Arduino
7-segmentový displej
Multiplexing

NSWI170: Lab 05

Patrik Dokoupil

Credit: Martin Kruliš

Shrnutí 4. úlohy

```
byte segmentMap[] = {
  0xC0, // 0   0b11000000
  0xF9, // 1   0b11111001
  0xA4, // 2   0b10100100
  0xB0, // 3   0b10110000
  0x99, // 4   0b10011001
  0x92, // 5   0b10010010
  0x82, // 6   0b10000010
  0xF8, // 7   0b11111000
  0x80, // 8   0b10000000
  0x90  // 9   0b10010000
};
```

- Problémy:
  - Zbytečná deklarace masek pro segmenty
    - Máte k dispozici ve funshield.h
  - Nadbytečné a chybné používání glob. proměnných
    - Viz pravidlo z minulého cvičení
  - Deklarace polí která vede na kopii objektů

```
int number1 = 0, number2 = 0, number3 = 0, ..
int numbers[4] = {number1,number2,number3,number4};
```

Hodnoty v poli budou inicializovány na 0, ale modifikace prvků v poli nezmění proměnné number1,… ! Není tam žádný odkaz

```
Button btn1(pin1), btn2(pin2), btn3(pin3);
Button btns[3] = { btn1, btn2, btn3 };
```

To stejné jako příklad nahoře, ale kopie struktury Button už může být „dražší“. Navíc v případě struktury/třídy máme odlišné chování než C#

Tímto způsobem se vyhnete kopiím:

```
Button btns[3] = {Button(pin1), Button(pin2), Button(pin3) };
```

Pravidlo č. 5

- # 5 **Encapsulation (zapouzdření)**
  - Logické celky se hodí seskupovat dohromady -> `struct/class`
  - Struktury/třídy musí mít jasný význam a účel (Button, Display, …)
  - Manipulace s objektem resp. využití objektu pro daný účel probíhá pomocí metod (≈ interface) a funkcí
  - Stav objektu je pokud možno privátní a z venku není možné jej měnit jinak než pomocí metod
- Nápovědy
  - „Interface first“ – nejprve se zamyslete nad tím, k čemu bude sloužit a jak se bude používat (navrhněte metody) a teprve pak je implementujte.
  - Myslete na to, že implementace se může měnit (např. u chování tlačítka)

Pravidlo č. 5

- Ukázka pro tlačítko
  - Co od něj očekáváme za funkcionalitu? (návrh rozhraní)
    - Detekce stisku
    - Inicializaci (v **`setup()`**)
    - Vyvolání události (inkrement/dekrement/posun zobrazené pozice)
      - Několik méně či více rozumných možností
        - **`Update()`** funkce kde provedenou akci určíte dle pinu/druhu tlačítka (pokud budete mít více druhů, tak není moc udržitelné)
        - „Událostní“ tlačítko, dostane ukazatel na callback funkci, která se po stisku zavolá
        - Dědičnost (společný typ Button s funkcí Update, odvozené typy určují implementaci)

```
using callback_t = void (*)(int *);

class button {
  public:
    callback_t on_press;
    button(int pin, callback_t
on_btn_press) {
      pin_ = pin;
      on_press = on_btn_press;
    }

    bool pressed() const {...}
  private:
  ...
}

button btns[] = {
  button(button1_pin, increment),
  ...
}

if (btn.pressed()) {
  btn.on_press(&counter);
}
```

Cvičení – zobrazení více znaků najednou

- T1: Naivní způsob
  - Začněte s kódem z minulé domácí úlohy
  - Zrušte funkcionalitu třetího tlačítka (již nebude měnit pozici)
    - První dvě tlačítka budou dělat +1/-1
    - Ve funkci **`loop()`** projděte cyklem všechny 4 pozice a ty vykreslete
  - Pozn.: pozorujete zvláštní chování displeje?
    - Pokud ne, zkuste za cyklus uvnitř **`loop()`** přidat **`delay(2)`**

Cvičení

- T2: Zajištění stejného „jasu“ u všech znaků
  - V každém volání funkce **`loop`** použijete jen jednu pozici displeje
  - Pozici měníte mezi voláními **`loop()`**
  - Jednotlivé iterace by měly v průměru trvat stejně dlouhou dobu
    - Může být problém v případě, že jednou za čas uděláte něco hodně složitého, co znatelně zpomalí daný loop (pro naše použití můžete ignorovat)
  - Oddělte logiku pro displej od zbytku kódu
    - Rozumný základ je class/struct s metodami **`set()`** a **`show()`** kde **`set()`** nastaví zobrazované číslo a **`show()`** se volá opakovaně v **`loop()`** s tím, že se postará to, která číslice se má zobrazit

```
class display {
public:
    void set(int value){…}
    void show(){
        ...
        write_digit
        pos_ = ... // update pos
        ...
    }
private:
    int pos_
    ...
}
```

```
display disp;
void setup() {
    disp.setup(150);
}
void loop() {
    disp.show();
}
```

Cvičení

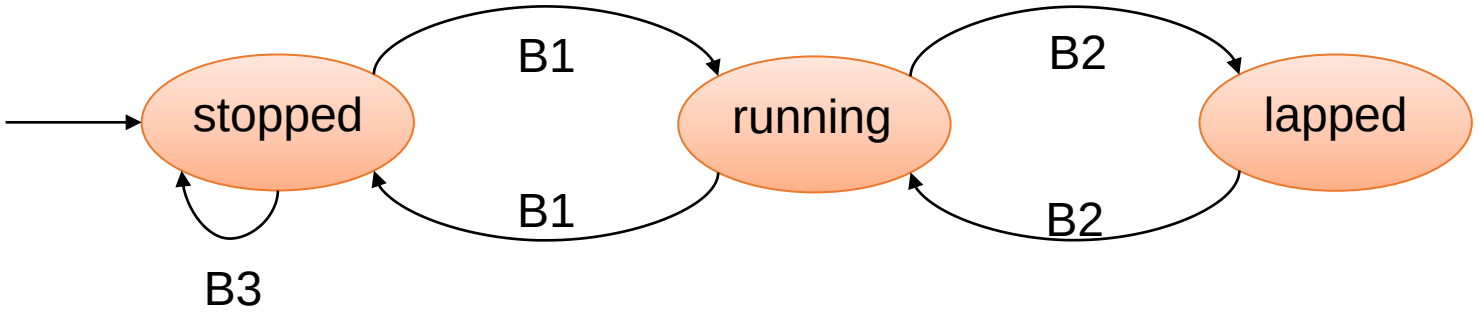
- T3: Vylepšení zobrazení
  - Nezobrazujte úvodní nuly
    - 12 bude zobrazeno jako 12, nikoliv jako 0012
    - 0 bude zobrazena jako 0
  - Upravte strukturu pro displej tak, aby umožnila zobrazení desetinné tečky
    - Pozice tečky jako privátní proměnná, její nastavení pomocí metody
    - Tečka na nejpravější pozici se nezobrazuje
  - Vyzkoušejte upravenou strukturu
    - Pomocí tlačítka č. 3 posouvejte pozici desetinné tečky

Domácí úloha

- Implementujte stopky
  - 1. Tlačítko vypne/zapne stopky
  - 2. Tlačítko „zmrazí“ displej, ale interní časování běží dále
  - 3. Tlačtítko resetuje stopky do 0
    - Pouze pokud jsou stopky vypnuté, jinak nemá efekt
  - Stopky zobrazují čas v 0.1s (100ms), interně však chcete rozlišit jemněji, alespoň 1ms -> použijte **`millis()`**
    - Poslední číslice je oddělena tečkou
    - Vždy zobrazujte alespoň 2 číslice (tj. nulu zobrazte jako 0.0)

Domácí úloha

- Detailnější popis stavů
  - Začínáte ve stavu stopped
  - Diagram/stavový automat popisující možné přechody mezi stavy
    - Stavy stopped/running/lapped
    - Hrany odpovídají stiskům tlačítek a vedou od starého k novému stavu (př. ze stavu running se lze pomocí B1 dostat do stopped, pomocí B2 do lapped)



Cvičení*

- Extension 1: Rozšiřte funkcionalitu čítače z domácí úlohy č. 4
  - Vidíte všechny cifry najednou
  - Pomocí třetího tlačítka vyberete pozici, kterou budete zvyšovat/snižovat, ale jelikož jsou vykresleny všechny cifry najednou, označte aktivní pozici pomocí segmentu tečky „.“
  - Možnost záporných čísel (na nejlevější pozici zobrazíte „-“)
  - Čítač od -999 do 9999
  - Místo modula detekujte podtečení/přetečení čítače a zobrazte nějakou chybovou hlášku (ideálně odlišnou pro každý z těchto stavů)