Arduino
7-segmentový displej
Multiplexing

NSWI170: Lab 05

Patrik Dokoupil

Credit: Martin Kruliš

Shrnutí 4. úlohy

- Problémy:
  - Drobná porušení pravidel 1-4
    - Konstanty, modifikace glob. proměnných, nevhodná dekompozice, …
  - Nevhodně navržený typ pro tlačítko / displej

```
// TOHLE NE
class button {
public:
    // ...
    void on_click(int & counter) {
        switch (kind_) {
            case kind::INCREMENT:
                do_increment(counter);
                break;
            case kind::DECREMENT:
                do_decrement(counter);
                break;
            default:
                // unknown kind
                break;
        }
    }
};
```

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 - metod)
    - Detekce stisku (**`is_pressed()`**)
      - Může být i komplikovanější než obyčejný stisk (např. lze brát v potaz debouncing, držení delší než X apod, pak se hodí metodu pojmenovat výstižněji)
    - Inicializaci (volanou v **`setup()`**)
    - [OPTIONAL] Vyvolání události (inkrement/dekrement/posun zobrazené pozice) -> tohle není nezbytné, viz obrázek vpravo
      - Několik méně či více rozumných možností
        - „Událostní“ tlačítko, dostane ukazatel na callback funkci, která se po stisku zavolá `using callback_t = void (*)(int *);`
        - Dědičnost (společný typ **`Button`** s metodou **`OnClick`**, odvozené typy určují implementaci)
      - Alternativně nechat zpracování události mimo tlačítko (jednodušší varianta)

```
// OK pokud je spravne implementovano
for (int i = 0; i < num_buttons; ++i) {
    if (buttons[i].is_pressed()) {
        buttons[i].on_click(counter);
    }
}

// TOHLE NE
class button {
public:
    // ...
    void on_click(int & counter) {
        switch (kind_) {
            case kind::INCREMENT:
                do_increment(counter);
                break;
            case kind::DECREMENT:
                do_decrement(counter);
                break;
            default:
                // unknown kind
                break;
        }
    }
};

// OK
if (incrementing_btn.is_pressed()) {
    do_increment(counter);
}
if (decrementing_btn.is_pressed()) {
    do_decrement(counter);
}
//...

// OK
for (int i = 0; i < num_buttons; ++i) {
    if (buttons[i].is_pressed()) {
        actions[i](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(200)`**

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

Cvičení

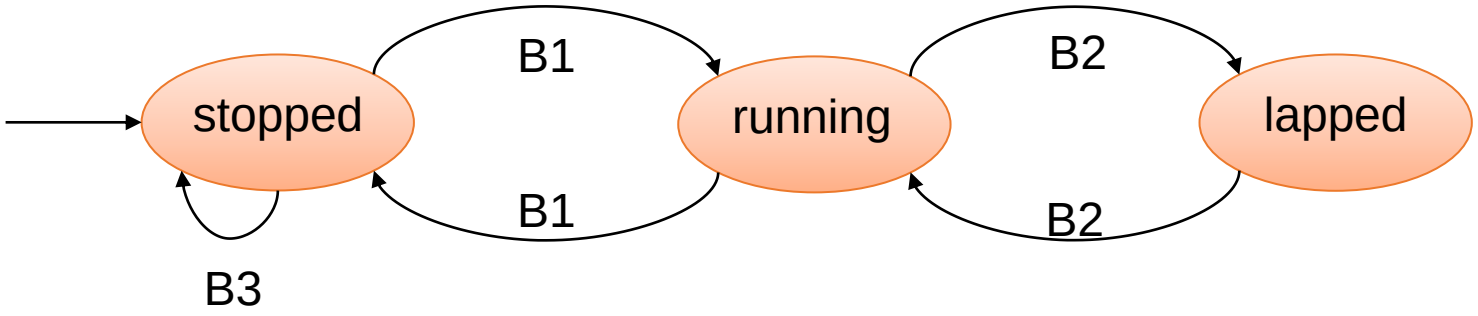
- T3: Vylepšení zobrazení
  - Nezobrazujte úvodní nuly
    - 12 bude zobrazeno jako 12, nikoliv jako 0012
    - 0 bude zobrazena jako 0
  - Upravte strukturu pro displej tak, aby umožnila zobrazení desetinné tečky
    - Pozice tečky jako privátní proměnná, její nastavení pomocí metody
    - Tečka na 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)