

Arduino Microcontroller Guide

W. Durfee, University of Minnesota

Available on-line at www.me.umn.edu/courses/me2011/arduino/

ver. oct-2011

1 Введення

1.1 Короткий огляд

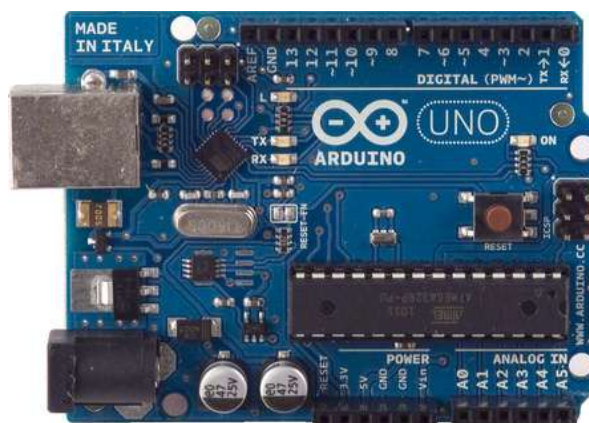
Мікроконтролер на платформі Arduino – це простий у використанні, але потужний одноплатний комп'ютер, який здобув значне визнання як для хобі, так і на професійному ринку. Arduino є відкритим вихідним кодом, що означає: апаратні засоби за розумними цінами і безкоштовна розробка програмного забезпечення. Це керівництво призначене для студентів в ME 2011 або для студентів, які стикаються з Arduino вперше. Для досвідчених користувачів Arduino є багато ресурсів у Інтернеті.

Проект Arduino було розпочато в Італії як розробку недорогого обладнання для конструювання дизайну. Про Arduino можна прочитати у Вікіпедії. Домашня сторінка Arduino є <http://www.arduino.cc/>.

Апаратна Arduino поставляється в декількох різновидах. У Сполучених Штатах SparkFun (www.sparkfun.com) є хорошим кодом (джерелом) для обладнання Arduino.

Це керівництво охоплює плату Arduino Uno (SparkFun DEV-09950, \$ 29.95), яка є вдалим вибором для студентів і викладачів. Із платою Arduino ви можете без особливих зусиль писати програми, створити інтерфейс для читання схеми перемикачів та інших датчиків, для контролю двигунів і ліхтарів. Багато малюнків і схем були взяті для цього посібника із документації на сайті Arduino, куди ви можете звернутися, якщо вам потрібна додаткова інформація. У розділі Arduino на веб-сайті ME 2011 <https://sites.google.com/a/umn.edu/me2011/>, охоплює більше інтерфейсу Arduino в реальному світі.

Такий вигляд має плата Arduino.



Плата Arduino Duemilanove має мікроконтролер Atmel ATmega328, який працює на 5 В із 2 Кб ОЗУ, 32 Кб флеш-пам'яті для зберігання програм і 1 Кбайт EEPROM для зберігання параметрів. Тактова частота 16 МГц дозволяє виконувати близько 300000 рядків вихідного коду на С за секунду. Плата має 14 цифрових каналів вводу/виводу і 6 контактів аналогового вводу. Існують роз'єм USB для з'єднання із комп'ютером і роз'єм для живлення постійним струмом при підключенні зовнішнього джерела живлення 6-20 В, наприклад батареї 9 В, при запуску програми без підключення до комп'ютера. Накінечники призначені для сполучення з виводами входу/виходу з використанням 22 г одножильних проводів або штирьових роз'ємів. Для отримання додаткової інформації про обладнання див. <http://arduino.cc/en/Main/ArduinoBoardUno>.

Мова програмування Arduino – це спрощена версія C/C++. Якщо ви знаєте С, то програмування в Arduino буде вам рідним. Якщо ви не знаєте С, то не слід турбуватися, бо лише декілька команд необхідні для виконання основних функцій.

Важлива особливість Arduino така: ви можете створити програму управління на ПК, завантажити її в Arduino, і вона працюватиме автоматично. Якщо ви від'єднаєте USB-кабель від ПК, то програма однак працюватиме до тих пір, доки ви не натиснете кнопку скидання. Вийміть акумулятор і поставте плату Arduino в шафу на півроку. Коли ви підключите батарею, то остання програма, яку ви зберегли, буде працювати. Це означає, що ви підключаєте плату до ПК лише для того, щоб розробити і налагодити вашу програму. А як тільки це буде зроблено, вам більше не потрібний ПК, щоб запустити програму.

1.2 Що потрібно для роботи системи

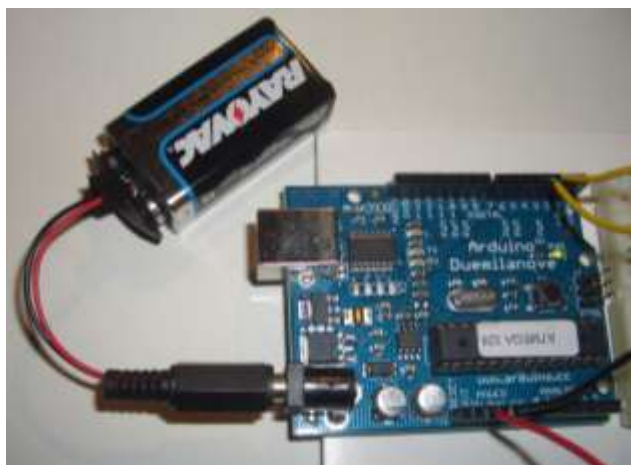
1. Плата Arduino Duemilanove.
2. USB-кабель для програмування (А до В).
3. 9 В батареї або зовнішнє джерело живлення (для автономної роботи)
4. Накінечники для конструювання зовнішніх ланцюгів, 22 г одножильного дроту для з'єднань.
5. Головний ПК працює в середовищі розробки Arduino. Існують версії для ОС Windows, Mac і Linux.

1.3 Встановлення програмного забезпечення

Дотримуйтесь інструкцій у розділі Getting Started веб-сайту Arduino - <http://arduino.cc/en/Guide/HomePage>. Ідіть крок за кроком, доки ви не побачите, що у штифті 13 світлодіод блимає. Це означає, що у вас є все програмне забезпечення, драйвери успішно встановлені і можуть почати вивчати ваші власні програми.

1.4 Підключення акумулятора

Для автономної роботи плата живиться від акумулятора, а не через USB-підключення до комп'ютера. У той час як зовнішнє джерело може бути де завгодно в діапазоні від 6 до 24 В (наприклад, ви могли б використовувати автомобільний акумулятор), то стандартна батарея 9 В зручніша. Щоб не застрягли дроти батареї кріплення в гніздах Vin і Gnd на платі, краще припаяти оснастки батареї до вилки живлення постійного струму і підключити до гнізда живлення на платі. Відповідний плагін є частиною номер 28760 від www.jameco.com. Ось як це виглядає:



Увага: Слідкуйте за полярністю при підключенні батареї до оснащення, адже зворотня орієнтація може підірвати вашу плату.

Вимкніть Arduino від комп'ютера. Підключіть батарею 9 В до гнізда живлення Arduino за допомогою адаптера батареї прив'язки. Переконайтеся, що програма працює - блимає. Це показує, що ви можете жити Arduino від батареї і що програма завантаження працює без необхідності підключення до вашого ПК.

1.5 Moving On (переміщення по)

Підключіть Arduino до комп'ютера за допомогою кабелю USB. У цей час не потрібна батарея. Зелений світлодіод PWR загориться. Якщо раніше запрограмували Arduino, то він буде працювати.

