Arduino

Co budeme potřebovat ?

1. Software :
   a) Arduino (IDE) - windows (64 b.), windows app, mac os,
      - Linux, online web
      https://www.arduino.cc

   b) UnoArduSim - windows (32b.)
      https://www.sites.google.com/site/unoardusim/

2. Hardware :
   a) Arduino Uno

   b) Propojovací USB kabel

   c) Shield + propojovací kabely

   d) PC pro programování, simulaci a nahrání

Co je to vůbec za jazyk?

Jazyk je odvozený z Wiringu, upravený a jmenuje se jednoduše Arduino Language.

Základní syntaxe

; středník

Středníkem musí být ukončena deklarace i jednotlivé prvky programu. Středník je také používán k oddělení prvků ve smyčce. int x = 13; // deklaruje proměnnou 'x' jako datový typ integer s hodnotou 13 Poznámka: Pokud zapomenete řádek programu středníkem ukončit, dojde k chybě kompilátoru. Z chybového hlášení může být zřejmé, že se jedná o zapomenutý středník, ovšem také nemusí. Pokud se objeví hlášení o záhadné nebo zdánlivě nelogické chybě kompilátoru, zkontrolujte nejprve, zda v zápisu programu nechybí středník v blízkosti místa, kde kompilátor chybu ohlásil. Nedává se jen za řádky s if, else, void, atd a řádky, kde jsou jen závorky, které tyto části kódu ohraničují. A jelikož výjimka potvrzuje pravidlo a abychom to neměli tak jednoduché, tak víceřádková deklarace 2D pole má jen jeden středník za ukončením deklarace.

{ } složené závorky

Složené závorky definují začátek a konec bloku kódu. Používají se ve funkcích i ve smyčkách. type function() { příkazy; } Za úvodní složenou závorkou [{] musí vždy následovat závorka uzavírací [}]. Proto se často uvádí, že složené závorky musí být párovány. Samozřejmě výjimka potvrzuje pravidlo, takže pokud se po if, else nachází jen jeden jediný řádek, lze závorky vynechat. Samostatně umístěné závorky (úvodní bez uzavírací a naopak) mohou často vést k záhadným, špatně dohledatelným chybám kompilátoru. Programové prostředí Arduino obsahuje praktickou funkci pro kontrolu párování složených závorek. Stačí vybrat závorku, nebo kliknout myší bezprostředně pod závorku a související závorka bude zvýrazněna.

/*… */ blokové komentáře

Blokové komentáře, nebo víceřádkové komentáře jsou oblasti textu, které jsou programem ignorovány. Jsou používány pro obsažnější komentování kódu nebo poznámky, které pomohou pochopit ostatním význam částí programu. Blokové komentáře začínají / * a končí * / a mohou obsahovat více řádků textu. /* Toto je blokový komentář, nezapomeňte ho ukončit. Znaky pro jeho začátek a konec musí být vždy v páru! */ Vzhledem k tomu, že komentáře jsou programem ignorovány, nezabírají žádný paměťový prostor; mohou tedy být hojně používány. Mohou být také použity k dočasnému znefunkčnění celých bloků kódu programu pro účely ladění. Poznámka: Do blokového komentáře je možno vložit i jednořádkové komentáře (uvozené //), ale není možno do blokového komentáře vložit další blokový komentář.

// jednořádkové komentáře

Jednotlivé řádky komentáře musí začínat // a končí na konci řádku. Stejně jako blokové komentáře jsou jednořádkové komentáře programem ignorovány a nezabírají žádný paměťový prostor. // toto je jednořádkový komentář ARDUINO – příručka programátora 3 hobbyrobot.cz Jednořádkové komentáře jsou často používány za příkazy k vysvětlení jejich funkce nebo jako poznámka pro další použití.

Struktura programu

```
/* 22.09.2018 Novák Jan IT4
 Úkol č.1 - Blikání led D13*/

#include <L298N.h>
L298N motor(6, 11, 12); //vytvoření instance pro jeden motor

byte stav=0;

#define led 13

void setup() {
  pinMode(led, OUTPUT);  //pin 13 ve stavu digitálního výstupu
}

void loop() {
  digitalWrite(led, HIGH);  // zapiš logickou úroveň 1
  digitalWrite(led, LOW);   // zapiš logickou úroveň 0
}
```

Do této části programu píšeme informace o programu.

Knihovny, které se mají přilinkovat při překladu programu. Jejich inicializace.

Definice proměnných a konstant a jejich základních hodnot použitých v programu. Definice podprogramů použitých v programu.

#define umožňuje programátorovi pojmenovat konstantní hodnotu před kompilací programu. Definované konstanty v arduinu nezabírají žádné místo v paměti programu na čipu. Kompilátor nahradí odkazy na tyto konstanty definovanou hodnotou v době kompilace.

Tato část se provede jen jednou na začátku programu. Zde se nastaví vstup/výstup pinů, inicializace displeje apod.

Hlavní program. Tato část programu se vykonává v nekonečné smyčce

Uživatelem definované nové funkce (podprogramy)

Konstanty

Konstanty si můžeme představit jako proměnné, které mají přednastavenou hodnotu, buď definovanou tvůrci Arduina nebo námi. Mají za úkol zpřehlednit práci a přiblížit kód lidskému jazyku

Logické konstanty

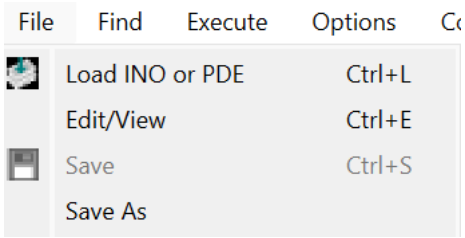
Jsou pouze dvě hodnoty, a to pravda a nepravda. V programování jim odpovídají konstanty true a false. Používají se tam, kde je třeba rozhodovat pouze mezi dvěma stavy. LOW a HIGH mají též hodnotu 0 nebo 1, ale požívají se spíše pro určení logické úrovně na výstupu.

False nebo **LOW** Konstanta má hodnotu 0.

True nebo **HIGH** Konstanta má hodnotu **1**. U konstanty true je situace trochu komplikovanější. Mohlo by se zdát, že má hodnotu 1, ale není to úplně přesné. Při logických operacích totiž jazyk Wiring vnímá jakékoliv nenulové číslo typu integer jako true.

Blikání s led (pin 13)

Program můžeme psát v jakémkoliv textovém editoru. V programu Arduino je jednoduchý editor bez našeptáván příkazů. Jediná velká výhoda je možnost použití příkladů. V programu UnoArduSim je editor s možností si vybrat základní příkazy. Pro první ukázkový program využijeme editor v simulátoru.



1. Spustíme UnoArduSim.

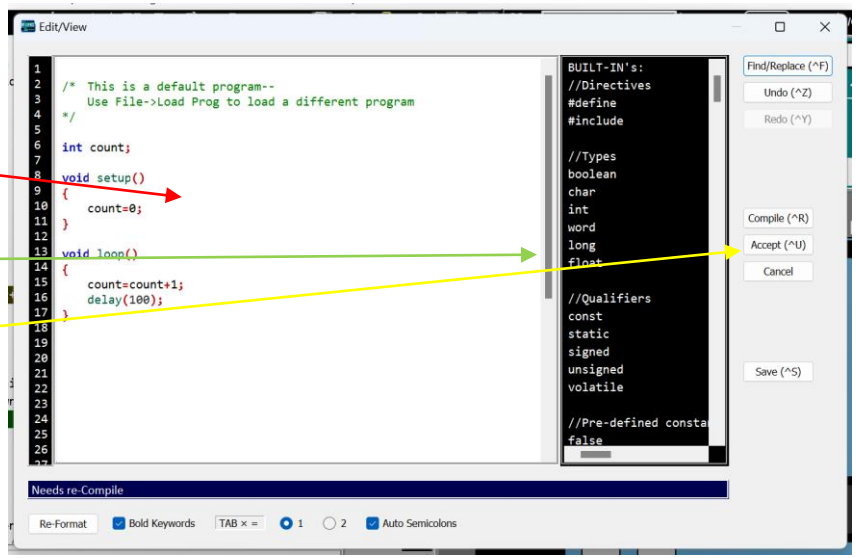
2. Uložíme si nový projekt (File => Save As). Nejlépe do samostatného adresáře se stejným jménem jako bude mít soubor. Např. blikani Nepoužívejte v pojmenování adresářů i souborů české znaky a mezery. Překladač nedokáže tyto cesty přeložit.

3. Spustíme editor programu : Ctrl+E (nebo 2x kliknout na text s programem) a otevře se editor

;Oblast pro psaní programu. Zde vložíte ukázkový program

Výběr základních příkazů

Příkazy simulátoru - kompilace apod.



Pro jednoduchost programování se v překladači pro arduino požívá číslování výstupů místo přesného popisu portů procesorů (možnost záměny arduino desek). Arduino uno má 14 digitálních vstupů/výstupů (D0 – D13) a 6 analogových vstupů nebo digitálních vstupů/výstupů (A0 – A5). Při programování je vždy zapotřebí zadat je-li pin ve stavu výstupu nebo vstupu (popřípadě zapnout pullup rezistor). To se zapisuje většinou do části setup příkazem „**pinMode**“.

Zápis logické úrovně na konkrétní port se provede příkazem „**digitalWrite**“. Lze zapisovat jen jeden pin.

Příkaz „**delay**“ zaměstná procesor na dobu v ms zadanou v závorkách

Úkol č.1 – Blikání led D13

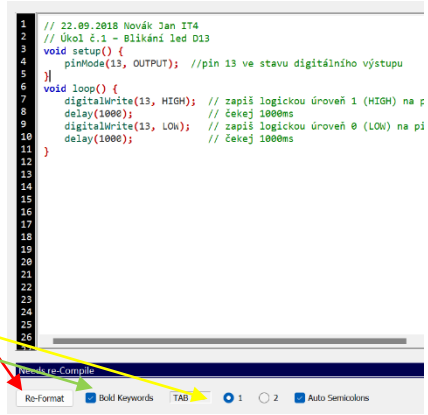
Napište program pro blikání led na D13. 1s svítí a 1000ms je zhaslá. Odzkoušejte jak simulátoru tak i v arduinu.

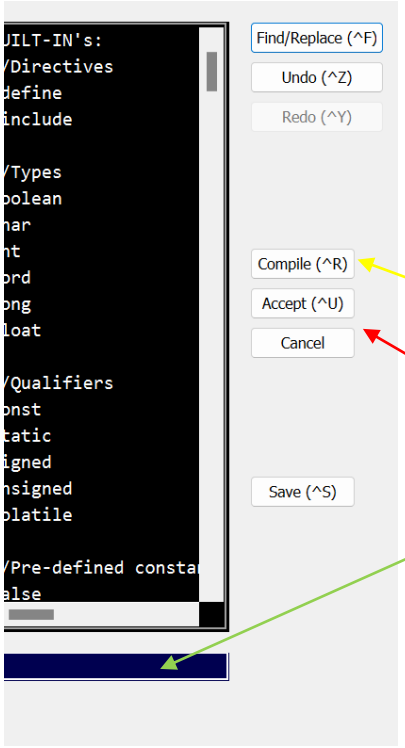
Všechny programy, které vytvoříte, uložte na disk H do adresáře s číslem úkolu. Do tohoto adresáře uložte program i konfigurační soubor simulátoru. U všech úkolů na první řádek do poznámky napište datum poslední úpravy programu příjmení, jméno a třídu. Na druhý řádek informaci o který úkol se jedná.

```
// 22.09.2018 Novák Jan IT4
// Úkol č.1 – Blikání led D13
void setup() {
  pinMode(13, OUTPUT);  //pin 13 ve stavu digitálního výstupu
}
void loop() {
  digitalWrite(13, HIGH);  // zapiš logickou úroveň 1 (HIGH) na pin 13
  delay(1000);             // čekej 1000ms
  digitalWrite(13, LOW);   // zapiš logickou úroveň 0 (LOW) na pin 13
  delay(1000);             // čekej 1000ms
}
```

Formátování textu :

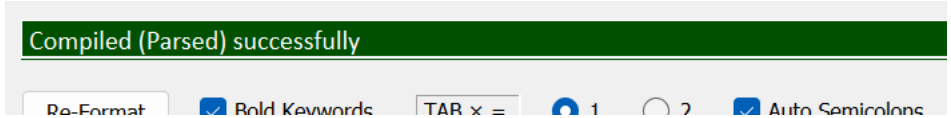
1. Klikněte na Re-formát a do textu se doplní mezery a tabulátory.
2. Zvýraznění určitých znaků
3. Počet tabulátorů (velikost odskoků před příkazem)





Compilace programu

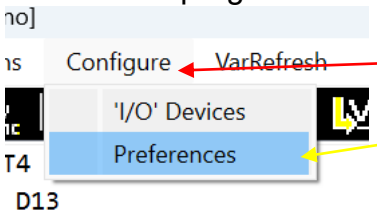
1. Compile - Při správné kompilaci se v dolním řádku vypíše nápis Compiled (Parsed) Succesfully. Při chybě je vypsána chyba a označen první řádek, kde překladač našel chybu.



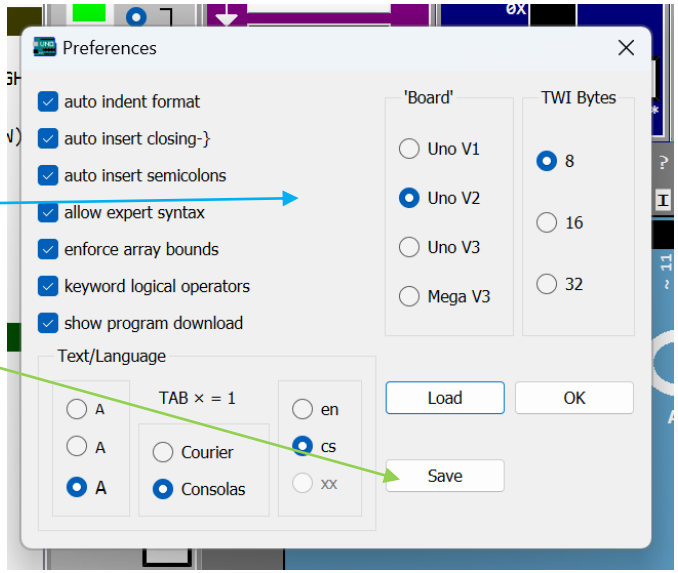
2. Accept - Zavře editor, přepne do simulace

Ověření programu v simulátoru

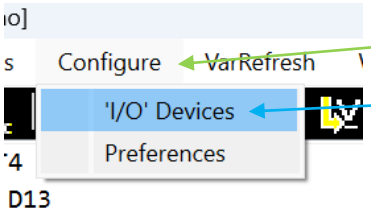
Pro ověření programu v simulátoru je zapotřebí nejprve vybrat Arduino Uno.



1. Configure
2. Preferences
3. Nastavení
4. Uložení - Save



Vybrat připojené periférie (led, tlačítka …)

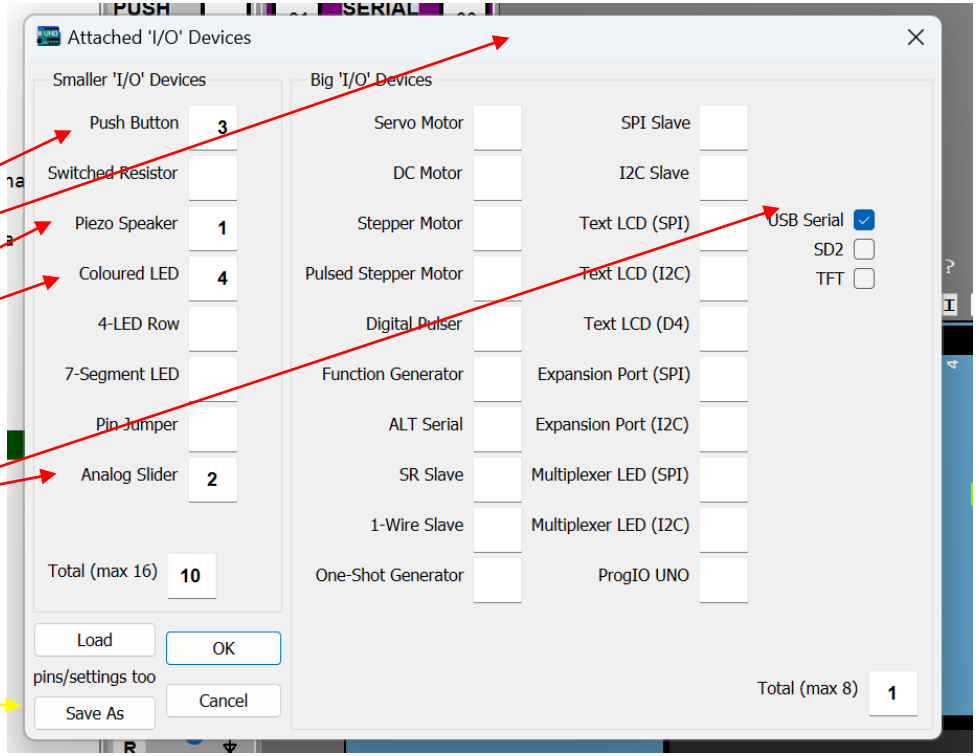


1. Configure

2. I/O Devices

3. Nastaveni:
   3x Tlačítko
   1x speaker (repro)
   4x led
   2x analogový vstup
   1x sériový vstup/výstup

4. Uložení – Save As



Ještě je zapotřebí připojit jednotlivé komponenty k pinům arduina. U patřičné komponenty zapište číslo pinu

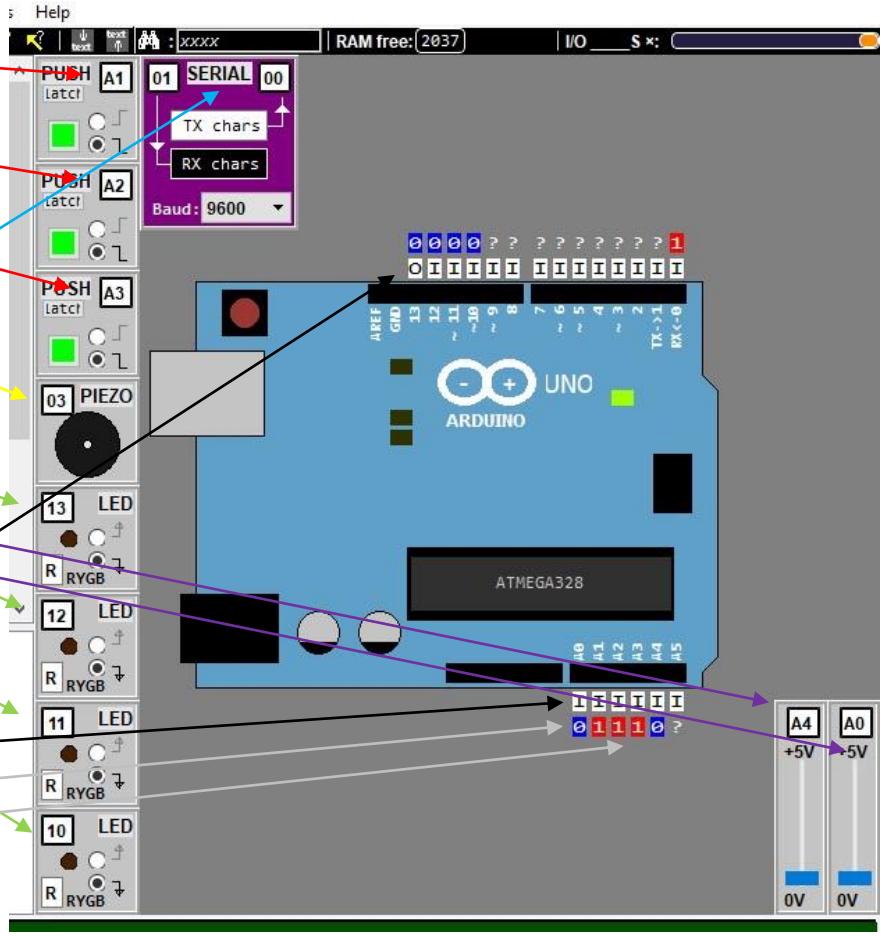
Tlačítko – pin A1
Tlačítko – pin A2
Tlačítko – pin A3

speaker (repro) – pin 03

led – pin 13
led – pin 12
led – pin 11
led – pin 10

analogový vstup – pin A4
analogový vstup – pin A0

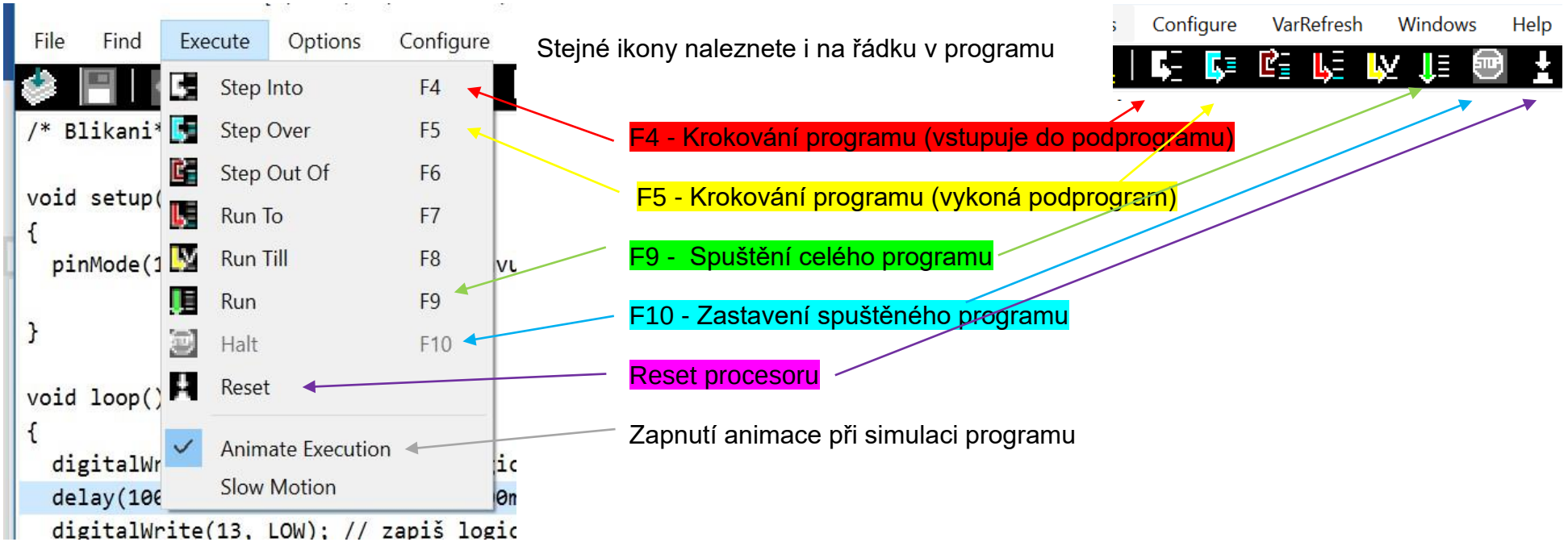
sériový vstup/výstup 01/00



U pinu na arduina naleznete tyto zkratky:

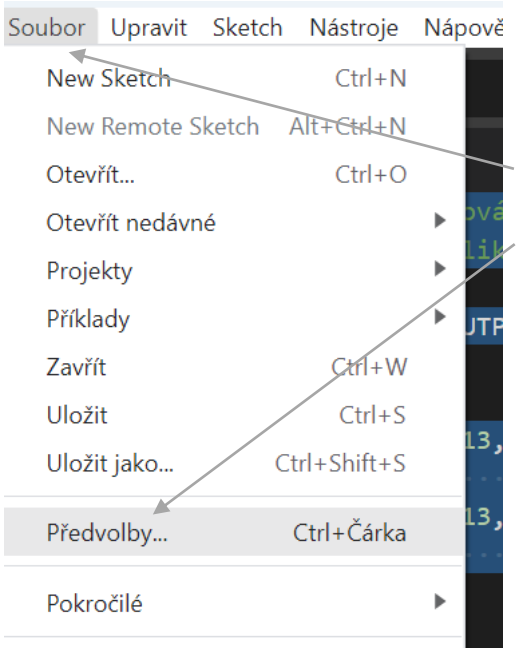
O – pin je stavu OUT – výstup z arduina
I – pin je stavu IN - vstup do arduina

0 – logická úroveň 0
1 – logická úroveň 1



Simulaci spustíte klávesou F9, ukončení simulace F10 - Zkontrolujte blikání led 13 v simulátoru

Ověření programu na Arduino Uno



Nejprve nastavíme program :

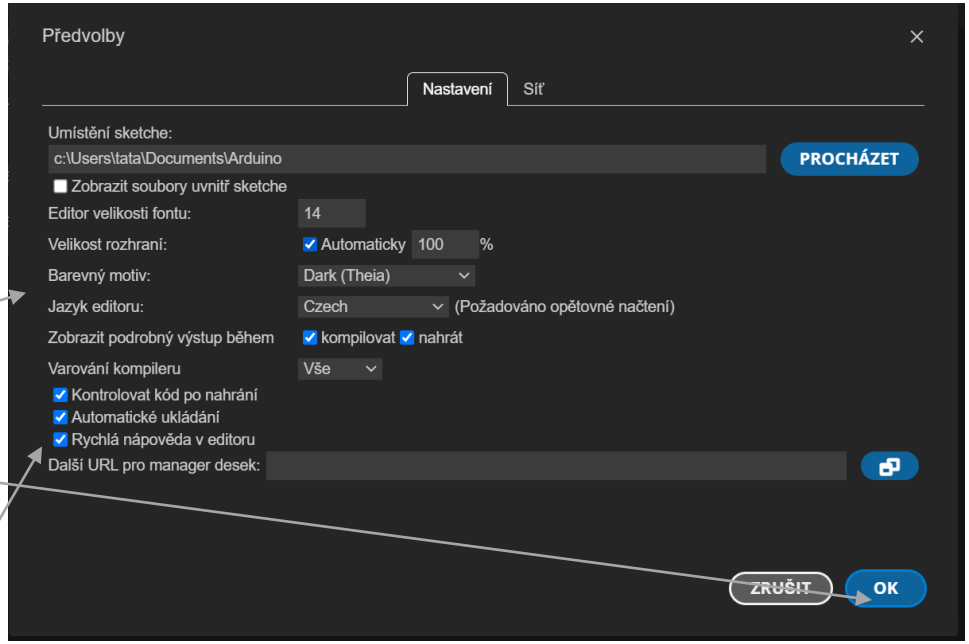
Vzhled a kompilace :

1.Soubor

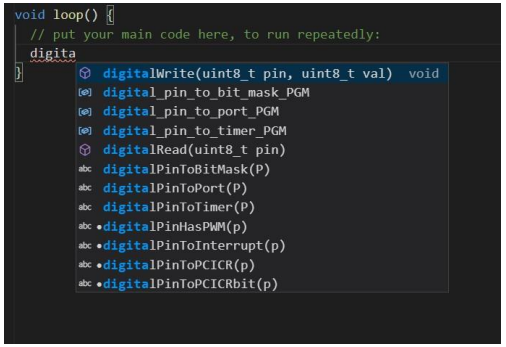
2.Předvolby

3.Nastavení

4.Uložení

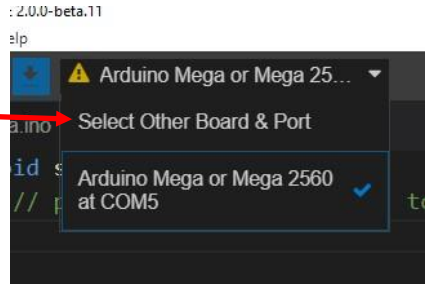


Tato položka zapne našeptávání příkazů

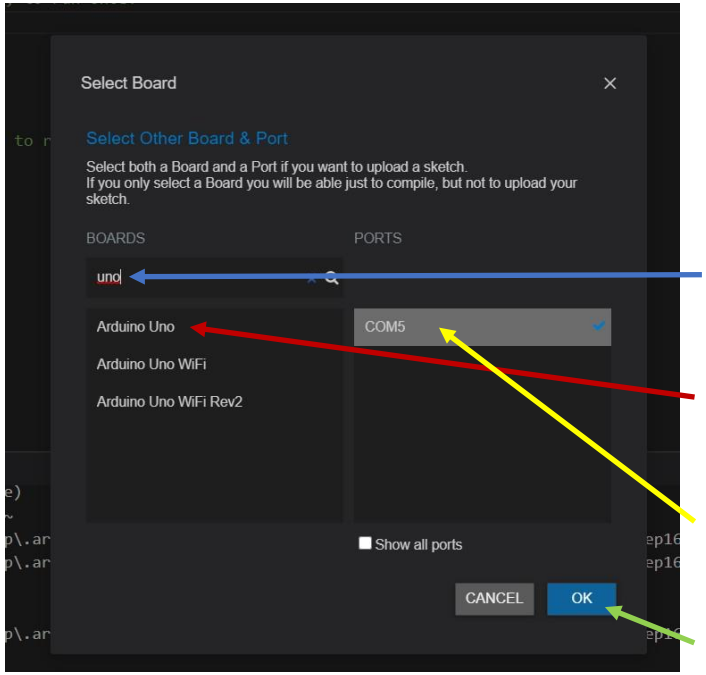


Nastavení vývojové desky arduino Uno

1. Klikněte na šipku výběru



2. Potvrďte výběr nové desky a portu připojení



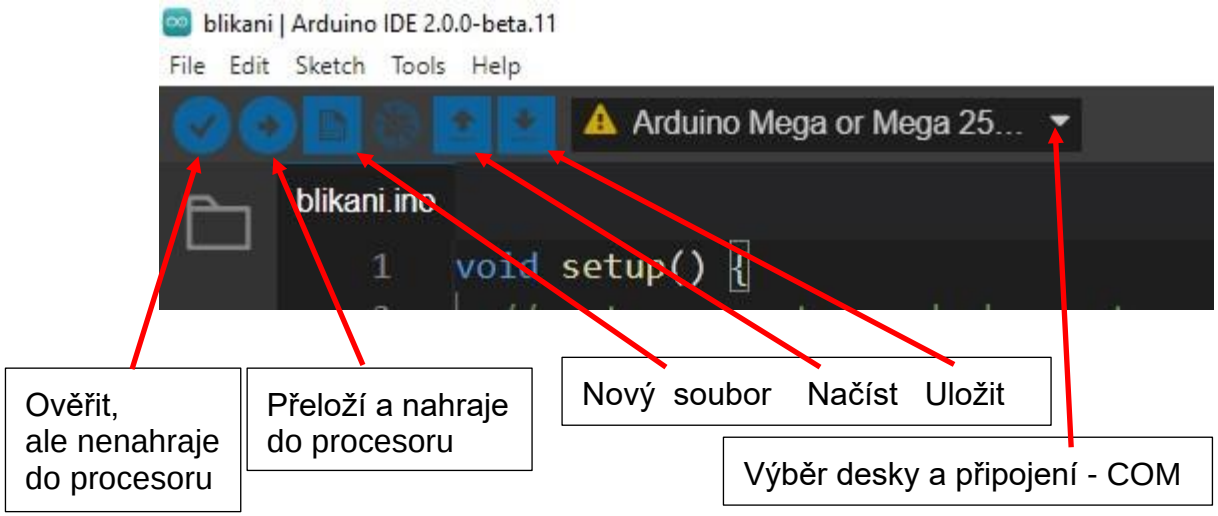
3. Do vyhledání začněte psát uno

4. Vyberte Arduino Uno

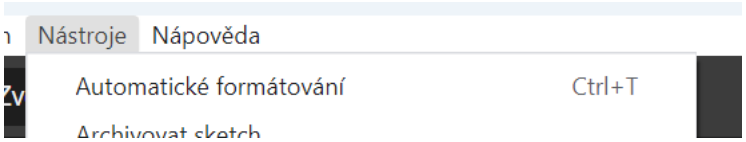
5. Potvrďte port připojení

6. Potvrďte nastavení

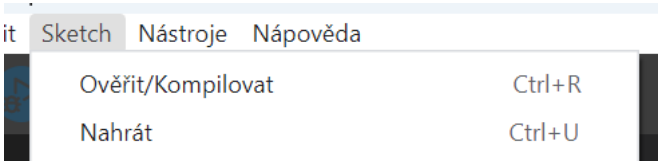

sketch_sep16a | Arduino IDE 2.0.0-beta.11
File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Preferences... Ctrl+Čárka
Advanced
Quit Ctrl+Q
setup(
loop()
10 }
Vytvořte nový projekt
Otevřete již vytvořený projekt
Otevře poslední projekty
Ukázkové programy. Univerzální i pro jednotlivé desky
Ukončení projektu
Uložit projekt
Uložit projekt na jiné místo
Vlastnost celého programu
Ukončit program


blikani | Arduino IDE 2.0.0-beta.11
File Edit Sketch Tools Help
Arduino Mega or Mega 25...
blikani.ino
1 void setup() {
Ověřit, ale nenahraje do procesoru
Přeloží a nahraje do procesoru
Nový soubor
Načíst
Uložit
Výběr desky a připojení - COM

Po vložení programu do editoru je možné použít automatické formátování Ctrl-T

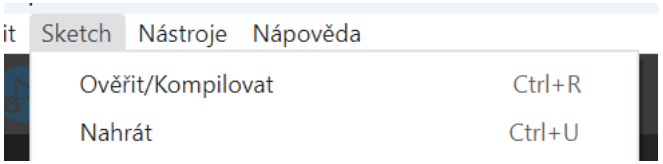


Pro kontrolu napsaného kódu Ctrl-R



nebo

Vložení programu do destičky arduina Ctrl-U



nebo

Proměnné a konstanty

Konstanta – hodnota, které se za celý běh programu nemění. Např. Pi=3,141592…..

Proměnná – hodnota, která se běhu programu může měnit.

Datové typy

Byte	- celé číslo od 0 do 255	1 B
Char	- jedno písmeno	1 B
Integer	- celé číslo od -32 768 do 32 767	2(4) B
Long	- celé číslo, od -2 147 483 648 do 2 147 483 647	4 B
Float	- číslo v pohyblivé řádové čárce jednoduché přesnosti,	4 B
Double	- číslo v pohyblivé řádové čárce dvojnásobné přesnosti,	4(8) B

Proměnné mohou být deklarovány v proceduře, na úrovni modulu nebo v jiných modulech. Pokud jsou deklarovány v proceduře, jsou LOKÁLNÍ a jsou viditelné jen v té proceduře, v níž jsou deklarovány.

Pokud jsou definovány v modulu, jsou viditelné pro všechny procedury tohoto modulu a jsou GLOBÁLNÍ k procedurám v modulu.

Z toho vyplívá, pokud budeme chtít deklarovat proměnnou pro celý program, musíme ji deklarovat v první části. Pokud ji budeme deklarovat v části start, bude platit pouze v této části a nelze ji použít v hlavním programu loop.

Proto, abychom si nemuseli pamatovat číslo výstupu v celém dlouhém programu, můžeme použít proměnnou, nebo konstantu, ale ještě lépe konstantu a do té si uložíte hodnotu. Potom v programu používáme jen jméno.

Můžeme též použít definici jména a ji přiřadit ekvivalent. To znamená, když překladač narazí na slovo zadané jako první v #define nahradí jej v překladu druhým výrazem. Tato možnost zabírá nejméně paměti při překladu.

Ještě jednu výhodu to má. Na začátku programu na jediném místě určím konkrétní pin a když je zapotřebí jej změnit mění se na jediném místě.

Aritmetické operátory

+	sčítání
-	odčítání
*	násobení
/	dělení
=	přiřazení
%	zbytek po dělení

Úkol č. 2 – Proměnná, konstanta definice

Odzkoušejte použití konstant, proměnných nebo definice. V programu odkomentujte pouze jednu možnost a zkuste přeložit a odzkoušet na arduinu. #define je příkaz pro překladač, který v celém programu vyhledá řetězec a nahradí jej jiným. Tímto způsobem můžete v hlavičce programu pojmenovat např. led a určit ji port. Pokud se port změní můžeme v jednom místě změnit hodnotu a ta se promítne do celého programu. V při psaní programů se používá # define. Ostatní dvě možnosti zabírají větší prostor v paměti.

```
/* Blikani*/
// úkol č.2
// const int led1=13;
// int led1=13;
// #define led1 13
void setup() {
  pinMode(led1, OUTPUT);  //pin 13 ve stavu digitálního výstupu
}
void loop() {
  digitalWrite(led1, HIGH);  // zapiš logickou úroveň 1 (HIGH) na pin 13
  delay(1000);               // čekej 1000ms
  digitalWrite(led1, LOW);   // zapiš logickou úroveň 0 (LOW) na pin 13
  delay(1000);               // čekej 1000ms
}
```

Pro funkčnost programu je jedno, kterou možnost použijete. Rozdíl je jen v tom, že některé varianty zabírají více či méně paměti.

V praxi se používá #define, má více výhod a nezabírá místo v RAM paměti (int) nebo v programové paměti (const). Většinou bývá i kratší program při použití #define

Úkol č.3 – Blikání led D13

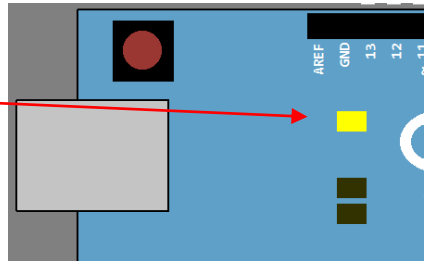
Upravte předešlí program tak, aby byl poměr svícení (logická 1 = HIGH) 2000 ms a zhasnutí byl 200ms (logická 0 = LOW). Všimněte si, jak svítí a nesvítí led. Podle očekávání i simulátoru by při HIGH měla led dlouze svíti a při LOW krátce zhasnout.

```
/* Blikani*/
#define led 13

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

void loop() {
  digitalWrite(led, HIGH);
  delay(2000);
  digitalWrite(led, LOW);
  delay(200);
}
```

Odzkoušejte tento program na arduinu. Blikání led přímo na destičce arduina je správné (led dioda uvnitř).



Na shieldu je však svícení opačné z důvodu zapojení led (zatěžitelnosti portu a pinů). Led diody jsou zapojeny proti kladnému napětí, proto pro rožnutí je zapotřebí logická úroveň LOW. Proto musíme program upravit tímto způsobem. Na začátku programu použijeme #define, které nazveme podle druhu svícení a přiřadíme hodnotu. Potom budeme v celém programu používat pouze námi vytvořené definice pro popsání výstupu pro led

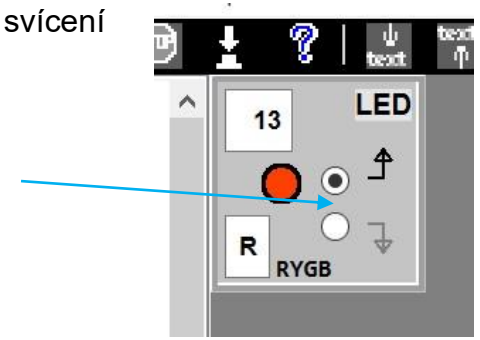
```
/* Blikani*/
#define led 13
#define sviti LOW
#define nesviti HIGH

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

void loop() {
  digitalWrite(led, sviti);
  delay(2000);
  digitalWrite(led, nesviti);
  delay(200);
}
```

Proto, aby i v simulátoru byla správná indikace simulace je zapotřebí přepnout logickou úroveň svícení na LOW.

V simulátoru bude zapotřebí též změnit vlastnosti led diod tímto způsobem přepnutím vlastností diody. Potom bude souhlasit svícení led jak simulátoru tak na sheeldu.



Úkol č. 4 - Blikání s led (pin 10, 11, 12, 13)

Použijte program č.3. V programu měňte hodnotu #define led 13. Odzkoušejte si měnit hodnoty (13,12,11,10) a tím měnit blikání na jinou led diodu. V simulátoru použijte čtyři diody a připojte každou na jiný pin (10, 11, 12, 13). Uložte variantu pro led na D10.

Úkol č. 5 - Blikání dvou diod –
výstražná světla na železničním přejezdu

Napište program, který bude blikat s dvěma diodami (D12 + D13). Nikdy nebudou svítit obě diody. Vždy bude svítit alespoň jedna dioda. Délka svícení jedné diody bude 1s.

Úkol č. 6 - Blikání diod v2

Upravte úkol č.5 tak, že D10 má stejný stav jako D12 a D11 má stejný stav jako D13.

Úkol č. 7 - Blikání čtyř diod

Napište program, který bude blikat s čtyřmi diodami (D10 + D11 + D12 + D13). Nikdy nebudou svítit dvě a více diod. Vždy bude svítit jen jedna dioda. Délka svícení jedné diody bude 1s.

Podmínka if - else

Příkaz if testuje, zda bylo dosaženo určité podmínky, třeba zda analogová hodnota je větší než zadaná, a provádí všechny příkazy uvnitř závorek, pokud tvrzení je pravdivé (TRUE). Pokud tvrzení pravdivé není (FALSE), program příkazy uvnitř závorek přeskočí.

```
if (cislo1 > cislo2) {
        příkazy;
}
```

Příklad porovnává hodnotu proměnné cislo1 s hodnotou cislo2, která může být opět buď proměnná nebo konstanta. Pokud je výsledek porovnání hodnot v závorce pravda (TRUE), jsou vykonány příkazy uvnitř složených závorek. Pokud ne, program je přeskočí a pokračuje za nimi. Dejte si pozor na náhodné použití '=' místo '==' v příkazu if (x = 10), zápis je syntakticky správný, ale nastavuje hodnotu proměnné x na hodnotu 10 a výsledkem je tedy vždy pravda (TRUE). Místo '=' je nutno použít výraz '==', tedy (x == 10), který jen testuje, zda x se rovná hodnotě 10, nebo ne. Myslete na '=' jako na "rovná se" na rozdíl od '==' které znamená "se rovná".

if .. else

Operace 'if .. else' umožňuje rozhodování stylem 'buď – nebo' a větvení programu podle výsledku operace. Například, pokud chcete otestovat stav digitálního vstup a pak provést nějakou činnost, pokud je vstupní pin ve stavu HIGH nebo naopak provést něco jiného, pokud je vstupní úroveň nízká, můžete to zapsat tímto způsobem:

```
if ( digitalRead (A1) == HIGH) {
        Program - pravda;
 } else {
        Program - nepravda;
}
```

V tomto příkladu 'if' pouze zkontroluje, zda opravdu má zadaný vstup logickou úroveň HIGH neboli 5 V.

Digitální vstup – tlačítko

V praxi se používají dva druhy zapojení tlačítek:

1. Tlačítko připojené na Ucc a pin.
   Nevýhoda: Při nestisknutém tlačítku možná elektromagnetická indukce na vodiči. Můžou nastat náhodné logické stavy na vstupu. Jde tomu zabránit připojením rezistoru na GND a pin tak zvaný puul-down rezistor.
   Nevýhoda: Pokud potřebujete spínat vstup místo tlačítkem tranzistorem je složitější zapojení
   Nevýhoda: Nutno dodržet stejné Ucc napětí pro procesor i tlačítko (+ 0,5V max). Výhoda: Na vstupu čteme logické úrovně přesně jak je stav tlačítk (stisknuto logická 1 na vstupu).

2. Tlačítko připojené na GND a pin.
   Nevýhoda: Při nestisknutém tlačítku možná elektromagnetická indukce na vodiči. Můžou nastat náhodné logické stavy na vstupu. Jde tomu zabránit připojením rezistoru na Ucc a pin takzvaný puul-up rezistor. Je možné jej zapnout i v procesoru.
   Výhoda: Pokud potřebujete spínat vstup místo tlačítkem tranzistorem je stačí k tomu jakýkoliv NPN tranzistor zapojený proti GND. Úbytky napětí na tranzistoru nejsou moc krytické.
   Nevýhoda: Čteme opačné logické úrovně (stisknuto logická 0 na vstupu)

V praxi se nejčastěji používá zapojení 2 s využitím puul-up rezistorů použitých přímo v procesoru. Odpadá další externí součástka a tlačítko se připojí přímo na GND a pin. Pro zapojení s velkým rušením se doporučuje připojení externího rezistoru a tím i možnost zvětšení proudu přes rezistor.

V praxi se setkáme ještě s jedním nepříjemným jevem, a to je zákmit tlačítka. To nastane, když přepneme mechanicky tlačítko a jeho prvotní dotyk není úplně ideální. Procesor však je dostatečně rychlí a zaregistruje tyto zákmity. Řešení je několik:

1. Snížit frekvenci procesoru – nelze využít matematický výkon procesoru (nepoužitelné)

2. Použít kvalitní tlačítka – i kvalitní tlačítka po určité době ztratí své vlastnosti.

3. Zvětšit proud přes puul-down nebo puul-up rezistory – velkými proudy se ničí tlačítka (nejsou na to stavěna)

4. Software napsat tak, aby tento jev hlídal a upravil vstupy do programu. V praxi dost často používaný způsob. Není finančně vůbec náročný, pouze se využije více vlastností procesoru.

5. Využití přerušení procesoru.

6. Hadware ošetření. Použití několika hradel a kondenzátorů. Je zapotřebí více součástek i integrovaných obvodů, větší a složitější DPS, finančně více náročné.

Úkol č. 10 – Test tlačítka

Pro naše potřeby budeme číst tlačítka bez nutnosti kompenzovat zákmity tlačítka. Proto může dojít k zákmitu.

```
/* Tlacitko a led*/
#define led 13
#define sviti LOW
#define nesviti HIGH
#define tlacitko A1

int pomocna;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(tlacitko, INPUT_PULLUP);
  //pin A1 ve stavu digitálního vstupu + pull-up rezistor v procesoru
}
void loop() {
  pomocna = digitalRead(tlacitko);  //přečti stav log. úrovně
  if (pomocna == 1) {               // porovnej s číslem 1
    digitalWrite(led, sviti);
  } else {
    digitalWrite(led, nesviti);
  }
}
```

Protože máme připojený tlačítko proti mínusu je logika přivedená na pin otočená. Logická 1 při nestisknutém tlačítku. Proto, aby program fungoval tak, že při stisku tlačítka se rozsvítí led jsou možné úpravy tohoto programu:

Úkol č. 11

V testování podmínky „`if (pomocna==1)`“ vyměnit 1 za 0. Otočení (negace) podmínky

Úkol č. 12

Prohodit jednotlivé části podmínky

„`digitalWrite(led1, HIGH)`“ ⇔ „`digitalWrite(led1, LOW)`“ a

„`digitalWrite(led1, LOW)`“ ⇔ „`digitalWrite(led1, HIGH)`“

Úkol č. 13

Programově čistější je již při čtení údajů z pinu tuto hodnotu znegovat. Výkřičník totiž znamená logickou negaci výsledku.

```
„pomocna=!digitalRead(tlacitko1);
```

Toto řešení se používá v praxi.

Úkol č. 14 - Tři tlačítka a tři led

Vyjděte z programu č.13. Napište program, který bude signalizovat stlačení jednotlivých tlačítek (A1, A2, A3) na diodách (D11, D12, D13). Každému tlačítku odpovídá jedna dioda.

Porovnávací operátory

Porovnávací operátory slouží k zápisu podmínek. Jedná se o systém značek, které jsou pro počítač srozumitelné. Výsledkem porovnávací operace je logická hodnota true, nebo false. Rozlišujeme šest operátorů.

A == B	A je rovno B. Vrátí hodnotu true, pokud A má stejnou hodnotu, jako B.
A != B	A není rovno B. Vrátí hodnotu true, pokud má A jinou hodnotu než B.
A < B	A je menší než B. Vrátí hodnotu true, pokud je A menší než B.
A > B	A je větší než B. Vrátí true, pokud je A větší než B.
A <= B	A je menší nebo rovno B. Vrátí true, pokud je A menší nebo rovno B.
A >= B	A je větší nebo rovno B. Vrátí true, pokud je A větší nebo rovno B.

Složené podmínky

Někdy dospějeme do situace, ve které je potřeba pracovat s nějakou složitější podmínkou. K tomuto účelu slouží tzv. logické operátory. Můžeme si je představit jako definici vztahu mezi více porovnávacími operátory.

X && Y	a (konjunkce).	Výsledkem je true pouze v případě, když jsou true X i Y.
X || Y	nebo (disjunkce).	Výsledkem je true v případě, kdy je alespoň jedna z X a Y true.
!X	negace.	Výsledkem je true, pokud je X false a naopak.

Úkol č. 15 - Tři tlačítka a čtyř led

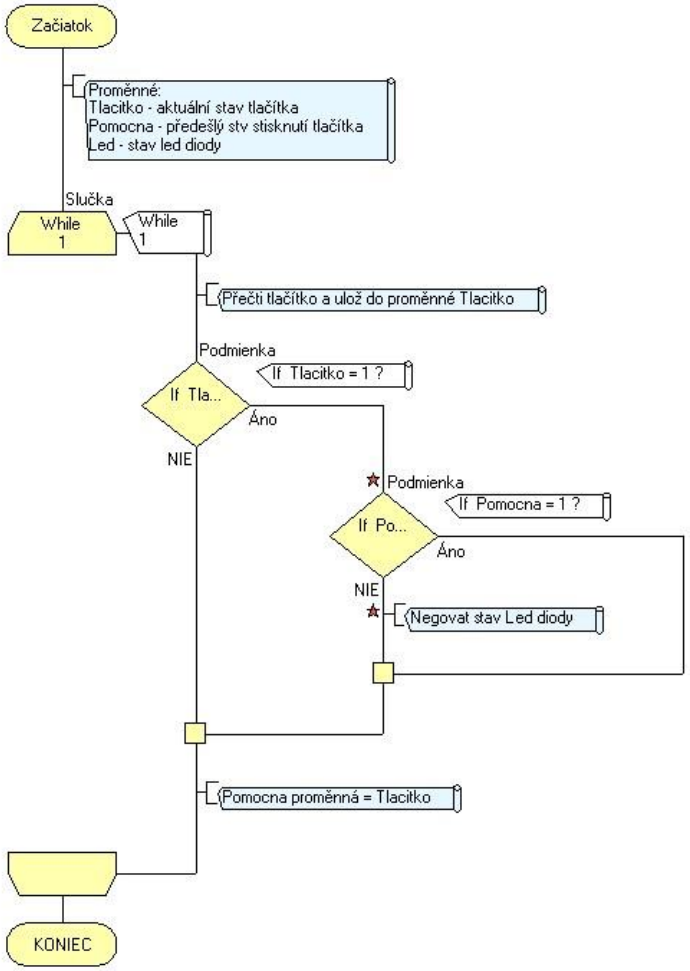
Upravte předchozí úkol tak, že přidáte funkci, kdy dioda D10 bude svítit jen, když budou stlačeny všechny tři tlačítka.

Úkol č. 16 - Tři tlačítka a čtyř led v2

Upravte předchozí úkol tak, že program, bude kontrolovat kombinaci stisku dvou libovolných tlačítek a indikovat je na D10.

Úkol č. 17 - Tlačítko s pamětí

Napište program, který bude při stisku a uvolnění tlačítka zapne D13. Při dalším zamáčknutí a uvolnění vypne D13. Bude to vlastně tlačítko s pamětí. Budete potřebovat tři proměnné, ve kterých budete uchovávat minulý stav tlačítka a aktuální stav a stav výstupu. Podle stavu tlačítka budete měnit výstup.



Úkol č. 18 – 3x tlačítko s pamětí

Rozšiřte tuto funkci i pro ostatní tlačítka a led diody.

Úkol č. 19 – 3x tlačítko s pamětí + 4x led

Upravte předchozí úkol tak, že přidáte funkci, kdy dioda D10 bude svítit jen, když budou svítit všechny tři led diody tlačítek.

Čas

V základní výbavě mají desky Arduino čtyři funkce pro práci s časem. Jsou to funkce delay(), delayMicroseconds(), millis() a micros(). První dvě a druhé dvě fungují na stejném principu, jenom pracují s jinými jednotkami. Jsou to milisekundy a mikrosekundy. Důležité je si připomenout převodní vztah mezi jednotkami času kdy: 1 sekunda = 1 000 milisekund = 1 000 000 mikrosekund.

delay()

S touto funkcí jsme se již setkali. Má jediný parametr, a to čas čekání v milisekundách. Rozsah parametru je od 0 do 4,294,967,295. Velkou nevýhodou funkce delay i následující funkce delayMicroseconds() je fakt, že dojde k zastavení téměř veškeré činnosti (pozastavení čtení hodnot ze senzorů, nemožnost ovládat logické hodnoty na pinech atd.). Nedojde však k zastavení těch funkcí, které nejsou přímo závislé na procesoru. Jedná se zejména o příjem informací z Rx linky, kdy se přijatými byty naplňuje buffer a ke zpracování dojde až po skončení funkce delay a také o funkci analogWrite(). Generování PWM signálu totiž probíhá mimo hlavní blok procesoru.

delayMicroseconds()

Funkce je obdobná, jako delay(), jenom s tím rozdílem, že parametr je zde čas v mikrosekundách. Rozsah parametru je od 0 do 65,535.

millis()

Pomocí funkce millis() se dá zjistit hodnota uložená ve vnitřním časovači procesoru. Zde je uchována informace o délce běhu programu od jeho spuštění. Tato funkce tedy nepotřebuje žádný parametr a vrací počet milisekund od začátku programu. Tento počet však není nekonečný. Maximální vrácená hodnota je 4,294,967,295. Po překročení dojde k takzvanému přetečení časovače, který poté znovu začne počítat od nuly. Funkce millis() se využívá například tam, kde je třeba čekat, ale není žádoucí, aby byl přerušen chod programu.

Poznámka: K přetečení časovače dojde přibližně jednou za 50 dní. (4 294 967 295 ms = 4 294 967 s = 71 582 min = 1193 h = 49,7 dní)

micros()

Funkce micros() je stejná jako millis(), pouze vrací hodnotu v mikrosekundách. Rozsah hodnot je stejný, ale jelikož platí, že 1 milisekunda = 1000 mikrosekund, doba přetečení bude tisíckrát menší, tedy asi 71,5 minuty. Nutno dodat, že rozlišení funkce je u 16 MHz procesorů 4 mikrosekundy a u 8 MHz 8 mikrosekund. Výstupem funkce tedy bude násobek čtyř, nebo osmi.

Poznámka: Možná se ptáte, proč jsou maximální hodnoty parametrů, nebo vrácených čísel takové, jaké jsou. Je tomu tak, protože funkce pro práci s časem používají dva datové typy. Jsou to unsigned int a unsigned long. Datový typ unsigned int má stejný rozsah hodnot jako int, jenom je tento rozsah posunut směrem do kladných čísel. Typ int může uchovat čísla od -32,768 do 32,767. U typu unsigned int se nepracuje se zápornými čísly. Rozsah je u něj tedy od 0 do 65,535. Stejná situace je i u unsigned long, jen s větším rozsahem.

delay() pozastaví program na dobu milisekund zadanou jako parametr.

millis() , na druhé straně, je funkce, která vrací množství milisekund, které uplynuly od spuštění programu. Na první pohled můžete pochybovat o užitečnosti této funkce. Skutečnost je taková, že je velmi užitečná v mnoha scénářích, často „nahrazuje“ delay() .

Proč použít millis () místo delay ()? Nyní se podíváme na dvě výhody s millis() ve srovnání se delay() .

1. Přesné načasování

První výhoda, kterou probereme, je **přesné načasování**. S millis() můžeme zajistit, že smyčka běží tak často, jak chceme, bez ohledu na dobu provádění jiných činností procesoru (samozřejmě pokud je doba provedení kratší než požadovaná doba). S delay() to není možné, protože nevíme, jak dlouho je doba provedení smyčky. Přesné načasování, je velmi užitečné, mimo jiné při vzorkování na určité frekvenci nebo při běhu filtrů.

2. Neblokující

Další výhodou systému millis() je, že nám nezabrání spuštění kódu během „čekání“.

Řekněme, že chceme blikat led diodou (1x 1s) a přitom testovat stav tlačítek. Při použití delay(1000) bude procesor čekat 1s, ale nemůže testovat změnu tlačítka. To provede 1x za 1s. Co se ale stalo mezi tím neví proto je lepší použít millis()

Úkol č. 20 – delay() – blikání led D13 + tlačítko

Takto by vypadal program s použitím delay() – zkuste reakci na tlačítko

```
/* Tlacitko a led*/
#define led_t 12
#define led_b 13
#define sviti LOW
#define nesviti HIGH
#define tlacitko A1
int pomocna;
#define blikani 1000

void setup() {
  pinMode(led_t, OUTPUT);
  pinMode(led_b, OUTPUT);
  pinMode(tlacitko, INPUT_PULLUP);  //pin A1 ve stavu digitálního vstupu + pull-up rezistor v procesoru

}
void loop() {
  pomocna = !digitalRead(tlacitko);  //přečti stav log. úrovně
  if (pomocna == 1) {                // porovnej s číslem 1
    digitalWrite(led_t, sviti);
  } else {
    digitalWrite(led_t, nesviti);
  }
  delay(blikani);
  digitalWrite(led_b, !digitalRead(led_b));  //přečte stav led, zneguje a potom uloží

}
```

Úkol č. 21 – millis() – blikání led D13 + tlačítko

Tímto způsobem můžeme kontrolovat třeba stlačení tlačítka a přitom blikat

```
/* Tlacitko a led*/
#define led_t 12
#define led_b 13
#define sviti LOW
#define nesviti HIGH
#define tlacitko A1
int pomocna;
unsigned long cas;
#define blikani 1000
void setup() {
  pinMode(led_t, OUTPUT);
  pinMode(led_b, OUTPUT);
  pinMode(tlacitko, INPUT_PULLUP);  //pin A1 ve stavu digitálního vstupu + pull-up rezistor v procesoru
  cas = millis() + blikani;
}
void loop() {
  pomocna = !digitalRead(tlacitko);  //přečti stav log. úrovně
  if (pomocna == 1) {                // porovnej s číslem 1
    digitalWrite(led_t, sviti);
  } else {
    digitalWrite(led_t, nesviti);
  }
  if (cas < millis()) {
    cas = cas + blikani;
    digitalWrite(led_b, !digitalRead(led_b));
  }
}
```

Úkol č. 22 – smyčka s počítáním – blikání led D13 + tlačítko

Tento způsob má nevýhodu v tom, že je nutné odhadnou počet průchodů ve smyčce. Tento počet bude jiný pro simulátor i arduino. Bude jiný i po prodloužení hlavní smyčky.

```
/* Tlacitko a led*/
#define led_t 12
#define led_b 13
#define sviti LOW
#define nesviti HIGH
#define tlacitko A1
int pomocna;
unsigned long cas;
#define blikani 50000
void setup() {
  pinMode(led_t, OUTPUT);
  pinMode(led_b, OUTPUT);
  pinMode(tlacitko, INPUT_PULLUP);  //pin A1 ve stavu digitálního vstupu + pull-up rezistor v procesoru
}
void loop() {
  pomocna = !digitalRead(tlacitko);  //přečti stav log. úrovně
  if (pomocna == 1) {                // porovnej s číslem 1
    digitalWrite(led_t, sviti);
  } else {
    digitalWrite(led_t, nesviti);
  }
  cas = cas + 1;
  if (cas > blikani) {
    digitalWrite(led_b, !digitalRead(led_b));
    cas=0;
  }
}
```

Úkol č. 22 – přerušení – blikání led D13 + tlačítko

Zde je využito přerušení v procesoru. Toto přerušení může být vyvoláno změnou logické úrovně určitého vstupu nebo uplynutím času. V tomto ukázkovém programu je použito uplynutí nastaveného času. V hlavním programu se nejprve dokončí instrukce, zavolá se podprogram (funkce) blikat. Po skončení funkce se pokračuje dalším příkazem z hlavního programu.

```
#include <TimerOne.h>
/* Tlacitko a led*/
#define led_t 12
#define led_b 13
#define sviti LOW
#define nesviti HIGH
#define tlacitko A1
int pomocna;
#define blikani 1000000
void setup() {
  pinMode(led_t, OUTPUT);
  pinMode(led_b, OUTPUT);
  pinMode(tlacitko, INPUT_PULLUP);  //pin A1 ve stavu digitálního vstupu + pull-up rezistor v procesoru
  Timer1.initialize(blikani); //inicializuje přerušení a nastaví jak často se bude volat
  Timer1.attachInterrupt(blikat); //nastaví funkci která se bude volat v přerušení
}
void loop() {
  pomocna = !digitalRead(tlacitko);  //přečti stav log. úrovně
  if (pomocna == 1) {                // porovnej s číslem 1
    digitalWrite(led_t, sviti);
  } else {
    digitalWrite(led_t, nesviti);
  }
}
void blikat (){
  digitalWrite(led_b, !digitalRead(led_b));
}
```

Sériová komunikace – odeslání dat

Při propojení destičky Arduino Uno a PC se nainstaluje COM port. Přes tento port je možno zasílat i přijímat data do počítače. Pro správnou funkci je zapotřebí nejprve v části start inicializovat sériovou komunikaci

Serial.begin(9600); 9600 je nejčastěji používaná přenosová rychlost pro komunikaci. Další standartní rychlosti jsou např. 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200. Tuto rychlost musí zvládat jak vysílací strana ta i přijímací. V sériovém protokolu není možno se na začátku domluvit na rychlosti. Je nutno ji dopředu pevně zvolit.

Odeslání dat po sériové lince do počítače

Serial.print( ); Odešle to co je v závorce
Serial.println( ); Odešle to co je v závorce a na konci řádky odešle znak pro zalomení řádky

Parametry při použití tisku s **Serial.print( ) nebo Serial.println( )** :

```
Serial.print(97);                 //vypíše: 97
Serial.print(2.123456);           //vypíše: 2.12
Serial.print('a');                //vypíše: znak "a"
Serial.print("ABCDEFGHIJ");       //vypíše: "ABCDEFGHIJ"
Serial.print(ABCDEFGHIJ);         //vypíše obsah proměnné ABCDEFGHIJ
```

Nepovinné parametry při tisku – číselná soustava:

```
Serial.print(97, DEC);        //vrátí: 97
Serial.print(97, BIN);        //vrátí: 1100001 (což je 97 ve dvojkové soustavě)
Serial.print(97, OCT);        //vrátí: 141 (=97 v osmičkové soustavě)
Serial.print(97, HEX);        //vrátí: 61 (=97 v šestnáctkové soustavě)
```

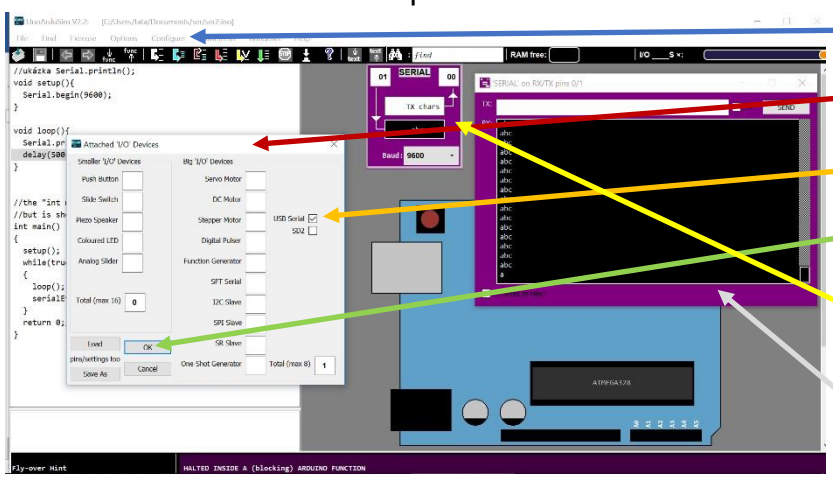
Nepovinné parametry při tisku – délka čísel

```
Serial.print(4.56789, 0);         //vrátí: 5
Serial.print(4.56789, 1);         //vrátí: 4.6
Serial.print(4.56789, 3);         //vrátí: 4.568
```

Ukončení sériové komunikace

Serial.end();

Nastavení simulátoru pro zobrazení sériové komunikace:



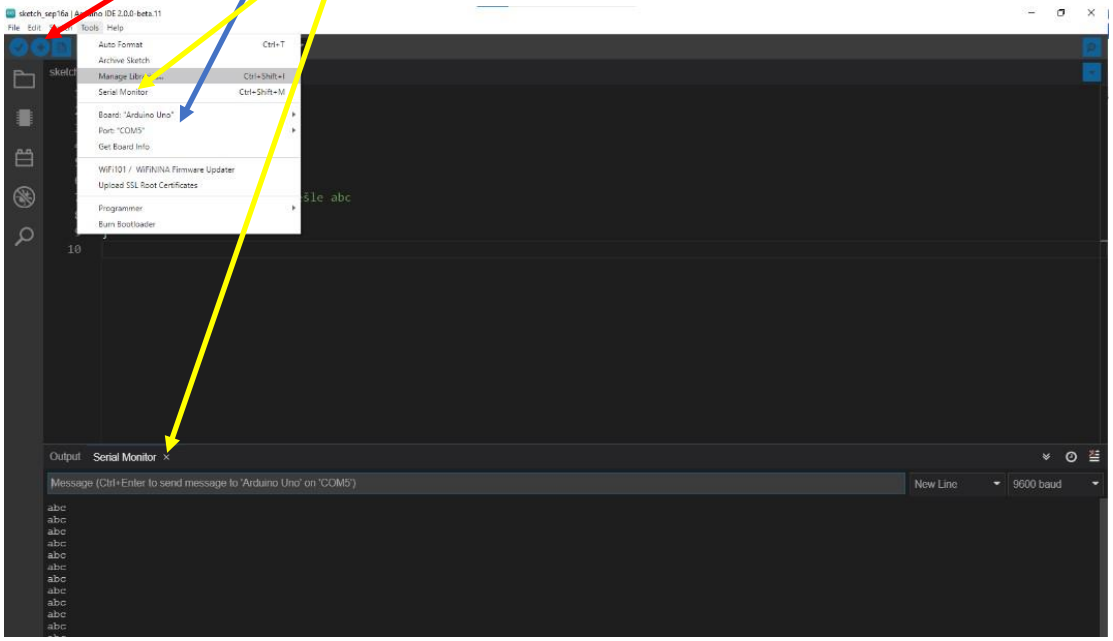
1. Configure
2. ‚I/O' Devices
3. USB Serials
4. OK
5. Kliknout na komponentu
6. Otevře se okno

```
//ukázka Serial.println(); void setup(){
      Serial.begin(9600);
}
void loop(){
      Serial.println("abc "); //odešle abc
      delay(1000); //čeká 1000ms
}
```

Odzkoušejte tento program na simulátoru.
Budete vidět, jak program každou 1s odešle abc

Odzkoušení na Arduinu :

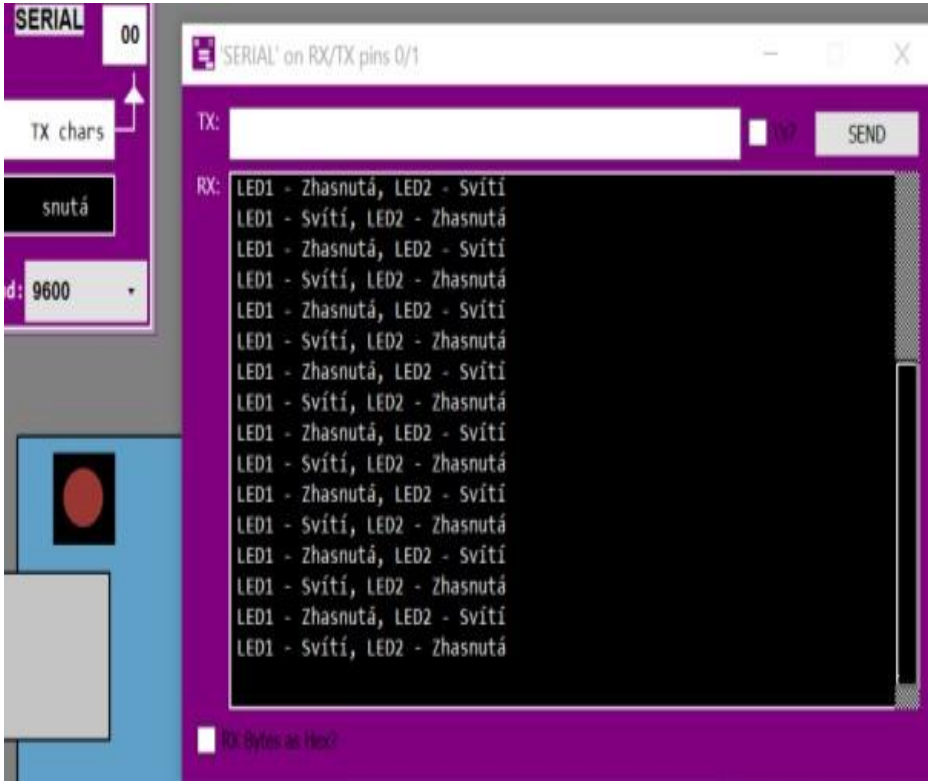
1. Zkontrolovat připojení desky na USB
2. Nahrát do procesoru
3. Otevřít sériový monitor

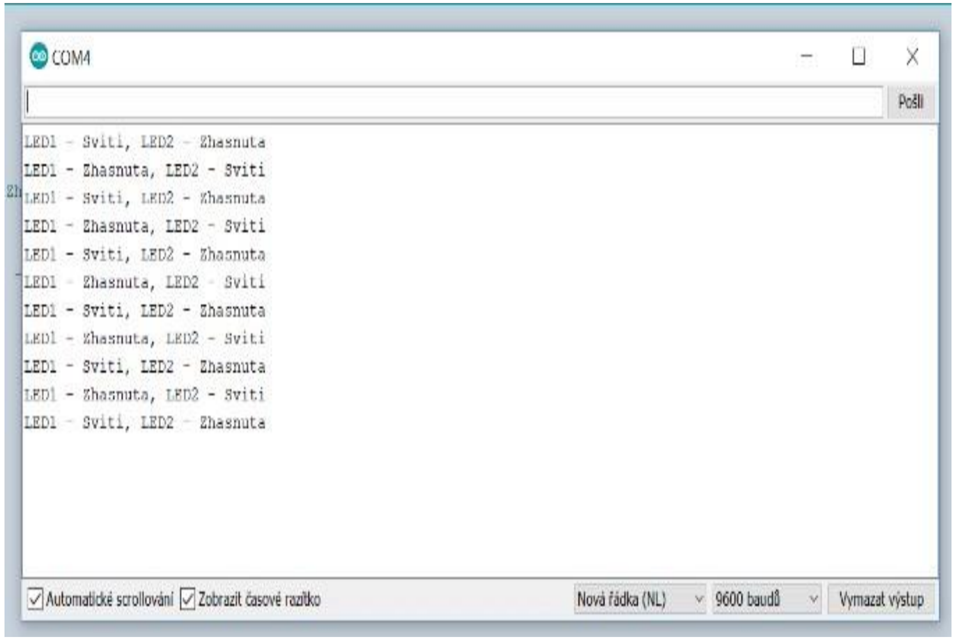


Úkol č. 30 – Odeslání dat, blikání led

Upravte úkol č.5 tak, že pomocí sériové linky vyšlete stav jednotlivých led diod.

Odzkoušejte jak na simulátoru tak i na Arduinu.





Úkol č. 31 – Odeslání dat – úkol 7

Upravte tímto způsobem úkol č. 7, který jste vytvořili. Zasílejte aktuální stavy led diod a při výpisu stavu tlačítek použijte proměnnou (výpis 0/1).

Úkol č. 32 – Odeslání dat – úkol 16

Upravte tímto způsobem úkol č. 16 který jste vytvořili. Zasílejte aktuální stavy led diod a při výpisu stavu tlačítek použijte proměnnou (výpis 0/1).

Úkol č. 33 – Odeslání dat - – úkol 19

Upravte tímto způsobem úkol č. 19, který jste vytvořili. Zasílejte aktuální stavy led diod a při výpisu stavu tlačítek použijte proměnnou (výpis 0/1).

Sériová komunikace – příjem dat

Nyní se podíváme na možnosti čtení informací, které do Arduina posílá PC, nebo jiné zařízení. Pro správnou funkci je zapotřebí nejprve v části start inicializovat sériovou komunikaci

Serial.begin(9600); 9600 je nejčastěji používaná přenosová rychlost pro komunikaci. Další standartní rychlosti jsou např. 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200. Tuto rychlost musí zvládat jak vysílací strana ta i přijímací. V sériovém protokolu není možno se na začátku domluvit na rychlosti. Je nutno ji dopředu pevně zvolit.

Nejdříve si musíme říci o tom, jak vlastně posílání dat vypadá na nejnižší úrovni. Když do Arduina přijdou informace po sériové lince, nezpracovávají se hned, ale jsou uchovány v „zásobníku“ (anglicky buffer). Ten dokáže uchovat až 64 bytů. Čtení poté probíhá tak, že se vezme první byte z bufferu, zpracuje se procesorem, jeho místo se uvolní a uchované byty se posunou dopředu. Poté se vezme další byte, zpracuje se atd.

Když chceme Arduinu odeslat nějakou hodnotu, nejjednodušším způsobem je napsat ji do textového pole v horní části Serial monitoru. Jelikož se odesílá ASCII hodnota znaků, má každý znak (i čísla) vlastní byte paměti. Když posíláme číselnou hodnotu musíme prvně převést ASCII znaky na čísla a ty potom složit do číselné hodnoty.

Serial.available() Tento příkaz přečte počet znaku ve vyrovnávacím bufferu.

Serial.read(); Tento příkaz přečte 1 znak nebo ASCII hodnotu znaku (záleží na typu poměnné) a předá ji proměnné, kterou uvedete před tímto příkazem.

Úkol č. 35 – Přečti a odešli znak po sériové lince

Přečti znak a potom jej pošli na zpět po sériové lince do terminálu. Vyzkoušejte použít proměnnou typu char i integer. Porovnejte výsledek přenosu.

```
//int pomocna;
char pomocna;

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

void loop(){
      if (Serial.available()>0){
            pomocna=Serial.read();
            Serial.println(pomocna);
      }
}
```

Inicializace sérové linky pro rychlost 9600

Zjistí počet znaků v bufferu. Pokud je 1 nebo více znaků v bufferu vykoná se podmínka.

Přečti znak a ulož do proměnné „pomocna“

Pošli proměnnou „pomocna“ po sériové lince zpět

Pokud použijete definici proměnné typu char výstup bude znovu písmeno, i další zpracování v programu musí být jako text

Pokud ale použijete typ integer bude návratová hodnota rovna ASCII kódu přijmutého symbolu. Proto když se podíváte do ASCII tabulky je pro znak „0“ hodnota 48, pro znak „1“ hodnota 49. Proto když budeme chtít testovat odeslané znaky „0“ a “1“ a chceme mít hodnoty 0 a 1 musíme od přečtené hodnoty znaku odečíst 48.

Úkol č. 36 – Obsluha led D13 po sériové lince

Napište program, který bude obsluhovat led diodu D13. Pokud pošlete na sériové lince znak 1 led svítí, když pošlete znak 0 led zhasne. Informujte uživatele o stavu D13 na sériové lince. Pozor ! při odesílání dat z monitoru na PC je odeslán na konci přenosu znak Enter(kod 13). Tento znak znamená nový řádek (ukončení přenosu). Proto je zapotřebí tento znak ignorovat při přijímání dat.

Úkol č. 37 – Obsluha led D13 po sériové lince v2

Upravte předešlý program tak, že D13 půjde obsluhovat jak tlačítkem A1, tak příkazem po sériové lince. Po sériové lince povolíte nebo zakážete používání tlačítka pro ovládání led diody. Informujte uživatele o stavu D13 a A1 na sériové lince.

Analogový vstup

Jsou to tedy piny označené písmenem A (například A4). Čtení analogových hodnot je užitečné u různých senzorů (teplota, vlhkost atd.). Většina desek Arduina má rozlišení 10 bitů, což odpovídá hodnotám od 0 do 1023 Proměnná musí být typu integer.

proměnná = analogRead(pin);

Úkol č. 40 – Načti analogový vstup

Napište program, který bude číst analogovou hodnotu fotorezistoru připojeného na A4. Načtenou analogovou hodnotu posílejte po sérové lince.

Úkol č. 41 – Načti analogový vstup + D9, D10

Upravte úkol 30. Pokud bude načtená hodnota větší než 600 led na D10 svítí a D9 zhasnutá. Pokud je hodnota menší než 600 led na D9 svítí a D10 zhasnutá.

Úkol č. 42 – Načti analogový vstup + D9, D10 v2

Upravte program tak že, pokud je hodnota rovna 600 svítí obě dvě D9, D10. To lze vyzkoušet jen v simulátoru.

Úkol č. 43 – Načti analogový vstup + D9, D10 v3

Napište program, aby platila následující tabulka:

Vstup ➔	0 – 299	300 – 599	600 - 799	800 – 1024
D9	OFF	ON	OFF	ON
D10	OFF	OFF	ON	ON

To lze vyzkoušet jen v simulátoru.

Úkol č. 44 – Načti 2x analogový vstup + D9, D13

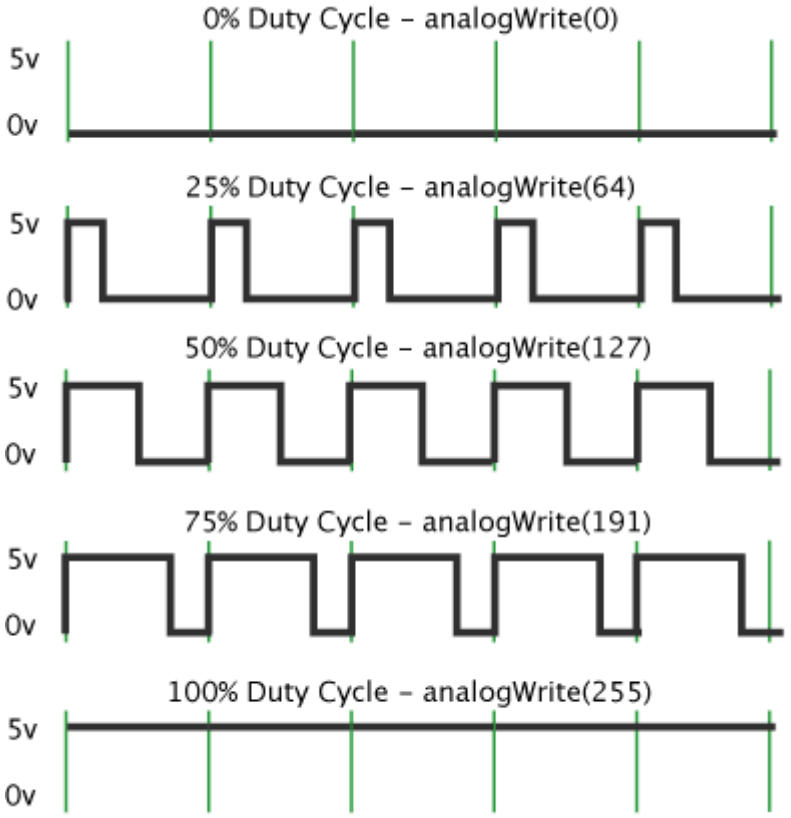
Analogový vstup A0 je trimr. Analogový vstup A4 je fotorezistor. Trimr slouží jako rozhodovací úroveň a porovná se s fotorezistorem. Když je osvětlení menší než nastavené, svítí D9. Když je osvětlení větší než nastavené, D9 je zhasnutá. D13 neguje stav D10. Údaje posílejte po sériové lince.

Úkol č. 45 – Načti 2x analogový vstup + D9, D13

Pomocí dvou tlačítek (+ a -) nastavte rozhodující úroveň pro analogový vstup A4. Když je osvětlení menší než nastavené, svítí D9. Když je osvětlení větší než nastavené, D9 je zhasnutá. D13 neguje stav D10. Údaje posílejte po sériové lince.

Analogový výstup - PWM (D10, D11)

Jak už z názvu vyplývá, jedná se o funkci sloužící k nastavení „analogové“ hodnoty na pinu. Můžeme ji použít pouze na pinech označených PWM (u Arduina UNO jsou to piny: 3, 5, 6, 9, 10, 11). Používá se u ní syntaxe analogWrite(číslo_pinu, hodnota), kdy hodnota může být v rozsahu 0 až 255. Slovo analogové jsem dal do uvozovek, protože se ve skutečnosti o žádné analogové hodnoty nejedná. Pokud bychom chtěli skutečně analogovou hodnotu v rozsahu například 0-5V, museli bychom použít externí D/A převodník. Tato funkce totiž na vybraných pinech generuje PWM signál, což je jakási digitální „náhražka“ analogového signálu. Ta v praxi funguje tak, že rychle střídá 0 a 5V. To se projeví sníženou ‚účinností‘. LED svítí slaběji (ve skutečnosti rychle bliká a střídá pouze dva stavy napětí a snížená intenzita je způsobena setrvačností oka), motor se točí pomaleji atd. Podle poměru času, ve kterém je na výstupu +5V ku stavu 0V se pak odvíjí intenzita svícení LED diody a podobně. Při volání funkce analogWrite(pin, 127) je tedy přibližně 50% času nastaveno napětí +5V a 50% času 0V.



Úkol č. 50 – PWM (D10) v.1

Program, který načte analogovou hodnotu z A4 a pošle ji na výstup jako PWM. Načtená hodnota z analogového vstupu se musí vydělit 4-ma, z důvodu převodu 10-ti bytového čísla na 8-mi bytové

```
#define led1  10  //pin s LED diodou
#define led2  11  //pin s LED diodou
#define pot   A4  //pin s fotorezistorem
int val;          //proměnná připravená k uchování hodnot

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(pot, INPUT);
}

void loop() {
  val = analogRead(pot)/4; //čtení hodnoty na A4 a úprava rozsahu
  analogWrite(led1, val);  //generování PWM
}
```

Úkol č. 51 – PWM (D10) v.2

Z úkolu č.40 zjistěte nejmenší a největší hodnotu, kterou dodává fotorezistor na A4 a zapište si jí.

Minimální hodnota (fotorezistor_min):

Maximální hodnota (fotorezistor_max):

Pro převod použijeme funkci map()

Slouží k rovnoměrnému „roztažení“, nebo „zmáčknutí“ celé stupnice. Dá se použít například k úpravě hodnoty získané při čtení analogového vstupu (0 – 1023) nebo jenom určitého rozsahu (190-850) a jejich přepočítání pro použití ve funkci analogWrite, která pracuje s hodnotami od 0 do 255. Syntaxe je následující:

val = map(nactena_hodnota, fotorezistor_min, fotorezistor_max, 0, 255);

0 – minimální nová hodnota (minimální hodnota může být větší maximální hodnota. Potom je přírůstek klesající.)

255 – maximální nová hodnota

```
#define led1  10  //pin s LED diodou
#define led2  11  //pin s LED diodou
#define pot   A4  //pin s fotorezistorem
int val;         //proměnná připravená k uchování hodnot
#define fotorezistor_min 190
#define fotorezistor_max 850

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(pot, INPUT);
}

void loop() {
  val = analogRead(pot); //čtení hodnoty na A4
  analogWrite(led1, map(val, fotorezistor_min, fotorezistor_max, 0, 255));  //generování PWM
}
```

Úkol č. 52 – PWM (D10) + sériová linka

Upravte předešlý úkol, aby posílal analogový vstup na sériovou linku.

Úkol č. 53 – PWM (D10, D11) + sériová linka

Upravte předešlý úkol, že jas led na D11 je opačně než na D10. Informace o D10 i D11 odesílejte po sériové lince

Generování zvuku

Pro výstup zvuku můžeme použít pin 3 a příkaz

tone (výstupní_pin, frekvence, délka_tonu); // délka tónu není povinný parametr. Pro hraní určitého tonu slouží následující tabulka na konci popisu.

Vypne hraní tonu na výstupním pinu **noTone**(výstupní_pin);

```
Veřejné konstanty pro generování tónů

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Úkol č. 60– Tón při stisku

Při stisku tlačítka zahraj tón o délce 0,5s

Úkol č. 61 – Tón při stisku v2

Při stisku tlačítka hraj tón trvale i při uvolnění tlačítka. Druhým tlačítkem vypni hraní tohoto tónu

Složené operátory

Anglicky nazývané compound operators jsou operátory, které nám usnadní práci. Zkracují totiž zápis operací, kdy upravujeme hodnotu pouze jedné proměnné

Delší zápis		Kratší zápis
x = x + 2;	zvětši hodnotu x o 2	x += 2;
x = x - y	od x odečteme hodnotu y a výsledek zapíšeme do x	x -= y
x= x * 20	Vynásobí x 20	x *= 20
x= x / 20	Vydělí x 30	x /= 30
	Provede celý matematický úkon nebo proceduru a po skončení přičte jedničku k x	x++
	Provede celý matematický úkon nebo proceduru a po skončení odečte jedničku od x	x--
	Nejprve přičte jedničku k x a potom teprve provede celý matematický úkon nebo proceduru	++x
	Nejprve odečte jedničku od x a potom teprve provede celý matematický úkon nebo proceduru	--x

Zápis a čtení EEPROM

Velikost paměti EEPROM je závislá na použitém procesoru. Pro arduino uno to je 1kb (0-1023). Arduino mega 2560 má 4kb paměti. Pro zápis a čtení je zapotřebí přilinkovat knihovnu `#include <EEPROM.h>`

Úkol č. 70 – Zápis do EEPROM

```
#include <EEPROM.h>  // V úvodní části programu je nutno přilinkovat knihovnu pro práci s EEPROM.
int count = 0;  // Tento program definuje proměnnou count typu integer (0- 65535)
byte b1;        // Pomocná proměnná b1
byte b2;        // Pomocná proměnná b2
void setup() {
}
void loop() {
  count++;   // Před zápisem do paměti EEPROM proměnnou count zvýší její hodnotu o 1 (count= count+1)
  b1 = count / 256;  // Rozdělení proměnné typu integer (v paměti uložena do dvou bytes) do dvou proměnných typu byte.
//Celá část po dělení 256
  b2 = count % 256;  // Rozdělení proměnné typu integer (v paměti uložena do dvou bytes) do dvou proměnných typu byte.
//zbytek po dělení 256
  EEPROM.write(100, b1);  //zápis do EEPROM na adresu 100, hodnotu z proměnné b1
  EEPROM.write(101, b2);  //zápis do EEPROM na adresu 101, hodnotu z proměnné b2
  count = 0;  //vynuluj proměnnou count
  b1 = 0;     //vynuluj proměnnou b1
  b2 = 0;     //vynuluj proměnnou b2
  b1 = EEPROM.read(100);  //přečti z EEPROM adresy 100 hodnotu a ulož do proměnné b1;
  b2 = EEPROM.read(101);  //přečti z EEPROM adresy 101 hodnotu a ulož do proměnné b2;
  count = (b1 * 256) + b2; //vypočítej hodnotu do cout
}
```

Správnou funkci zjistíte tím způsobem, že v simulátoru je vypsaná hodnota proměnné a ta se bude měnit.

Cyklus (počítádlo) - FOR

Úkol č. 80 – FOR

Ověřte následující program na simulátoru UnoArduSim. V tomto simulátoru je vidět proměnné a jejich aktuální hodnota a lze krokovat program po jednotlivých instrukcí. Odzkoušejte tuto možnost.

```
int count;
void setup() {
  pinMode(9, OUTPUT);
}
void loop() {
  for (count = 10; count < 100; count++) {
    analogWrite(9, count);
  }
}
```

V proměnné count počítej od 10

Smyčka bude prováděna, dokud bude platit podmínka, že proměnná count je menší než 100

V každém cyklu provde matematický úkon zde uvedený. V našem případě přičti hodnotu 1.

Cyklus (s podmínkou) – WHILE

Všechny příkazy v cyklu se provádí, dokud je podmínka pravdivá. Za zmínku stojí, že program kontroluje platnost podmínky vždy na začátku cyklu, pokud je nepravdivá, cyklus skončí. Vyjádřeno slovy: „Pokud je podmínka pravdivá, udělej tohle a vrať se na začátek. Pokud není, skonči“. Cyklus while() se tedy nemusí provést vůbec.

Úkol č. 81 – WHILE

Ověřte následující program na simulátoru UnoArduSim. V tomto simulátoru je vidět proměnné a jejich aktuální hodnota a lze krokovat program po jednotlivých instrukcí. Odzkoušejte tuto možnost.

```
int count;
void setup() {
  pinMode(9, OUTPUT);
}
void loop() {
  count = 10;
  while (count < 100) {
    analogWrite(9, count++);
  }
}
```

V proměnné count počítej od 10

Smyčka bude prováděna, dokud bude platit podmínka, že proměnná count je menší než 100

Vykonej příkaz a po skončení přičti jedničku k proměnné count.

Cyklus (s podmínkou) – DO WHILE

Cyklus do..while() se od while() liší pouze v tom, že se podmínky kontrolují až na konci. Dříve tedy dojde k provedení příkladů a poté až ke kontrole podmínek. Slovně: „Udělej něco, a když platí podmínky, vrať se na začátek. Jinak skonči.“ V praxi to tedy znamená, že se tento cyklus provede minimálně jednou.

Úkol č. 82 – DO, WHILE

Ověřte následující program na simulátoru UnoArduSim. V tomto simulátoru je vidět proměnné a jejich aktuální hodnota a lze krokovat program po jednotlivých instrukcí. Odzkoušejte tuto možnost.

