С.А. Лєхан

для школярів

Програмування

2018

Нова українська школа потребує інноваційних підходів до навчання. Arduino – це портативна платформа з відкритим вихідним кодом, на базі якої легко пристосовуються апаратні засоби і безкоштовне програмне забезпечення для побудови простих систем автоматики та робототехніки. Arduino дозволяє навіть новачкам робити дійсно дивовижні речі.

За допомогою цього посібника Ви можете підключати до Arduino різні типи датчиків, джерела світла, електродвигуни і багато інших пристроїв, а також використовувати програмне забезпечення для зручного керування вашими приладами, а також обробки різної інформації з датчиків. Ви самостійно зможете створити інтерактивний дисплей або рухомий робот, моделі, які керуватимуться по радіо або створити систему керування «Розумний дім».

Посібник буде корисний учителям інформатики, студентам технічних навчальних закладів та учням, що цікавляться практичною інформатикою.

Автор:

Лєхан Сергій Антонович – учитель інформатики та технологій (категорія вища, старший учитель) Білгород-Дністровської загальноосвітньої школи І-ІІІ ступенів № 3

Методичний посібник: «**ARDUINO** для школярів. Програмування»

Рецензенти:

1. Веріс Д.В. — завідувач ММК Управління освіти, сім'ї, молоді та спорту Білгород-Дністровської міської ради
2. Бойченко О.П. – заступник директора з НВР Білгород-Дністровської загальноосвітньої школи І-ІІІ ступенів № 3
3. Пилипчук О.П. – вчитель інформатики і фізики загальноосвітньої школи І-ІІІ ступенів, с. Гаврилівка, Хмельницької області (автор підручників 5-11 клас)
4. Шестопалов Є.А. – вчитель інформатики загальноосвітньої школи І-ІІІ ступенів №8, м. Шепетівка, Хмельницької області (автор підручників 5-11 клас)

Схвалено методичною радою міського методичного кабінету УПРАВЛІННЯ ОСВІТИ, СІМ'Ї, МОЛОДІ ТА СПОРТУ Білгород-Дністровської міської ради (Протокол № __ від _____2018 року)

Управління освіти, сім'ї, молоді та спорту Білгород-Дністровської міської ради

Білгород-Дністровської загальноосвітньої школи І-ІІІ ступенів № 3

С.А. Лєхан

ARDUINO

для школярів.

Програмування

(методичний посібник)

м. Білгород-Дністровський, 2018 рік

Зміст

Філософія Arduino полягає в тому, що, якщо ви захочете навчитися електроніці, ви зможете вивчати її вже з першого дня, замість того, щоб спочатку вивчати алгебру.

Девід Куартилльз

ВСТУП

Нова українська школа потребує інноваційних підходів до навчання. На думку експертів, майбутні професії не обійдуться без навичок програмування. Такі вимоги висуває розвиток нашого сучасного суспільства, а, значить, такі вимоги вже сьогодні треба враховувати під час навчанні школярів.

Arduino – це портативна платформа з відкритим вихідним кодом, на базі якої легко пристосовуються апаратні засоби і безкоштовне програмне забезпечення для побудови простих систем автоматики та робототехніки. Платформа Ардуіно повністю відкрита для розробників і доступні ціни на плати, модулі та датчики зробили її найпопулярнішою платформою для радіоаматорів в цілому світі.

Arduino дозволяє навіть новачкам робити дійсно дивовижні речі. Ви можете підключати до Arduino різні типи датчиків, джерела світла, електродвигуни і багато інших пристроїв, а також використовувати програмне забезпечення для зручного керування вашими приладами, а також обробки різної інформації з датчиків. Ви можете створити інтерактивний дисплей або рухомий робот, моделі, які керуватимуться по радіо або розробити систему керування «Розумний дім».

А найголовніше – дитина, зробивши проект, бачить результат! Вже не виникне питання: «А навіщо це мені потрібно?»,- як це досить часто відбувається на шкільних уроках.

Цей посібник описує стандарт для мови C++, та надає деякі приклади використання Arduino.

Корисна інформація

Структурування під час написання коду Arduino

Для автоматичного структурування[1] коду використовуйте комбінацію CTRL+T на клавіатурі, або в головному меню:

Інструменти/Автоформатування.

Структурований код значно полегшує пошук та виправлення помилок, які обов'язково будуть супроводжувати вас під час програмування Arduino!

Також рекомендується згортати довгі функції та інші фрагменти коду для зручності редагування. Увімкнути цей режим можна так: *Файл/Налаштування,* у вікні, що відкриється поставити прапорці: *Увімкнути згортання тексту.* Також тут можна увімкнути нумерацію рядків: *Показувати нумерацію рядків*.

Сполучення клавіш для зручного набору коду:

Ctrl+← , Ctrl+→ – перемістити курсор ліворуч/праворуч НА ОДНЕ СЛОВО.

Home , End – перемістити курсор на початок/кінець рядка.

Shift+← , Shift+→ – виділити символ ліворуч/праворуч від курсора.

Shift+Ctrl+← , Shift+Ctrl+→ – виділити слово ліворуч/праворуч від курсора.

Shift+Home , Shift+End – виділити всі символи від місця встановлення курсора до початку/кінця рядка.

Ctrl+Z – відмінити останню дію.

Ctrl+Y – повторити дію, щ обула відмінена.

Ctrl+C – копіювати виділений текст.

Ctrl+X – вирізати виділений текст.

Ctrl+V – вставити текст з буфера обміну.

Сполучення клавіш Arduino:

Ctrl+U – вивантажити прошивку в Arduino

Ctrl+R – зкомпілювати (перевірити)

Ctrl+Shift+M – відкрити монітор послідовного порту

[1] Красиве виведення коду, з використанням відступів

Урок 1. Дані та змінні

Структура програми

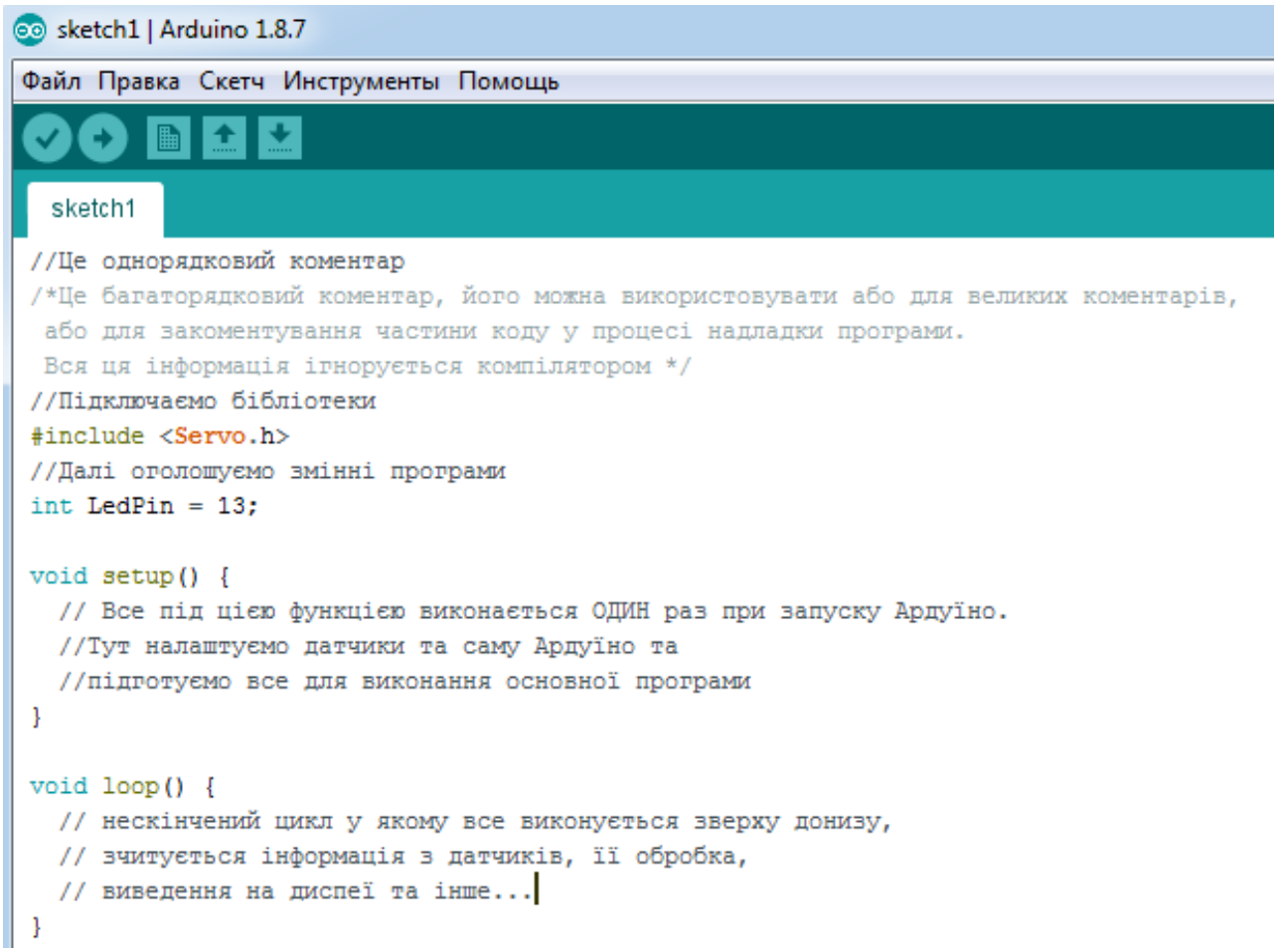
Роздивимося з яких складових складається програмний код (спрощена версія C++) Arduino.

Коментарі у програмі не сприймаються компілятором і потрібні лише для пояснення роботи оператора чи частини програми. Читаючи коментарі легше розібратися з кодом.

// Однорядковий коментар

/* Багаторядковий коментар (Коментарі не займають пам'яті) */

Директиви препроцесора - вказівки компілятора, які виконуються на початку компіляції програми і їх відповідно записують на початку тексту програми — у мові C++ починаються із символу #.



Деякі прилади, що приєднуються до Arduino потребують використання окремих файлів (бібліотек), вже створених програмістами для успішної

роботи цього приладу. Для підключення бібліотек використовують директиву **#include:**

#include – підключити файл (бібліотеку), наприклад:

#include <UIPEthernet.h>

Будь-яка програма Arduino складається з функцій і обов’язково має у складі функцію **setup().**

void setup() {} – все всередині дужок {} буде зроблено один раз, коли ви завантажуєте Arduino.

Тут **void** означає, що функція нічого не повертає зовні (за межі функції).

Також програма Arduino має циклічну функцію **loop():**

void loop() {} – все всередині дужок {} нескінченно повторюється зверху вниз.

Після кожної «дії» ставиться крапка з комою;

Типи даних

Для раціонального використання пам'яті Arduino треба обов'язково оголошувати типи даних, що будуть використані у вашому коді. Таким чином, компілятор буде визначати скільки місця виділяти у пам'яті під ті чи інші дані.

< Тип даних > < Ім'я>; так можна оголосити змінну **LedPin**:

Int LedPin;

<тип даних> <ім’я> = <значення>; оголосити змінну **LedPin** і привласнити їй значення 13

Int LedPin = 13;

Знак **«=»** означає «присвоїти» – записати у комірку пам’яті з зазначеним іменем вказане після нього число, символ або текст (Не плутати зі знаком дорівнює!).

Нижче приведена таблиця типів даних:

Тип	Роз-мір, байт	Значення	Особливість
boolean	1	0 або 1	Логічна змінна **true** або **false**
char	1	-128… 127	Зберігає номер символа з таблиці символів ASCII
byte	1	0… 255	
int	2	-32 768… 32 767	
unsigned int	2	0… 65 535	
word	2	0… 65 535	Те ж саме, що і **unsigned int**
long	4	-2 147 483 648… 2 147 483 647	
unsigned long	4	0… 4 294 967 295	
float	4	-3.4028235e-38… 3.4028235e+38	Зберігає числа з плаваючою комою з точністю 6-7 знаків
double	8	-3.4028235e-38… 3.4028235e+38	Те ж саме, що і **float**

Особливості використання змінних

- Уважно стежте за значенням, що змінна приймає. Якщо значення стане більше максимального або менше мінімального (тобто вийде з діапазону) для цього типу даних, змінну буде скинуто до 0 або на випадкове число. Цю помилку буде важко простежити пізніше.

- Тип даних вказується при оголошенні змінної тільки один раз. В подальшому змінна використовується виключно за іменем. Якщо

спробувати змінити тип змінної, з'являється повідомлення про помилку. Але це відбудеться тільки, якщо змінна глобальна[2], або коли локальну змінну знову оголосили в межах функції, в якій вона вже була оголошена.

Особливості використання чисел з плаваючою комою float

- Присвоювати лише значення з комою, навіть якщо число ціле: **(Name = 25.0)**.
- Операцію ділення також проводити тільки на цифри з комою: **(Name/5.0).**
- Під час ділення числа цілого типу з ціллю отримати число з плаваючою комою перед обчисленням потрібно написати **(float)**.

float Name = (float)celoe_chislo / 3.789;

Операції з числами типу **float** займають набагато більше часу, ніж з цілими. Для підвищення швидкості обчислень у такому випадку використовують різні способи, щоб обійти цю ситуацію.

Приклад використання змінних

```
byte sound_sensor_pin = 5; //звуковий сенсор підключено до 5-го піну
void setup() {
  pinMode(sound_sensor_pin,INPUT);
}
void loop()  {
  digitalRead(sound_sensor_pin);
}
```

Таким чином, змінна – це комірка пам’яті, яка зберігає значення відповідного типу. Наприклад, вище оголошено змінну **sound_sensor_pin** і присвоєно їй значення 5-го піну Arduino. Якщо треба буде змінити значення піну, наприклад, на 7-й, достатньо тільки змінити значення змінної на 7 на початку програми. Всі інші значення у програмі, де зустрічається змінна автоматично приймуть задане значення. Таким чином полегшується процес написання коду.

[2] Дивись наступний урок

Урок 2. Операції зі змінними та константами

Типи змінних

Глобальна змінна оголошується на початку скетча поза межами всіх функцій. Глобальна змінна буде видима у будь-якому місці скетча. Приклад глобальної змінної **sound_sensor_pin**:

```
byte sound_sensor_pin = 7; //займає 1 байт пам'яті
void setup() {
  pinMode(sound_sensor_pin,INPUT);
}
```

Локальна змінна оголошується у функції, і доступ до неї можна отримати лише в межах цієї функції. Приклад локальної змінної **sound_sensor_pin**:

```
void setup() {
  byte sound_sensor_pin = 6;
  pinMode(sound_sensor_pin,INPUT);
}
```

Локальних змінних може бути кілька з однаковими іменами, але з різними значеннями. Це пов'язано з тим, що локальна змінна вивантажується з пам'яті мікроконтролера при виході із функції. Тобто дві змінні з однаковими іменами та різними значеннями не існують одночасно.

Константи

const<тип><ім'я> = <значення> – оголосити константу.

Наприклад оголошуємо константу **LedPin** та надаємо їй значення 13:

const int LedPin = 13;

```
const int LedPin = 13;
const int pins_in_count=6;      //Входи для підключення перемикачів
void setup() {
}
void loop()  {
```

Можна в окремих випадках оголошувати константу та надавати їй значення за допомогою директиви **#define:**

#define <ім'я> <значення>

Наприклад, такий рядок на початку скетчу оголосить змінну **sound_sensor_pin** та присвоїть їй значення 7:

#define sound_sensor_pin 7 //Тут крапка з комою не ставиться!

```
#define sound_sensor_pin 7 //не займає пам'ять
void setup() {
  pinMode(7,INPUT);
}
void loop()  {
  digitalRead(7);
}
```

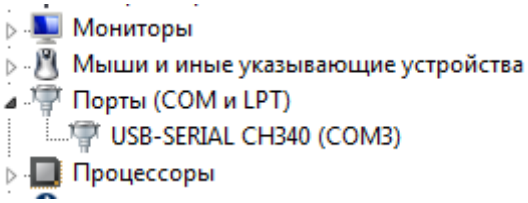
Якщо ви спробуєте змінити значення постійної після її оголошення, ви отримаєте помилку!

Математичні оператори

+, -, *, /	додавання, віднімання, множення та ділення.	
pow(x, y);	x^y, функція pow може навіть зводити в дробовий ступінь: **pow(2, 3.5);**	
sq (x);	Означає: x^2	
sqrt (x);	$\sqrt{x}$	
abs(x);	модуль числа x: | x |	
sin (x), cos (x), tan (x);	тригонометричні функції синус, косинус, тангенс	
round(x);	математичне округлення (якщо значення після коми більше або дорівнює 5, округлює у більший бік)	
ceil (x);	округлює у більший бік	
floor (x);	округлює в менший бік	
x + = y;	додасть "y" до "x", або:	x=x+y
x – = y;	відніме "y" з "x", або:	x=x-y
x * = y;	множення "x" на "y", або:	x=x*y
x /= y;	розділить "x" на "y", або:	x=x/y
x + +;	збільшить "x" на 1, або:	x=x+1
x – -;	зменшить "x" на 1, або:	x=x-1

Урок 3. Послідовний порт. Оператори бібліотеки Serial

Підключивши Arduino до комп'ютера, необхідно дізнатися, до якого порту вона приєдналася. Це можна зробити, відкривши «Диспетчер приладів» Windows:



У нашому випадку – це COM3. Тепер необхідно встановити прапорець у меню Arduino: «Інструменти» – «Порт» – вибрати COM3.

Тепер роздивимося, як використовується «Монітор порту». Це меню знаходиться також в «Інструменти». Натискаємо:

Serial – об'єкт бібліотеки Serial для роботи з послідовним портом (COM-портом).

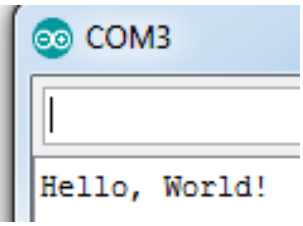
Serial.begin (<швидкість>); відкрити порт.

Наприклад, щоб відкрити порт на 9600 БОД (значення по замовчуванню, як на малюнку):

Serial.begin(9600); Увага! Швидкість, встановлена в **begin()** повинна дорівнювати швидкості монітору порту (правий нижній кут на самому моніторі – див. малюнок вище).