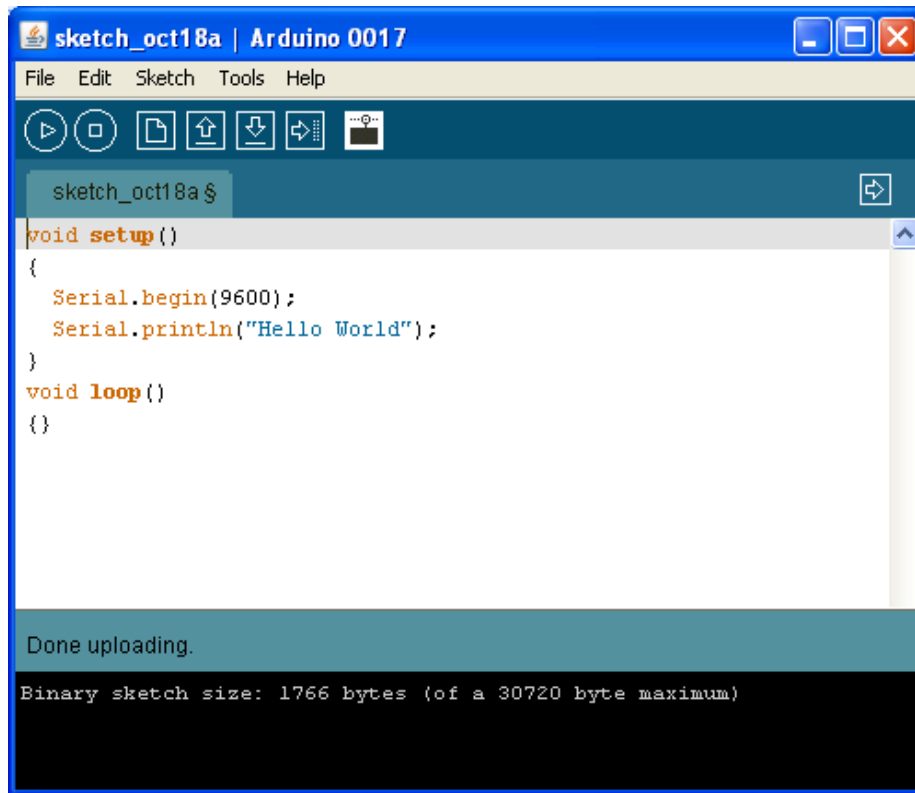
Увага: Не кладіть низ плати на струмопровідну поверхню: буде коротке замикання. Покладіть плату на підложку!


Стартуйте в середовище розробки Arduino. У Arduino кажуть: програми називаються "ескізи", але тут ми будемо називати їх просто програми.


У вікні редагування, що з'явиться, введіть наступну програму, звертаючи увагу, де крапка з комою з'являється в кінці командного рядка.

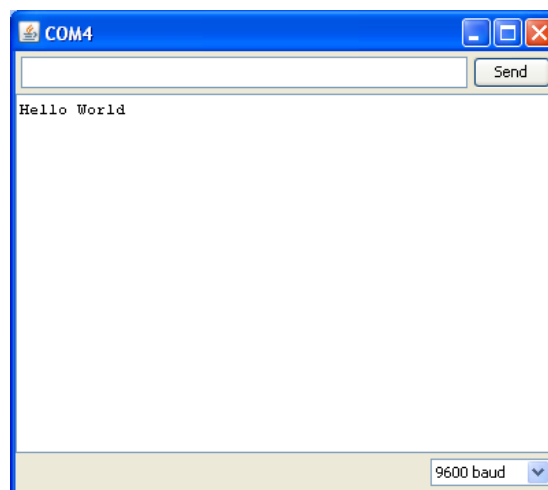
```
void setup()
{
  Serial.begin(9600);
  Serial.println("Hello World");
}
void loop()
{}
```

Ваше вікно буде виглядати приблизно так:




Натисніть кнопку Завантажити  або Ctrl-U, щоб скомпілювати програму і завантажити на плату Arduino.

Натисніть кнопку Serial Monitor.  . Якщо все пройшло добре, вікно монітора покаже повідомлення на зразок цього:



Вітаємо! Ви створили і запустили першу програму Arduino!

Натисніть кнопку reset (скидання) в Arduino кілька разів і подивіться, що відбувається.

Підказка: Якщо ви хочете перевірити синтаксис коду без підключення плати Arduino, натисніть кнопку Verify  або Ctrl-R.

Підказка: Якщо ви хочете побачити, скільки програма займає пам'яті, підтвердьте Verify, а потім подивіться на повідомлення в нижній частині вікна програмування.

1.6 Вирішення проблем

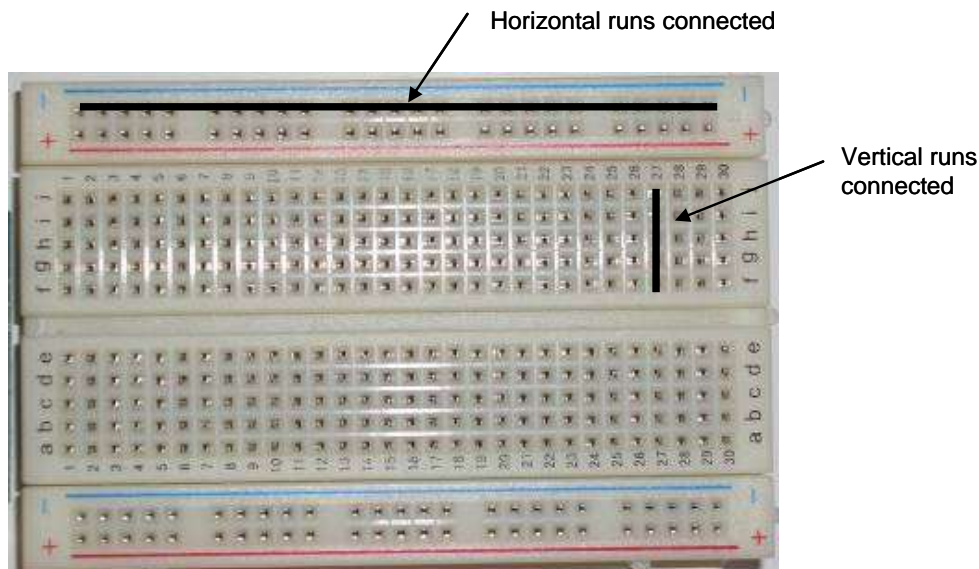
Якщо є синтаксична помилка в програмі, то в типізації повідомлення про помилку з'явиться в нижній частині вікна програми. Взагалі, програма разом із помилками покаже проблему. Якщо і раніше були проблеми, спробуйте ці ідеї

- Запустіть програму Arduino знову
- Переконайтеся, що USB-КАБЕЛЬ є безпечним на обох кінцях.
- Перезавантажте комп'ютер, тому що іноді послідовний порт може замкнути
- Якщо "Послідовний порт ... вже використовується", то з'являється повідомлення про помилку при завантаженні

Попросіть друга про допомогу

1.7 Накінечники

Накінечники є важливим інструментом для швидкого створення прототипів електронних схем. Компоненти й дроти вставте в отвори. Рядки і стовпці отворів внутрішньо пов'язані, щоб зробити з'єднання легким. Дроти біжать від накінечників до виводів I/O на платі Arduino. Зробіть з'єднання, використовуючи короткі відрізки 22 г одножильного дроту, на якому слід зняти ізоляцію 0,25 мм на кожному кінці. Ось фото макета, який показує внутрішньо пов'язані ділянки. Пари горизонтальних ділянок на верхній і нижній корисні для живлення і заземлення. Домовлено зробити перехід +5 В червоного кольору і перехід Gnd синього кольору. Траси живлення іноді називають "електричні шини".



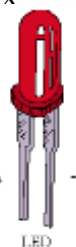
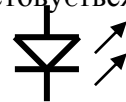
Увага: використовуйте тільки міцний одножильний дріт на платі. Пасма багатожильного дроту можуть розірватися і заповнити отвори.

Підказка: обріжте дроти і деталь підведіть так, щоб дроти і деталі прилягали близько до плати.