```
int count;
void setup() {
  pinMode(9, OUTPUT);
}
void loop() {
  count = 10;
  do {
    analogWrite(9, count++);
  } while (count < 100);
}
```

V proměnné count počítej od 10

Smyčka bude prováděna, dokud bude platit podmínka, že proměnná count je menší než 100

Vykonej příkaz a po skončení přičti jedničku k proměnné count.

Switch case

Podobně jako if , kontroluje switch…case běh programu a umožňuje programátorovi napsat rozdílné kusy programu, které budou spuštěny v závislosti na podmínkách. Switch výraz porovnává hodnotu proměnné a hodnoty uvedené v jednotlivých case větvích. Když bude nalezena shoda tak provede následující kus programu. Tímto způsobem nelze ale zadat podmínku od – do nebo větší nebo menší.

Dále se používá příkaz break a to typicky na konci každé case větve. Bez použití příkazu break by program pokračoval vykonáváním další case větve dokud by nenarazil na příkaz break a nebo konec switch bloku. Pokud není ani jedna podmínka nalezena je možno pokračovat částí default.

V našem příkladu do proměnné paměť je uložena hodnota pro vypsání. V proměnné cislice je aktuální číslice která bude zobrazena. Toto složité vypisování je použito z důvodu omezení počtu součástek (IO) a možnosti připojit display pomocí tří vodičů. Pro přenos dat je použit sériový přenos, kdy první byte určuje zapnuté segmenty na display a druhý byte zapíná, která číslice bude zobrazována. O vlastní přenos se nemusíme starat. Stačí na začátku povolit přenos (pin pinLatch). Pro vlastní přenos slouží instrukce shiftOut, která má parametry

1. Kam je připojen datový vodič pinData
2. Kam je připojen datový vodič pinClk
3. Směr přenosu dat. Od nejvýznamnějšího k nejnižšímu nebo opačně
4. Přenášená hodnota

pinLatch slouží pro povolení přenosu dat pinClk slouží k synchronizaci přenosu (hodiny)
pinData slouží k přenosu dat

Úkol č. 90 – 7 segmentovky v1

Na display zobraz údaj analogového vstupu A4

```
#define pinLatch 4
#define pinClk 7
#define pinData 8
byte cislice = 3;
int pamet = 0;
void setup() {
  pinMode(pinLatch, OUTPUT);
  pinMode(pinClk, OUTPUT);
  pinMode(pinData, OUTPUT);
}
void loop() {
  pamet = analogRead(A4);
// delay(20);
  zapisCisloNaSegment(pamet);
}
void zapisCisloNaSegment(int hodnota) {
  // mapa čísel pro segmentový displej - čísla 0 až 9
  const byte mapaSegment[] = { 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90 };
  switch (cislice--) {
    case 1:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[hodnota / 1000]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF1);
      digitalWrite(pinLatch, HIGH);
      break;
    case 2:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[((hodnota / 100) % 10)]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF2);
      digitalWrite(pinLatch, HIGH);
      break;
    case 3:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[(hodnota / 10) % 10]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF4);
      digitalWrite(pinLatch, HIGH);
      break;
    default:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[hodnota % 10]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF8);
      digitalWrite(pinLatch, HIGH);
      cislice = 3;
  }
}
```

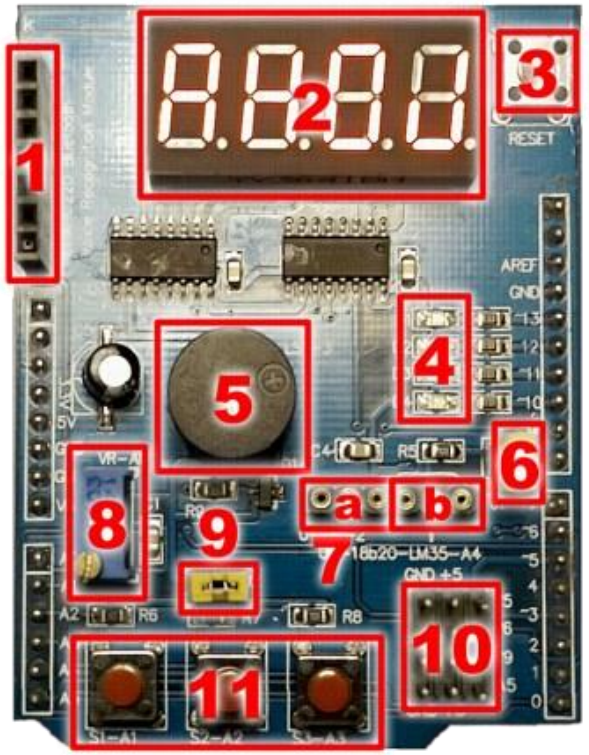
Tento program vypíše údaj, ale číslice moc neblikají. Stačí jenom v hlavní smyčce odkomentovat příkaz delay(20) a po novém nahrání údaje blikají, protože obnovovací frekvence je nízká. Kvalita zobrazení je velice závislá na rychlosti hlavní smyčky. Aby se tomu předešlo je zapotřebí zobrazení volat přes přerušení.

Úkol č. 91 – 7 segmentovky v2

Na display zobraz údaj analogového vstupu A4

```
#include <TimerOne.h>
#define pinLatch 4
#define pinClk 7
#define pinData 8
byte cislice = 3;
int hodnota = 0;
void setup() {
  pinMode(pinLatch, OUTPUT);
  pinMode(pinClk, OUTPUT);
  pinMode(pinData, OUTPUT);
  Timer1.initialize(6000);
  Timer1.attachInterrupt(zapisCisloNaSegment);
}
void loop() {
  hodnota = analogRead(A4);
  delay(50);
}
void zapisCisloNaSegment() {
  // mapa čísel pro segmentový displej - čísla 0 až 9
  const byte mapaSegment[] = { 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90 };
  switch (cislice--) {
    case 1:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[hodnota / 1000]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF1);
      digitalWrite(pinLatch, HIGH);
      break;
    case 2:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[((hodnota / 100) % 10)]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF2);
      digitalWrite(pinLatch, HIGH);
      break;
    case 3:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[(hodnota / 10) % 10]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF4);
      digitalWrite(pinLatch, HIGH);
      break;
    default:
      digitalWrite(pinLatch, LOW);
      shiftOut(pinData, pinClk, MSBFIRST, mapaSegment[hodnota % 10]);
      shiftOut(pinData, pinClk, MSBFIRST, 0xF8);
      digitalWrite(pinLatch, HIGH);
      cislice = 3;
  }
}
```

Popis: Arduino Multi-function Shield



1. Konektor pro modul Bluetooth APC220 (nebo Voice Recognification Module)

Popis (směrem shora)
- 7 – nepoužito
- 6 – nepoužito
- 5 – pin 0
- 4 – pin 1
- 3 – nepoužito
- 2 – napětí +5 V
- 1 – GND

2. Čtveřice LED zobrazovačů řízených obvodem 74HC595 (jeden pro všechny čtyři zobrazovače)

Ovládaní zobrazovačů:

Zápis do Latch – pin 4 (start zápisu LOW, konec HIGH)

Hodinový pulz (takt) – pin 7

Sériová data – pin 8 Zápis 2×8 bitů (2× Byte)

První Byte – data rozsvícených segmentů – rozsvícení LOW, vypnutí HIGH)

Bity 

odpovídají segmentu: 

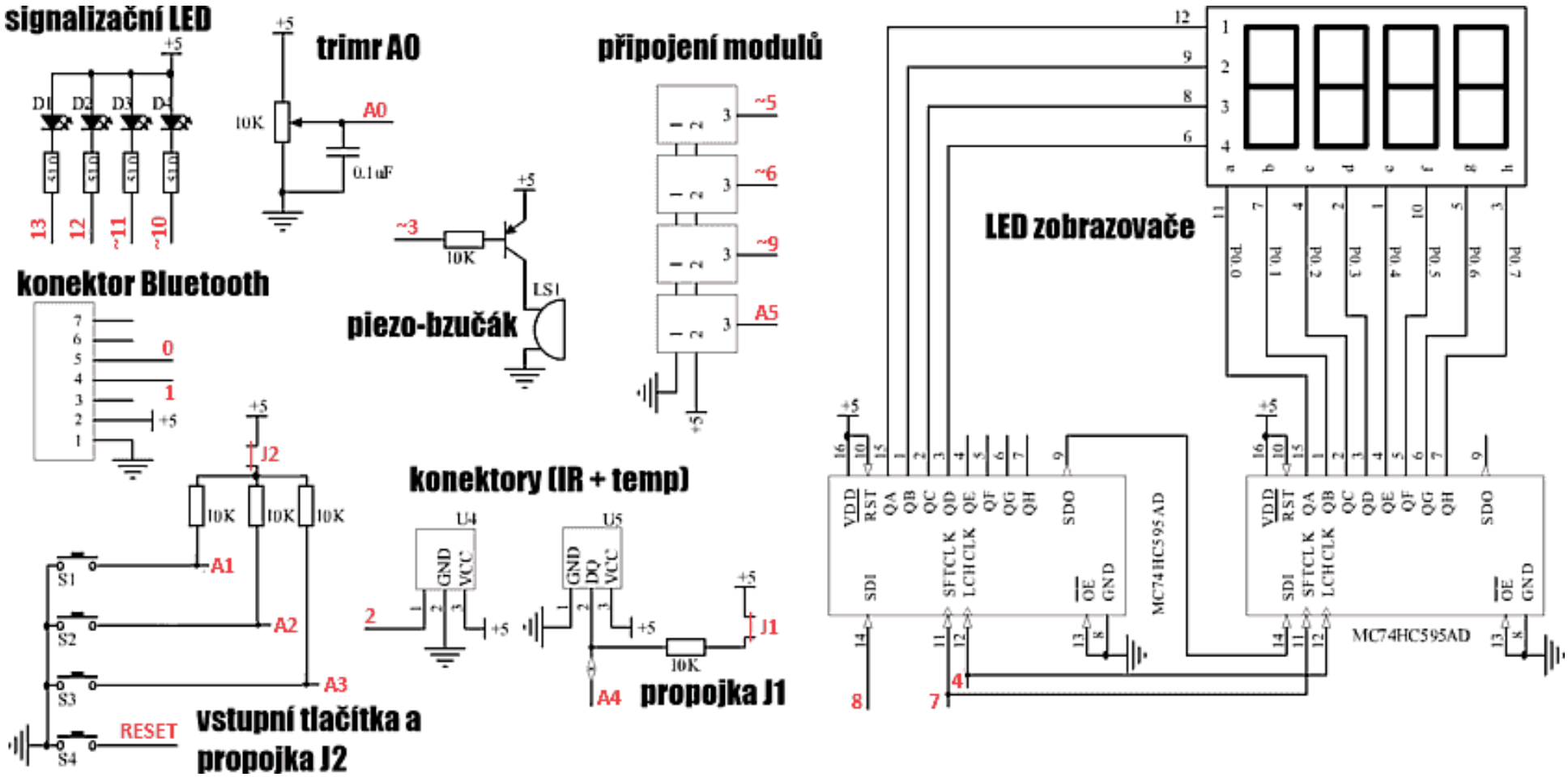
V druhém Byte se zapisuje adresa zobrazovače (zleva):
- první – ××××0001
- druhý – ××××0010
- třetí – ××××0100
- čtvrtý – ××××1000

3. Reset – resetuje modul Arduino (nikoliv Multi-function Shield)

4. Signalizační LED (rozsvícení LOW, vypnutí HIGH) Popis LED (směrem shora)
- LED D1 – pin 13
- LED D2 – pin 12
- LED D3 – pin 11 (PWM)
- LED D4 – pin 10 (PWM)

5. Piezo-bzučák pasivní – připojen k pinu 3 (PWM)

6. Propojka (jumper) – připojuje Pull Up rezistor 10 kΩ na vstup A4 (vyvedeno na střední pin konektoru 7b)
7. Konektor pro připojení čidla IR (7a) a tepelných čidel Dallas 18B20 a LM35 (7b)
   - Popis 7a (zleva) 1 – pin 2
     2 – GND
     3 – napětí +5 V
   - Popis 7b (zleva) 1 – GND
     2 – pin A4
     3 – napětí +5V
8. Odporový trimr 10 kΩ mezi referenční napětí (+5 V) a GND, jezdec vyveden na pin A0
9. Propojka (jumper) – připojuje Pull Up rezistory (10 kΩ) k tlačítkům S1, S2 a S3 (viz 11)
10. Konektor pro připojení čidel – např. sonarové čidlo HCSR04, servo…
    - Popis (zleva) První sloupec – GND (všechny 4 piny)
      Druhý sloupec – napětí +5 V (všechny 4 piny)
      Třetí sloupec (shora) – pin 5 (PWM), pin 6 (PWM), pin 9 (PWM) a pin A5
11. Vstupní tlačítka – připojena k pinům A1-3 (stisknutí – hodnota LOW) – možnost připojení Pull Up rezistorů k napětí +5 V (propojka 9)
    - Popis (zleva) Tlačítko S1 – pin A1
      Tlačítko S2 – pin A2
      Tlačítko S3 – pin A3


signalizační LED
trimr A0
připojení modulů
konektor Bluetooth
piezo-bzučák
konektory (IR + temp)
LED zobrazovače
vstupní tlačítka a
propojka J2
propojka J1
MC74HC595AD