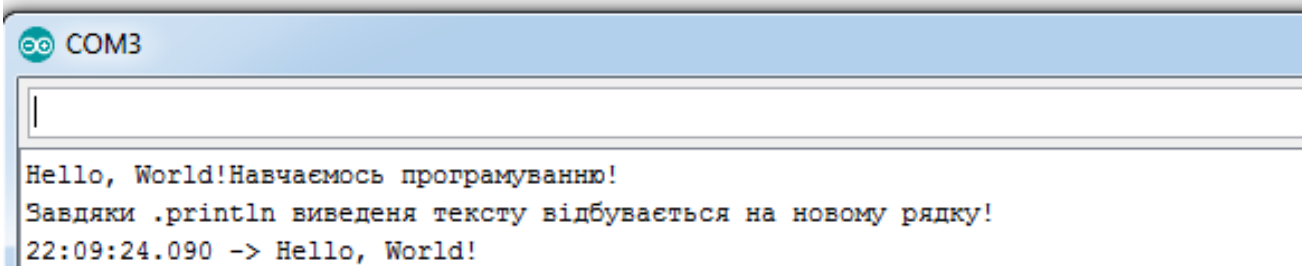
Serial.print (); – виведення в порт. Змінні та цифри пишемо безпосередньо, як є, текстові повідомлення беремо в лапки " ".

```
void setup() {
  Serial.begin(9600);
  Serial.print("Hello, World!");

}
```



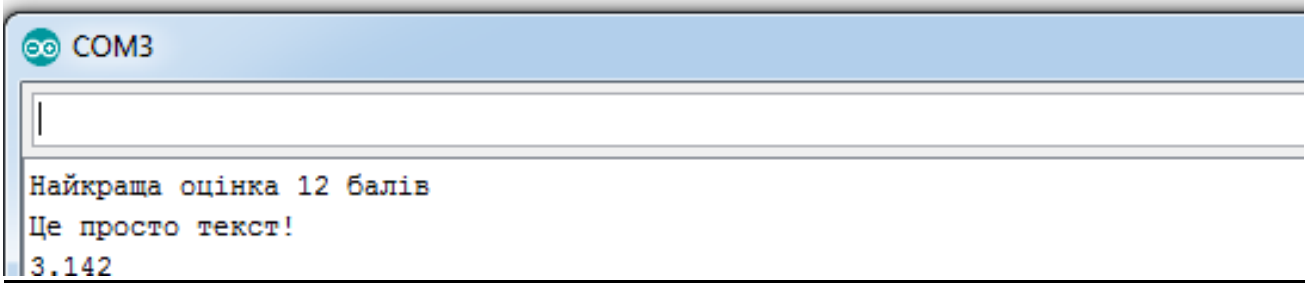
Serial.println (); – виведення з переходом на новий рядок.

```
void setup() {
  Serial.begin(9600);
  Serial.print("Hello, World!");
  Serial.println("Навчаємось програмуванню!");
  Serial.println("Завдяки .println виведеня тексту відбувається на новому рядку!");
  }
```



Serial.println (Name, n); – виведення змінної **Name** (типу **float**), де **n** – кількість десяткових чисел після коми. Знімемо прапорець «Показати відмітки часу» у моніторі порту та запустимо такий код:

```
void setup() {
  Serial.begin(9600);
  Serial.print("Найкраща оцінка ");Serial.print(12);Serial.println(" балів");
  String var = "Це просто текст!";
  Serial.println(var);
  float pi = 3.141592653589793238;
  Serial.println(pi, 3);
  }
```



Serial.println (Name, <система>); – виведення із зазначенням системи обчислення:

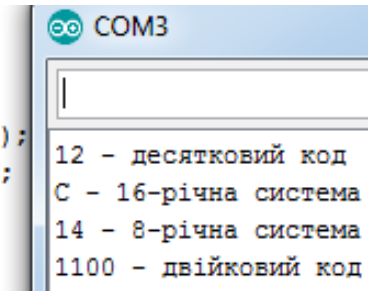
DEC – десяткова система

OCT – 8 – річна система

NEX – 16 -річна система

BIN – двійкова система

```
void setup() {
  Serial.begin(9600);
  Serial.print(12,DEC);Serial.println(" - десятковий код");
  Serial.print(12,HEX);Serial.println(" - 16-річна система");
  Serial.print(12,OCT);Serial.println(" - 8-річна система");
  Serial.print(12,BIN);Serial.println(" - двійковий код");
  }
```



З виведенням все зрозуміло, а ось з прийняттям даних все складніше. Коли ми відправляємо на Ардуіно дані, вони складаються в буфер, і чекають поки Arduino їх прочитає. Обсяг буфера складає 64 байти. Щоб тисячі разів в секунду не читати порожній буфер, є функція перевірки буфера, **Serial.available().** Вона повертає число байт, які лежать в буфері. Тобто ми можемо використовувати умову “якщо в буфері більше 0 байт”, і написати код, який буде виконуватися поки в буфері щось є. Тепер можна спробувати прийняти дані. Створимо змінну типу **integer**, і надамо їй дані з порту за допомогою функції читання **Serial.read().** І відразу відішлемо назад те, що отримали за допомогою вже відомої нам функції **Serial.println().** Тобто Arduino повинна відправити нам те, що ми відправили їй.

Перевіряємо. Відправляю 5 і отримую 53. Відправляю 35 і отримую цілих 2 числа, 51 і 53.

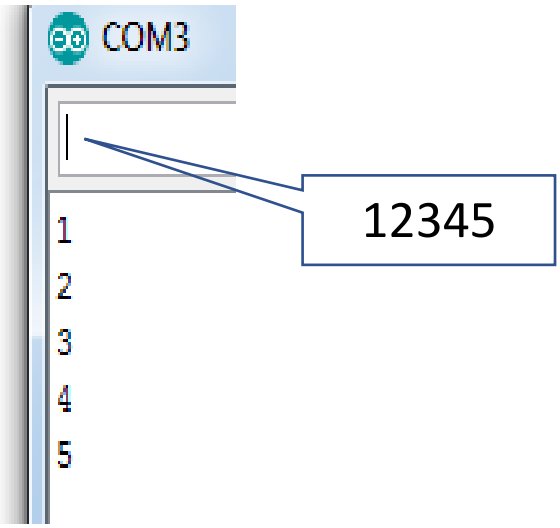
```
void setup() {
  Serial.begin(9600);
  }
void loop() {
  if (Serial.available() >0) {
    int in_data = Serial.read();
    Serial.println(in_data);
  }
}
```

Такий результат нас не задовольняє. В чому ж справа? Функція **Serial.read()** приймає не чисельні значення, а символьні. А у зв’язку з тим, що тип даних у нас **int**, ми отримуємо код символу згідно зі стандартною таблицею символів ASCII (Дивись таблицю на сторінці 15). Відправимо букву А. Бачите – це код букви в таблиці, номер 65. Щоб прийняти символ, можна змінити тип даних на **char**, і тоді ми зможемо відправляти і приймати символи. Але це все – символи, а не чисельні значення. Тобто

виконувати з ними математичні операції не можна. А якщо ми хочемо приймати саме цифри?

Тоді потрібно відразу переводити дані з порту в чисельне значення. Для цього потрібно відняти символ 0, в одинарних лапках. Це потрібно просто запам'ятати.

```
void setup() {
  Serial.begin(9600);
  }
void loop() {
  if (Serial.available() >0) {
    int in_data = Serial.read()- '0';
    Serial.println(in_data);
  }
}
```



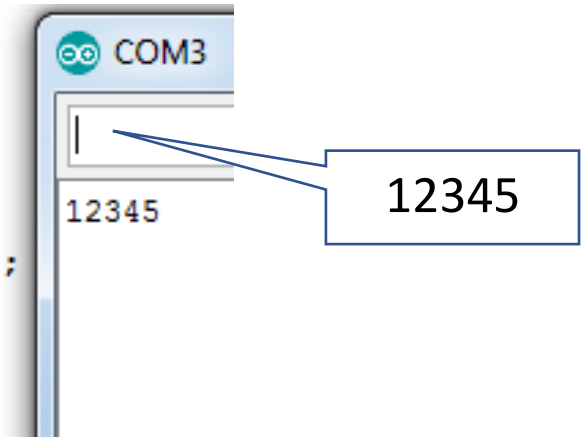
Тепер ми відправляємо і отримуємо саме цифри! Отримані таким чином цифри можна використовувати в математичних та логічних операціях. Але, на жаль, тільки однозначні. Число, що складається більше ніж з однієї цифри, розбивається на шматочки, і виходить, що наша змінна приймає значення тільки останньої цифри відправленого числа. Тобто відправили 12345, і змінна перезаписується 5 разів, і врешті-решт приймає значення 5. Для того, щоб прийняти все число цілком, є функція: **Serial.parseInt();**

Ця функція почекає, поки прийдуть всі відправлені вами цифри і складе з них число, ніяких нулів віднімати більше не потрібно. Тепер наша змінна прийме число повністю, і можна працювати з ним далі. У зв’язку з тим, що ця функція чекає нові цифри, ви можете помітити невелику затримку виконання. Ще є функція:

Serial.flush(); - вона очищає буфер порту.

Приклад використання функції **Serial.parseInt():**

```
void setup() {
  Serial.begin(9600);
  }
void loop() {
  if (Serial.available() >0) {
    int in_data = Serial.parseInt();
    Serial.println(in_data);
  }
}
```



Виділимо головне з описаного вище:

Дані з комп'ютера попадають в буфер з об'ємом 64 байти і чекають обробки.

Serial.available(); – перевірити буфер на наявність вхідних даних.

Serial.read(); – читати вхідні дані в форматі символів, згідно з ASCII.

Serial.read() - '0'; – читання даних у цілочисельному форматі по одній цифрі.

Serial.parseInt(); – читання даних у цілочисельному форматі цілком всього числа.

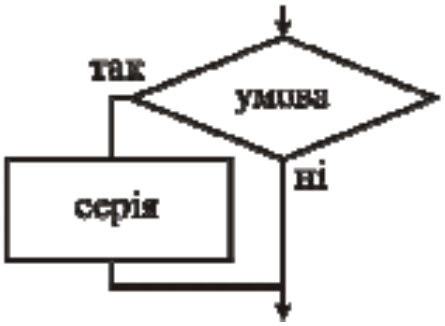
Serial.flush(); – очистити буфер порту.

Таблиця 2. ASCII кодування клавіатури

Escape 27		F1 112	F2 113	F3 114		F4 115	F5 116	F6 117	F7 118	F8 119	F9 120	F10 121	F11 122	F12 123	Print Screen	Scroll Lock 145	Pause 19			
` 192	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	0 48	- 189	=+ 187	Back Space 8	Insert 45	Home 36	Page Up 33	Num Lock 144	/ доп. 111	* доп. 106	+ доп. 107
Tab 9	Q 81	W 87	E 69	R 82	T 84	Y 89	U 85	I 73	O 79	P 80	[219	] 221		Delete 46	End 35	Page Down 34	7 доп. 103	8 доп. 104	9 доп. 105	
Caps Lock 20	A 65	S 83	D 68	F 70	G 71	H 72	J 74	K 75	L 76	; 186	' 222	Enter 13					4 доп. 100	5 доп. 101	6 доп. 102	Enter доп. 13
Shift 16	Z 90	X 88	C 67	V 86	B 66	N 78	M 77	,< 188	.> 190	/ 191	Shift 16	| 220			Up 38		1 доп. 97	2 доп. 98	3 доп. 99	
Ctrl 17	win	Alt 18	Space Bar 32						Alt 18	win	list	Ctrl 17		Left 37	Down 40	Right 39	Ins/0 доп. 45/96		Del/. доп. 46/110	

Урок 4. Умовний оператор

Оператор розгалуження (або умовний оператор) дозволяє здійснювати під час виконання програми перевірку умови, та на основі перевірки виконувати ту або іншу операцію. Загальний вигляд оператора розгалуження такий:



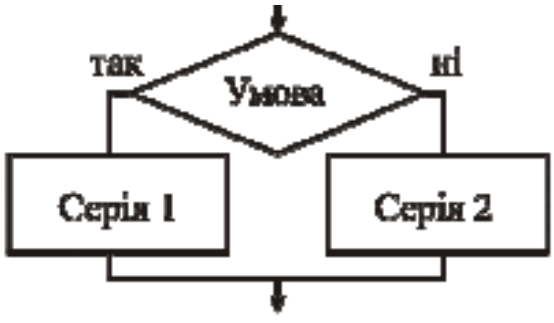
if (умова, яку слід перевірити) {код};

Оператор розгалуження if виконує перевірку, використовуючи операції порівняння. Якщо результат перевірки є істиною, то буде виконаний код, записаний після умови у фігурних дужках. Наведені блок-схема та запис умовного оператора є його скороченою формою.

Дуже часто виникає потреба при виконанні умови в операторі розгалуження if виконувати одну серію команд (код1), а при не виконанні – іншу (код2). В такому випадку використовують повну форму оператора розгалуження:

```
if (умова, яку слід перевірити)
   {код 1;}
else
   {код 2;}
```

Блок-схема повної форми умовного оператора має такий вигляд, як на малюнку. Якщо відповідь на питання-умову є позитивною, то виконується



серія команд «серія 1» (гілка «так»), якщо ж відповідь негативна – серія команд «серія 2» (гілка «ні»). Після виконання однієї з серій команд виконавець переходить до наступної після розгалуження команди. Оператор розгалуження if у цьому випадку діє так: якщо результат перевірки є істиною, if виконує код 1, записаний у фігурних дужках після умови, інакше – код 2 у фігурних дужках після ключового слова else.

Слід пам’ятати, що в кінці оператора-виразу ставиться крапка з комою.

Оператори порівняння та логічні оператори, які використовують при написанні умови:

==	дорівнює	
!=	не дорівнює	
<	менше	
<=	не більше	
>	більше	
>=	не менше	
&&	Логічне «і»	Одна умова і друга
||	Логічне «або»	або перша або друга
!	Заперечення «Ні»	**Val=false; if (!Val)** – значення умови буде: **true**

Використовувати тип **Boolean, (bool)** краще зі значеннями **true** і **false**. С++ прирівнює нуль до **false,** а будь-яке інше число до **true.**

Приклад:

```
bool x = 2;
if (x == 1) then
   {
   Serial.println ("істина");
   }
   else
   {
   Serial.println ("хибність");
   }
//Слово "істина" буде виведено через порт, хоча x дорівнює 2!
```

У С++ існує також скорочений запис умов:

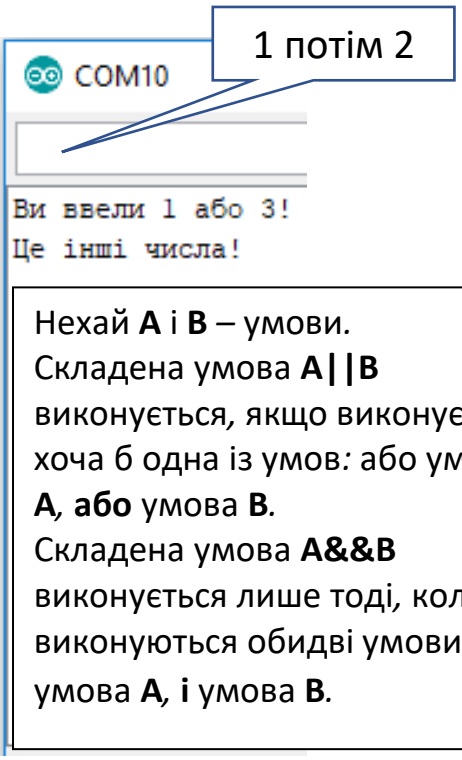
```
(a > b) ? c == true : c == false;
```

Якщо **a** більше, ніж **b**, то **c** дорівнює **true**, а в іншому випадку **c** дорівнює **false**. Також має місце присвоювання результату порівняння:

```
c = = (a> b);
```

Задача: якщо числа прийняті з порту дорівнюють 1 або 3, то виводимо про це сповіщення, інакше повідомляємо, що це інші числа. Результат виведено для введених чисел 1 і 2:

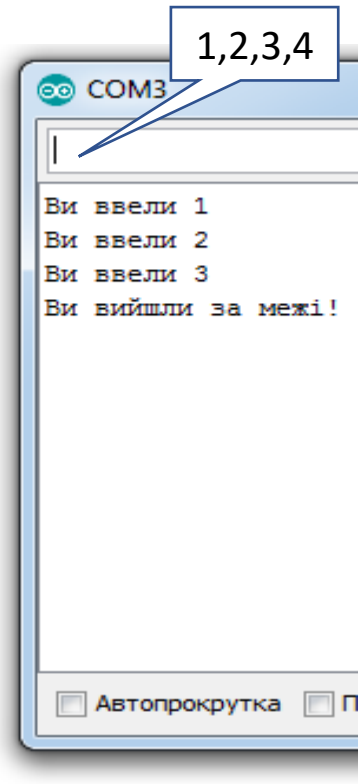
```
byte val;
void setup() {
  Serial.begin(9600);
}
void loop() {
  if(Serial.available())
  {
    val = Serial.parseInt();
    if (val==1 || val==3)
    {
      Serial.println("Ви ввели 1 або 3!");
    }
    else
    {
       Serial.println("Це інші числа!");
    }
  }
}
```



Нехай **A** і **B** – умови.
Складена умова **A||B** виконується, якщо виконується хоча б одна із умов: або умова **A**, **або** умова **B**.
Складена умова **A&&B** виконується лише тоді, коли виконуються обидві умови: **і** умова **A**, **і** умова **B**.

Можна багаторазово вкладати один умовний оператор в інший. Роздивимся приклад:

```
void loop() {
  if (Serial.available())
  {
    val = Serial.parseInt();
      if (val==1)
        {
        Serial.println("Ви ввели 1");
        }
        else if (val==2)
              {
              Serial.println("Ви ввели 2");
              }
              else if (val==3)
                    {
                    Serial.println("Ви ввели 3");
                    }
                    else
                    {
                    Serial.println("Ви вийшли за межі!");
                    }
   }
}
```



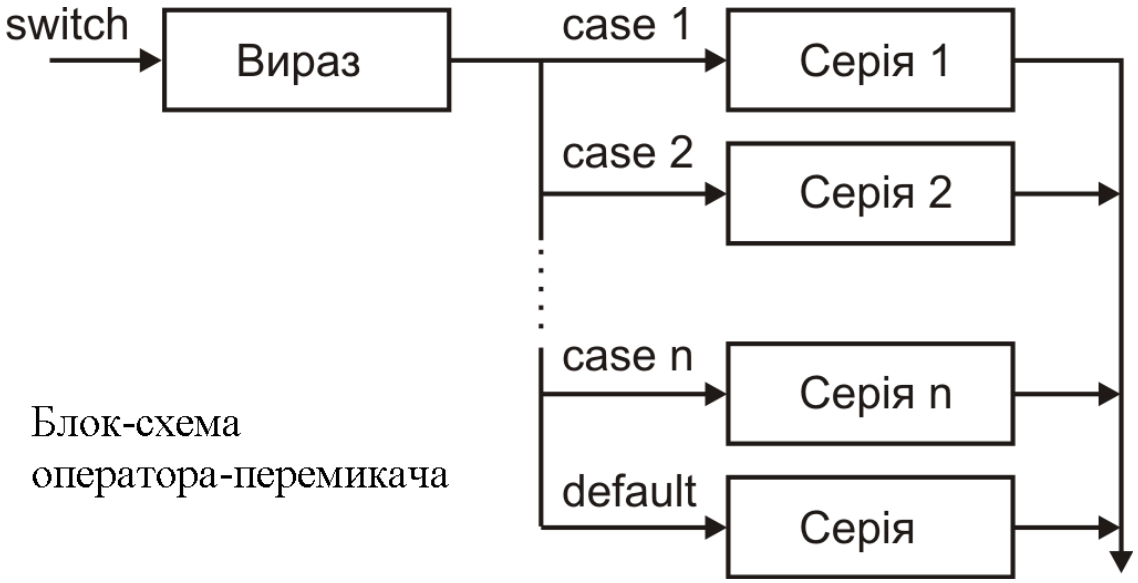
Результат наведено до послідовного введення чисел 1, 2, 3, 4. Цю ж задачу можна виконати значно легше, використавши наступний оператор **switch.. case**.

Урок 5. Оператор вибору switch.. case

У попередній програмі ми використали оператори розгалуження, але при глибині вкладеності цих операторів понад три, така конструкція робить програму складною для сприйняття людиною. Більш зручним у такій ситуації виявляється застосування оператора **switch**.

Загальна структура оператора така:

```
…
switch (вираз)
{
   case константний вираз 1: оператори 1;
         break;
   case константний вираз 2: оператори 2;
         break;
   …
   case константний вираз n: оператори n;
         break;
   default: оператори;
}
…
```



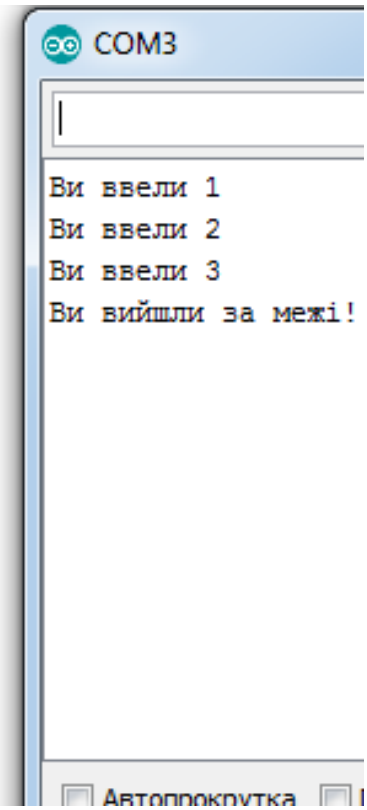
Блок-схема оператора-перемикача

Виконання оператора починається з обчислення виразу (він повинен бути цілочисельним), а потім керування передається першому оператору зі списку, поміченого константним виразом, значення якого співпало з обчисленим. Вихід із перемикача найчастіше виконується за допомогою

оператора **break**. Зверніть увагу на використання оператора **break**. Якщо в програмі зустрічається варіант (**case…**), що дорівнює значенню керуючого виразу оператора **switch**, то подальша перевірка умов припиняється і виконуються всі оператори, аж до кінця оператора **switch**. Оператор **break** є вказівкою завершити поточний оператор **switch** і продовжити виконання програми з першого оператора, що є наступним за оператором **switch**.

Приклад виконання останньої в попередньому уроці задачі:

```
byte val;
void setup() {
  Serial.begin(9600);
}
void loop() {
  if (Serial.available())
  {
    val = Serial.parseInt();
      switch (val) {
        case 1:Serial.println("Ви ввели 1");
          break;
        case 2:Serial.println("Ви ввели 2");
          break;
        case 3:Serial.println("Ви ввели 3");
          break;
        default:Serial.println("Ви вийшли за межі!");
       }
   }
}
```



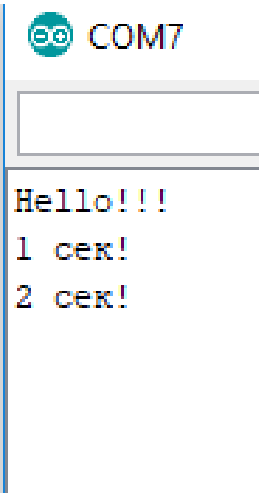
Не важко помітити, що таке рішення більш зрозуміле та наочне.

Висновок: оператори умовний та вибору, які ми розглянули у попередніх 2-х уроках дозволяють обробляти значення з датчиків, переключати режими роботи приладів, робити текстове меню та багато інших цікавих речей.

Урок 6. Функції затримки

Функція **delay()** – затримка, кількість мілісекунд вказана в дужках (в 1 сек 1000 ms (мілісекунд)). Максимальне значення типу **unsigned long** (4 байти), 4 294 967 295 ms, або близько 1200 годин, або 50 днів.

```
void setup() {
 Serial.begin(9600);
 Serial.println("Hello!!!");
 delay(1000);//Призупинити роботу на 1 сек/
 Serial.println("1 сек!");
 delay(2000);//Призупинити роботу на 1 сек/
 Serial.println("2 сек!");
}
void loop() {
}
```



Функція **delayMicroseconds()** – затримка, кількість мікросекунд вказується в дужках (1 сек дорівнює 1000 мілісекунд (ms) або 1000000 мікросекунд (µs)). Максимальне значення становить 16383 µs, або 16 ms. Наприклад:

```
delayMicroseconds(50);      // чекаємо 50 мікросекунд
```

Не рекомендується використовувати затримки! Їх використання в деяких випадках приводить до зависання системи Arduino!

Функції таймера

Функція **millis()** – повертає кількість мілісекунд, що минули після увімкнення Arduino.

Максимальне значення: 4 294 967 295 ms або 50 діб.

Роздільна здатність: 1 мілісекунда.

Функція **micros()** – повертає кількість мікросекунд, які минули з моменту увімкнення МК.

Максимальне значення: 4 294 967 295 µs або 70 хвилин.

Роздільна здатність: 4 мікросекунди для більшості плат Arduino (Повернуте значення завжди кратне 4).

Пишемо **(unsigned long)** або **(long)** перед множенням так, щоб компілятор виділив необхідну кількість пам'яті для проведення операції! (для роботи з великими числами).

Приклад: **(unsigned long) 28 * 24 * 60 * 60 * 1000**, якщо ми хочемо отримати правильний результат множення, що дорівнює числу мілісекунд у 28 днях.

Вирішаємо проблему переповнення таймера в структурі:

```
if (millis() – last_time > 1000)
{ // оператори
      last_time = millis();
}
```

Таймер на 1 секунду таким чином можна запрограмувати так:

```
unsigned long last_time;
void setup() {
 Serial.begin(9600);
}
void loop() {
  if (millis() - last_time > 1000) {
    last_time = millis();
    Serial.println("sek");
  }
}
```

```
sek
sek
sek
sek
sek
sek
sek
sek
```

Замінивши в умові 1000 на 5000 чи 10000, матимемо таймер на 5 сек та 10 сек відповідно. Якщо нам потрібно виконувати якусь дію, наприклад, раз у три дні – замінимо 1000 на вираз**:**

(unsigned long)(3*24*60*60*1000)

(де 3 – дні, 24 – число годин у добі, 60 – число хвилин у годині, 60 – число секунд у хвилині).

Таким чином ми можемо запрограмувати таймер для поливу рослин, або запрограмувати видачу корма тваринам по графіку.

Але, що відбудеться з цією умовою, коли пройде 50 діб? Врешті-решт, **last_time** буде містити число ближче до максимуму, а **millis()** ближче до початкового. Тому потрібно зробити ще одну умову, щоб після завершення 50 діб все не зруйнувалося через переповнення таймера.

if (millis() < last_time) last_time = millis(); // виконається, коли таймер переповниться.

Остаточно програма триденного таймера виглядатиме так:

```
unsigned long last_time;
void setup() {
 Serial.begin(9600);
}
void loop() {
   if (millis() – last_time > (unsigned long)(3*24*60*60*1000)) {
   last_time = millis();
   Serial.println("виведення повідомлення про спрацювання таймера");
   if (millis() < last_time) last_time = millis();
  }
}
```

Можна зробити по іншому:

```
if (nextTime <= millis())
{ // оператори
nextTime = millis() + 5000;
}
```

У даному випадку теж получаємо таймер, що не переповнюється!

Приклад:

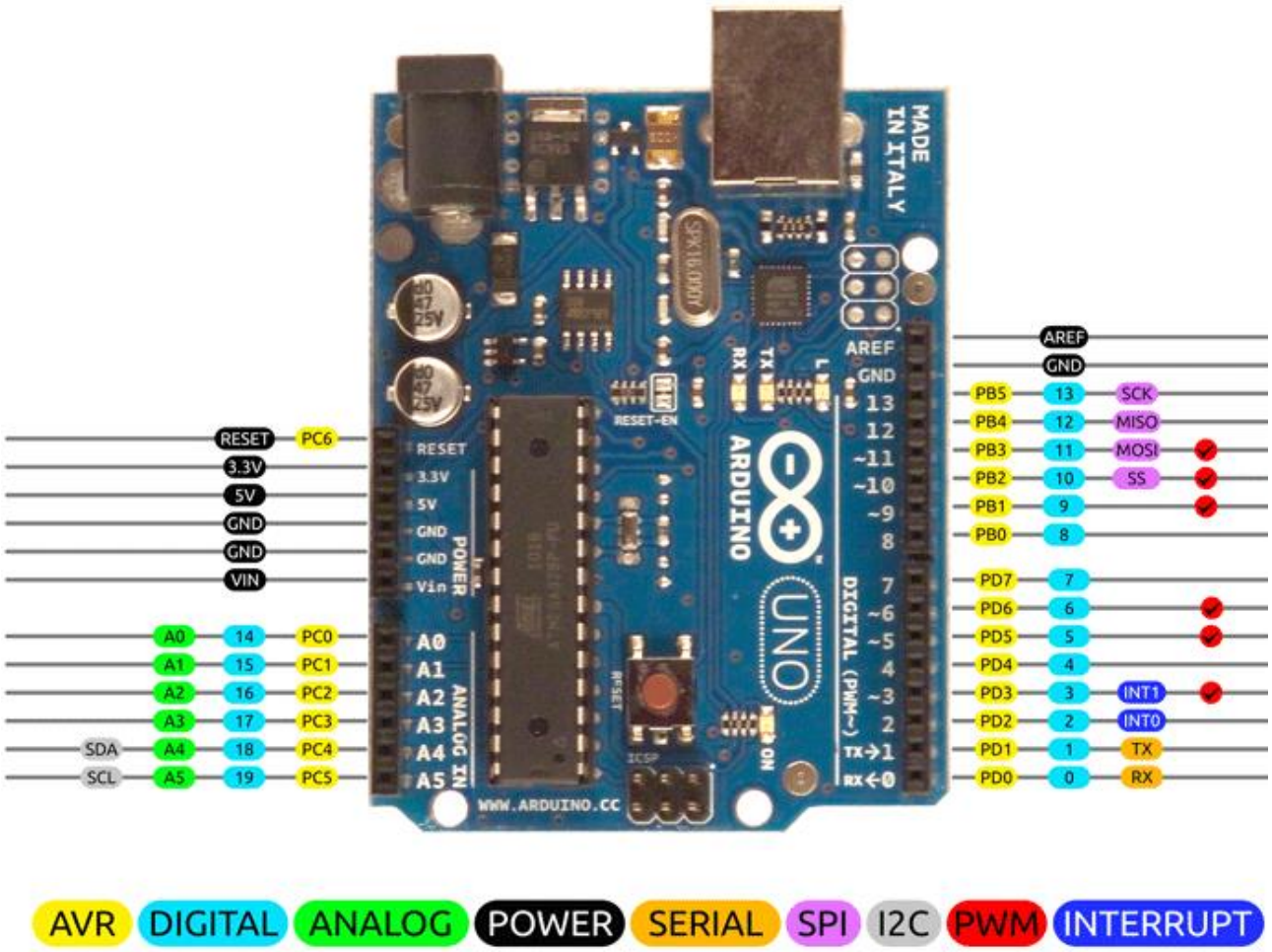
Цей скетч буде виводити повідомлення у монітор послідовного порту кожні 5 секунд:

```
unsigned long nextTime;
void setup() {
 Serial.begin(9600);
}
void loop() {
 if (nextTime <= millis())
  {
  Serial.println("Пройшло 5 секунд");
  nextTime = millis() + 5000;
  }

}
```

```
Пройшло 5 секунд
Пройшло 5 секунд
Пройшло 5 секунд
Пройшло 5 секунд
Пройшло 5 секунд
```

Урок 7. Режими роботи цифрових портів

На цьому занятті Arduino роздивимося порти для введення та виведення даних. Порти бувають цифрові та аналогові. Роздивимося спочатку, як працюють цифрові порти, а також дізнаємось, як підключити кнопки та керувати деякими цифровими пристроями Arduino.



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Цифрові порти позначаються літерою D, Digital, аналогові літерою A, Analog (Дивись малюнок). Усі порти можуть отримувати та видавати цифровий сигнал, тобто 0 або 5 вольт. До будь-якого аналогового або цифрового порту можна підключити цифровий датчик, з яким Arduino спілкується через цифровий сигнал, тобто набір одиниць і нулів. Аналогові порти можуть приймати аналоговий сигнал, тобто вимірювати напругу від 0 до 5 вольт, з точністю приблизно 5 мілівольт. Деякі з цифрових портів можуть видавати «PWM або в перекладі: ШІМ – широтно імпульсна модуляція» сигнал, який являє собою такий собі «гребінець» із значень 0 і 5 вольт, які перемикаються кілька тисяч разів на секунду. Цей сигнал може регулювати яскравість ламп або світлодіодів,

швидкість двигунів, ступінь нагрівання та інше. Ці піни різні на різних платах Arduino, значення їх можна знайти на сайті Arduino. У нашому випадку для Arduino УНО – дивіться малюнок вище.

Зупинимося на цифрових портах

Будь-який порт може працювати як вхід, так і як вихід. За замовчуванням всі порти налаштовані як входи. Щоб повідомити Arduino, що ми хочемо використовувати порт, існує команда **pinMode**. В дужках вказується номер піну, а через кому – режим роботи.

pinMode(pin, mode);

Для нашої плати цифрові піни занумеровані від 0 до 13, а при невикористанні аналогових – ще й з 14 по 19. Аналогові піни з 14 по 19 або A0…A5.

pinMode(pin, mode); – налаштування порту, де:

- **pin –** номер порта. Цифрові: 0 – 13. Аналогові: 14-19, або a0-a5
- **mode –** Режим роботи порту
- **INPUT –** вхід, сигнал приймає
- **OUTPUT** – вихід, видає 0 або 5 вольт
- **INPUT_PULLUP** – вхід з підтягуванням до 5 В (підключення внутрішнього підтягуючого резистора)

Генерація цифрового сигналу

digitalWrite(pin, signal); – подати цифровий сигнал

- **pin –** номер порту. Цифровий: 0 – 13. Аналоговий: 14-19, або a0-a5
- **signal –** який сигнал подається
- **LOW**, або 0 (нуль), або ХИБНІСТЬ (false) або 0 Вольт
- **HIGH**, або 1, або ІСТИНА (true) або 5 Вольт

Читання цифрового сигналу

digitalRead(pin); – читати цифровий сигнал

- **pin –** номер порту. Цифровий: 0 – 13. Аналоговий: 14-19, або a0-a5

Приклад 1.

```
void setup() {
  pinMode(A4,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);

  digitalWrite(13,HIGH);
  digitalWrite(12,1);
  digitalWrite(A4,0);
}
void loop() {
  // put your main code h
}
```

Тестером[3] заміряємо напругу на **пінах**:
Пін А4 – буде 0 Вольт.
Пін 13 – буде 5 Вольт, і світиться встановлений на цей пін світлодіод на платі. Записавши код: **digitalWrite(13, 0);** або **digitalWrite(13, LOW);** та завантаживши скетч – ми згасимо світлодіод на платі.
Пін 12 – буде 5 Вольт.

Зауважимо, що вихідна напруга, що видає тестер, може коливатися навколо 5 Вольт, це залежить від вхідного живлення плати Arduino.

Приклад 2.

Подивимося, як надані нам значення на пінах будуть виведені у порт. Напишемо такий код:

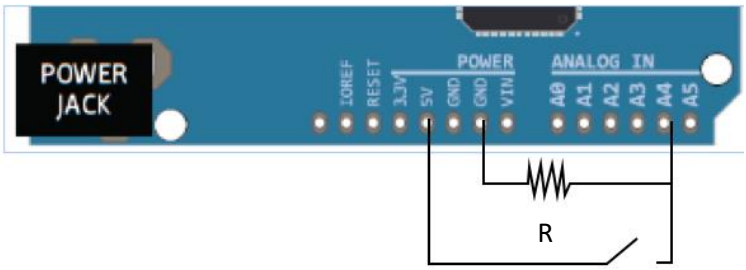
```
void setup() {
  pinMode(A4,INPUT);
  pinMode(12,INPUT);
  pinMode(13,INPUT);
  Serial.begin(9600);
}
void loop() {
  boolean button1 = digitalRead(A4);
  boolean button2 = digitalRead(12);
  Serial.print("Button1 - ");Serial.print(button1);
  Serial.print(", Button2 - ");Serial.println(button2);
}
```

```
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
```

Будемо виводити в порт значення з пінів А4 та 12. Як бачимо – обидва значення нулі. Це відбувається тому, що ми не надали ніяких значень пінам. Тепер спробуємо причепити кнопку між піном А4 та значенням +5 В на нашій платі. Що при цьому відбувається: натискаємо кнопку і значення Button1 змінюється на 1-цю, але, відпустивши кнопку, побачимо, що 1-ця залишилася! Потім слідує хаотичне змінювання нулів та одиниць… І нарешті – знову нуль…

[3] Прилад для вимірювання напруги, сили току, та інших параметрів електричних схем

Для того, щоб цього не відбувалося потрібно підтягнути потрібний цифровий пін до землі (земля позначається на платах як **GND**) через резистор R= 10 Ком, як показано на схемі:



Тепер проблема зникла! Натискаємо на кнопку, маємо 1-цю, відпускаємо – одразу 0!

На щастя, Arduino має вмонтований резистор і вмикається він за допомогою команди **INPUT_PULLUP**. Цей режим зручно використовувати для підключення кнопок, але з невеликими відмінностями. Провід від цифрового піна, через кнопку, повинен йти не до 5 вольт, а до землі **(GND)**. При натисканні кнопки ми матимемо – 0, а при відпущеній – 1. Для зручності інвертуємо сигнал, за допомогою логічного виразу «NOT» мови С++, що позначається, як «!».

```
void setup() {
  pinMode(A4,INPUT_PULLUP);
  pinMode(12,INPUT);
  pinMode(13,INPUT);
  Serial.begin(9600);
}
void loop() {
  boolean button1 = !digitalRead(A4);//інвертуємо сигнал
  boolean button2 = digitalRead(12);
  Serial.print("Button1 - ");Serial.print(button1);
  Serial.print(", Button2 - ");Serial.println(button2);
}
```

```
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
Button1 - 0, Button2 - 0
```

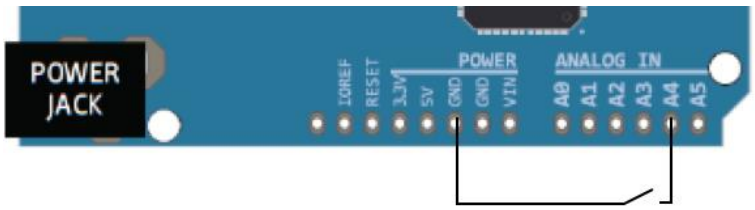
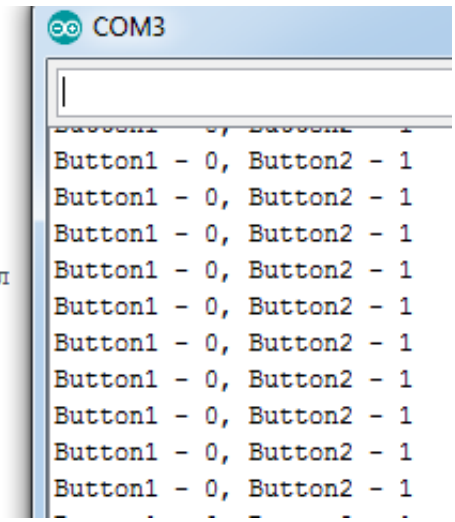


Схема підключення тепер стане такою, і все чудово працюватиме! Надалі рекомендую підключати кнопки та обробляти їх стан таким чином!

На завершення уроку спробуємо налаштувати вмикання та вимикання світлодіоду на платі Arduino. Пишемо умову: якщо кнопка натиснута, то подати 1-цю на 13 порт, якщо кнопка не натиснута, то подати 0. Таким

чином ми керуємо вмонтованим світлодіодом на платі Arduino. Скетч дивись нижче:

```
void setup() {
  pinMode(A4,INPUT_PULLUP);
  pinMode(12,INPUT);
  pinMode(13,INPUT);//тут вмонтований світлодіод
  Serial.begin(9600);
}
void loop() {
  boolean button1 = !digitalRead(A4);//інвертуємо сигнал
  boolean button2 = digitalRead(13);
  if(button1 == 1) digitalWrite(13,1);//УВІМКНУТИ
  else digitalWrite(13,0);//ВИМКНУТИ
  Serial.print("Button1 - ");Serial.print(button1);
  Serial.print(", Button2 - ");Serial.println(button2);
}
```



Наведений скетч дозволяє нам змінювати значення Button1 з нуля на 1-цю шляхом натискання кнопки, підключеної між землею GND та піном A4, і вмикати-вимикати світлодіод, підключений до піну 13.