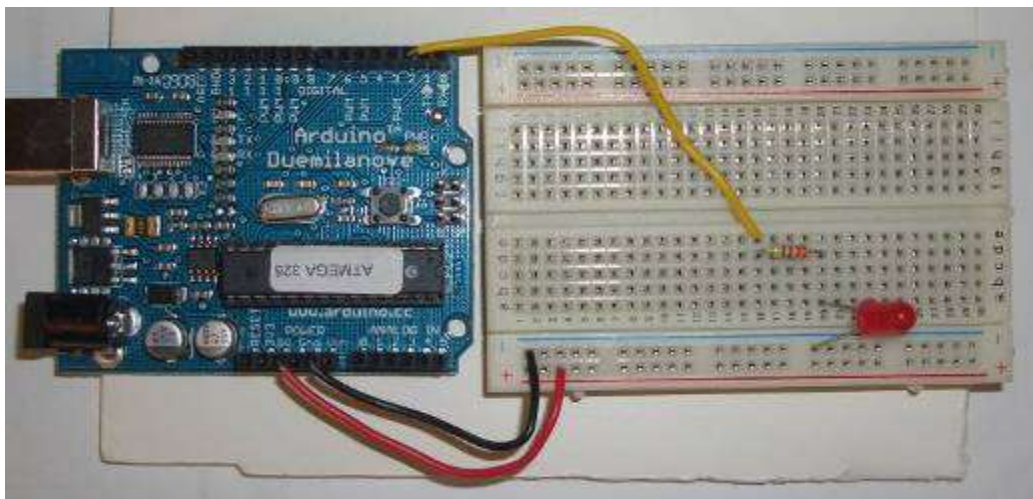
Щоб зберегти плати Arduino і макетів разом, ви можете скріпити частини з використанням двостороннього скотчу або іншими засобами.

2 Блимання світлодіодів

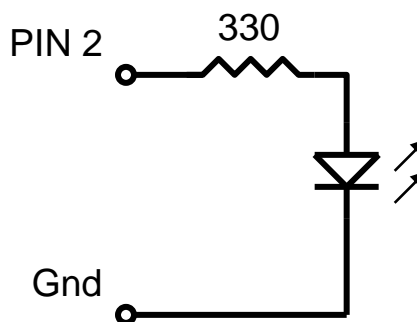
Світлодіоди (LED) зручні для перевірки, що Arduino може робити. Для вирішення цього завдання необхідні світлодіод, 330 Ом резистор і кілька коротких шматків 22 або 24 г дроту. На малюнку праворуч представлений ескіз світлодіода і його символ, що використовується в електронних схемах.



Використовуючи 22 г одножильний провід, підключіть силовий штифт 5 В на Arduino до н червоної шини живлення на платі і штифт заземлення на Arduino до нижньої блакитної на . Підключіть зубчасту або плоску сторону світлодіода (виїмка або площина знаходяться на краю, який оточує LED базу, подивіться уважно, бо їх важко знайти) до шини Gnd і з іншого боку до вільного отвору на головній дошці макету. Помістіть резистор так, щоб один кінець знаходився в тому ж стовпці, що й світлодіод, а інший кінець - у вільному стовпці. Від цього стовпця підключіть провід до цифрового виводу 2 на платі Arduino. Ваша установка буде виглядати наступним чином:



Щоб перевірити, чи працює індикатор, тимчасово відключіть провід від контакту 2 на платі Arduino і дотику до силової шини 5V. Світлодіод повинен загорітися. Якщо ні, то спробуйте змінити орієнтацію світлодіода. Помістіть провід назад у контакт 2. На LED струм проходить від анода (+) до катода (-), який відзначений насічкою. Схема представлена в схематичній формі на малюнку справа.



Створити і запустити цю програму Arduino

```
void setup()
{
  pinMode(2, OUTPUT);

  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
}

void loop()
{}
```

Світлодіод мигнув одну секунду? Натисніть кнопку reset (скидання) Arduino і знову запустіть програму.

Тепер спробуйте цю програму, в якій буде блимати світлодіод на 1,0 Гц. Всі записи після // у рядку є коментарем, як текст між "/" "*" і "*" "/" у верхній частині. Завдяки цьому завжди можна додавати коментарі до програми.

```
/*-----
Blinking LED, 1.0 Hz on pin 2
-----*/
```

```

void setup()                // one-time actions
{
  pinMode(2,OUTPUT);        // define pin 2 as an output
}

void loop()                  // loop forever
{
  digitalWrite(2,HIGH);     // pin 2 high (LED on)
  delay(500);                // wait 500 ms
  digitalWrite(2,LOW);      // pin 2 low (LED off)
  delay(500);                // wait 500 ms
}

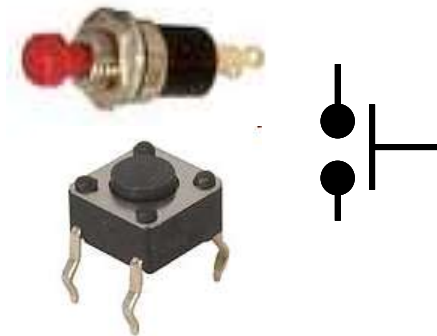
```

Команда `pinMode` приключає LED кнопкою до контакту виходу. Перша команда `digitalWrite` встановлює контакт 2 Arduino із HIGH, або +5 вольт. Таким чином посилається струм від штифта, через резистор, через світлодіод (він світить) і на 0 - землю. Команда `delay(500)` очікує протягом 500 мс. Друга команда `digitalWrite` встановлює контакт 2 НИЗЬКИЙ або 0 В зупинки подачі струму, і таким чином світлодіод вимикається. Код в дужках, що визначає функцію `loop()`, повторюється безперервно, тому індикатор блимає.

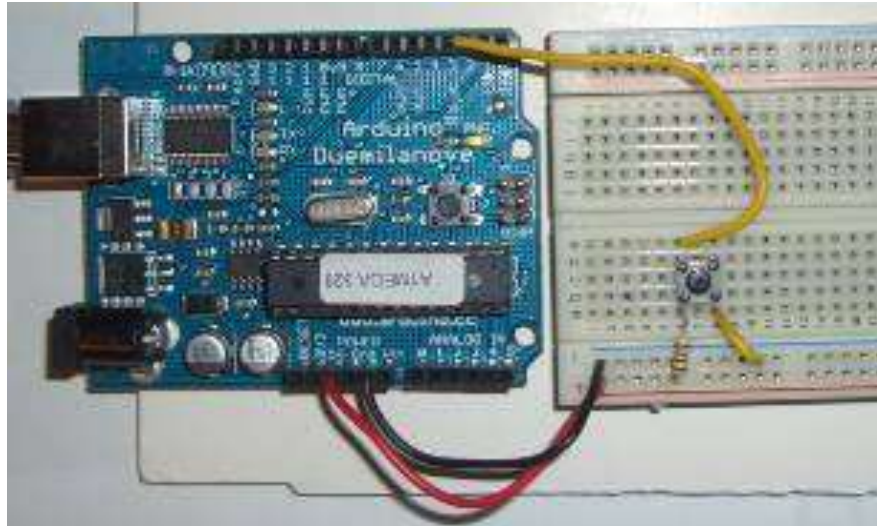
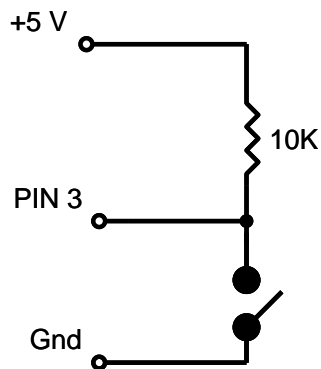
Ця вправа показує, як Arduino може контролювати зовнішній світ. При відповідному інтерфейсі один і той же код може вмикати і вимикати двигуни, реле, соленоїди, електромагніти, пневматичні клапани або будь-який інший тип пристрою.

3 Читання (тлумачення) вимикача

Вправа зі світлодіодом показує, як Arduino може контролювати зовнішній світ. Багато додатків вимагають читання стану датчиків, у тому числі вимикачів. На малюнку праворуч показано картинку кнопкового вимикача і його умовне позначення. Зверніть увагу, що символ - це вимикач, чиї контакти зазвичай роз'єднані, але замкнуті тоді, коли натиснута кнопка. Якщо у вас є вимикач, використовуйте функцію безперервності (звукового) цифрового мультиметра (DMM), щоб зрозуміти, коли дроти роз'єднані і коли вони з'єднані при натиснутій кнопці.



Для цієї вправи Arduino читатиме стан нормально-відкритого кнопкового вимикача і відображення результатів на ПК за допомогою команди `serial.println()`. Вам знадобиться вимикач, 10 кОм резистор і кілька шматків 22 г дроту для підключення. Якщо у вас немає вимикача, то візьміть два дроти і вручну під'єднайте вільні кінці для імітації замикання перемикача. На малюнку нижче показана схема для схеми зліва і реалізації справа.



Створити і запустити цю програму Arduino

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

void loop()
{
  Serial.println(digitalRead(3));
  delay(250);
}
```

Відкрийте вікно Serial Monitor. Коли вимикач відкритий, ви повинні побачити потяг 1 на екрані. Коли закритий, відбувається зміна від 1 до 0. З апаратного боку, коли вимикач відкритий, струм не тече через резистор. Якщо струм не тече через резистор, то не існує падіння напруги на резисторі. Це означає, що напруга на кожній стороні така ж. У вашій схемі, коли перемикач відкритий, контакт 3 знаходиться на 5 вольт, які комп'ютер зчитує як стан 1. Коли перемикач замкнутий, контакт 3 підключений безпосередньо до землі, яка знаходиться на 0 вольт. Комп'ютер зчитує це як стан 0.

Тепер спробуйте цю програму, котра є прикладом того, як ви, маючи комп'ютер, можете сидіти і чекати датчика для зміни стану.

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

void loop()
{
  while (digitalRead(3) == HIGH)
  {
    Serial.println("Somebody closed the switch!");
  }
}
```

```

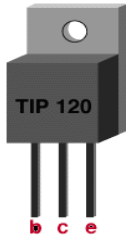
while (digitalRead(3) == LOW)
;
Serial.println("The switch is now open!");
}

```

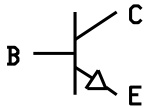
Дивіться, що відбудеться у вікні Serial Monitor, коли ви натиснете і відпустите кнопку.

4 Керування малим двигуном постійного струму

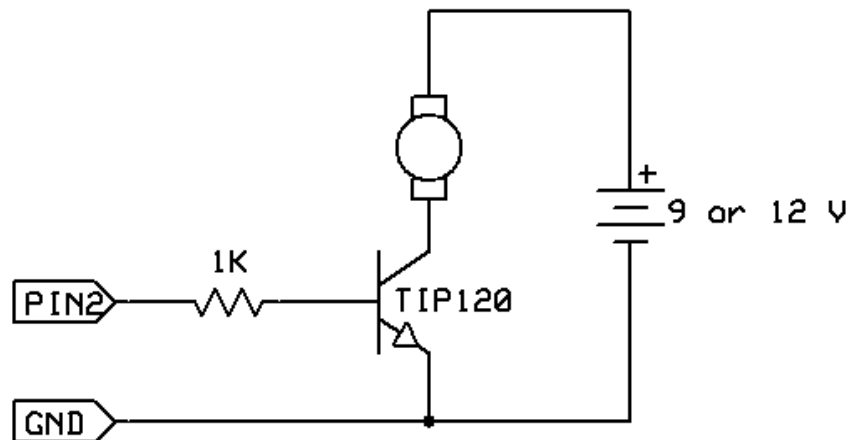
На платформі Arduino можна керувати невеликим двигуном постійного струму через транзисторний вимикач. Вам знадобляться TIP120 транзистор, 1K резистор, 9 В батареї, зі змінними батареями, і двигун.



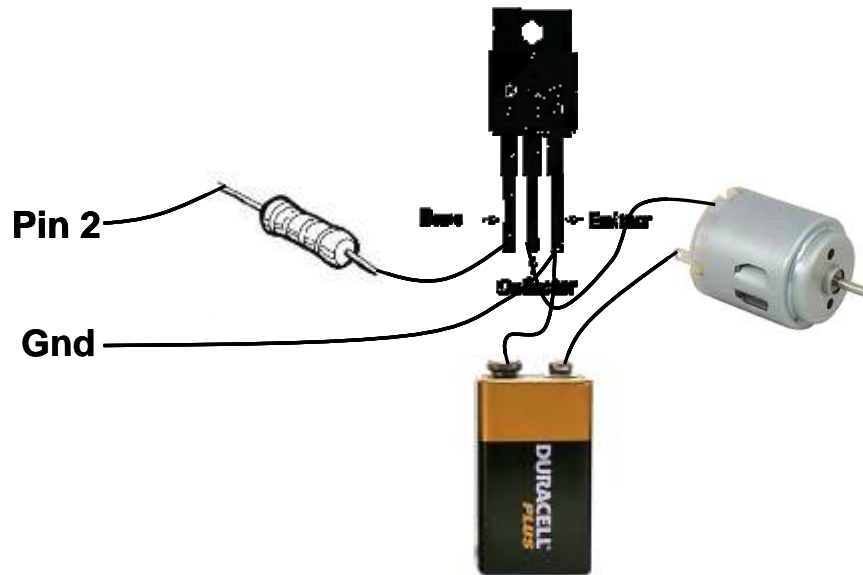
Кнопки TIP120 мають такий вигляд, а це схема кнопки.



Ось схема підключення двигуна.



А ось наочна схема того, як з'єднати компоненти. З'єднання можуть бути припаяні або вони можуть бути зроблені через накінецьники



Вивід 2 може бути будь-яким цифровим контактом I/O (pin) для вашого Arduino. Підключіть мінус батареї з емітером транзистора (Е-контактний) і також під'єднайте емітер транзистора до Gnd (заземлення) на платі Arduino.

Щоб перевірити, чи все працює, влаштуйте перемичку і короткий колектор до контактів емітера кнопки транзистора. Двигун повинен включитися. Далі відключіть резистор 1K від контакту 2 і переключіть його на + 5 В. Двигун повинен включитися. Поставте резистор назад в контакт 2 і запустіть наступну тестову програму:

```
void setup()
{
  pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
}

void loop()
{}
```

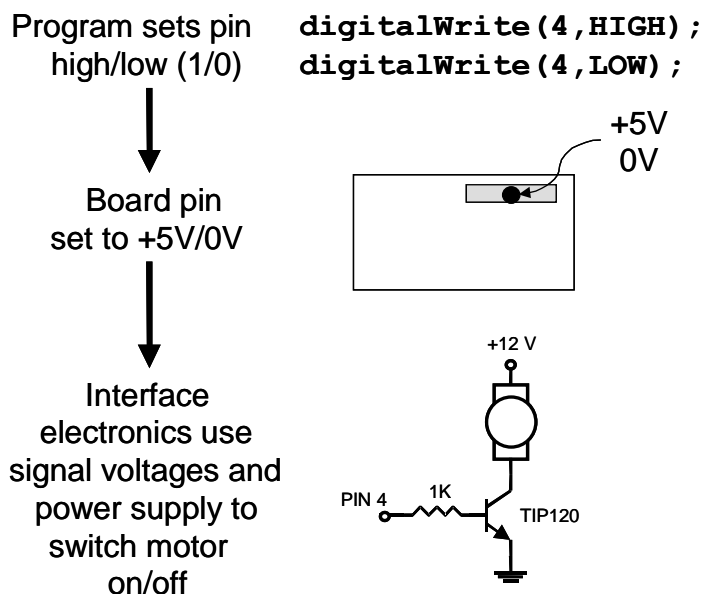
Двигун повинен включитися протягом 1 секунди.

5 Апаратні засоби Arduino

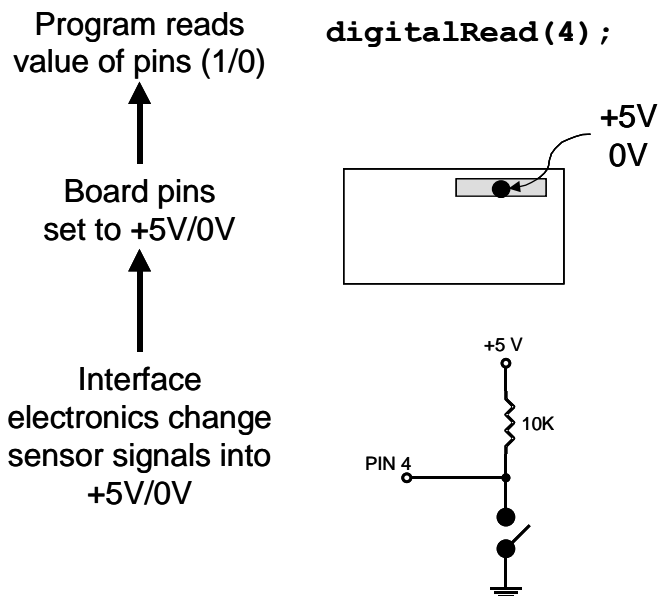
Сила Arduino не в його відкритому коді, а в його здатності взаємодіяти із зовнішнім світом через кнопки вводу-виводу (коротко - I/O). Arduino має 14 цифрових I/O, позначених від 0 до 13, які можуть бути використані, щоб включити двигуни і світло, читати стан вимикачів.

Кожен цифровий вивід повинен мати джерело близько 40 мА струму. Це більш ніж достатньо для взаємодії з більшістю пристроїв, але не означає, що інтерфейс схеми необхідний для керування іншими, ніж прості світлодіоди, пристроями. Іншими словами, ви не можете запустити наявний двигун безпосередньо в даний час з плати Arduino, а повинен бути штифт приводу схеми інтерфейсу, який, у свою чергу, приводить у дію електродвигун. В останньому розділі цього документа показаний інтерфейс для невеликого мотора.

Для взаємодії із зовнішнім світом програма встановлює цифрові контакти на високому або низькому значенні, використовуючи інструкції C коду, який відповідає +5 В або 0 В на виведенні. Штифт підключений до зовнішніх інтерфейсів електроніки, а потім до пристрою, щоб міняти включення-виключення. Послідовність подій - на цьому малюнку:



Для визначення стану перемикачів та інших датчиків, то Arduino в змозі прочитати значення напруги, прикладеної до його кнопки у вигляді двійкового числа. Схема інтерфейсу перетворює сигнал датчика в 0 або +5 сигналу, що подається на цифровий контактний ввід/вивід. Через команди програми Arduino опитує стан виводу. Якщо висновок при 0 В, то програма буде читати його як 0 або LOW. Якщо він знаходиться на +5 В, то програма буде читати його як 1 або HIGH. Якщо більше, ніж +5, то ви можете підірвати вашу плату, так що будьте обережні. Послідовність подій для читання кнопки показано на цьому малюнку.



Взаємодія зі світом має дві сторони. По-перше, розробник повинен створювати електронні схеми інтерфейсів, які дозволяють двигунам та іншим пристроям, котрі будуть управлятися з низьким (1-10 мА) струмовим сигналом, що перемикає між 0 і +5 В, та інші схеми, які перетворюють показання датчиків 0 або 5 В у сигнал, що комутується. По-друге, розробник повинен написати програму, використовуючи набір команд, які Arduino читатиме завдяки кнопкам I/O. Приклади можна знайти в розділі Ресурси Arduino веб-сайту ME 2011.

При читанні входів кнопки повинні мати значення 0 або +5. Якщо вивід залишається відкритим або "плаваючим", то він буде читати випадкові напруги і викликати помилкові результати. Ось чому у вимикачів завжди є 10K підтягуючий резистор, підключений при взаємодії з штифтом Arduino.

Примітка: Причина, щоб уникнути використання штифтів 0 і 1 та, що ці контакти використовуються для послідовного зв'язку між Arduino і комп'ютером.

На платформі Arduino також є шість аналогових входів для зчитування постійної напруги в діапазоні від 0 до 5 В від датчиків, таких як потенціометри.

6 Концепції програмування

У цій главі розглядаються деякі основні поняття програмування, припускається, що читач є повним новачком.

Комп'ютерна програма - це слідування крок за кроком інструкції для комп'ютера. Комп'ютер буде робити саме те, що ви не скажете йому не робити, не більше і не менше. Комп'ютер знає, що в програмі не те, що ви хотіли. Звідси оригінальність фрази "Сміття в, сміття з".

Набір допустимих інструкцій залежить від використання конкретної мови програмування. Є багато мов, у тому числі C, C++, Java, Ada, Fortran, Lisp, Basic, Pascal, Perl, і тисяча інших. На платформі Arduino використовується спрощена варіація мови програмування C.

Для будь-якої мови програмування інструкції повинні бути введені в особливому синтаксисі для того, щоб комп'ютер міг інтерпретувати їх правильно. Як правило, інтерпретація - це двоступінчастий процес. Компілятор приймає введений в програмі певний мовний текст і перетворює його в форму, що може прочитати машина, який затим завантажується в процесор. Коли програма виконується, то процесор виконує машинний код порядково.

6.1 Основи мов програмування

Всі послідовні мов програмування мають чотири категорії інструкції. Перша - це команди управління, які оцінюють вираз, виконання арифметичних, стан перемикання ліній I/O, і багато інших операцій. По-друге, це команди, які викликають стрибки, - програма стрибає відразу в іншу частину програми, яка позначається міткою. Стрибки - єдиний спосіб вирватися зі звичайного рядка за рядком режиму обробки. Наприклад, якщо ви хочете, щоб програма повторялася знову і знову, не зупиняючись, то в останньому рядку програми буде команда стрибка, який змушує програму повернутися до своєї першої лінії. По-третє, це команди, які оцінюють стан і перехід, якщо умова істинна. Наприклад, ви, можливо, захочете перейти, тільки якщо число більше нуля. Або ви можете стрибати, тільки якщо стан I/O рядка є низьким. По-четверте, є команди loop, які повторюють секцію коду задане число разів. Наприклад, з loop ви можете блимати світлом рівно шість разів.

Більшість мов програмування містять відносно невелике число команд. Складність комп'ютерів залежить від комбінування і повторення інструкцій кілька мільйонів разів на секунду.

Ось загальна програма.

```
1.   Do this
2.   Do that
3.   Jump to instruction 6
4.   Do the other thing
5.   All done, sleep
6.   If switch closed, do that thing you do
7.   Jump to instruction 4
```

Комп'ютер буде виконувати цей рядок за рядком. Мистецтво програмування - це просто вміння перевести свій намір у послідовності до відповідних інструкцій.

Тут як приклад для команди for loop слідує команда гілкування, яку використовує оператор IF

```
for (i=0;i<6,i++) {
    instructions
}
```



```
if (j > 4) goto label
instructions
```

Команди всередині циклу будуть повторюватися шість разів. Після цього, якщо значення змінної *j* більше 4, програма перейде до інструкції на ярлику з вказаною міткою, а якщо ні, то буде виконаний наступний рядок.

На додаток до основних команд, мови мають здатність викликати функції із незалежних ділянок коду, які виконують певне завдання. Функції є одним із способів виклику секції коду із різних місць програми, а потім повертається з цього розділу на рядок, який слідує за викликаним рядком. Ось приклад:

```
apples();
instructions
apples();
more instructions

void apples() {
    instructions
}
```

Усі функціональні яблука є між безліччю фігурних дужок, що слідують після "apples()". Коли функція завершується, програма повертається до наступного рядка після рядка, який викликає функцію.

6.2 Цифрові числа

При роботі з мікроконтролером, який взаємодіє з реальним світом, ви повинні «копати» трохи глибше, щоб зрозуміти системи нумерації і розміри даних.

Двійкова (підстава 2) змінна має два стани: від і до, або 0 і 1, або низькі і високі. За своєю суттю, всі комп'ютери працюють у двійковій системі, бо їхні внутрішні транзистори можуть бути тільки або відключені, або нічого між ними. Числа будуються з багатьох цифр двійкових чисел, багато в чому таким же чином, що і в системі з основою 10, коли ми створюємо більші від 9 числа за допомогою декількох цифр.

Трохи про двійковий розряд, який може приймати значення 0 або 1. Байт складається з 8 біт, або 8 двійкових цифр. За угодою, біти, які складають байт, позначені справа наліво, біт 0 є крайнім справа з найнижчим значенням, як показано нижче:

b7	b6	b5	b4	b3	b2	b1	b0
----	----	----	----	----	----	----	----

Таким чином, у двійковому числі 011, біти 0 і 1, а 1 біт 2 дорівнює 0. У двійкового числа 1000001, біти 0 і 7 1, а інші рівні нулю.

Ось кілька перетворень із двійкової в десяткову систему чисел розміром байт.

Binary	Decimal
00000011	3
00000111	7

11111111	255
----------	-----

У комп'ютері змінні використовуються для зберігання чисел. Змінна може приймати два значення 0 і 1 і, як правило, використовується в якості істинного/помилкового прапора в програмі. Змінна байт може приймати цілі значення 0-255 десяткової, а 16-бітна змінна може взяти значення 0-65,535. Змінні можуть бути або підписані (позитивні і негативні значення) або без знака (тільки позитивний).

7 Мова програмування Arduino

На платформі Arduino працює спрощена версія мови програмування C, з деякими розширеннями для доступу до обладнання. У цьому керівництві ми розглянемо підмножину мови програмування, яка є найбільш корисною для початківця - дизайнера Arduino. Для отримання більш докладної інформації про мову Arduino див. the Language Reference довідкового розділу веб-сайту Arduino - <http://arduino.cc/en/Reference/HomePage>.

Всі інструкції Arduino мають одну лінію. Плата може провести програму із сотні довгих рядків і є місце для приблизно 1000 двобайтових змінних. На платформі Arduino програма виконує близько 300,000 джерел рядків коду за секунду.

7.1 Створення програми

Програми створюються в середовищі розробки Arduino, а потім завантажуються в плату Arduino. Код повинен бути введений в правильному синтаксисі, що означає використання дійсних імен команд і дійсної граматики для кожного коду рядка. Компілятор зловить синтаксичні помилки і поставит прапор перед завантаженням. Інколи повідомлення про помилку може бути прихованим, і ви повинні пошукати і знайти актуальну помилку, що була позначена прапором.

Хоч ваша програма може пройти чисто через перевірки синтаксису, він, як і раніше, не може робити те, що ви хотіли. Ось де у вас є шанс відточити свої навички в налагодженні коду. На платформі Arduino зробіть те, що ви задали робити, а не те, що ви хотіли робити. Кращий спосіб знайти помилки - прочитати код порядково і бути як комп'ютер. Може допомогти пройти код досвідчена людина. Кваліфіковане налагодження вимагає практики.

7.2 Програма форматування і синтаксис

Програми вводяться порядково. Код чутливий до регістру, що означає "myvariable" відрізняється від "MyVariable".

Заявлення будь-яких команд. Заявлення припиняються крапкою з комою. Класична помилка - забути поставити крапку з комою, отож, якщо ваша програма не компілюється, слід вивчити текст помилки і побачити, чи не забули ввести крапку з комою.

Коментарі можуть бути будь-яким текстом, який слід розміщувати в "//" у рядку. Для багаторядкових блоків коментарів вони починаються з "/" "*" і закінчуються "*/".

Константи - це фіксовані числа, які можуть бути введені у вигляді звичайних десяткових чисел (тільки число) або в шістнадцятковому (основа 16) або в двійковому (основа 2), як показано в таблиці нижче

Decimal	Hex	Binary
---------	-----	--------

100	0x64	B01100100
-----	------	-----------

Мітки використовуються для посилання на місцеположення у вашій програмі. Вони можуть бути будь-якою комбінацією букв, цифр і підкреслення (_), але перший символ повинен бути літерою. Коли використовується для позначення місця розташування, виконайте ярлик з двокрапкою. При зверненні до адресної етикетки в керівництві рядка не використовуйте двокрапку. Ось приклад:

```
repeat: digitalWrite(2,HIGH);
        delay(1000);
        digitalWrite(2,LOW);
        delay(1000);
        goto repeat;
```

Використовуйте етикетки обережно, бо вони можуть насправді зробити програму важко виконуваною і складно відлагоджуваною. Дійсно, деякі програмісти C порадять вам ніколи не використовувати мітки.

Змінні виділяються, оголосивши їх у програмі. Кожна змінна повинна бути оголошена. Якщо змінна оголошена поза дужками функції, то її можна побачити всюди в програмі. Якщо вона оголошена всередині дужок функції, змінна може розглядатися тільки в рамках цієї функції.

Змінні бувають в декількох модифікаціях, включаючи байт (8-біт, без знака, від 0 до 255), слово (16-bit, без знака, від 0 до 65,536), int (16-bit, підписаний, -32,768 до 32,767.) і long (32-bit, підписаний, -2,147,483,648 до 2,147,483,647). Не використовуйте байт змінні, якщо вам не потрібні негативні числа або числа більше, ніж 255, а потім використовуйте внутрішні змінні. Використання більших розмірів, ніж потрібно, заповнює дорогий цінний простір пам'яті.

Оголошення змінних, як правило, з'являються у верхній частині програми

```
byte i;
word k;
int length;
int width;
```

Імена змінних можуть бути будь-якою комбінацією букв і цифр, але повинні починатися з літери. Імена, зарезервовані для інструкцій з програмування, не можуть бути використані для імен змінних, бо це дасть вам повідомлення про помилку.

Символи використовуються для перевизначення, коли переназваний код може бути зручнішим для прийняття і більш читабельним. Символи визначаються за допомогою команди "#define", і рядки, що визначають символи, повинні йти на початку вашої програми. Ось приклад без символів для випадку, коли світлодіод підключений до контакту 2.

```
void setup()
{
    pinMode(2, OUTPUT);
}

void loop()
{
    digitalWrite(2,HIGH); // turn LED on
```

```

    delay(1000);
    digitalWrite(2,LOW);    // turn LED off
    delay(1000);
}

```

Тут те ж саме - використовується символ, щоб визначити "LED"

```

#define LED 2    // define the LED pin

void setup()
{
    pinMode(LED,OUTPUT);
}

void loop()
{
    digitalWrite(LED,HIGH);
    delay(500);
    digitalWrite(LED,LOW);
    delay(500);
}

```

Зверніть увагу, як використання символів зменшує потребу в коментарях. Символи є надзвичайно корисними для портативних пристроїв, підключених до контактів, тому якщо вам треба змінити кнопку підключення пристрою, то ви зміните визначення тільки одного символу, а не йтимете через всю програму в пошуках посилань на цю кнопку.

7.3 Структура програми

Всі програми Arduino мають дві функції: настройка - setup() та цикл - loop(). Інструкції ви розміщуєте у функції автозапуску - startup (), яка виконується тільки один раз, коли починається програма, і використовується для ініціалізації. Використовуйте його, щоб встановити напрямки кнопки або ініціалізувати змінні. Інструкції, поміщені в циклі, виконуються багаторазово і утворюють основні завдання програми. Тому кожна програма має цю структуру

```

void setup()
{
    // commands to initialize go here
}

void loop()
{
    // commands to run your machine go here
}

```

Дійсно, просто-мінімум програма не зробить нічого, це ви можете скомпілювати і запустити її

```
void setup() {} void loop() {}
```

Програма не виконує жодної функції, але корисна для очищення від старих програм. Зверніть увагу, що компілятор не хвилює рядок повернення, і яка причина, чому ця програма працює, якщо знаходиться в одному рядку.

7.4 Математика

На платформі Arduino можна робити стандартні математичні операції. У той час як з плаваючою точкою (наприклад, 23.2) числа допускається, якщо оголошений як плаваючий, операції на плаваючих дуже уповільнюються, так що рекомендуються цілі змінні і математичні числа. Якщо у вас є байт змінних, ніякий номер, і не результат якоїсь математичної операції не можуть виходити за межі діапазону від 0 до 255. Ви можете розділити номери, але результат буде усічений (не округляється) до найближчого цілого числа. Таким чином, в цілочисельній арифметиці $17/3 = 5$, а не 5,666, а не 6. Математичні операції виконуються строго в порядку зліва направо. Ви можете додати дужки для групування операцій.

