# Arduino

Michal Šerý

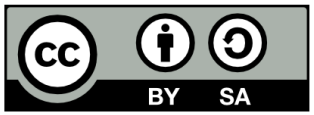
Jednočipové počítače

# Podpora

Tento vzdělávací materiál vznikl v rámci projektu
CZ.02.3.68/0.0/0.0/16_036/0005322 **Podpora rozvíjení informatického myšlení**.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání



# Obsah

# Obsah

# Obsah

# Co budeme dělat

- Návrh zapojení
- Jazyk WIRING

# Zajímavé odkazy

Taháky:

https://dlnmh9ip6v2uc.cloudfront.net/learn/materials/8/Arduino_Cheat_Sheet.pdf

https://github.com/liffiton/Arduino-Cheat-Sheet/blob/master/Arduino%20Cheat%20Sheet.pdf

# Několik rad

Při vymýšlení obvykle postupujete od prvotního nápadu ke zpřesňování jednotlivých detailů (analýza).

Při realizaci obvykle postupujete od jednotlivostí k celku (syntéza).

Opakováním získáváte na zručnosti, jistotě a spolehlivosti.

Je to úplně stejné jako třeba u houslisty. Čím více cvičíte, tím do složitějších „projetů“ se můžete pustit.

Důležitá je chuť a nenechat se odradit počátečními neúspěchy.

# Několik rad

Při vytváření výrobků s Arduinem je třeba mít na paměti, že produkt se skládá z hardware (HW) části a programové (SW) části. Pro dobrou funkčnost musí být obě části v pořádku. I když se někdy mohou některé chyby HW opravit pomocí SW a naopak, je lepší se již od počátku snažit chyby nedělat.

Skoro každý problém má více jak jedno řešení. Proto je dobré si vymyslet více možných řešení.

# Několik rad

Postup si vždy dobře rozmyslete a snažte se pochopit co nejlépe funkčnost jednotlivých komponent.

Postupujte pomalu. Každý krok tvorby konstrukce se snažte „nezávisle“ otestovat.

Proto je dobré si jednotlivé kroky vymyslet tak, aby se daly otestovat.

I když to stojí více (někdy i dost) času, ve finále ho uspoříte.

Praxe ti dodá na jistotě.

# Shrnuto, podtrženo

## Ničemu nevěř!

Proč? Všichni děláme chyby. Začni vždy u sebe. Pokud u sebe chybu nenajdeš hledej chybu u druhých. Zdravá míra nedůvěry je velmi prospěšná. I zdroje informací by si měly tvou důvěru získat, nikoliv ji mít automaticky.

## Chyba není tragédie. Chyba ti pomáhá!

Proč? Hledáním chyb se nejvíc naučíš. Ale, snaž se chyby neopakovat.

## Neboj se začít znovu

Proč? Někdy je nejlepší to „zbourat“ a začít znovu a pořádně. To se stává, když se chyby stanou již na počátku. Chce to odvahu, ale často se to vyplatí.

# WIRING

# Používané prvky

Při psaní programu používáme tyto prvky:

- proměnné jejich deklarace
- řídicí a běhové příkazy
- operátory
  - matematické
  - logické
  - porovnávací
- funkce
  - I/O
  - komunikační
  - časové
  - matemetické
  - vlastní

# Základní struktura

```
1 // deklarace
2 void setup() {
3 // proběhne pouze 1 x
4 }
5
6 void loop() {
7 // běží v nekonečném cyklu
8 }
```

# Blokové komentáře

```
1 /*
2 Toto je blokový komentář, nezapomeňte
3 ho ukončit. Znaky pro jeho začátek a
4 konec musí být vždy v páru!
5 */
```

# Řádkové komentáře

```
1 // Toto je řádkový komentář
```

Řádkový komentář může být od začátku řádku nebo na řádku za příkazem.

**Poznámka**

Komentářů není nikdy dost. Alespoň v programu.

# Složené závorky

## { }

Každá otevřená levá { musí být uzavřena pravou } !

Za } se nepíše středník!

## Poznámka

Editor v Arduino IDE zvýrazňuje párové závorky.

# Středník

## ;

Středníkem musí být ukončeny řádky:

- deklarací
- příkazů
- řízení toku programu (**`return, break, continue`**)

Středníkem se neukončují řádky řídicích struktur!

## Poznámka

Zapomenutý středník vyvolá chybu kompilátoru. Ne vždy však musí být jasné, že chybí středník. Pokud vám chybové hlášení nedává smysl, zkontrolujte nejprve, zda v okolí místa, kde kompilátor chybu ohlásil, nechybí středník.

# Konstanty a proměnné

## Pravidlo 1

Konstanty a proměnné musí být deklarované před jejich prvním užitím (fyzicky v rámci programu). Proto je dobrým zvykem umístnit deklarace na úplný začátek programu. Před **`setup`** i rutiny.

## Pravidlo 2

Názvy proměnných volte tak, aby, pokud možno, bylo z kontextu jasné jaký mají význam.

# Programování

## Strukruta programu

Program se skládá minimálně z dvou modulů (funkcí).

`void setup()` a `void loop()`

# Základní struktura

```
1  void setup() {
2
3  // proběhne pouze 1x
4
5  }
6
7
8
9  void loop() {
10
11 // nekonečná smyčka
12
13 }
```

# Programování

## Pravidlo 3

Snažte se při psaní programů každou informaci uvádět v programu pouze jednou (na jednom místě). Výrazně se tím sníží riziko chyb. Používejte konstanty a vlastní funkce.

# Programování

## Funkce

Jedná se o část kódu, která se ve vašemm programu bude několikrát opakovat.

# Ukázka funce

```
1  void setup(){
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    int a = 2;
7    int b = 6;
8    int k;
9
10   k = myVynasobCisla(a, b); // k bude 12
11   Serial.println(k);
12   delay(500);
13 }
14
15 int myVynasobCisla(int x, int y){
16   int result;
17   result = x * y;
18   return result;
19 }
```

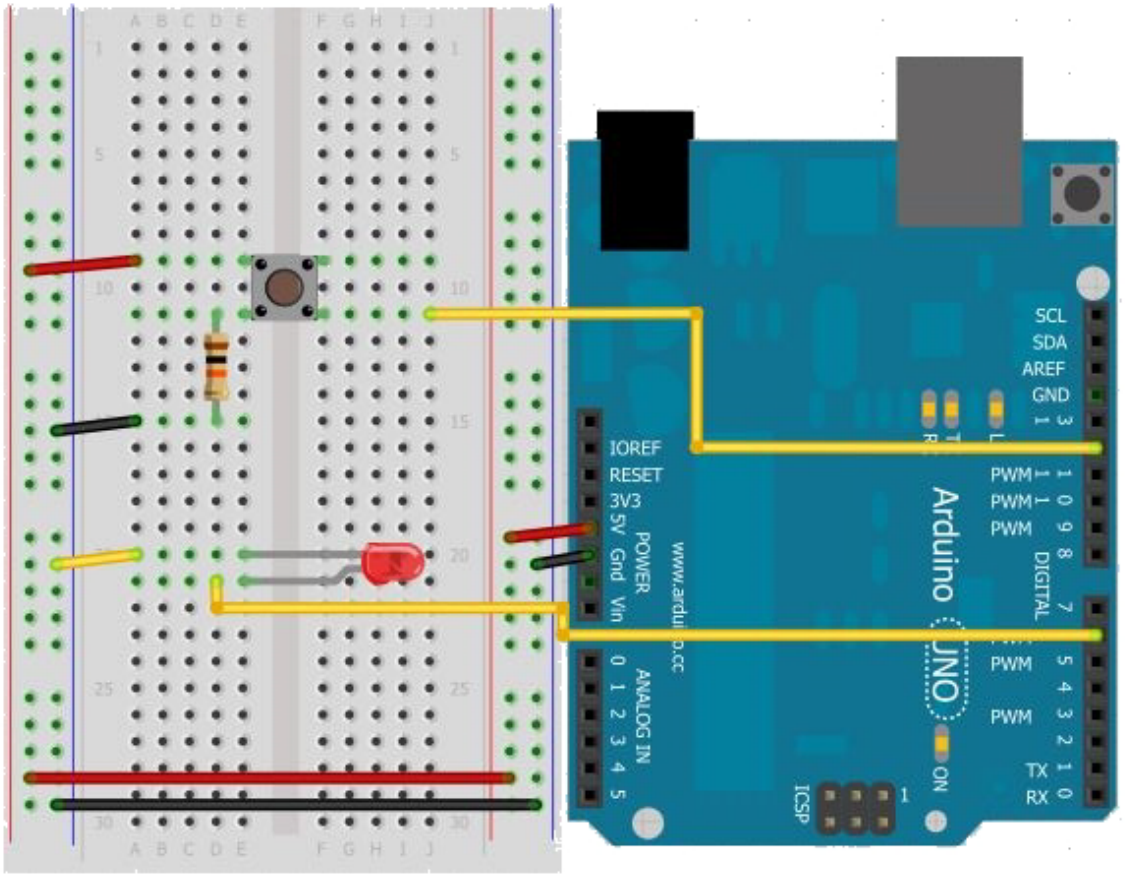
# Blikající LED dioda - interní

```
1  void setup() {
2     pinMode(13, OUTPUT);
3  }
4
5  void loop() {
6     digitalWrite(13, HIGH);
7     delay(1000);
8     digitalWrite(13, LOW);
9     delay(1000);
10 }
```

# Blikající LED dioda - externí

```
1  void setup() {
2     pinMode(5, OUTPUT);
3  }
4
5  void loop() {
6     digitalWrite(5, HIGH);
7     delay(500);
8     digitalWrite(5, LOW);
9     delay(500);
10 }
```

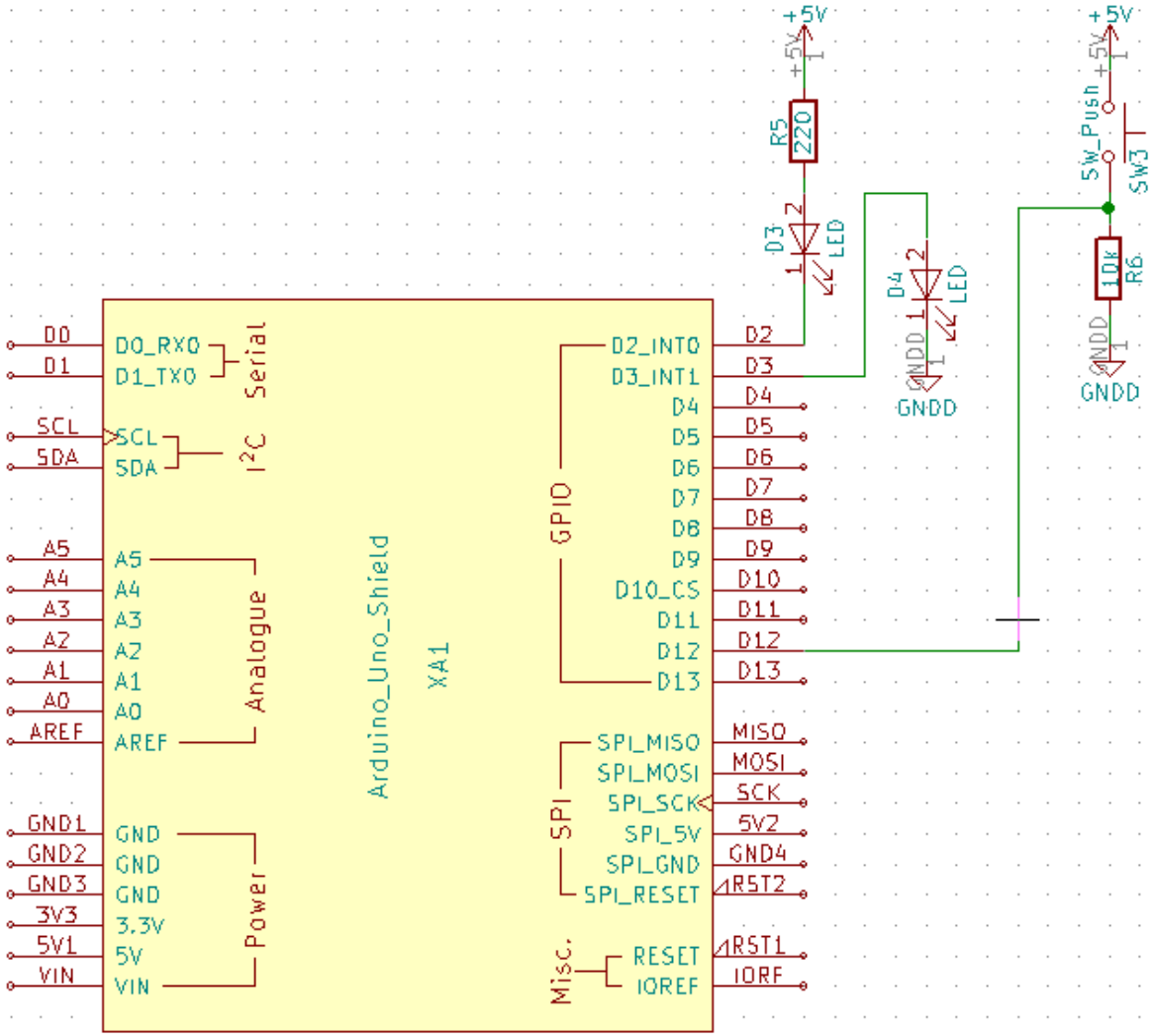
# LED + tlačítko




**Obrázek:** Zapojení FRITZING

Zdroj: www.arduino.cc

# LED + tlačítko




**Obrázek:** Zapojení KiCAD

# Příklad

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4
5  void setup() {
6  pinMode(led, OUTPUT);
7  pinMode(tlacitko, INPUT);
8  }
9
10 void loop() {
11 cteni = digitalRead(tlacitko);
12 if(cteni == HIGH){
13   digitalWrite(led, HIGH);
14   }
15 else{
16   digitalWrite(led, LOW);
17   }
18 }
```

# Příklad

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4
5  void setup() {
6  pinMode(led, OUTPUT);
7  pinMode(tlacitko, INPUT);
8  }
9
10 void loop() {
11 cteni = digitalRead(tlacitko);
12 digitalWrite(led, cteni);
13 }
```

# Příklad

```
1
2  if(inputPin == HIGH)
3  {
4  vykonej A;
5  }
6  else
7  {
8  vykonej B;
9  }
10
11 ...
12
13 if(inputPin < 500)
14 {
15 vykonej A;
16 }
17 else if(inputPin >= 1000)
18 {
19 vykonej B;
20 }
21 else
22 {
23 vykonej C;
24 }
```

# Příklad

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4  int tecka = 200;
5
6  void setup() {
7  pinMode(led, OUTPUT);
8  pinMode(tlacitko, INPUT);
9  }
10
11 void loop() {
12 cteni = digitalRead(tlacitko);
13 if(cteni == HIGH){
14   ...
15   }
16 else{
17   digitalWrite(led, LOW);
18   }
19 }
```

# Příklad

```
1  //S:
2    digitalWrite(led, HIGH);
3    delay(tecka);
4    digitalWrite(led, LOW);
5    delay(tecka);
6    digitalWrite(led, HIGH);
7    delay(tecka);
8    digitalWrite(led, LOW);
9    delay(tecka);
10   digitalWrite(led, HIGH);
11   delay(tecka);
12   digitalWrite(led, LOW);
13   delay(3*tecka);
14 //O:
15   digitalWrite(led, HIGH);
16   delay(3*tecka);
17   digitalWrite(led, LOW);
18   delay(tecka);
19   digitalWrite(led, HIGH);
20   delay(3*tecka);
21   digitalWrite(led, LOW);
22   delay(tecka);
23   digitalWrite(led, HIGH);
24   delay(3*tecka);
25   digitalWrite(led, LOW);
26   delay(3*tecka);
```

# Příklad

```
1  ...
2  void pismenoS(){
3  //S:
4    digitalWrite(led, HIGH);
5    delay(tecka);
6    digitalWrite(led, LOW);
7    delay(tecka);
8    digitalWrite(led, HIGH);
9    delay(tecka);
10   digitalWrite(led, LOW);
11   delay(tecka);
12   digitalWrite(led, HIGH);
13   delay(tecka);
14   digitalWrite(led, LOW);
15   delay(3*tecka);
16   }
```

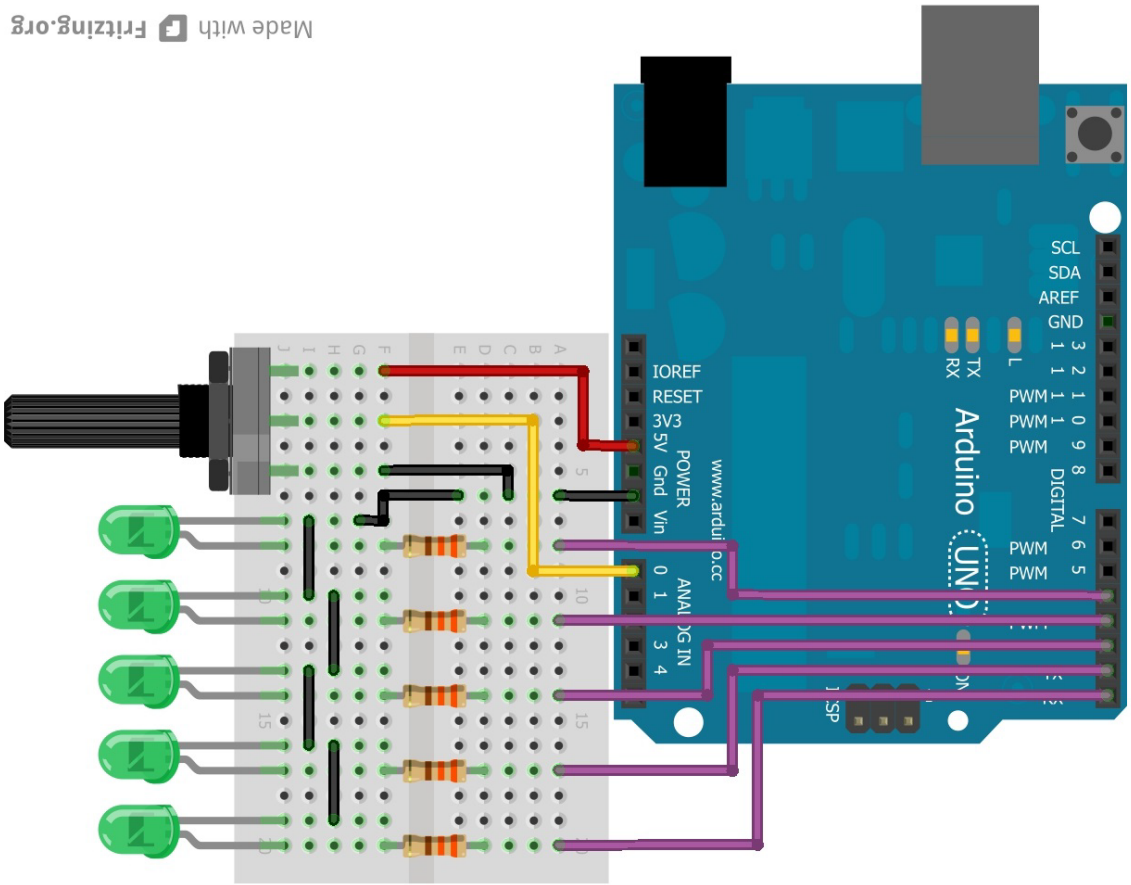
# Příklad

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4  int tecka = 200;
5
6  void setup() {
7  pinMode(led, OUTPUT);
8  pinMode(tlacitko, INPUT);
9  }
10
11 void loop() {
12 cteni = digitalRead(tlacitko);
13 if(cteni == HIGH){
14   pismenoS();
15   pismenoO();
16   pismenoS();
17   konecSlova();
18   }
19 else{
20   digitalWrite(led, LOW);
21   }
22 }
```

# Příklad COM

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4  int ukazatel_high;
5  int ukazatel_low;
6  void setup() {
7    Serial.begin(9600);
8    pinMode(led, OUTPUT);
9    pinMode(tlacitko, INPUT);
10 }
11
12 void loop() {
13 //    Serial.println(" xx ");
14 //    delay(10);
15   cteni = digitalRead(tlacitko);
16   if(cteni == HIGH){
17
18     digitalWrite(led, HIGH);
19     int ukazatel_high
20     if(ukazatel_high == 0{
21     Serial.print(cteni);
22     Serial.println("jdu ==");
23     ukazatel_high=1;
24     ukazatel_low=0;
25     }
26     }
```

# LED + potenciometr




**Obrázek:** Zapojení

Zdroj: www.arduino.cc

```
byte led[] = {0,1,2,3,4}; //pole s piny pripojenych LED diod
byte pot = A0;
int val;
void setup() {
  pinMode(led[0], OUTPUT);
  pinMode(led[1], OUTPUT);
  pinMode(led[2], OUTPUT);
  pinMode(led[3], OUTPUT);
  pinMode(led[4], OUTPUT);
}

void loop() {
  val = analogRead(pot);
  if(val > 800){digitalWrite(led[0],HIGH);}
  else if(val > 600){digitalWrite(led[1],HIGH);}
  else if(val > 400){digitalWrite(led[2],HIGH);}
  else if(val > 200){digitalWrite(led[3],HIGH);}
  else{digitalWrite(led[4],HIGH);}
  delay(250);
  digitalWrite(led[0],LOW);
  digitalWrite(led[1],LOW);
  digitalWrite(led[2],LOW);
  digitalWrite(led[3],LOW);
  digitalWrite(led[4],LOW);
}
```

```
1  void loop() {
2    val = analogRead(pot);
3    if(val > 800){
4      digitalWrite(led[0],HIGH);
5      }
6    else if(val > 600){
7      digitalWrite(led[0],HIGH);
8      digitalWrite(led[1],HIGH);
9      }
10   else if(val > 400){
11     digitalWrite(led[0],HIGH);
12     digitalWrite(led[1],HIGH);
13     digitalWrite(led[2],HIGH);
14     }
15   else if(val > 200){
16     digitalWrite(led[0],HIGH);
17     digitalWrite(led[1],HIGH);
18     digitalWrite(led[2],HIGH);
19     digitalWrite(led[3],HIGH);
20     }
21   else{
22     digitalWrite(led[0],HIGH);
23     digitalWrite(led[1],HIGH);
24     digitalWrite(led[2],HIGH);
25     digitalWrite(led[3],HIGH);
26     digitalWrite(led[4],HIGH);}
27   delay(250);
28
29 }
```

```
1  void loop() {
2    val = analogRead(pot);
3    if(val > 900){digitalWrite(led[0],HIGH);}
4
5    else if(val > 800){digitalWrite(led[1],HIGH);}
6    else if(val > 700){
7      ....
8      }
9    else if(val > 100){digitalWrite(led[8],HIGH);}
10   else{
11     digitalWrite(led[0],HIGH);
12     ....
13     digitalWrite(led[9],HIGH);}
14   delay(250);
15
16 }
```

# Termistor

**Obrázek:** Termistor

Při 0 °C odpor 10 kΩ

# Měřič vzdálenosti


**Obrázek:** Ultrazvukový měřič vzdálenosti

Vývody:
Vcc - napájení +5V
Trig
Echo
GND

# Měřič vzdálenosti

## Parametry a princip činnosti

Měří vzdálenost 2 cm až 3 m
Na Trig přivedem pulz alespoň 2 $\mu$s.
Poté vysílač vyšle 8 ultrazvukových impulzů.
Při zachycení odrazi jde signál Echo do „**1**“ po dobu trvání 8 pulzů.

## Výpočet

Zvuk se pohybue rychlostí 340 m/s (nebo 0,034 cm/$\mu$s).
Za určitou dobu $t$ urazí vzdálenost $s = v.t$.
Dobu $t$ získáme změřením délky pulzu na signálu Echo.
Získaný čas odpovídá dráze od senzoru k překážce a zpět. Proto budeme změřený čas dělit 2.

# Měřič vzdálenosti

## Měření

Nejprve vynulujem Trig na 2 $\mu$s.
Pak na Trig přivedem pulz 10 $\mu$s „**1**“.
Poté vysílač vyšle 8 ultrazvukových impulzů a nahodí „**1**“ na Echo.
Při zachycení odrazu, jde signál Echo do „**0**“.
Ke změření délky pulzu použijeme příkaz `pulseIn()`

```
1  // Čísla pinů
2  const int trigPin = 9;
3  const int echoPin = 10;
4  // Definice proměnných
5  long duration;
6  int distance;
7
8  void setup() {
9  pinMode(trigPin, OUTPUT); // Nastavení trigPin jako Output
10 pinMode(echoPin, INPUT); // Nastavení echoPin jako Input
11 Serial.begin(9600); // Starts the serial communication
12 }
13 void loop() {
14 digitalWrite(trigPin, LOW);// trigPin do LOW na 2 microsekundy
15 delayMicroseconds(2);
16 digitalWrite(trigPin, HIGH);// trigPin do HIGH na 10 microsekund
17 delayMicroseconds(10);
18 digitalWrite(trigPin, LOW);
19 // Změříme na pinu echoPin dobu impulzu
20 duration = pulseIn(echoPin, HIGH);
21 distance= duration*0.034/2;   // Výpočet vdálenosti
22 // Odeslání údaje distance na Serial monitor
23 Serial.print("Distance: ");
24 Serial.println(distance);
25 }
```

```
1 // Čísla pinů
2
3 tone(13,440,200);
```

## Nebo

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);
9    delayMicroseconds(100); // Approximately 10% duty cycle @ 1KHz
10   digitalWrite(13, LOW);
11   delayMicroseconds(1000 - 100);
12 }
```

# „Manuální“ PWM

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);
4    int intPerioda
5  }
6
7  void loop()
8  {
9    digitalWrite(13, HIGH);
10   delayMicroseconds(100); // Approximately 10% duty cycle @ 1KHz
11   digitalWrite(13, LOW);
12   delayMicroseconds(1000 - 100);
13 }
```

# Operátory

## Algebraické

| | |
|---|---|
| `=` | přiřazení |
| `+` | sčítání |
| `-` | odčítání |
| `*` | násobení |
| `/` | dělení |
| `%` | modulo (zbytek po dělení) |

# Operátory

## Komparační

| | |
|---|---|
| `==` | rovná se |
| `!=` | nerovná se |
| `<` | menší než |
| `>` | větší než |
| `<=` | menší nebo roven |
| `>=` | větší nebo roven |

# Operátory

## Logické

| | |
|---|---|
| `&&` | logický součin (AND) |
| `\|\|` | logický součet (OR) |
| `!` | negace (NOT) |
| `&` | logický bitový součin |
| `\|` | logický bitový součet |
| `~` | bitová negace |
| `^` | bitová exkluzivní disjunkce (XOR) |
| `<<` | posunout o bit vlevo |
| `>>` | posunout o bit vpravo |

# Operátory

## Složené operátory

| | |
|---|---|
| `++` | inkrementace (zvětšení o 1) |
| `--` | dekrementace (zmenšení o 1) |
| `+=` | složené sčítání (`x+=y` odpovídá `x=x+y`) |
| `-=` | složené odčítání (`x-=y` odpovídá `x=x-y`) |
| `*=` | složené násobení (`x*=y` odpovídá `x=x*y`) |
| `/=` | složené dělení (`x/=y` odpovídá `x=x/y`) |
| `&=` | složený logický bitový součin (`x&=y` odpovídá `x=x&y`) |
| `\|=` | složený logický bitový součet (`x\|=y` odpovídá `x=x\|y`) |

# Operátory

## Ukazatele

| | |
|---|---|
| `&` | reference |
| `*` | dereference |

Viz příklad na následujícím snímku.

## Poznámka

Ukazatele jsou pro začátečníky celkem komplikovaná látka na pochopení. Většina programů se bez nich obejde. Proto není nezbytně nutné je ihned používat. Jejich výhodou je, že dokáží výrazně zjednodušit manipulace s datovými strukturami.

# Příklad na ukazatele

```
1 ...
2 int *p;        // deklaruje ukazatel na int data typ
3 int i = 5, result = 0;
4 p = &i;        // nyní 'p' obsahuje adresu proměnné 'i'
5 result = *p;   // 'result' získá hodnotu na adrese ukazatele v 'p'
6
7 // jinak: 'result' obsahuje hodnotu proměnné 'i', která je 5
8 ...
```

# Možné typy proměnných - bitové

**`byte`**

obsadí jeden byte v paměti a reprezentuje celé číslo bez znaménka v rozsahu 0 až 255.

**`unsigned int`**

obsadí 2 byte v paměti a reprezentuje celé číslo bez znaménka v rozsahu 0 až 65 535.

**`unsigned long`**

obsadí 4 byte v paměti a reprezentuje celé číslo bez znaménka v rozsahu 0 to 4 294 967 295.

# Možné typy proměnných - číselné

## `int`

obsadí 2 byte v paměti a reprezentuje celé číslo se znaménkem v rozsahu -32 768 až 32 767.

## `long`

obsadí 4 byte v paměti a reprezentuje celé číslo se znaménkem v rozsahu -2 147 483 648 až 2 147 483 647.

## `float`

obsadí 4 byte v paměti a reprezentuje desetinné číslo se znaménkem v rozsahu -3.4028235E+38 až 3.4028235E+38.

## Poznámka

`float` čísla mají omezenou přesnost. Díky tomu může dojít k problémům při porovnávání dvou takových čísel. Matematické operace s nimi jsou také mnohem pomalejší. Pokud to jde - vyhněte se jim.

# Možné typy proměnných - textové

## `char`

obsadí jeden byte v paměti. Pokud ho použijete jako číslo pak jeho hodnota je v rozsahu -128 až +127. Jako řetězec obsahuje ASCII kód znaku.

## `char[]`

takto získáme proměnnou, do které lze uložit text. Pak je použit 1 byte na každý znak řetězce + jeden byte (symbol NULL), který indikuje konec řetězce.

# Deklarace a inicializace proměnné

```
...
//deklarace proměnné x
int x;
//přirazenií hodnoty do proměnné x
x = 10;
//tyto operace se dají spojit do jedné
int x = 10;
...
```

# Deklarace a inicializace proměnné

```
Tabulka priorit přerušení ATMega328

 1  Reset
 2  External Interrupt Request 0  (pin D2)          (INT0_vect)
 3  External Interrupt Request 1  (pin D3)          (INT1_vect)
 4  Pin Change Interrupt Request 0 (pins D8 to D13) (PCINT0_vect)
 5  Pin Change Interrupt Request 1 (pins A0 to A5)  (PCINT1_vect)
 6  Pin Change Interrupt Request 2 (pins D0 to D7)  (PCINT2_vect)
 7  Watchdog Time-out Interrupt                     (WDT_vect)
 8  Timer/Counter2 Compare Match A                  (TIMER2_COMPA_vect)
 9  Timer/Counter2 Compare Match B                  (TIMER2_COMPB_vect)
10  Timer/Counter2 Overflow                         (TIMER2_OVF_vect)
11  Timer/Counter1 Capture Event                    (TIMER1_CAPT_vect)
12  Timer/Counter1 Compare Match A                  (TIMER1_COMPA_vect)
13  Timer/Counter1 Compare Match B                  (TIMER1_COMPB_vect)
14  Timer/Counter1 Overflow                         (TIMER1_OVF_vect)
15  Timer/Counter0 Compare Match A                  (TIMER0_COMPA_vect)
16  Timer/Counter0 Compare Match B                  (TIMER0_COMPB_vect)
17  Timer/Counter0 Overflow                         (TIMER0_OVF_vect)
18  SPI Serial Transfer Complete                    (SPI_STC_vect)
19  USART Rx Complete                               (USART_RX_vect)
20  USART, Data Register Empty                      (USART_UDRE_vect)
21  USART, Tx Complete                              (USART_TX_vect)
22  ADC Conversion Complete                         (ADC_vect)
23  EEPROM Ready                                    (EE_READY_vect)
24  Analog Comparator                               (ANALOG_COMP_vect)
```

# Platnost deklarace proměnné

```
...
int x = 10; //zde deklarovanou proměnnou lze použít v celém programu
...

void setup() {
int y = 11;
//uvnitř této funkce lze použít proměnne x a y
}

void loop() {
int z = 12;
//uvnitř této funkce lze použít proměnne  x a z
}
...
```