Урок 8. Прапорці та розширене керування кнопкою

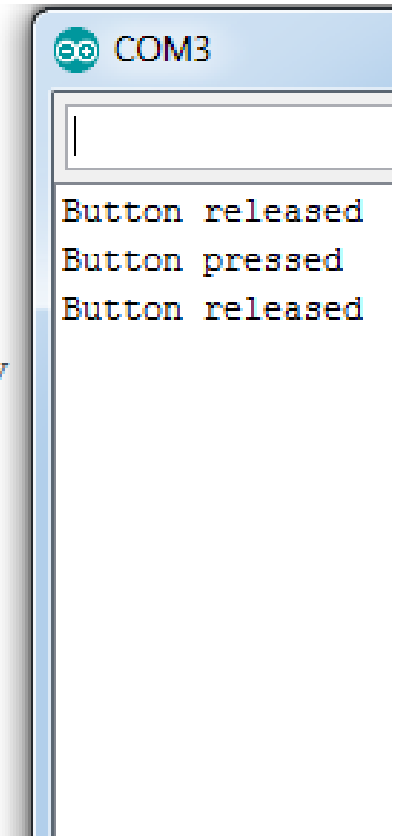
Недолік попереднього скетчу для вмикання світлодіода на 13 цифровому піні Arduino в тому, що він вмикається тільки при утриманні кнопки натиснутою. Якщо кнопку відпустити – світлодіод одразу згасне. Логічно запрограмувати кнопку так, щоб кнопка перемикала стан світлодіода – перше натискання приводило до вмикання світлодіода, наступне – до вимикання.

Цикл **loop(),** як ми пам'ятаємо з попередніх уроків, – нескінчений і виконується 1000 разів за секунду. Нам же потрібно виконати всього одну дію. Для цього введемо поняття логічної змінної – прапорця, яка зберігатиме стан об'єкта. Назвемо її **butt_flag** та присвоїмо їй початкове значення нуль. Кнопку підключимо на 5-й пін, підтягнувши його до землі за допомогою **INPUT_PULLUP**. Змінна **butt** буде приймати поточне значення кнопки. У зв'язку з використанням підтяжки **INPUT_PULLUP**, інвертуємо значення піну 5 на протилежне. Таким чином, якщо кнопка натиснута, **butt** дорівнює одиниці, якщо не натиснута, **butt** дорівнює нулю.

Створимо умову. На початку виконання коду **butt_flag = 0**. Якщо сигнал на піні дорівнює одиниці (кнопка натиснута) **butt = 1**, то перша умова виконується і прапорець змінює значення на 1-цю: **butt_flag = 1**. Коли буде виконуватися наступний цикл **loop** умова вже не виконається. Таким чином, код у фігурних дужках виконається тільки один раз, при натисканні кнопки.

При відпусканні кнопки **butt = 0**, а **butt_flag =1**, виконається друга умова і значення прапорця знову зміниться на протилежне. Для наочності виведемо в порт значення, які будуть з'являтися при натисканні і відпусканні кнопки. Дивись скетч нижче:

```
boolean butt_flag =0;
boolean butt;
void setup() {
  pinMode(5,INPUT_PULLUP);
  Serial.begin(9600);
}
void loop() {
  butt = !digitalRead(5);//зчитати стан піну
  if (butt == 1 && butt_flag ==0) {
    butt_flag = 1;
    Serial.println("Button pressed");
  }
  if (butt == 0 && butt_flag ==1) {
    butt_flag = 0;
    Serial.println("Button released");
  }
}
```

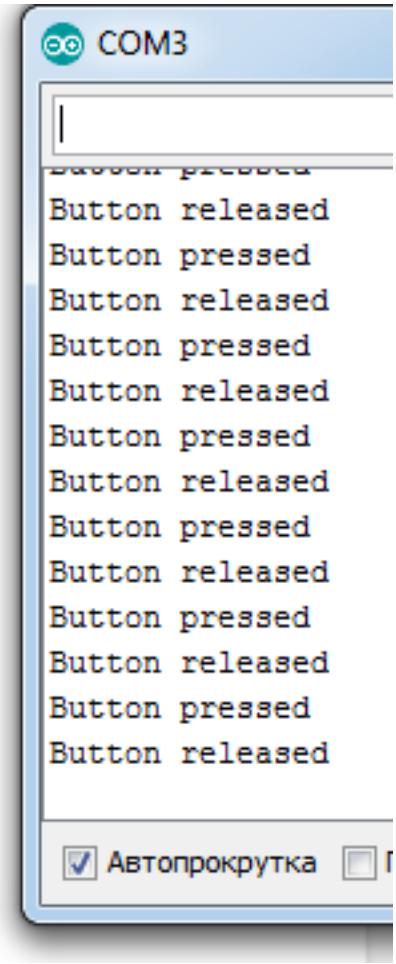


Пам'ятка з попередніх уроків!

Нехай **A** і **B** – умови.
Складена умова **A||B** виконується, якщо виконується хоча б одна із умов: або умова **A**, **або** умова **B**.
Складена умова **A && B** виконується лише тоді, коли виконуються обидві умови: **і** умова **A**, **і** умова **B**.

Напишемо скетч, що буде вмикати і вимикати кнопкою на 5-му піні світлодіод, що підключений до 13 піну:

```
boolean butt_flag = 0;
boolean butt;
boolean led_flag = 0;
void setup() {
  pinMode(5,INPUT_PULLUP);
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}
void loop() {
  butt = !digitalRead(5);//зчитати стан піну
  if (butt == 1 && butt_flag ==0) {
    butt_flag = 1;
    Serial.println("Button pressed");
    led_flag =!led_flag;
    digitalWrite(13,led_flag);
  }
  if (butt == 0 && butt_flag ==1) {
    butt_flag = 0;
    Serial.println("Button released");
  }
}
```



Натискаючи кнопку, ми побачимо небажане явище, яке називають «брязкіт контактів[4]». Воно з'являється завдяки тому, що під час натискання кнопки може кілька разів замкнутися та розімкнутися контакт. Для виключення «брязкіта контактів» доопрацюємо програму затримкою після натискання кнопки. Можна використати **delay(50)**, але використання цієї функції небажане, тому підемо іншим шляхом, використавши функцію **millis**, як це ми робили в попередніх уроках.

[4] Небажане замикання й розмикання контактів в момент комутації

Скетч що виключає «брязкіт контактів»:

```
boolean butt_flag = 0;
boolean butt;
boolean led_flag = 0;
unsigned long last_press;
void setup() {
  pinMode(5,INPUT_PULLUP);
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}
void loop() {
  butt = !digitalRead(5);//зчитати стан піну
  if (butt == 1 && butt_flag == 0 && millis()-last_press>50) {
    butt_flag = 1;
    Serial.println("Button pressed");
    led_flag =!led_flag;
    digitalWrite(13,led_flag);
    last_press = millis();
  }
  if (butt == 0 && butt_flag ==1) {
    butt_flag = 0;
    Serial.println("Button released");
  }
}
```

А як же відпрацювати не просто натискання, а наприклад, утримання кнопки або подвійного натискання? Для цього існує бібліотека **OneButton**, яка дозволяє відпрацювати всі перераховані варіанти натискання кнопки.

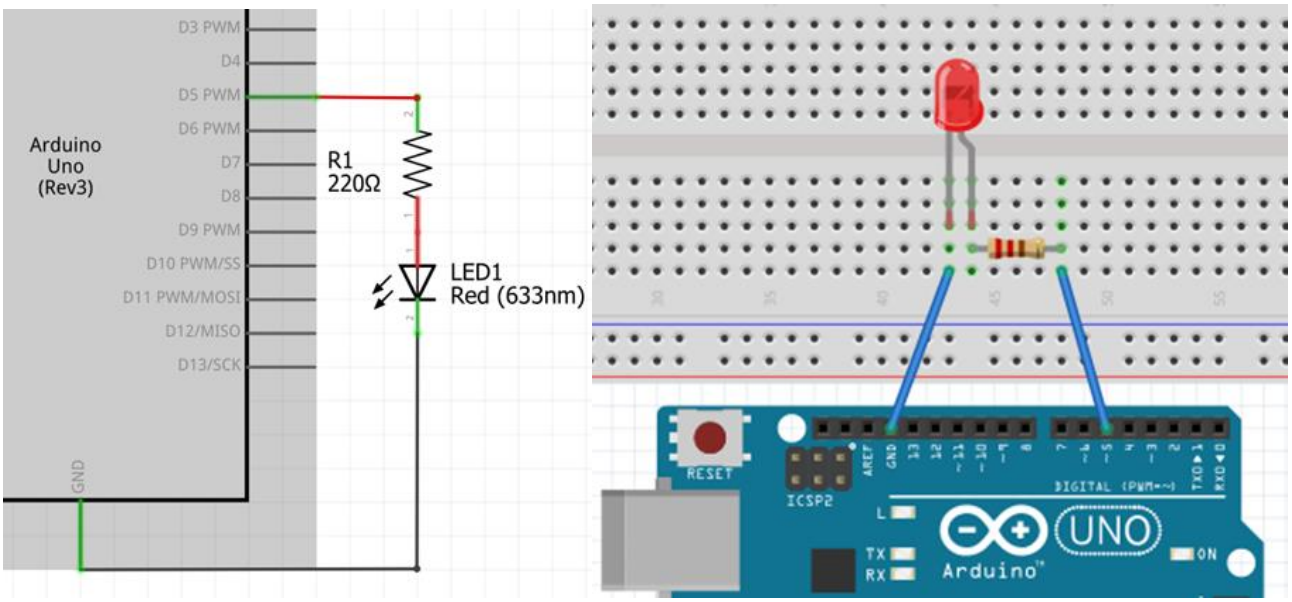
Скачати можна за посиланням:

https://github.com/mathertel/OneButton

Як працювати з бібліотеками ми роздивимося на 19 уроці.

Урок 9. Як підключити світлодіод до Arduino

Виходи Arduino служать тільки для керування іншими приладами. До плати Arduino можна безпосередньо підключати деякі елементи з споживаним струмом до 40 міліампер. Це можуть бути 2 світлодіоди, якась пищавка, датчик. Світлодіоди підключають через резистор, який обмежує струм.



Розрахуємо номінал резистора. Для цього ознайомимося з деякими параметрами нашої плати Arduino. Плата живиться від джерела живлення 7-12 В, або від ЮСБ комп'ютера 5 В. Але ця напруга на самій платі зменшується до 5 В, і є один вихід 3,3 В. Таким чином, на будь-якому виході Arduino може бути присутня напруга 5 В. Якщо подивитися на технічні характеристики світлодіодів, то побачимо, що падіння напруги на них приблизно 3 В а струм 0,02 Ампера. Тому маємо такі вхідні дані:

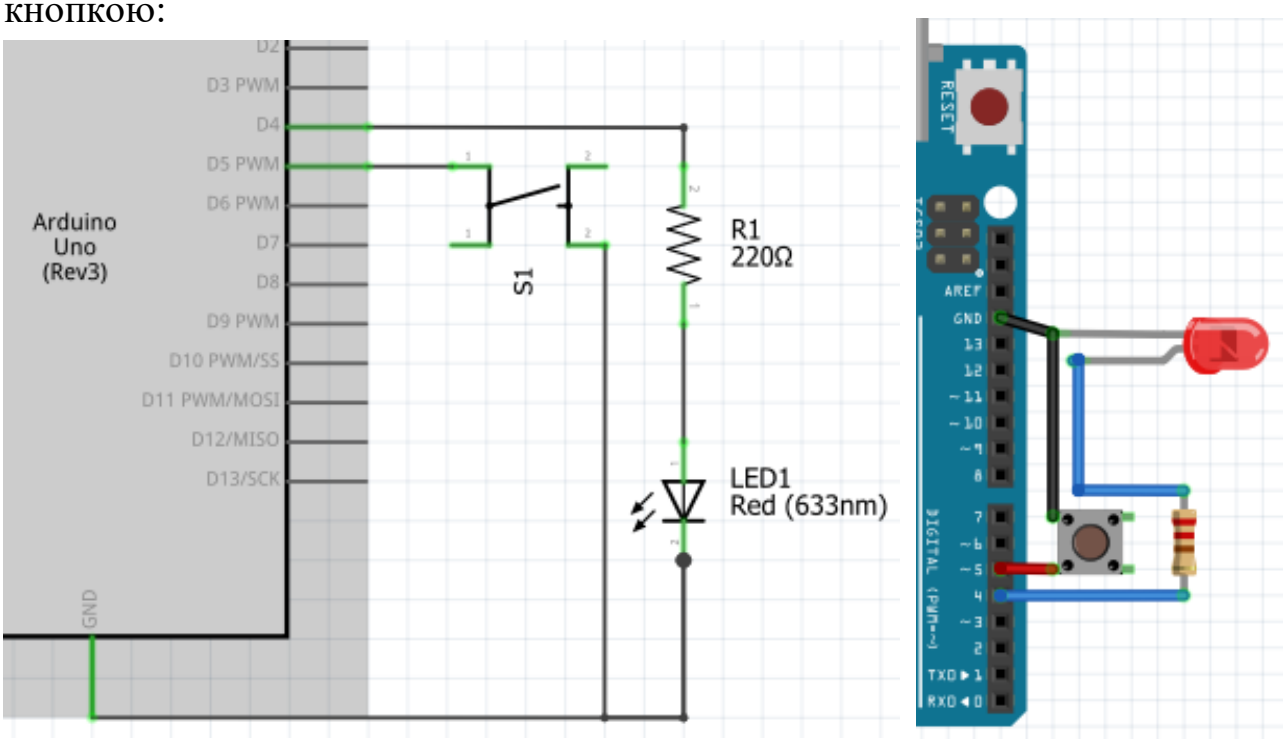
$I_{сп}$ – споживаний світлодіодом струм, V_R – падіння напруги на резисторі, $V_{дж}$ – напруга джерела живлення. Згідно закону Ома знайдемо значення резистора, що обмежує струм:

$$R = \frac{V_{дж} - V_R}{I_{сп}} = \frac{5-3}{0{,}02} = 100 \text{ Ом}$$

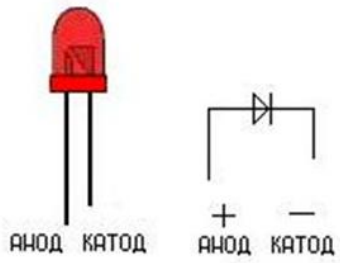
Це буде R для максимального значення струму, що витримає світлодіод. Для номінального значення можемо збільшити значення R до 220 Ом. Для того, щоб зібрати макетну схему, потрібно ще знати, що світлодіод має полярність – анод та катод. Звичайно, мінусовий вихід

(катод) має меншу довжину. Це обов’язково треба приймати до уваги, збираючи схему.

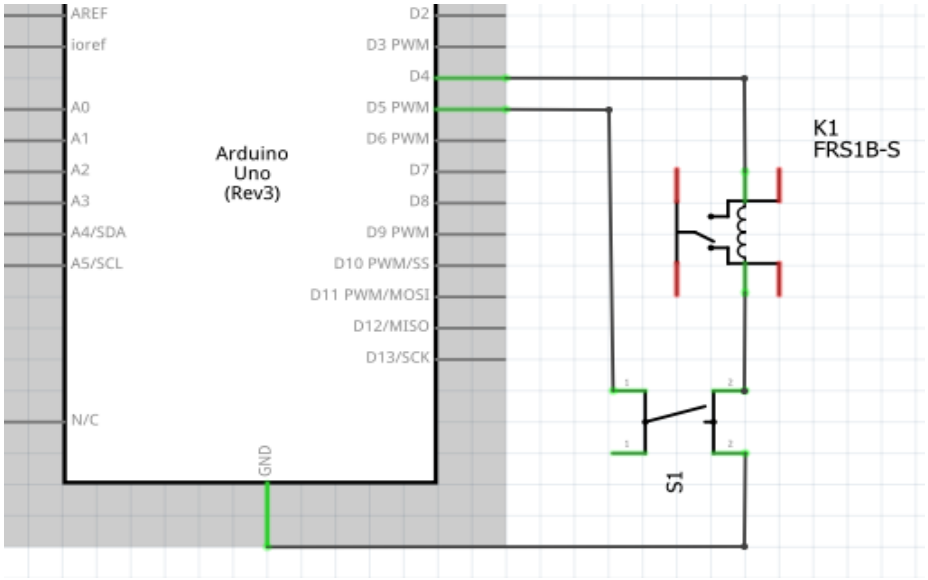
Вивчивши теорію, спробуємо на основі теорії попереднього уроку створити схему, де зовнішній світлодіод буде вмикатися та вимикатися кнопкою:

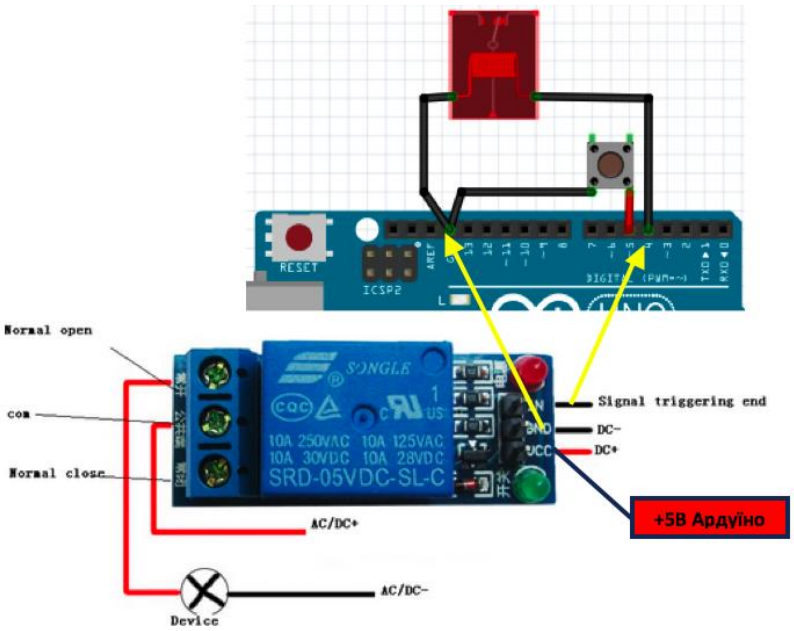



```
boolean butt_flag = 0;
boolean butt;
boolean led_flag = 0;
unsigned long last_press;
void setup() {
  pinMode(5, INPUT_PULLUP);
  Serial.begin(9600);
  pinMode(4, OUTPUT);
}

void loop() {
  butt = !digitalRead(5); // считать текущее положение кнопки

  if (butt == 1 && butt_flag == 0 && millis() - last_press > 100) {
    butt_flag = 1;
    Serial.println("Button pressed");
    led_flag = !led_flag;
    digitalWrite(4, led_flag);
    last_press = millis();
  }
  if (butt == 0 && butt_flag == 1) {
    butt_flag = 0;
    Serial.println("Button released");
  }
}
```



А тепер спробуємо замість світлодіода підключити РЕЛЕ, яке буде перемикати контакти і вмикати споживача з великим навантаженням, наприклад, лампу або бойлер у системі «Розумний дім»:





На схемі вище відсутнє підключення входу реле DC+. Його треба підключити до +5В на платі Arduino. До силових клем можна підключати велике навантаження, величина якого обмежена параметрами реле. Для нашого варіанту – це 10А та 250В, або 2 кВт. Але краще використовувати навантаження вдвічі менше за максимальне – тоді реле служитиме довго.

Скетч підходить від попереднього завдання, просто замість світлодіода з резистором на ті ж піни потрібно причепити реле. Можна звичайно замінити деякі назви – замість led прописати reley. Але це не обов’язково.

Урок 10. Робота з аналоговими портами

На відміну від цифрового, аналоговий вхід Arduino приймає сигнали від 0 до 5В. Але для того, щоб з цим сигналом можна було працювати далі, він оцифровується. Для цього служить АЦП – аналого-цифровий перетворювач з розрядністю 10 біт. Це означає, що коли на вході АЦП напруга змінюється від 0 до 5В, на виході АЦП маємо значення від 0 до 1023. Тобто $2^{10} = 1024$ різних значення. А щоб получити значення напруги, потрібно порахувати просту пропорцію:

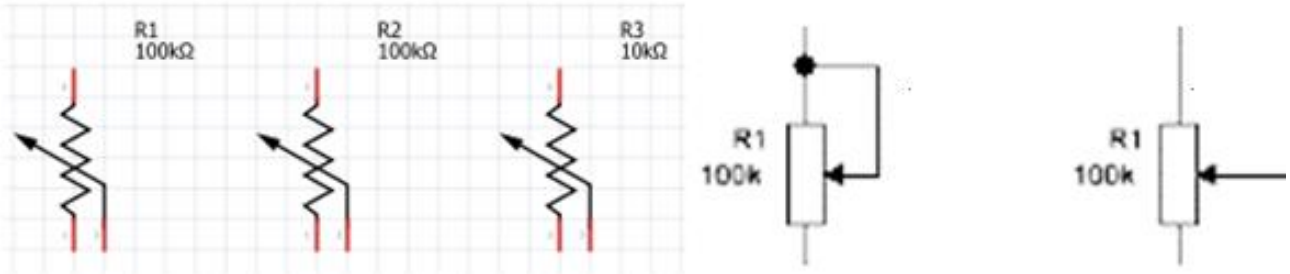
$$V_{real} = \frac{значення\ з\ АЦП * 5}{1024}$$

Значення 5В може трохи змінюватися, в залежності від блоку живлення Arduino.

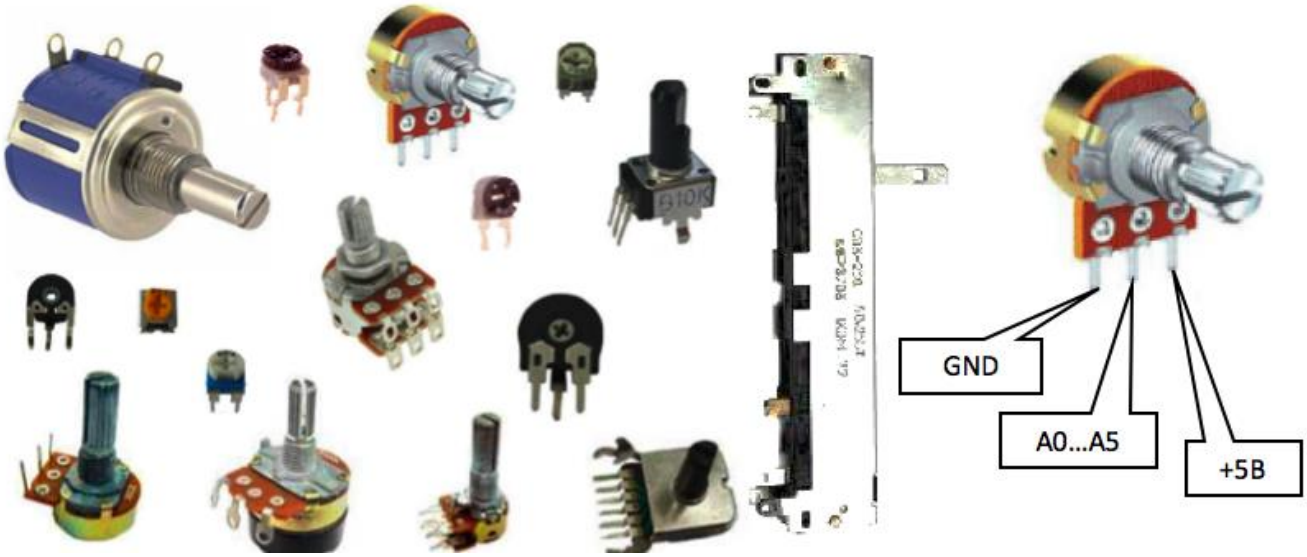
Аналогові порти використовують для прийняття сигналів з аналогових датчиків, таких як датчики звуку, світла, джойстики, а також змінні резистори або потенціометри.

Роздивимося останні. На малюнках виглядають вони ось так:

На принциповій схемі позначаються так:



Реальні виглядають ось так:



Два крайні виводи підключаємо до землі GND та піну +5В. Середній вивід можна підключати до будь-якого аналогового порту A0…A5.

Ознайомимося з основними функціями для опрацьовування аналогового сигналу.

Читання аналогового сигналу:

Функція **analogRead(pin);** зчитує аналоговий сигнал (оцифровує), де

pin – номер Аналогового піна: 0-5 (Для Arduino UNO).

Функція повертає значення 0.. 1023, в залежності від напруги на піні від 0 до приблизно 5 В.

Змінення діапазону значень:

Функція **map(val, min, max, new_min, new_max);** повертає величину в новому діапазоні, де

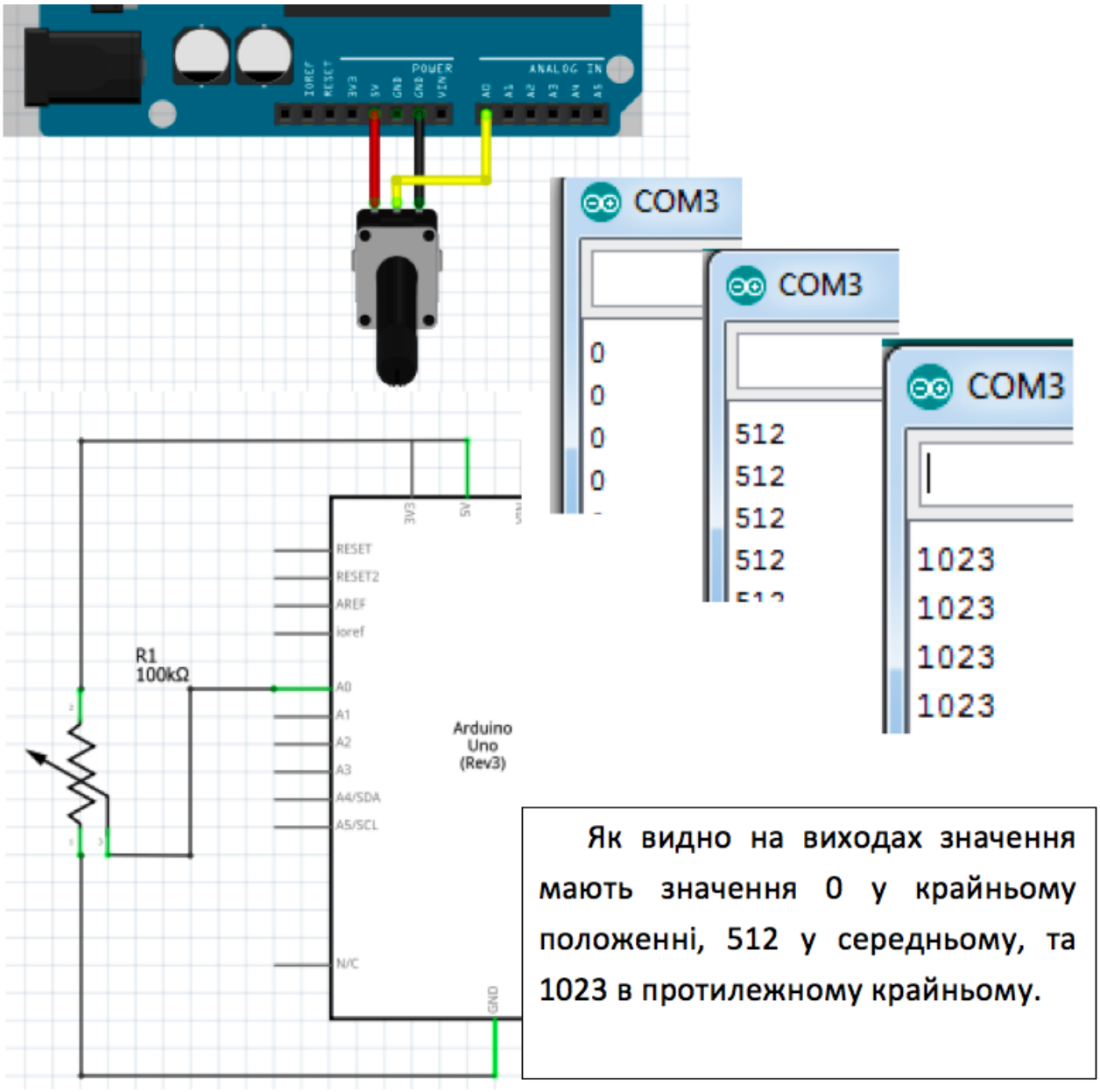
val – вхідна величина.

min, max – мінімальне і максимальне значення при вході в **map.**

new_min, new_max – відповідно, мінімальне і максимальне значення на виході.

Функція **constrain(val, min, max);** обмежує діапазон змінної **val** до вказаних значень **min** і **max.**

Побудуємо схему:



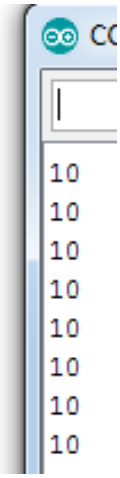
Як видно на виходах значення мають значення 0 у крайньому положенні, 512 у середньому, та 1023 в протилежному крайньому.

Напишемо скетч:

```
#define potent_pin 0 //Сюди підключена середня ніжка потенціометра
int val;             //Змінна для зберігання значень потенціометра
void setup() {
  Serial.begin(9600);
}

void loop() {
  val = analogRead(potent_pin);//Запам'ятати значення з потенціометра
  Serial.println(val);//Вивести в порт
  delay(30);
}
```

Зменшимо діапазон цифрових значень з 0 до 10, використавши вивчені функції:

```
#define potent_pin 0   // Сюди підключена середня ніжка потенціометра
int val;               // змінна для зберігання значень потенціометра
void setup() {
  Serial.begin(9600);
}
void loop() {
  val = analogRead(potent_pin);  // запам'ятати значення з потенціометра
  val = map(val, 0, 1023, 0, 10);// перевести в діапазон 0.. 10
  val = constrain(val, 0, 10); // обмежити діапазон 0.. 10
  Serial.println(val);              // вывести в порт
  delay(30);
}
```

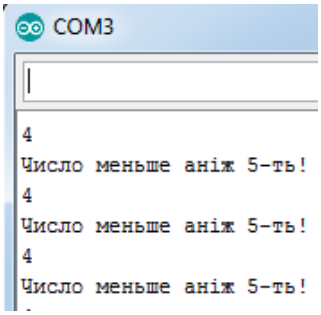


Функції **map** і **constrain** звичайно використовують в парі. Тепер ми легко зможемо регулювати кут повертання сервоприводу, яскравість світла, швидкість обертання, потужність нагріву, силу утримання електромагніту, і навіть кількість світлодіодів, що світяться!

Якщо додати просту умову, то можна дізнатися про перевищення якогось порогового значення з датчика.

У наступному прикладі навчимо Arduino повідомляти нам, чи число більше, чи менше аніж п’ять з діапазону 0…10 виводиться у COM порт:

```
#define potent_pin 0 //сюди підключена середня ніжка потенціометра
int val;             //змінна для зберігання значень потенціометра
void setup() {
  Serial.begin(9600);
}
void loop() {
  val = analogRead(potent_pin);//запам'ятати значення з потенціометра
  val = map(val, 0, 1023, 0, 10);    //перевести в діапазон 0...10
  val = constrain(val, 0, 10);          //обмежити діапазон 0...10
  if(val>5)Serial.println("Число більше 5-ти!");  //вивести в порт
  else Serial.println("Число меньше аніж 5-ть!"); //вивести в порт
  Serial.println(val);                           //вивести в порт
  delay(30);
}
```

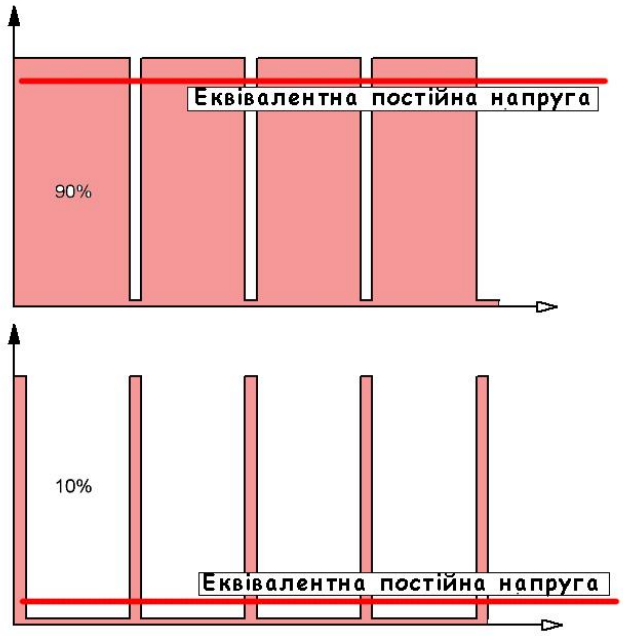


Таким чином, можна, наприклад, впіймати момент про перевищення рівня з датчика звуку або світлового потоку з датчика світла та вирішити інші задачі з використанням порогових значень.

Урок 11. Широтно-Імпульсна Модуляція (ШІМ, PWM)

Після вивчення попередніх тем перед нами обов'язково постане питання, як можна регулювати потужність споживача, наприклад, керувати яскравістю світлодіода або регулювати швидкість двигуна. Найпростіший спосіб – послідовно навантаженню, наприклад, – світлодіоду, включити резистор. Але ж він буде грітися і забирати дорогоцінну енергію, і чим потужніше світлодіод, тим сильніше буде грітися резистор. Такий варіант нам не підходить. А давайте спробуємо дуже швидко вмикати і вимикати світлодіод, при цьому змінюючи тривалість включень! Наприклад, якщо включати світлодіод на 0,5 мілісекунд кожну мілісекунду, то світлодіод засвітиться, але не на повну яскравість. Аналогічно з двигуном – включати двигун на 30 секунд кожну хвилину – тоді двигун розкрутиться, але не на повну швидкість, останні 30 секунд він буде рухатися по інерції, та гальмуватися за рахунок сили тертя. Таким чином, двигун буде крутитися на 50 відсотків своєї потужності.

Широтно-імпульсна модуляція (ШІМ, PWM)



ШІМ є імпульсний сигнал постійної частоти і змінної скважності, тобто відношення періоду проходження імпульсу до його тривалості. З допомогою завдання скважності (тривалості імпульсів) можна міняти середню напругу на виході **ШІМ**. У цифровій техніці, виходи якої можуть приймати тільки одне з двох значень, наближення бажаного середнього

рівня виходу за допомогою ШІМ є абсолютно природним. Давайте на практиці спробуємо змінювати яскравість світлодіода. Але спочатку ознайомимося з функціями та особливостями Arduino, при роботі з ШІМ сигналами:

Генерація ШІМ-сигналу відбувається за допомогою функції:

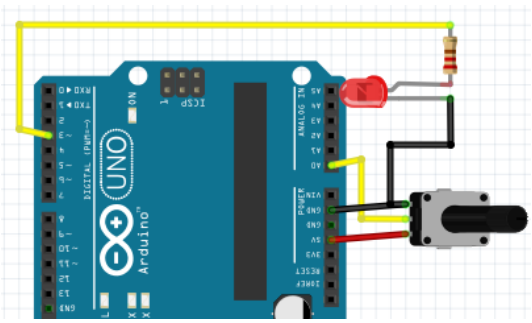
analogWrite(pin, duty);

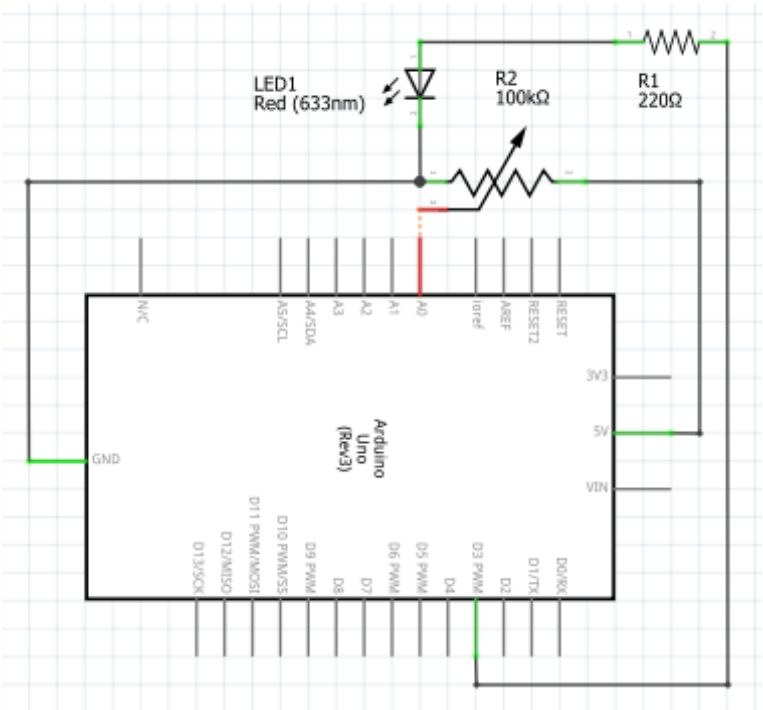
pin – пін, на якому генерується ШІМ

duty – величина 0.. 255, відповідає значенню ШІМ 0.. 100%.

ШІМ-піни Arduino NANO, UNO: 3, 5, 6, 9, 10, 11 (інші моделі – дивись інструкцію).

А тепер змонтуємо макет з використанням світлодіода, резистора, що обмежує струм, та потенціометра, за допомогою якого будемо змінювати скважність сигналу на на ШІМ піні 3 Arduino:





Напишемо скетч:

```
int pwm;
void setup() {
}
void loop() {
 pwm = analogRead(0);
 pwm = map(pwm, 0, 1023, 0, 255);
 pwm = constrain(pwm,3,pwm);
 analogWrite(3, pwm);
}
```

Змінній цілого типу **pwm** присвоюється значення з аналогового входу A0, яке змінюється потенціометром від 0 до 1023. Функція **map** змінює цей діапазон до значень 0…255. Функція constrain на всяк випадок обрізає цей діапазон. Після цього змінна **pwm** передається на ШІМ вихід на 3 пін. Якщо тепер покрутити потенціометр, світлодіод буде змінювати яскравість світла!

Урок 12 Організація циклів

Цикл – це форма організації дій, при якій одна і та ж послідовність дій виконується кілька разів доти, поки виконується деяка умова. Пам'ятаймо! У всіх скетчах Arduino маємо нескінчений цикл **loop().** Тому цикли, що ми роздивимося нижче являтимуть собою вкладені цикли!

Цикл з лічильником for

Розглянемо синтаксис оператора циклу з лічильником у C++:

for (оператор1; вираз1; вираз2)

оператор_тіла_циклу;

Послідовність його виконання така:

крок 1: виконується **оператор1**;

крок 2: обчислюється **вираз1**, і якщо він істинний, то виконується **оператор_тіла_циклу**. Якщо хибний, то виконання циклу припиняється і виконується наступний оператор;

крок 3: обчислюється **вираз2**;

крок 4: повторюються кроки 2-4.

Наприклад, у скетчі може зустрітися такий цикл з лічильником:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  for (int i = 0; i < 10; i++) {
    Serial.println(i);
    delay(100);
  }
delay(5000);
}
```

```
5
6
7
8
9
0
1
2
3
4
```

Цей скетч виводитиме в порт Arduino послідовність цифр від 0 до 9-ти з невеликою затримкою 100 ms, потім відбудеться затримка 5000 ms і процес почнеться знову. У зв'язку з тим, що ми маємо цикл в циклі – це відбуватиметься нескінченно.

for – ключове слово, що означає початок циклу;

Оператор1: int i = 0, де **i** – змінна лічильника циклу, якій присвоюється початкове значення лічильника 0;

вираз1: i < 10 – умова, після досягнення якої цикл завершується;

вираз2: i++ – зміна лічильника циклу. Якщо крок не дорівнює одиниці, можна, наприклад, написати: **i = i + 5**, чи **i = i + 0.25**. Можливе скорочення запису: **i += 5** у першому випадку та **i += 0.25** у другому. Не забувайте, що у другому випадку змінна циклу повинна мати тип **float**!

В даному випадку виконання циклу здійснюється за такою схемою:

1. Параметр і одержує значення **0**.
2. Робиться перевірка, чи не перевищує значення змінної циклу кінцевого значення лічильника **9**.
3. Якщо не перевищує, то виконуються оператори у фігурних дужках. Якщо оператор у циклі один, його можна у дужки не брати.
4. Цикл закінчує роботу, як тільки умова **i < 9** стане хибною.

Розглянемо ще один приклад, щоб закріпити наші знання:

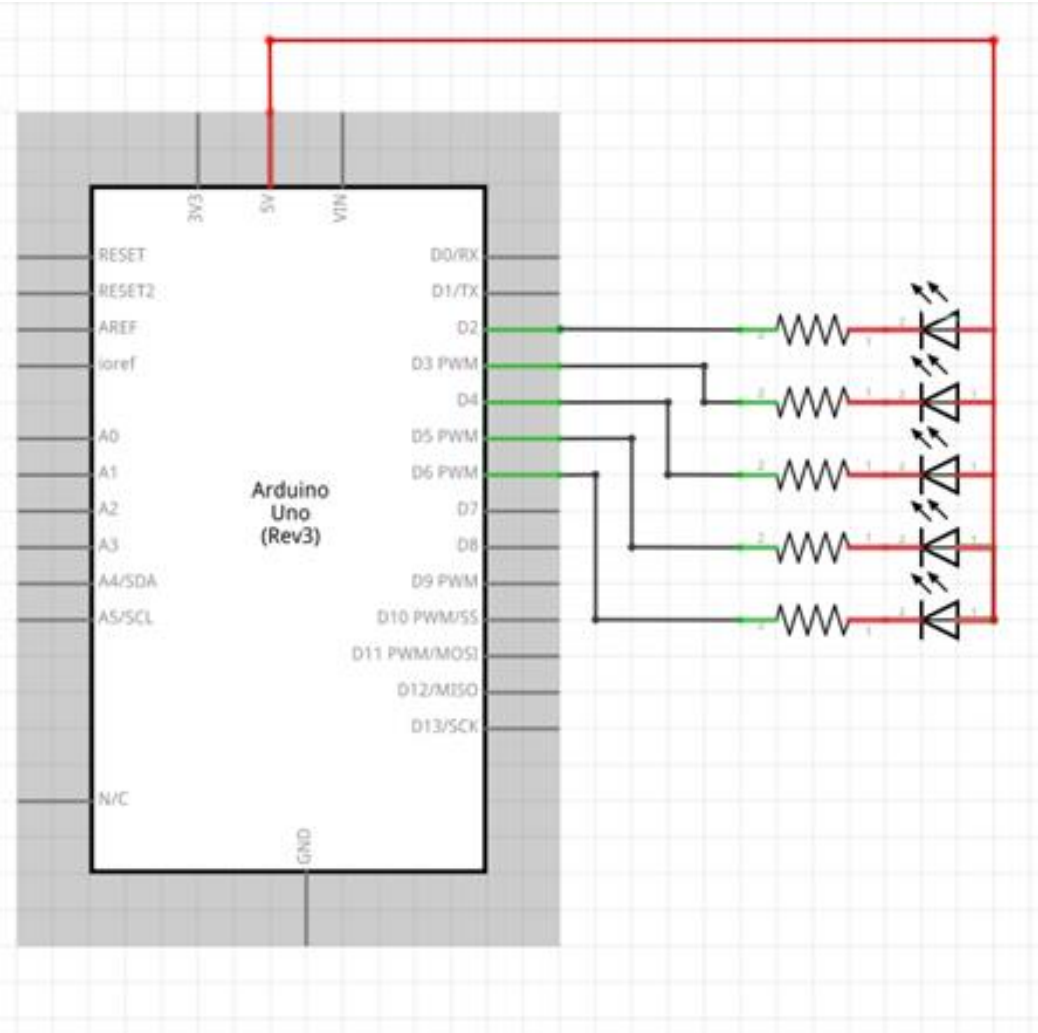
```
void setup() {
  Serial.begin(9600);
}
void loop() {
  for (float i = 2; i <= 4; i+=0.25) {
    Serial.println(i);
    delay(100);
  }
delay(5000);
}
```

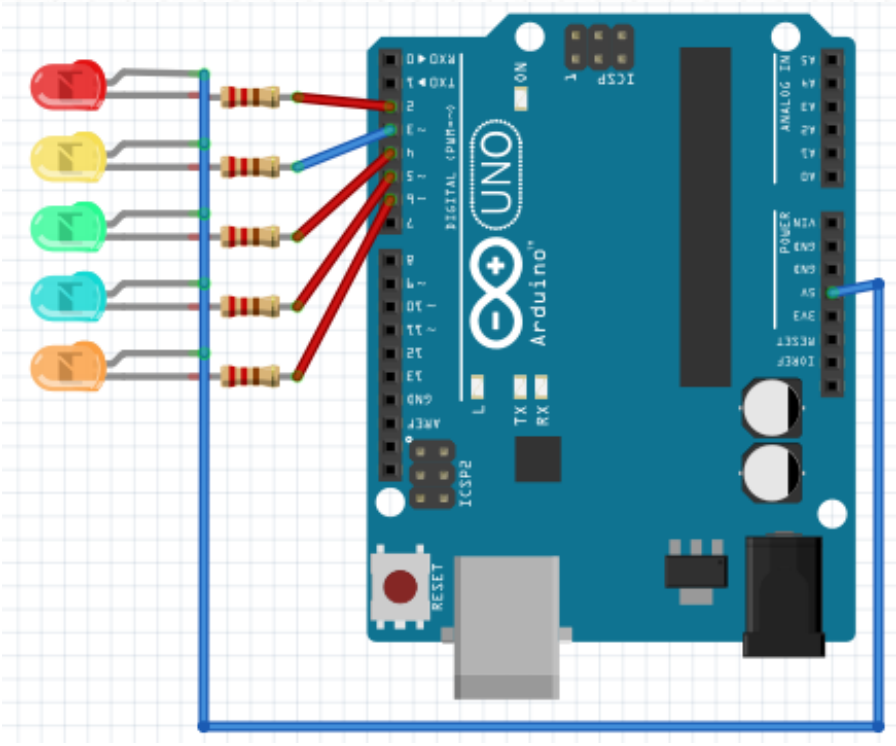
```
4.00
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
```

Цей скетч виводитиме в порт Arduino послідовність цифр від 2 до 4 включно з кроком 0.25. Не забуваємо оголосити змінну циклу як число з плаваючою комою – float!

Попрацюємо з елементами Arduino. Напишемо скетч та змонтуємо схему, яка буде автоматично перемикати зовнішні світлодіоди один за одним:

```
void setup() {
  Serial.begin(9600);
  for(int i=2;i<7;i++){
     pinMode(i, OUTPUT);//зробити піни 2..6 виходами
  }
}
void loop() {
  for(int i=2; i<7; i++){
      digitalWrite(i, HIGH);//увімкнути світлодіод
      delay(200);
  }
  delay(1000);
  for(int i=6; i>1; i--){
      digitalWrite(i, LOW);//вимкнути світлодіод
      delay(200);
  }
delay(1000);
}
```





Принципова схема та монтаж автоматичного перемикача світлодіодів

Цикл while

Цикл **while** зручно застосовувати у випадку, якщо в програмі необхідно повторювати дії, поки виконується якась умова. Заздалегідь ми не знаємо, коли вона перестане виконуватися й скільки разів виконається, відповідно, цикл.

Існує дві форми написання циклу **while**:

```
while (умова)
оператор_тіла_циклу;
```

```
do
оператор_тіла_циклу;
while(умова);
```

У першому випадку, спочатку проводиться перевірка умови, і якщо вона істинна – виконуються оператори, що складають тіло циклу. Цикл буде виконуватися, поки умова залишатиметься істинною. При першому ж порушенні істинності умови цикл припиниться. У такій формі запису циклу **while** можлива ситуація, коли тіло циклу взагалі не буде виконане: якщо умова при першій перевірці виявиться хибною.

У другій формі запису тіло циклу обов'язково буде виконане хоча б один раз, тому що умова перевіряється після першого виконання операторів тіла циклу.

Як у першій, так і в другій формі оператор тіла циклу може бути складеним:

```
while(умова)
{оператори}
```

```
do
{оператори}
while(умова);
```

Вдалий вибір виду циклу (**for, while** чи **do…while**) робить скетч зрозумілішим, а отже зменшує ймовірність помилки.

```
void setup() {
 Serial.begin(9600)
}
void loop() {
    while(1){
      if(digitalRead(3)) break;
    }
}
```

Наприклад: наведений код буде у нескінченому циклі чекати наявність сигналу на піні 3 з якогось датчика. Як тільки сигнал з’явиться, оператор **break** передасть управління наступному після **while** оператору.

Урок 13. Функції

Функція являє собою цілком самостійний блок програми. Як правило, вона одержує певні дані при виклику й повертає якесь значення. Щоб викликати функцію, потрібно вказати її ім'я і, за потреби, в дужках – вхідні дані, відокремлені комами. Це означає, що якщо в програмі зустрівся рядок

k = myMultiplyFunction(i, j);

то буде викликана функція **myMultiplyFunction** з відповідними параметрами (цілі числа **i** та **j**), а по завершенні роботи функції змінній **k** буде присвоєне значення, повернуте цією функцією.

Застосування функцій доцільне тоді, коли певна дія, або послідовність дій повинна повторюватися в різних частинах програми.

В Arduino, як ви вже знаєте існує дві обов'язкові функції **setup()** і **loop().** Всі інші функції можна створювати за межами цих функцій, надаючи їм власні імена. Наприклад, створимо функцію, що множитиме два числа:

```
void setup(){
Serial.begin(9600);
}
void loop(){
int i = 2; int j = 3; int k;
k = myMultiplyFunction(i, j); // k содержит 6
Serial.println(k);
delay(500) ;
}
int myMultiplyFunction(int x, int y){
int result;
result = x * y;
return result;
}
```

```
6
6
6
6
6
6
6
6
6
6
6
```

Як бачимо, крім двох обов'язкових функцій Arduino у нас створена функція **myMultiplyFunction. Int** перед назвою означає, що ця функція повертатиме у фрагмент коду, звідки її буде визвано результат дії функції

– число цілого типу. Зробить це оператор **return** (повертати). Значення у дужках після назви функції **(int x, int y)** – це аргументи, які надаються функції при зверненні до неї. У нашому прикладі звернення до функції виглядає так: **myMultiplyFunction(i,j)** і значення цієї функції буде присвоєно змінній цілого типу **k.** Ясна річ, що перед зверненням до функції необхідно надати якісь значення аргументам та повинна бути оголошена змінна, куди буде записано результат обчислення. Наприклад, в нашому скетчі це виглядає так:

int i=2; int j=3; int k;

У результаті при зверненні до нашої функції змінній цілого типу **x** буде присвоєно значення змінної **i** і відповідно змінній **y** – значення **j.** Після опрацювання функцією аргументів, оператор **return** поверне значення **result**, яке буде присвоєне змінній **k.**

Таким чином, якщо нам у нашому скетчі 10 разів треба буде виконати множення двох змінних, ми просто звертаємося 10 разів до нашої функції і получаємо результат.

І ще – функція може мати стільки аргументів, скільки вам потрібно і вони можуть бути різних типів:

float myFunc(float a, float x, float z);

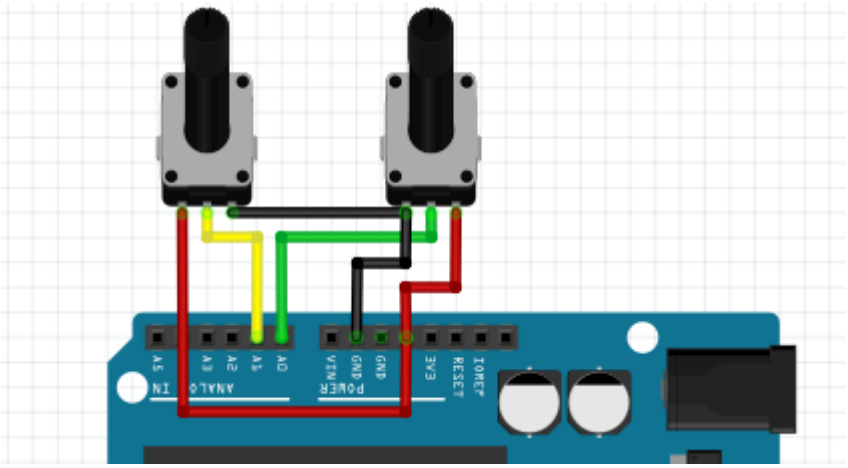
Функція може не повертати ніяких значень:

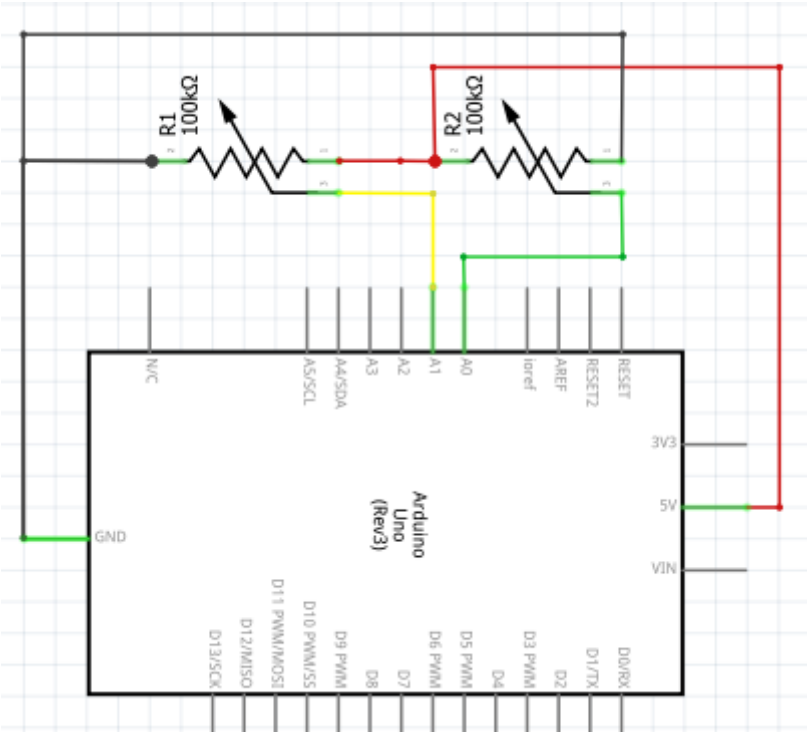
void myFunction(int k);

Функція може не мати аргументів:

bool myFun();

А тепер створимо такий собі обчислювач суми чисел на аналогових входах, що подаються з двох потенціометрів:






```
948 179
1127

948 179
1127

948 179
1127

948 180
1128
```

```
int potA, potB;
void setup() {
  Serial.begin(9600);
}
void loop() {
  potA = analogRead(0);
  potB = analogRead(1);
  myFunction(potA, potB);
}
void myFunction(int valA, int valB) {
  long SUM = valA + valB;
  Serial.print(valA); Serial.print(" "); Serial.print(valB);
  Serial.println(SUM); Serial.println();
  delay(100);
}
```

Тепер, якщо ми крутитимемо потенціометри, на аналогових входах A0 та A1 будуть з’являтися числа з діапазона 0…1023, їх значення будуть передаватися у функцію, яка відображатиме ці числа а також їх суму у вікні монітора послідовного порту. Дивись малюнок вище.

Урок 14. Масиви

Поняття масиву. Опис та ініціалізація масиву

Масив – це сукупність елементів одного типу, звернення до яких здійснюється за допомогою імені масиву та індексу (тобто порядкового номера елемента).

i	0	1	2	3	4
A[i]	-4	3	2	7	-39

Елементи масиву пронумеровані. Завдяки нумерації можна звернутися до будь-якого елемента масиву як до простого значення базового типу. У C++ нумерація елементів починається з нуля. У таблиці (див. малюнок) зображений масив цілих чисел, названий **A**. Цей масив містить 5 елементів. Таким чином, перший елемент масиву **A** указують як **A[0],** другий елемент – як **A[1],** п'ятий – як **A[4]** тощо. Тим самим – імена масивів повинні задовольняти тим самим вимогам, які ставляться до інших імен змінних. Отже, зображений масив міг бути заповнений у такий спосіб: **A[0]= -4; A[1]= 3; … A[4]= -39** Як ви вже знаєте, номер позиції, зазначений всередині квадратних дужок, називається індексом. Індекс повинен бути цілим додатним числом або математичним виразом, результатом обчислення якого є ціле додатне число. Якщо в якості індексу записаний математичний вираз, то вираз обчислюється з метою визначення індексу.

Кожен масив займає певну частину у пам'яті комп’ютера. Програміст повинен вказати тип елемента, кількість елементів, необхідних для кожного масиву, тоді компілятор зможе зарезервувати відповідний обсяг пам'яті. Наприклад, щоб компілятор зарезервував пам'ять для 5 елементів масиву цілих чисел **A**, можна використати таке оголошення: **int A[5];** Пам'ять для декількох масивів може бути зарезервована за допомогою одного оголошення. Наступне оголошення резервує пам'ять для 200 елементів масиву цілих чисел **b** й 15 елементів масиву цілих чисел y: **int b[200], y[15];** Масиви можуть складатися з елементів не лише типу **int**, але й інших типів даних. Але всі елементи одного масиву повинні бути одного типу! Наприклад, для зберігання рядка символів можна використати масив типу **char: char st[50];**

Приклади масивів:

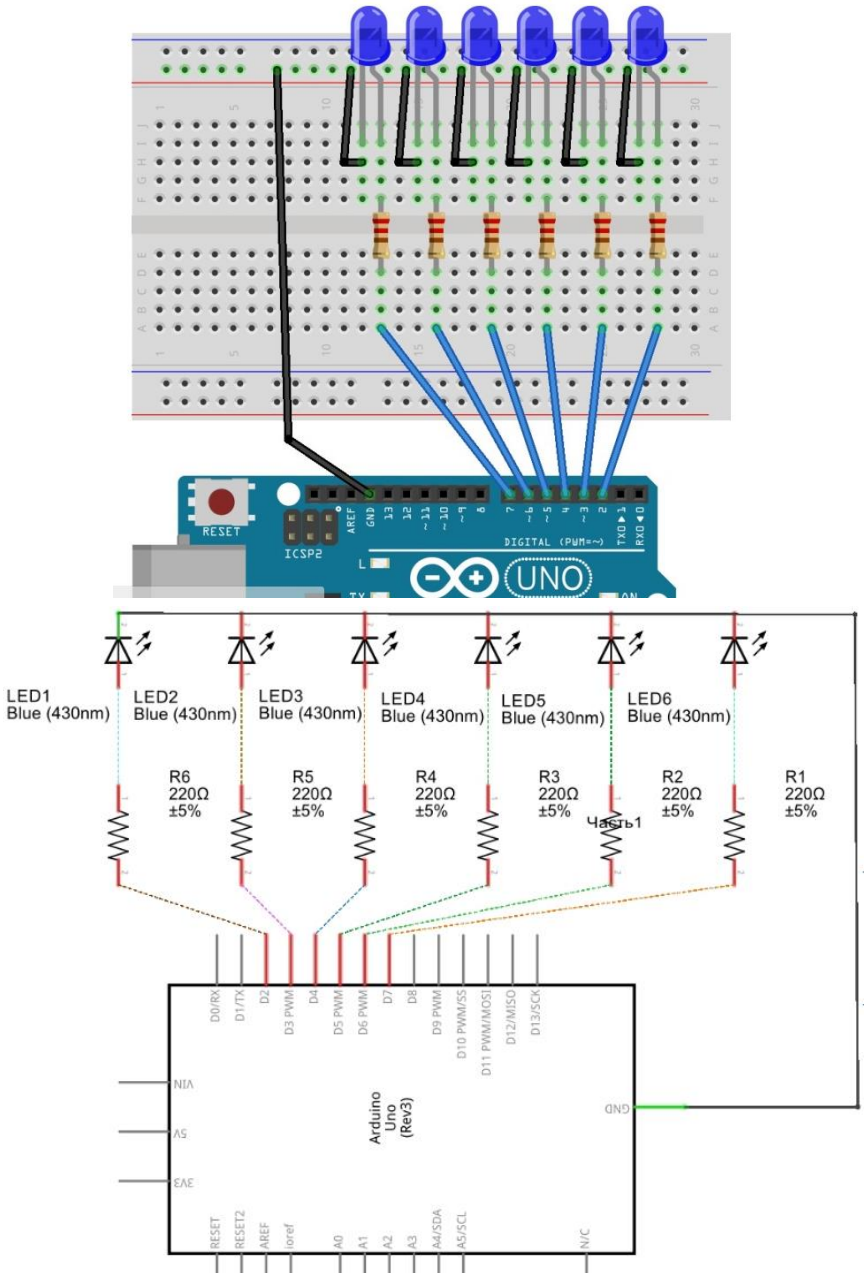
int myInts[6]; //оголошення масиву цілого типу розмірністю 6

int mySensVals[5] = {2, 4, 6, 3, 2}; //надання значень елементам масиву

int myPins[] = {3, 4, 8, 2, 6}; //Arduino сам порахує розмір масиву

char message[7] = "arduino"; //масив символів

Приклад 1: у цьому проекті ми, використовуючи масив для збереження номерів пінів, зможемо міняти послідовність вмикання світлодіодів, залежно від їх послідовності, прописаної в масиві. Змінюємо послідовність пінів у масиві – змінюється послідовність вмикання. Таким чином, відпадає необхідність фізично міняти підключення світлодіодів до відповідних пінів.



Скетч виглядатиме так:

```
/*Цей скетч використовує массив для зберігання даних про номери
пінів. Послідовно вмикає світлодіоди в різних комбінаціях.
Світлодіоди підключені до пінів 2-7 через резистори і до землі.*/
int timer = 500; // значення для таймеру.
int ledPins[] = {2, 7, 3, 6, 4, 5}; //масив з номерами пінів.
int pinCount = 6; // кількість пінів (розмір масиву)
void setup() {
  // елементи масиву нумеруються від 0 до (pinCount - 1).
  // використання циклу для ініціалізації пінів
  for (int Pin = 0; Pin < pinCount; Pin++) {
    pinMode(ledPins[Pin], OUTPUT);
  }
}
void loop() {
  //Цикл напрямку від 0 до (pinCount - 1)
  for (int Pin = 0; Pin < pinCount; Pin++) {
    // увімкнути світлодіоди:
    digitalWrite(ledPins[Pin], HIGH);
    delay(timer);
    // вимкнути світлодіоди:
    digitalWrite(ledPins[Pin], LOW);
  }
   //Цикл напрямку від (pinCount - 1) до 0
  for (int Pin = pinCount - 1; Pin >= 0; Pin--) {
    // увімкнути світлодіоди:
    digitalWrite(ledPins[Pin], HIGH);
    delay(timer);
    // вимкнути світлодіоди:
    digitalWrite(ledPins[Pin], LOW);
  }
}
```

Урок 15. Багатовимірні масиви

Масиви в C++ можуть бути також двовимірними, тривимірними й більше. Простими прикладами двовимірних масивів є сторінка класного журналу або таблиця EXCEL, що містять інформацію в рядках і стовпцях. Такий масив має два індекси: перший указує номер рядка, а другий – номер стовпця. На малюнку зображена схема нумерації елементів двовимірного масиву **a**, який містить три рядки й чотири стовпці.

стовпці (stovp)

рядки (ryad)

i \ j	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Взагалі, масиви з **ryad** рядками й **stovp** стовпцями називають масивами розміром **ryad** на **stovp** (наприклад, масив 3 на 4).

Кожен елемент у масиві **a** визначається ім'ям елемента у формі **a[i][j];** **a** – це ім'я масиву, а **i** та **j** – індекси, які однозначно визначають кожен елемент в **a**. Нумерація рядків і стовпців елементів масиву починається з нуля. Багатовимірні масиви можуть одержувати початкові значення у своїх оголошеннях так само, як масиви з єдиним індексом. Наприклад, двовимірний масив **c[3][2]** можна оголосити й дати йому початкові значення в такий спосіб:

```
int c[3][2] = {{4, 2}, {6, 7},{5, 8}};
```

Значення групуються в рядки, вміщені у фігурні дужки. Таким чином, елементи **c[0][0]** і **c[0][1]** одержують початкові значення 4 й 2, а елементи **c[1][0]** і **c[1][1]** одержують початкові значення 6 й 7 і т.д. Якщо початкових значень у даному рядку не вистачає для їхнього присвоєння всім елементам рядка, то елементам, що залишаються, присвоюються нульові початкові значення.

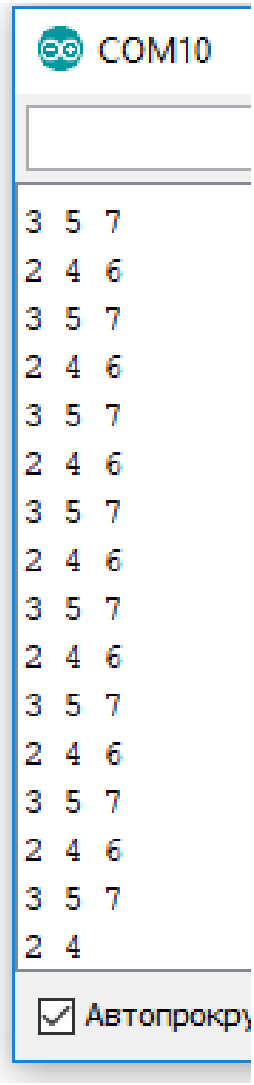
Таким чином, оголошення

int c[2][2] = {{10}, {9, 14}};

буде означати, що **c[0][0]** одержує початкове значення 10, **c[0][1]** одержує початкове значення 0, **c[1][0]** одержує початкове значення 9 і **c[1][1]** одержує початкове значення 14.

Приклад 2. Не будем змінювати принципову схему попередньої задачі, але спробуємо, використавши двовимірний масив, по черзі включати світлодіоди на парних та непарних пінах. Також виведемо значення масиву у монітор послідовного порту, для кращої наочності:

```
int A[2][3] = {{3, 5, 7},{2, 4, 6}};
void setup() {
  Serial.begin(9600);
  for (int i = 0; i < 2; i++)
    for(int j = 0; j < 3; j++)
    {
      pinMode(A[i][j],OUTPUT);
    }
}
void loop() {
  for (int i = 0; i < 2; i++)
  {
    for(int j = 0; j < 3; j++)
    {
      Serial.print(A[i][j]); Serial.print(" ");
      digitalWrite(A[i][j], HIGH);
      delay(2000);
      digitalWrite(A[i][j], LOW);
    }
   Serial.println(" ");
  }
}
```



Для опрацювання багатовимірних масивів використовуємо вкладені цикли.

Для того, щоб зручно користуватися масивами нам потрібно завжди точно обчислювати їх розміри. Для цього розглянемо ще одну із функцій мови C++ – **sizeof.**

Функція визначення розміру **sizeof** використовується для обчислення розміру значення виразу чи типу в байтах і має дві форми:

sizeof (вираз**)** **sizeof (**тип**)**

Нехай заданий масив **A**, елементи якого нумеруються від **0** до **n**. Отже, щоб знайти довжину масиву **A**, достатньо виконати таку операцію:

n = sizeof(A)/sizeof(int);

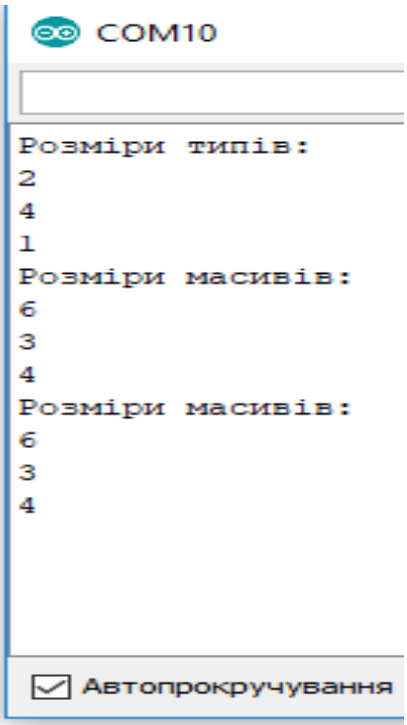
Завдяки використанню функції **sizeof** буде правильно опрацьований масив будь-якого розміру. Можна написати також:

n = sizeof(A)/sizeof(A[0]);

– тоді робота цього оператора не залежатиме навіть від типу елементів масиву.

Наведемо приклад скетчу, що рахуватиме розміри масивів різного типу описаними нами способами:

```
int A[] = {1,2,3,7,8,9};
float B[]={4.5,3.3,5.7};
char C[]={'a','b','c','d'};
void setup() {
  Serial.begin(9600);
  //Обчислюємо розміри типів!
  int a=sizeof(int);//розмір int
  int b=sizeof(float);//розмір float
  int c=sizeof(char);//розмір char
  Serial.println("Розміри типів:");
  Serial.println(a);
  Serial.println(b);
  Serial.println(c);
  //Обчислюємо розміри масивів!
  int a1=sizeof(A)/sizeof(int);
  int b1=sizeof(B)/sizeof(float);
  int c1=sizeof(C)/sizeof(char);
  Serial.println("Розміри масивів:");
  Serial.println(a1);
  Serial.println(b1);
  Serial.println(c1);
  // А щоб не заморочуватися типами
  // масиву можна зробити навіть так!!!
  int a2=sizeof(A)/sizeof A[0];
  int b2=sizeof(B)/sizeof B[0];
  int c2=sizeof(C)/sizeof C[0];
  Serial.println("Розміри масивів:");
  Serial.println(a1);
  Serial.println(b1);
  Serial.println(c1);
}
void loop() {
}
```



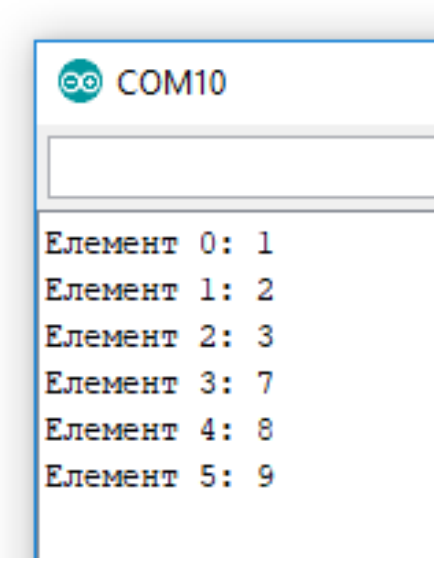
Як передати масив у функцію, роздивимся далі.

Урок 16. Передача масивів у функції

Для зменшення коду і зручності написання скетчів, як ми вже знаємо використовують функції. На цьому уроці роздивимося, як можна передавати масиви у функції.

Роздивимся приклад, коли нам потрібно вивести значення масиву у монітор послідовного виводу. Використавши знання попереднього уроку це не важко зробити так:

```
int A[] = {1,2,3,7,8,9};
void setup() {
  Serial.begin(9600);
  int k = sizeof(A)/sizeof(A[0]);//знаходимо розмір масиву
  printArray(A,k);//передаємо у функцію аргументи
}
void loop() {
}
//Функція друку значень масиву
void printArray(int arr[],int n) {
  for(int i=0; i<n;i++){
    Serial.print("Елемент ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println(arr[i]);
  }
}
```



Таким чином, ми легко організували виведення даних масиву в монітор послідовного порту Arduino, при цьому розмір масиву ми можемо і не знати. Програма автоматично порахує відповідні дані. Таким чином, ми можемо обробляти будь-яку кількість даних, наприклад, про температуру з відповідного датчика, або інших даних, що поступають ззовні на Arduino.

Заключення:

За допомогою масивів Arduino може оперувати великими наборами даних, наприклад, збирати інформацію з багатьох підключених датчиків, або генерувати сигнали для декількох підключених світлодіодів. Масиви використовуються для зберігання рядків і достатньо складних організованих структур даних. Використання масивів в Arduino таке саме, як у C++ і вимагає розуміння принципів адресної арифметики, постійного

контроля програміста над розмірністю і типами. Але завдяки цьому масиви надають гнучкі способи зберігання і динамічного змінювання наборів даних. Без них, написати досить складні програми, практично неможливо.

Для розширення знань про використання масивів, вказівників динамічних масивів, а також для вивчення інших тонкощів мови програмування С++ ви можете продовжити навчання за моєю книгою:

Інформатика. Мова програмування С++. Спецкурс. 10-12 класи. Навчальний посібник / Лєхан С.А. – Шепетівка, «Аспект», 2007

Урок 17. Випадкові числа

В Arduino реалізована функція для генерації псевдовипадкових чисел. Для цього використовується дві функції:

random() і randomSeed().

Функция **randomSeed()** дозволяє ініціалізувати генератор псевдовипадкових чисел.

Функция **random()** дозволяє генерувати псевдовипадкові числа з вказаного діапазона:

```
random(x);      // діапазон 0 ...(x — 1)
random(a, b);   // діапазон a ... (b — 1)
```

Вказавши один параметр **x**, ми генеруємо числа від нуля до числа на **один** менше ніж **x**. Для двох параметрів діапазон починається від першого параметра **a**, а закінчується значенням на **один** менше від другого параметра **b**.

У прикладі нижче генератор ініціюється з використанням значення, зчитаного з аналогового входу. Перша пара випадкових чисел знаходиться в діапазоні 4...9, друга — в діапазоні 0…199.

Скетч виглядає так:

```
long randNumber;
void setup()
{
Serial.begin(9600);
randomSeed(analogRead(0));
}
void loop()
{
randNumber = random(4, 10);
Serial.print(randNumber);
Serial.print("    ");
randNumber = random(200);
Serial.println(randNumber);
delay(300);
}
```

```
COM10

6     106
5     77
8     121
6     162
4     174
5     180
5     125
5     99
6     2
5     141
4     2
7     62
7     178
4     196
9     124
```

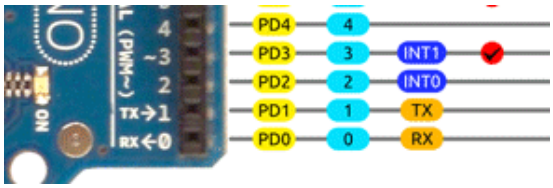
Приклад генерації випадкових наборів символів:

```
void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop() {
 for (int i=0; i< random(3, 12); i++) {
  Serial.write(random(97,122));
 }
Serial.println();
delay(500);
}
```

```
jtc
ndxrgha
siy
ardomiy
uwwlvcno
hfjockp
mgrlxi
yrvswg
avnyry
wxgasc
txhfk
wbvjgdp
```

Знаючи все вищесказане, ви легко створите скетч, де випадковим чином буде змінюватися яскравість світлодіода, швидкість руху сервопривода, зможете вибрати з масиву чисел чи текстових величин випадкове число або слово для тесту або гри.

Урок 18. Апаратні переривання



На цьому уроці ми ознайомимося з такою важливою функцією Arduino як переривання. У Arduino UNO і NANO є два переривання і виведені вони на піни 2 і 3 (Дивись малюнок) – **INT0** і **INT1**. В інших моделях може бути по іншому – дивіться специфікацію. Ці піни можуть оброблятися обробником переривань, який перевіряє наявність логічного нуля або одиниці на них та дає сигнал процесору на виконання якихось дій. На відміну від **digitalRead**, який прописаний у коді і залежить від наявності затримок **loop** та інших, обробник переривань опитує пін постійно, незалежно від того, яка частина коду виконується і від наявності затримок або складних обчислень, що потребують витрат часу у коді. Таким чином, ми можемо відслідкувати натискання на кнопку в любому місці коду, не звертаючись до піну зовсім. Наприклад, у нас написаний складний код обчислень з великою кількістю затримок, а нам потрібно постійно знімати покази з датчика Хола або тахометра – тут без переривань не обійтися!

Обробка сигналу переривання проходить миттєво, швидкість зчитування близько 4 мікросекунди.

Ще один плюс переривань – вони здатні вивести Arduino з режиму сну.

Підведемо підсумок: переривання використовують для опитування кнопок і датчиків постійно з великою швидкістю, незалежно від виконуваного коду, а також для виведення системи Arduino з режиму сну.

Для використання переривання, його необхідно підключити і настроїти. Для цього є команда **attachInterrupt**, де в дужках вказують номер переривання, ім’я функції що визивається та режим роботи. Номери переривань починаються з 0 і не співпадають з номерами цифрових пінів (Малюнок вище). Підключимо кнопку до переривання 1 (пін 3). Одна ніжка кнопки на землю (**GND**), інша через внутрішній підтягуючий резистор (**INPUT_PULLUP**) на пін 3.

Настроюємо переривання за допомогою функції **Interrupt**. Спочатку вказуємо номер переривання, у нас це 1 (не плутати з номером піну 3!!!). Далі вказуємо ім’я функції що буде визиватися обробником. Цю функцію необхідно створити. Наприклад, назвемо її **myInterrupt**. Назва функції

вписується як другий параметр підключення переривання без дужок. Третій параметр – режим роботи. Їх може бути чотири:

- **LOW** – викликає переривання, коли на порту **LOW.**
- **CHANGE** – викликає переривання, коли значення на порту змінюється від **LOW** до **HIGH** високого і навпаки.
- **RISING** – викликає переривання, коли значення на порту змінюється з **LOW** на **HIGH**.
- **FALLING** – викликає переривання, коли значення на порту змінюється з **HIGH** на **LOW.**

Третій і четвертий режими дуже підходять для кнопки. Як тільки ми натиснемо на кнопку, на піні буде встановлено нуль – замикання на землю. Це означає, що при натисканні маємо падіння логічного сигналу і в настройці переривання пишемо **FALLING**, якщо бажаємо, щоб переривання спрацювало при натисканні кнопки.

Як це працює: при спрацюванні переривання виконується набір команд, що входять у створену нами функцію, при чому зовсім неважливо, яка частина коду виконується в цей час. Виконання коду зупиняється і виконується набор функцій обробника переривань, в нашому випадку **myInterrupt**, а потім знову продовжується виконання основного коду, одразу після того місця, де було переривання.

Сама функція, що виконується в момент спрацювання переривання (у нас це **myInterrupt**) має деякі обмеження: по-перше, в ній не працює затримка **delay**, а також не змінює значення результат функції **millis**. Він завжди буде таким, яким був під час визивання нашої функції. Таким чином, якщо нам потрібно узнати час – це можна зробити лише один раз під час виклику. Для спрощення можна уявити, що блок функції виконується миттєво, навіть час не встигає пройти. Ну і ще – щоб змінити значення будь-якої змінної, вона повинна мати атрибут **volatile**, інакше робота буде некоректною. Наприклад, під час кожного переривання ми хочемо збільшити значення лічильника і використовувати його значення в основному циклі програми. Для цього змінна лічильника повинна мати атрибут **volatile**. Наприклад: змінна зберігає число натискань. При визові переривання збільшується на одиницю. Потім вона виводиться в

основному циклі програми кожні пів секунди. Можна поставити delay, щоб показати що переривання працює завжди:

```
volatile byte count;
void setup() {
  Serial.begin(9600);
  pinMode(3,INPUT_PULLUP);
  attachInterrupt(1,myInterrupt, FALLING);
}
void loop() {
  Serial.println(count);
  delay(500);
}
void myInterrupt(){
  count++;
}
```

```
0
1
3
5
5
5
5
5
5
5
5
```

І ще: якщо виводити щось в порт функції опрацювання переривання, частина символів може загубитися, у зв'язку з тим, що програма дуже швидко працює.

Для відключення обробника переривань використовуємо функцію **detachInterrupt(pin)**.

Розберемо приклад з обробкою переривань і прапорцем:

```
volatile boolean flag;
void setup() {
  Serial.begin(9600);
  pinMode(3,INPUT_PULLUP);
  attachInterrupt(1,myInterrupt, FALLING);
}
void loop() {
  if(flag){
  Serial.println("ПЕРЕРИВАННЯ!!!");
  flag = 0;
  }
}
void myInterrupt(){
  flag = 1;
}
```

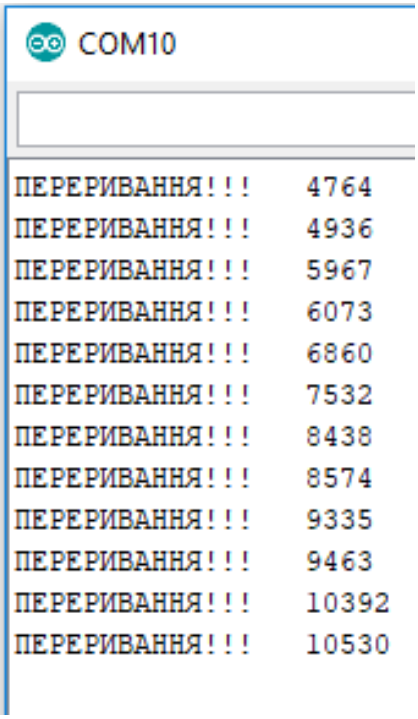
```
COM10

ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
ПЕРЕРИВАННЯ!!!
```

Створимо прапорець з атрибутом **volatile** , по замовчанню він дорівнює нулю. Під час обробки переривання піднімемо прапорець. В основному

циклі програми, якщо прапорець піднято, посилаємо в порт рядок, а прапорець опускаємо. Таким чином, можна передати в основний код факт спрацювання переривання.

Також можна запам`ятати час спрацювання і також його виводити:

```
volatile boolean flag;
volatile unsigned long Timer;
void setup() {
  Serial.begin(9600);
  pinMode(3,INPUT_PULLUP);
  attachInterrupt(1,myInterrupt, FALLING);
}
void loop() {
  if(flag){
  Serial.print("ПЕРЕРИВАННЯ!!!    ");
  Serial.println(Timer);
  flag = 0;
  }
}
void myInterrupt(){
  flag = 1;
  Timer = millis();
}
```



Якщо потрібно на деякий час відключити переривання існує функція **noInterrupts**, яка призупиняє обробку переривання і є функція **Interrupts**, яка включає переривання знову.

attachInterrupt(pin, function, state); – підключити переривання.

detachInterrupt(pin); – вимкнути переривання.

pin – пін переривання, для NANO і UNO це піни D2 і D3, відповідають номерам 0 і 1.

function – назва функції, яка буде визиватися після того, як спрацює переривання.

state режим обробки, їх кілька:

- **LOW** викликає переривання, коли на порту **LOW.**
- **CHANGE** викликає переривання, коли значення на порту змінюється від **LOW** низького до **HIGH** високого і навпаки.
- **RISING** – викликає переривання, коли значення на порту змінюється з **LOW** на **HIGH**.
- **FALLING** – викликає переривання, коли значення на порту змінюється з **HIGH** на **LOW.**

Приклад:

Ця кнопка підключена до D2 і GND

```
void setup() {
pinMode(2, INPUT_PULLUP); // пін D2 подтягнуто до живлення
attachInterrupt(0, myInterrupt, FALLING);
}
void myInterrupt() { // функція обрабника переривання
Serial.println("INTERRUPT!"); // при спрацюванні вивести в порт INTERRUPT
}
```

Урок 19. Використання бібліотек. Підключення сервоприводу

Сервоприводи використовують у всіх роботизованих пристроях та системах керування, таких як: управління по радіо різними моделями, приведення у дію виконуючих механізмів пристроїв та багатьох інших.

Сервопривід має двигун з редуктором[5] та потенціометр, який контролює кут відхилення валу сервоприводу.

Підключаємо сервопривід за допомогою трьох проводів, два з яких підключені до блоку живлення: GND та +5В, а один – для керування сервоприводом. Провід керування підключається до пінів, які дозволяють використовувати ШІМ – широтно імпульсну модуляцію (див. Урок 13).

Але щоб не витрачати багато часу на написання коду, існує бібліотека, завдяки якій управління сервоприводом здійснюється дуже просто!

Бібліотека – це програмний код для скетча, що зберігається у зовнішньому файлі, і який підключаєтеся до проекту. Всі бібліотеки Arduino ділять на стандартні — це бібліотеки, що не потребують встановлення, вони вже вмонтовані в середовище Arduino IDE і додаткові бібліотеки. Додаткові бібліотеки розробляє виробник датчиків і модулів. Вам потрібно завантажити і встановити їх. Для цього скачуємо відповідну бібліотеку собі в комп'ютер (звичайно це ZIP-архів), потім в меню *Скетч/Додати бібліотеку* вибираємо: *Додати ZIP бібліотеку*. Є сотні готових бібліотек для модулів в Інтернеті. Також ви можете написати бібліотеку самостійно.

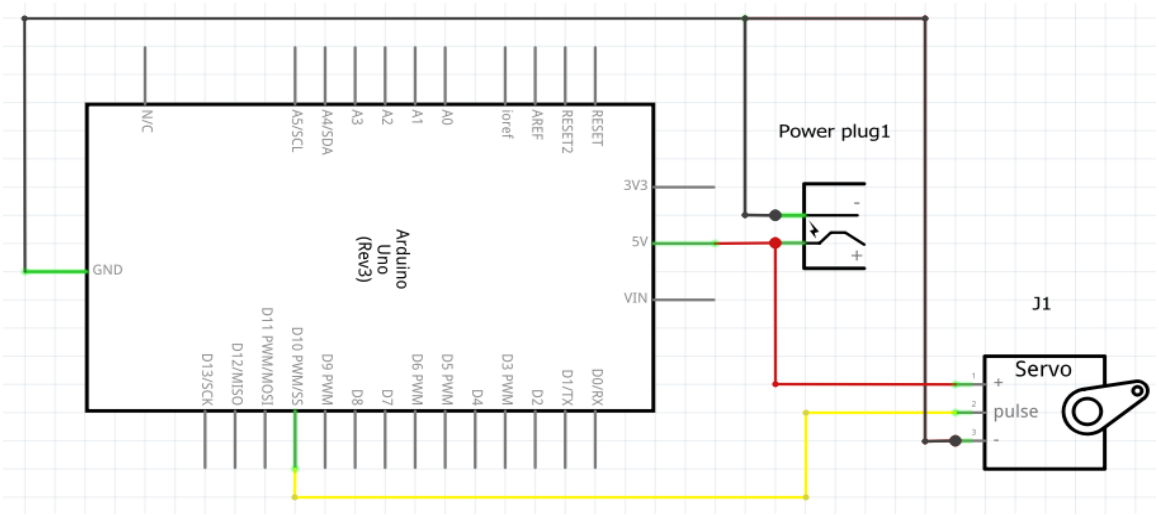
[5] Механізм, що зменшує швидкість руху двигуна

Для підключення бібліотек використовують директиву **#include**. Якщо бібліотека вже є, її можна підключити, вибравши в меню: *Скетч/Додати бібліотеку*, і вибрати в меню, що відкрилася необхідна бібліотека. У нашому випадку це **Servo**. Потрібний рядок сам з'явиться у вашому коді в потрібному місці:

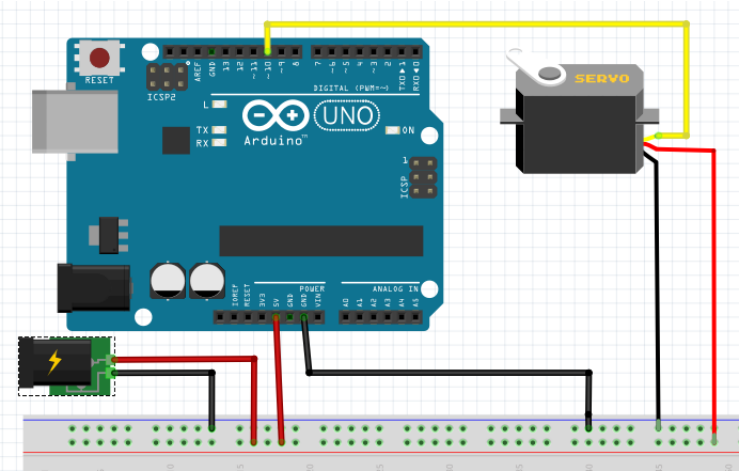
#include<Servo.h>

А тепер підключимо сервопривод до нашої Arduino. Краще для цього використовувати Breadboard[6], і підключити окремий блок живлення для сервоприводу (ви пам'ятаєте – піни Arduino не слід використовувати для живлення підключених пристроїв, а тільки для керування ними).

Принципова схема підключення виглядатиме ось так:



Зібраний макет за допомогою Breadboard виглядатиме так:



Підключимо окремі блоки живлення до Arduino та до Breadboard.

А тепер напишемо скетч, використавши підключену бібліотеку. У скетчі тепер ми можемо створювати змінні типу **Servo**, наприклад, як у

[6] Макетна плата для монтажу без пайки

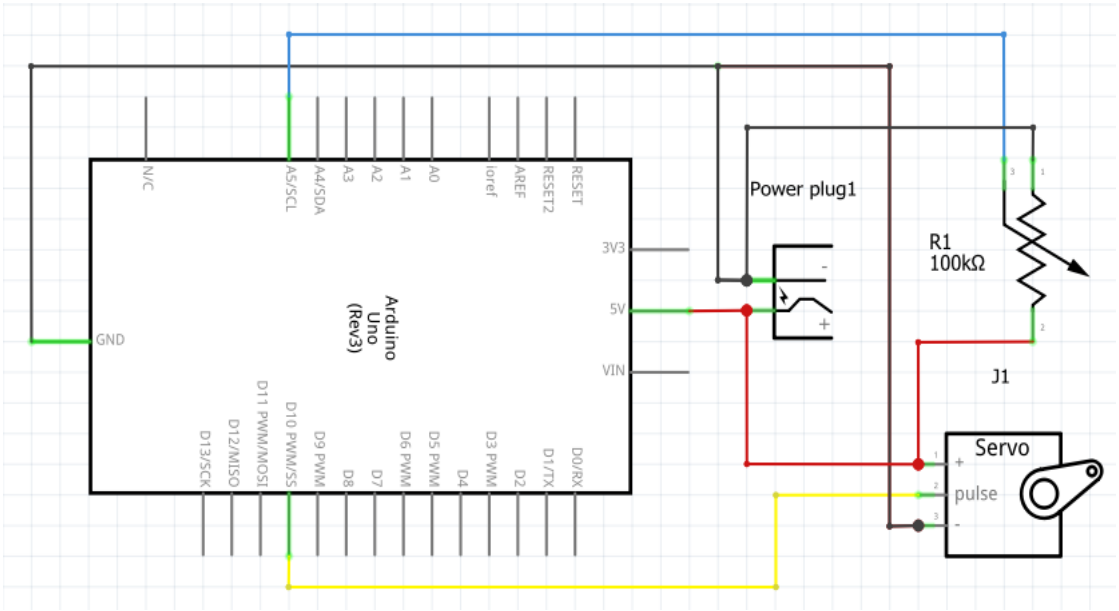
нас: **servo1** і потім використовувати для них різні методи об'єкту **Servo**. Наприклад, **servo1.attach(10)** прив’язує керуючий провід до піну 10 з ШІМ, а **servo1.write(kut)** встановлює значення кута повороту для валу сервоприводу від 0 до 180 градусів:

```
#include <Servo.h> //Підключаємо бібліотеку для роботи з сервоприводом
Servo servo1; //оголошуємо змінну servo1 типу Servo
void setup()
{
servo1.attach(10); //прив'язуємо керування до порту 10
}
void loop()
{
servo1.write(0); //встановлємо кут вала на 0
delay(2000); //затримка 2 сек
servo1.write(180); //встановлємо кут вала на 180
delay(2000); //затримка 2 сек
}
```

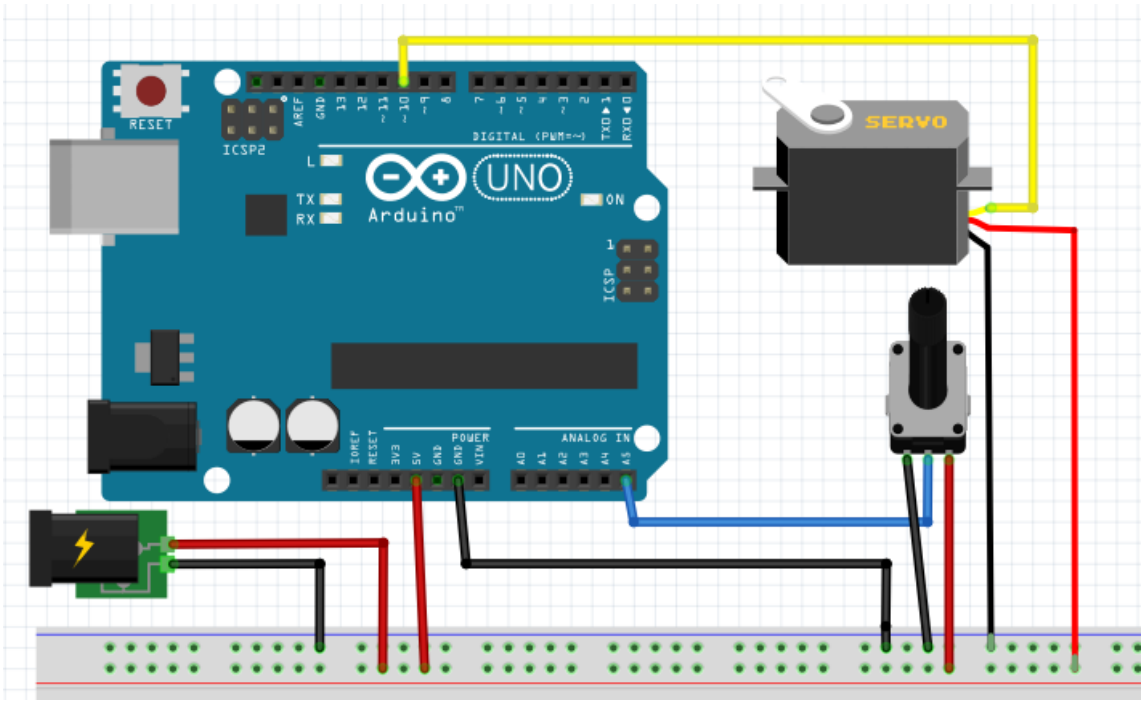
А тепер використаємо ШІМ модуляцію для керування кутом повороту сервоприводу, для цього існує другий варіант управління і метод **writeMicroseconds(a).** Значення параметру **a** можна знайти у специфікації до відповідного сервоприводу. У нашому випадку для стандартного сервоприводу з набору Arduino значення параметру **a** для кута повороту у 0 градусів буде 550, а для значення 180 градусів, відповідно 2350. Таким чином, нижче приведений скетч буде працювати абсолютно ідентично попередньому:

```
#include <Servo.h> //Підключаємо бібліотеку для роботи з сервоприводом
Servo servo1; //оголошуємо змінну servo1 типу Servo
void setup()
{
servo1.attach(10); //прив'язуємо керування до порту 10
}
void loop()
{
servo1.writeMicroseconds(550); //встановлємо кут вала на 0
delay(2000); //затримка 2 сек
servo1.write(2350); //встановлємо кут вала на 180
delay(2000); //затримка 2 сек
}
```

Створимо схему для керування сервоприводом з зовнішнього потенціометра. Доопрацюємо нашу схему, підключивши потенціометр. Як ви вже знаєте – крайні ніжки підключаються до землі GND та +5В а середня буде керувати сервоприводом через наприклад 5-й аналоговий порт:



Макетна схема нашого пристрою:



Скетч виглядатиме так:

```
#include <Servo.h> //Підключаємо бібліотеку для роботи з сервоприводом
Servo servo1; //оголошуємо змінну servo1 типу Servo
void setup()
{
servo1.attach(10); //прив'язуємо керування до порту 10
}
void loop()
{
int pot = analogRead(5);//підключаємо середню ніжку потенціометра на A5
pot = map(pot, 0, 1024, 0, 180);
servo1.write(pot);
delay(2);
}
```

Повертаючи потенціометр, ми будемо керувати кутом повороту сервоприводу.

У наступному прикладі ми підключимо два сервоприводи до нашої Arduino і вони будуть синхронно працювати від потенціометра. Гадаю, складності для самостійного збирання схеми у вас не буде. Головне, щоб зовнішній блок живлення забезпечив потужність всіх під’єднаних пристроїв.

Скетч до схеми керування двома сервоприводами:

```
#include <Servo.h> //Підключаємо бібліотеку для роботи з сервоприводом
Servo servo1; //оголошуємо змінну servo1 типу Servo
Servo servo2;
void setup()
{
servo1.attach(10); //прив'язуємо керування до порту 10
servo2.attach(11); //прив'язуємо керування до порту 11
}
void loop()
{
int pot = analogRead(5);//підключаємо середню ніжку потенціометра на A5
pot = map(pot, 0, 1024, 0, 180);
servo1.write(pot);
servo2.write(pot);
}
```

Тепер ваш багаж знань достатній, щоб самостійно розібратися в безлічі скетчів, що надаються до проектів Arduino. А також ви самі зможете створити скетчі для своїх розробок. Бажаю успіху!

ВИСНОВКИ

Протягом найближчих 5 до 10 років, Arduino буде використовуватися в кожній школі в галузі електроніки, фізики та інформатики.

Плюси Arduino:

Arduino IDE працює під Mac OS, Linux і Windows.

Arduino — це дешево.

Arduino — це Open Source.

Arduino — це просто, але не дуже легко ;)

Arduino є ідеальною платформою для початківців.

До Arduino можна підключати безліч різних датчиків, АЦП дозволяють отримувати аналогові дані (наприклад, датчик температури), а вбудовані інтерфейси SPI і i2c дозволяють працювати з майже всіма видами датчиків.

Для Arduino доступні багато різних бібліотек, як для складних завдань (робота з SD-картками, LCD, парсінг GPS-даних), так і для простих проблем, як, наприклад, прибирання «брязкіта» контактів.

Головні особливості Arduino — простота, відкритість і швидке освоєння! Всього 10 хвилин на ознайомлення і ви вже починаєте програмувати!

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://www.arduino.cc/
2. http://arduino.ru/
3. https://m.habr.com/post/397019/
4. http://edurobots.ru/kurs-arduino-dlya-nachinayushhix/
5. Інформатика. Мова програмування С++. Спецкурс. 10-12 класи. Навчальний посібник / Лєхан С.А. – Шепетівка, «Аспект», 2007 – 160 с.