У таблиці нижче наведені деякі з допустимих операторів математики. Повну інформацію про їх використання можна знайти в довіднику з мови Arduino.

Symbol	Description
+	addition
-	subtraction
*	multiplication
/	division
%	modulus (division remainder)
<<	left bit shift
>>	right bit shift
&	bitwise AND
	bitwise OR

8 Прості команди

У цьому розділі описується невеликий набір команд, які ви повинні зробити, щоб Arduino став ще кориснішим. Ці команди з'являються в порядку черговості. Ви можете зробити велику машину, використовуючи тільки цифрове зчитування, цифровий запис і відкладення команд. Вивчення всіх команд приведе вас до наступного рівня.

Якщо вам потрібно більше інформації, то зверніться до довідкової сторінки про мову Arduino в <http://arduino.cc/en/Reference/HomePage>.

pinMode

Ця команда, яка йде у функції setup(), використовується, щоб установити цифровий вивід на вхід/вихід. Встановіть кнопку OUTPUT для керування і світлодіода, двигуна або іншого пристрою. Встановіть кнопку INPUT, якщо контакт читає перемикач або інший датчик. При включенні живлення або скиданні всі виводи - за замовчуванням входів. Цей приклад встановлює контакт 2 до виходу і контакт 3 на введення.

```
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, INPUT);
}
void loop() {}
```

Serial.print

Команда `Serial.print` дозволяє бачити те, що відбувається всередині Arduino, з вашого комп'ютера. Наприклад, ви можете побачити результат математичної операції, щоб визначити, чи ви отримуєте правильний номер. Або ви можете побачити стан цифрового введення кнопки, щоб побачити, що Arduino є датчиком, і перейти належним чином. Коли ваші інтерфейсні схеми або програми, здається, не працюють, використовуйте команду `Serial.print`, аби пролити трохи світла на ситуацію. Для цієї команди, щоб показати що-небудь, ви повинні Arduino підключити до комп'ютера за допомогою кабелю USB.

Для команди до роботи команда `Serial.begin(9600)` повинна бути поміщена в функцію `setup()`. Після того, як програма буде завантажена, ви повинні відкрити вікно серійного монітора, щоб побачити відповідь.

Є дві форми команди друку. `Serial.print()` друкує на тій же лінії, а `Serial.println()` починає друк у новому рядку.

Ось коротка програма для перевірки, якщо ваша плата діє і підключена до комп'ютера

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Hello World");
}
void loop() {}
```

Тут програма циклів на місці показує значення вводу/виводу кнопки. Це корисно для перевірки стану датчиків і вимикачів, а також можна подивитися, чи Arduino читає датчик належним чином. Спробуйте на вашому Arduino. Після завантаження програми використовуйте дротяну перемикачу для підключення почергово контактів 2 + 5V і Gnd.

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(digitalRead(2));
  delay(100);
}
```

Якщо ви хочете побачити стан контактів 2 і 3, то одночасно можете зв'язати кілька команд друку, зазначивши, що остання команда є `println to start`, щоб почати нову лінію.

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print("pin 2 = ");
  Serial.print(digitalRead(2));
  Serial.print("    pin 3 = ");
  Serial.println(digitalRead(3));
}
```



```

    delay(100);
}

```

Ви, можливо, помітили - коли намагаються залишити один із виводів відключеним, то його стан слідує за іншим. Це тому, що контакт залишили плаваючим, який має невизначене значення і буде блукати від високої до низької. Таким чином, робиться два стрибки при спробі, як на цьому прикладі.

Ось що перевіряє значення змінної подалі. Оскільки розрахунок робиться тільки одноразово, весь код в функції `setup()`. Це `Serial.flush()`

```

int i,j,k;
void setup()
{
    Serial.begin(9600);
    i=21;
    j=20;
    k=i+j;
    Serial.flush();
    Serial.print(k);
}
void loop() {}

```

digitalWrite

Ця команда встановлює кнопку входу/виходу високою (+ 5 В) або низькою (0 В) і є робочою конячкою для командування зовнішнім світлом ламп, двигунів і всього іншого, сполученого з платою. Використовуйте команду `pinMode()` у функції настройки `setup()`, щоб встановити штифт на вихід.

```

digitalWrite(2,HIGH); // sets pin 2 to +5 volts
digitalWrite(2,LOW);  // sets pin 2 to zero volts

```

delay (затримка)

`delay` припиняє програму для вказаного числа мілісекунд. Через те, що більшість взаємодій зі світом залучає час, ця інструкція важлива. Затримка може бути для 0 до 4,294,967,295 мс. Цей фрагмент коду поверне на контакт 2 протягом 1 секунди.

```

digitalWrite(2,HIGH); // pin 2 high (LED on)
delay(1000);          // wait 500 ms
digitalWrite(2,LOW);  // pin 2 low (LED off)

```

if (якщо)

Це основна команда умовного гілкування, яке дозволяє вашій програмі зробити дві різні речі залежно від того, зазначена умова є дійсною чи ні.

Ось один із способів, щоб ваша програма чекала на місці, поки перемикач не буде закритий. Підключіть перемикач до контакту 3, як показано в розділі 3. Завантажте цю програму і спробуйте закрити перемикач

```

void setup()

```

```

{
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(3) == LOW) {
    Serial.println("Somebody closed the switch!");
  }
}

```

Умовна лінія зчитує стан кнопки 3. Якщо її положення високе, то це означає для цієї схеми відкритий вимикач, і код перестрибує через команди `Serial.println` та повторює цикл. Коли ви закриваєте вимикач 0 В на кнопку 3 і її стан стає LOW, то це означає, що (якщо умова дійсна) виконується код у фігурних дужках і друкується повідомлення. Синтаксис для підтвердження, якщо

```

if (condition) {
  //commands
}

```

Якщо умова чинна, то програма буде виконувати команди між дужками. Якщо умова не виконується, то програма перейде до оператора, наступного за дужками.

Умова порівнює один стан до іншого. У наведеному вище прикладі стан кнопки 1, у порівнянні з LOW with `==`, має стан рівності. Інші оператори: `!=` (не дорівнює), `>` (більше), `<` (менше), `>=` (більше або дорівнює) і `<=` (менше або дорівнює).

Ви можете мати відгалуження програми залежно від значення змінної. Наприклад, ця програма буде друкувати значення для `i` тільки тоді, коли це менше, ніж 30

```

int i;

void setup()
{
  Serial.begin(9600);
  i=0;
}

void loop()
{
  i=i+1;
  if (i<30) {
    Serial.println(i);
  }
}

```

for

for використовується для повторення блока програми. Цикли корисні, якщо ви хочете, щоб блок коду повторювався вказане число разів. Змінна використовується для підрахунку кількості повторень коду. Ось приклад, коли блимає світлодіод, прикріплений до контакту 2, п'ять разів

```
int i;

void setup()
{
  pinMode(2, OUTPUT);
  for (i=0; i<5; i++) {
    digitalWrite(2, HIGH);
    delay(250);
    digitalWrite(2, LOW);
    delay(250);
  }
}

void loop() {}
```

Змінна i - це лічильник циклу. for() заява складається з трьох частин: ініціалізація, перевірка і прирощення. Змінна i ініціалізується до нуля. Перевірка, щоб побачити, якщо i - менше 5. Якщо це так, то команди між дужками виконуються. Якщо ні, то ці команди пропускаються. Після перевірки i збільшується на 1 (командою i++). У той час як для затвердження може читати по (i=1; i==5; i++), це угода, щоб почати змінну лічильника з нуля і використовувати менше перевірки стану.

Ви хочете, щоб лічильник циклу збільшувався на два або на три або з будь-яким прирощенням. Наприклад, спробуйте цей фрагмент коду.

