փակվի: Նախագծեր կարելի է շատ թվարկել: Փորձեք ստեղծել նորը: Ցանկացած սահմանափակում գոյություն ունի միայն մեր գլխում: Ազատեք երևակայությունը և մուտք գործեք **Arduino** կոչված հետաքրքրաշարժ իրականություն:

Սիրելի ընթերցող, թող այս ձեռնարկը հանդիսանա հիմք՝Ձեր հետագա խորը գիտելիքների ձևավորման և օգտագործման ճանապարհին: