Základní syntaxe

Při psaní kódu je třeba dbát na několik pravidel. Nejspíše jste je odchytily z minulých dílů, ale pro úplnost je zde připomenu.

- Na konci každého řádku s voláním funkce nebo deklarací proměnné je ; (středník). 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.
- Ohraničování částí kódu s { a }. Vše, co pod něco spadá, tj. věci v setupu, loopu, námi definovaných funkcí, if, else atd. se musí dát mezi ty havraní závorky. Samozřejmě výjimka potvrzuje pravidlo, takže pokud se po if, else nachází jen jeden jediný řádek, lze závorky vynechat.

Proměnné

Bez proměnných by to nešlo. Proměnné jsou místo v paměti, do kterého můžeme uložit nějakou hodnotu, libovolně ji měnit a libovolně si ji pojmenovat. Máme hned několik typů proměnných:

char	Znak, nabývá hodnoty právě jednoho ASCII znaku
byte	Je schopen uložit jedno 8 bitové číslo (0-255)
int	Nabývá hodnot celého čísla od -32 768 do 32 767 (16 bitů)
unsigned int	Podobně jako int, jen se záporné hodnoty "přelijí" do kladných, tedy uloží hodnoty 0-65 535
long	Uchovává 32 bitové číslo v rozsahu -2 147 483 648 do 2 147 483 647
unsigned long	Obdobně jako unsigned int. 0 - 4 294 967 295
float	Desetinné číslo o velikosti 32 bitů
double	Desetinné číslo s dvojnásobnou přesností
boolean	Nabývá hodnoty 0 (False) až 1 (True)
word	Viz. unsigned int
string	Uchovává textový řetězec

Jak proměnné deklarovat?

Než proměnné začneme používat, musíme je deklarovat. Pokud se do ní bez deklarace rovnou pokusíme něco uložit, tak nám kompiler vynadá (dá se to přirovnat k tomu, když chcete něco uložit do trezoru, ale žádný trezor jste si nekoupili). Možností deklarace je hned několik:

```
//první způsob - konstanty, neměnitelné proměnné
#define hodnota = 10 // Hodnotu vytvoříme a rovnou jí nastavíme na 10. Sama se nastaví na int
#define hodnota // Někdy potřebujeme nastavit hodnotu až když je třeba, k tomu slouží tento zápis

//druhý způsob - klasické proměné
int hodnota = 10; // proměnná má nastavený typ int a hodnotu 10
int hodnota; // proměnná typu int bez přiřazené hodnoty
```

Pokud proměnnou deklarujeme, nepřiřadíme jí hodnotu a poté jí vyvoláme, tak nenastane chyba. Program se bude tvářit, že je v ní 0 (v případě int, float,..), prázdný řetězec(string),...

Proměnným pak můžeme nastavovat různé hodnoty:

```
vstup = Serial.read();
pocet = 10
```

A co kdybyste chtěli hodnotu *pocet* zvýšit nebo snížit o hodnotu v proměnné *zmena*? Pokud vás napadne:

```
pocet = pocet + zmena;
pocet = pocet - zmena;

//mimochodem toto se bude jevit jako že se s proměnnou nic neudělalo, protože
něco přidáme a hned to samé sebereme
```

špatně to není, ale není jednodušší napsat

```
pocet += zmena;
pocet -= zmena;
```

?

Podmínky

Bez podmínek se prakticky neobejdete. Prostě se stále dostáváte do situací, kdy budete potřebovat, aby se jedna část provedla jen když něco platí. Podmínka se řeší pomocí příkazu if (anglicky pokud). Dále tu je ještě else if (nebo pokud) a else (jinak). A jak to funguje? Pokud chceme jen jednu podmínku, tak použijeme jednoduše if

```
if( podmínka )
{
        // Udělej toto
}
```

Pokud chceme více podmínek, tak jsou dvě možnosti - záleží na tom, jestli nám je jedno, jestli se jich splní více, nebo jestli chceme, aby se něco stalo v případě, že žádná neplatí.

```
if( podminka1)
        cinnost1
if( podmínka 2)
        cinnost 2
else
        cinnost 3
```

Co se zde stane? Dejme tomu, že bychom chtěli, aby se v případě podmínky jedna vykonala činnost 1, v případě podmínky 2 činnost 2 a v případě žádné podmínky činnost 3. Pokud se nesplní ani podmínka 1, ani 2, tak se vykoná činnost 3. Pokud se splní podmínka 2, tak se vykoná činnost 2, ale pokud se splní podmínka 1, tak se vykoná činnost 1 a 3. Jakto? Protože druhé if program bere jako další oddělenou část, takže vykoná podmínku 1 a přejde k druhému if s tím, že se jedná o další věc, kterou třeba otestovat. Podmínka 2 se nesplní, takže se vykoná else. Jak tedy programu říci, aby to celé bral jako jednu část?