```
int i;
void setup()
{
  Serial.begin(9600);
  for (i=0; i<15; i=i+3) {
    Serial.println(i);
  }
}

void loop() {}
```

Цикли можуть бути вкладеними і всередині циклів. На цьому прикладі світлодіод буде блимати 10 разів, тому що для кожного з п'яти зовнішніх циклів, підрахованих для i, програма переходить двічі через внутрішній цикл розрахунку для j.

```
int i, j;
void setup()
{
  pinMode(2, OUTPUT);
  for (i=0; i<5; i++) {
    for (j=0; j<2; j++) {
      digitalWrite(2, HIGH);
      delay(250);
      digitalWrite(2, LOW);
      delay(250);
    }
  }
}
```

```

    }
    void loop() {}

```

while

Це час для іншої команди програми, яка робить безперервне зациклення. Якщо умова після `while` дійсна, то команди у фігурних дужках виконуються безперервно. Ось приклад, коли постійно читається вимикач на контакті 3, а потім при натисканні вимикача стан стає більше не дійсним, отож код уникає команди `while` і друкує.

```

void setup()
{
    Serial.begin(9600);
    while(digitalRead(3) == HIGH) {
    }
    Serial.println("Switch was pressed");
}
void loop() {}

```

goto

`goto` – за цим завданням команди комп'ютер «перестрибує» безпосередньо до визначеної мітки у самій програмі, пропускаючи виконання коду до цієї мітки, який виконується опісля. `goto` слід використовувати з обережністю, бо команда може запустити програму в безкінечний цикл. Але команда `goto` зручна для виходу із глибоких циклів або інших складних структур управління. Ось приклад:

```

void setup()
{
    Serial.begin(9600);
    while(true) {
        if (digitalRead(3) == LOW) {
            goto wrapup;
        }
    }
wrapup:
    Serial.println("Switch was pressed");
}
void loop() {}

```

В дійсності завдання працює безперервно, перевіряючи стан штифта 3 щоразу. Коли кнопка 3 низько (натиснута), то, якщо умова дійсна і виконується перехід до виконання завдання, то можна вирватися з моменту циклу.

functions

Функції – це потужна функція програмування, яка використовується тоді, коли ви хочете налаштувати дії, які можуть бути викликані з декількох місць у програмі. Припустимо, ви хочете, щоб підключений до контакту 2 світлодіод блимав у разі попередження 3 рази, але вам потрібно виконати таке попередження у трьох різних місцях в програмі. Одним із рішень могло б бути - ввести миготливий код у трьох різних місцях програми. При цьому використовується дорогий простір коду, а в разі, якщо ви змініте функцію спалаху, наприклад, із 3 до 4 разів, то вам доведеться змінити код у трьох місцях. Краще рішення - це написати функцію спалаху в якості підпрограми і викликати її з основної частини коду. Ось приклад:

```

int i;
void setup()
{
    pinMode(2, OUTPUT);
    Serial.begin(9600);

    Serial.println("Welcome to my program");
    delay(1000);
    flasher(); // call flasher function
    Serial.println("I hope you like flashing");
    delay(1000);
    flasher(); // call flasher again
    Serial.println("Here it is one more time");
    delay(1000);
    flasher();
}
void loop() {}

void flasher()
{
    for(i=0;i<3;i++) {
        digitalWrite(2, HIGH);
        delay(250);
        digitalWrite(2, LOW);
        delay(250);
    }
}

```

Тут слід зазначити кілька речей. Функція `flasher()` визначена поза `setup()` і `loop()`. Коли основна програма стикається з командою `flasher()`; то програма негайно переходить до функції і починає виконувати код. Коли вона досягає кінця функції, тоді програма повертає виконання команди, що слідує безпосередньо за командою `flasher()`. Ця особливість дозволяє викликати підпрограму з декількох різних місць у коді. Параметри можуть бути передані і повернутися з функції, але ця функція для досвідченого програміста.

Це завершує розділ про основні команди програми. Ви можете написати деякі дивовижні програми, використовуючи тільки те, що було описано тут. Існує набагато більше того, що Arduino може робити, і Вам настійно рекомендується прочитати про це повністю за посиланням сторінки мови Arduino он-лайн.

9 Стиль кодування

Стиль передбачає ваш власний особливий стиль для створення коду і включає в себе макет, конвенції в подальшому використанні, заголовки і використання коментарів. Весь код повинен мати правильний синтаксис, але є багато різних стилів, які можна використовувати. Ось деякі пропозиції:

- Почніть кожен рядок коментаря, який має ім'я програми і, можливо, короткий опис того, що робить програма.
- Використовуйте відступи в лінії рядка. Назва функції і дужки – в одній колонці, а потім використовуйте багаторазово в рядках 2 або 4 коду, щоб відзначити рядки всередині, і так далі.

- Всі основні розділи і функції можуть мати один або два рядки.
- Кількість коментарів повинна бути оптимальною - не замало і не забагато. Припустимо, читач знає мову програмування, то коментар буде інформативним. Ось приклад такого інформуючого коментаря:

```
digitalWrite(4,HIGH)    // turn on
motor і ось марний коментар
digitalWrite(4,HIGH)    // set pin 4 HIGH
```

Вам не потрібно коментувати кожен рядок. Насправді це погана практика.
- Додайте коментарі при створенні коду. Якщо ви кажете собі: "Додам коментарі після завершення коду", то ви ніколи не зможете цього зробити.

10 Поширені помилки кодування

- Забули поставити кому в кінці заяви
- Неправильно написані команди
- Неправильно написані команди

Будь ласка, надсилайте коментарі, пропозиції та повідомлення про помилки в цьому посібнику за адресою: wkdurfee@umn.edu

11 Додаток

На наступній сторінці наводиться короткий список основних команд Arduino, з прикладами.

Основи програмування Arduino

Command	Description
<code>pinMode(n, INPUT)</code>	Set pin <i>n</i> to act as an input. One-time command at top of program.
<code>pinMode(n, OUTPUT)</code>	Set pin <i>n</i> to act as an output
<code>digitalWrite(n, HIGH)</code>	Set pin <i>n</i> to 5V
<code>digitalWrite(n, LOW)</code>	Set pin <i>n</i> to 0V
<code>delay(x)</code>	Pause program for <i>x</i> millisec, <i>x</i> = 0 to 65,535
<code>tone(n, f, d)</code>	Play tone of frequency <i>f</i> Hz for <i>d</i> millisec on speaker attached to pin <i>n</i>
<code>for()</code>	Loop. Example: <code>for (i=0; i<3; i++) {}</code> Do the instructions enclosed by {} three times
<code>if (expr) {}</code>	Conditional branch. If <i>expr</i> true, do instructions enclosed by {}
<code>while (expr) {}</code>	While <i>expr</i> is true, repeat instructions in {} indefinitely

Більше команд див. МЕ 2011 " Керівництво мікроконтролера Arduino" і розділ Language Reference на веб-сайті Arduino.

Інструкції у функції настройки `setup()` виконуються тільки один раз. А після цього функції `loop()` виконуються безперервно.

Приклади

1. Turn on LED connected to Pin 2 for 1 s.

```
void setup() {
  pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
}
void loop()
{}
```

2. Flash LED connected to Pin 2 at 1 Hz forever.

```
void setup() {
  pinMode(2, OUTPUT);
}
void loop() {
  digitalWrite(2, HIGH);
  delay(500);
  digitalWrite(2, LOW);
  delay(500);
}
```

3. Turn on motor connected to Pin 4 for 1 s.

```
void setup() {
  pinMode(4, OUTPUT);
  digitalWrite(4, HIGH);
  delay(1000);
  digitalWrite(4, LOW);
}
void loop()
{}
```

4. Play 440 hz tone for one second on speaker connected to pin 5. Delay is needed because the program does not wait for the `tone()` command to finish but rather immediately goes to the command following `tone()`.

```
void setup() {
  pinMode(5, OUTPUT);
  tone(5, 440, 1000);
  delay(1100);
}
void loop()
{}
```

5. LED is on Pin 2 and switch is on Pin 6. Turns on the LED for one sec when switch is pressed.

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(6, INPUT);
  while (digitalRead(6) == HIGH)
    ;
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
}
void loop()
{}
```