Jednoduše, místo druhého if vložíme else if. Program tedy bude fungovat stylem:

```
Pokud souhlasí první podmínka (if) //pokud ne, přejde na další
        Udělej činnost 1
Nebo pokud souhlasí tato podmínka (else if) //pokud ne, přejde na další
        Udělej činnost 2
Jinak, pokud nic nesouhlasí (else)
        Udělej činnost 3
```

Takže to je systém fungování podmínek. A jak je používat?

Operátory k podmínkám

Takže co chceme v kódu kontrolovat? Velmi často se setkáme s tím, že nějaká funkce vrátí true nebo false, podle toho, jestli se akce povedla. Pokud chceme zkontrolovat výsledek na true, tak není třeba nic porovnávat, prostě napíšeme

```
if(vysledek)
```

A pokud je výsledek true, tak se if vykoná, pokud je false, tak nikoli. Co ale když chceme porovnat nějaká čísla? Zde máme několik operátorů. Ty napíšeme jednoduše mezi dvě hodnoty do závorek za if.

Takže máme zápis:

```
if( x <operator> y)
```

Co můžeme dosadit za operátory?

Operátor	Kompletní zápis	Význam
==	x == y	pokud se x rovná y
!=	x != y	pokud se x nerovná y
<	x < y	pokud je x menší než y
>	x > y	pokud je x větší než y
<=	x <= y	pokud je x menší nebo rovno y
>=	x >= y	pokud je x větší nebo rovno y

To jsou všechny operátory, které Arduino dokáže zpracovat. Ovšem musíme porovnávat věci, které se porovnávat dají. Chyba by byla napsat něco ve stylu if (pocet_kusu_nasklade(int) >= seznam_zakazniku(string))!

Kombinování podmínek & logické operátory

Dejme tomu, že potřebujeme podmínek ověřit více. Jako příklad si vezměme něco ze života. Někdo si jde koupit alkohol a my potřebujeme ověřit několik věcí - bylo mu 18 let? Je Becherovka na skladě? A má u sebe dostatečný obnos? Jak na to? Pokud začnete hned bušit

```
if( vek >= 18)
{
        if (napoj.jeNaSklade)
        {
                if( hotovost >= napoj.cena)
                {
```

Tak to sice taky půjde, ale použitelné to je jen v případě, když budeme chtít přijít na to, kde nastala

chyba (pomocí else ke každému if). Pokud ale chceme otestovat vše bez ohledu na to, kde to neplatí, tak můžeme všechny 3 podmínky sloučit do jednoho ifu. Jak?

Pomocí logických operátorů:

- && - neboli a (P Alt + C)
- || - neboli nebo (P Alt + W)

  && nám vrátí True, pokud platí obě podmínky, které spojujeme. Pokud některá z nich neplatí, vrátí false.

  || nám vrátí True, pokud alespoň jedna z podmínek je True. Pokud jsou obě False, vrací False.

A jak to tedy spojit?

```
if ((vek >= 18 & (napoj.jeNaSklade == true & (hotovost >= napoj.cena))))
```

Zde je důležité dbát na to, že pokud spojujeme podmínky s ==, > atd, tak je lepší, když je každá část v závorce, jinak to dělá neplechu.

Dále existuje ještě operátor *!*, který neguje, takže

```
!true //je false
!false //je true

if(!Serial.available())
{
        //se vykoná jen když sériový port nebude k dispozici
```

Funkce

Máme dvě základní funkce - *setup()* a *loop()*. Jak již víme, setup se vykoná jen jednou, loop ve smyčce. Jako příklad si vezměme situaci, kdy chceme zablikat diodou. Pokud budete otrocky ve všech případech vypisovat

```
...
digitalWrite(13, HIGH); // zabliká
delay(500);
digitalWrite(13, LOW);
delay(500);
digitalWrite(13, HIGH);
delay(500); ...
...
...
digitalWrite(13, HIGH); // zabliká
delay(500);
digitalWrite(13, LOW);
delay(500);
digitalWrite(13, HIGH);
delay(500); ...
...
...
digitalWrite... //zabliká
```

tak můžete, já vám v tom nebráním. Ale nebylo by jednodušší řešení? Samozřejmě, že je. Deklarujeme si vlastní funkci! Ta se deklaruje pomocí *void*, může brát i nějaké vstupní parametry a

případně i vracet nějakou hodnotu.

```
void blikniPin13() //nastavíme blikání
{
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
}
void loop()
{
    ...
    blikniPin13(); //zabliká
    ...
    ...
    blikniPin13();
    ...
    ...
    blikniPin13();
    ...
}
```

Není to jednoduší? A výhod je mnohem více. Pokud se rozhodnete, že LEDka nebude blikat po půl vteřině, ale po jedné vteřině, nemusíte v kódu hledat všechna místa, kde se bliká. Jen ve funkci přepíšete těch pár delay co tam jsou a rázem se změní úplně vše. A co kdybychom chtěli stejným způsobem klikat na více pinech? Nebudeme vypisovat funkce blinkiPin13, blikniPin12, blikniPin11, atd., ale dále funkci vstupní parametr:

```
void blik(int pin)
{
        digitalWrite(pin, HIGH);
        delay(500);
        digitalWrite(pin, LOW);
        ...
}
void loop()
{
        ...
        ...
        blikni(13); //blikání na pinu 13
        ...
        blikni(10); //blikání na pinu 10
        ...
}
```

Pokud se funkce dokončí, vrátí se kód na místo, kde se funkce zavolala, a pokračuje dál.

Pole

Omlouvám se zkušeným programátorům, vás teď budu chvíli nudit 🙄 V minulém díle jsme dořešili proměnné. Co kdybychom potřebovali udělat databázi uživatelů? Co třeba zkusit něco takovéto:

```
String user1 = "pepa";
String user2 = "karel";
...
```

Určitě si domyslíte, že takhle by to nešlo. A co kdybychom měli třeba 500 uživatelů (u Arduina to moc nehrozí, ale jeden nikdy neví). K tomuto je lepší použít pole (ty jsme nakousli už v tomto díle). Co to pole vůbec je? My si můžeme představit proměnnou jako kartičku se jménem uživatele. Pole si můžeme představit jako pořadač, ve kterém jsou kartičky narovnány do přihrádek. V tom jsou seřazeny podle čísel a můžeme je jednotlivě vyvolávat.

```
// databáze uživatelů
String uzivatele[5] = { "Pepa" , "David", "Franta", "Karel", "Michal" };
//v poli uzivatele je ulozeno 5 uzivatelu
```

není to jednodušší? Všimněte si, že před názvem musíme udat typ, který skladujeme. A jak vyvoláváme?

```
Serial.println(uzivatele[0]);
```

```
>>>Pepa
```

```
Serial.println(uzivatele[4]);
```

```
>>>Michal
```

```
Serial.println(uzivatele[1]);
```

```
>>>David
```

Protože programátoři začínají vždy od nuly, tak Pepa není pod indexem 1, ale pod indexem 0. Pod jedničkou je David, trojku má Franta... A co kdybychom chtěli kromě uživatelů ukládat i redaktory a administrátory? Je tu možnost pole uzivatele, pole redaktori, pole administratori, ale je i jednodušší možnost.

A v desátém díle jsme se setkali s pokročilejším polem - 2D polem. To si můžeme představit tak, že kartičky jsou v pořadačích a pořadače jsou umístěny do skříně. Opět zde použiji to pojmenování "pole polí", protože to je několik polí v poli. Vypadalo by to nějak takto:

```
String clenove[3][5] = {
        {"Honza", "Jenda", "Mirek", "Jirka", "user"}, //uzivatele
        {"Nikola", "Adam", "Zdenda"},   //redaktoři
        {"David", "Michal"},    //admini
};
//neberte ta jména osobně, je to jen příklad :)
```

Zas musíme napsat, co v poli ukládáme. V našem případě tam je string, takže String. Poté musíme do hranatých závorek napsat počet polí a maximální hodnotu, kterou každé pole pojme. Pokud bychom 5 snížili na 4, tak redaktoři a admini projdou, ale protože uživatelů je 5, tak to vyhodí chybu. A už si asi domyslíte vyvolání:

```
Serial.println(clenove[2][0]);
```

```
>>>David
```

```
Serial.println(clenove[0][3]);
```

```
>>>Jirka
```

To by byly pole, tak co tu máme dále?

Knihovny

Knihovny už jsme tak jednou nakousli a to v tomto díle, ale já je tu znovu připomenu, abychom tu měli všechno. Knihovny slouží k tomu, abychom nemuseli spoustu věcí složitě programovat, protože již to někdo dořešil za nás. Podrobněji se na ně podíváme v příštím díle, kde si i nějakou vytvoříme.

Escapování

Ve čtvrtém díle se objevil jeden nevysvětlený řádek:

```
Serial.println("Ajaj, asi sem ti nerozumel. Napis \"t\" pro teplotu a \"v\" pro vlhkost.");
```

```
>>> Ajaj, asi jsem ti nerozumel. Napis "t" pro teplotu a "v" pro vlhkost.
```

Co se tu stalo? Proč se nevypsala ta lomítka? A proč je to jeden řetězec, když tam je několik uvozovek? Řešení je jednoduché. Nastala nám zde situace, kde potřebujeme do stringu přidat uvozovky, ale uvozovky nám string ukončí. Co s tím? Použijeme tzv. escapování, kde se pomocí zpětného lomítka "zruší" význam následujícího znaku. Takže pokud chceme do stringu vypsat uvozovky, musíme napsat \". Co ale když chceme napsat zpětné lomítko? No, prostě ho odescapujeme - **.

Komentáře

Pokud si do kódu potřebujete něco poznamenat, nebo část deaktivovat bez mazání, existují komentáře. Máme dva typy - jednořádkové a více řádkové

```
// toto je jednořádkový komentář,
ale tato část se již bere jako kód

/* pokud chceme komentář na více
řádků, stačí použít
tento zápis */
```

Cykly

Pokud potřebujeme, aby se nám něco opakovalo, můžeme použít smyčku. Máme v podstatě dva typy - *for* a *while*. For je smyčka, která vykonává kód *pro každou hodnotu řídící proměnné*:

```
for (promena; podminka; akce)
```

Ve smyčce for nejdříve vytvoříme proměnnou. Poté za středníkem napíšeme podmínku, která musí platit, aby se smyčka vykonala. Za poslední středník pak přijde akce, která se po každém vykonání s proměnnou udělá. Pokud bychom třeba chtěli vypsat čísla od nuly do devíti, můžeme to udělat takto:

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

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

Otevření sériového portu známe. Pod ním je cyklus for. Nejdříve vytvoříme proměnnou int i s hodnotou nula. Podmínka je, že i bude menší než deset. Pokud bude rovno deseti, nebo větší (k čemuž tady nemůže dojít), tak se cyklus přestane vykonávat a kód přejde dál. Poslední je akce, která se s proměnnou vykoná, tudíž se k ní přičte jedna. Cyklus se tedy provede, když je v i nula, přičte se 1, i je jedna, znovu se provede a tak dále. A while?

While znamená dokud. Ten se hodí, pokud třeba chceme, aby při zmáčknutí tlačítka začala blikat LEDka a nepřestala dokud tlačítko neuvolníme.

```
while(digitalRead(10))
{
        // blikání
}
```

Tak dlouho, jak do pinu 10 bude proudit proud. Co ale kdybychom chtěli, aby se cyklus spustil při startu Arduina a neukončil se, dokud nestiskneme jedno tlačítko nebo druhé tlačítko? Šlo by to nějak zkombinovat do podmínky while, ale pokud by přibylo třetí tlačítko, pak čtvrté... Přece musí existovat lepší způsob, jak smyčku ukončit. Takže dejme tomu, že při spuštění Arduina se začne vykonávat cyklus, který bude něco dělat a ukončí se až při stisknutí jednoho, druhého, třetího nebo čtvrtého tlačítka bez nepřehledného zápisu v podmínce. Jak? Pomocí příkazu *break*, který cyklus ukončí.

```
...
while(true)     // nekonečná podmínka, ukončí se jen příkazem break,
{               // pokud k tomu nemáte pádný důvod, nepoužívejte jí
        ...
        if(digitalRead(10))
         break;
        if(digitalRead(11))
         break;
        ...
```

V nějakých situací to může být lepší než hromada podmínek v jednom řádku, neměli bychom konstrukci ovšem používat příliš často? To je snad všechno, v příštím dílu se podíváme na knihovny a jednu si i vytvoříme (snad).