NSWI170 – **Počítačové systémy** – 2024/25 LS – Martin Svoboda

Úkol **A3: Počítadlo**

301. [**Dekompozice do tříd**] Veškerý kód týkající se činnosti diod, tlačítek a časovačů, stejně jako i kód samotného počítadla dekomponujeme a zapouzdříme do vhodně navržených tříd. Jejich datové položky pak budou výhradně jen privátní.

302. [**Univerzální funkce**] Výjimkou z předchozího požadavku mohou být samostatné globální funkce, pokud jsou použitelné univerzálně i mimo kontext našeho konkrétního problému.

303. [**Oddělení aplikace od ovladačů**] Implementace počítadla, diod a tlačítek musí být od sebe navzájem důsledně odděleny. Počítadlo jakožto analogie k uživatelské aplikaci samozřejmě bude využívat služeb diod a tlačítek, z opačného pohledu na věc je ale potřeba striktně zajistit, že naše ovladače diod ani tlačítek nebudou řešit žádný aspekt počítadla, dokonce si ani nesmí uvědomovat jeho existenci.

304. [**Systematické názvy konstant**] Vzhledem k tomu, že množství nejrůznějších konstant v našem programu narůstá, je vhodné je začít pojmenovávat nějak systematicky, tedy aby jen z jejich názvů bylo už na první pohled patrné, čeho se týkají (diod, tlačítek, …).

305. [**Analogie tlačítek k diodám**] Pro práci s tlačítky máme v obecné rovině v podstatě stejné nebo analogické požadavky, jaké jsme už měli dříve vůči diodám. Reprezentujeme je tedy vhodnou třídou, udržovat budeme globální pole jejich instancí atd.

306. [**Logická čísla tlačítek**] Konkrétně to znamená i to, že při práci s tlačítky `B1` až `B3` budeme opět používat logická čísla `0` až `2` namísto nízkoúrovňových pinů odpovídajících konstantám `button1_pin` až `button3_pin`.

307. [**Třída pro reprezentaci tlačítka**] Třída ovladače tlačítka zapouzdří veškeré datové položky a metody, které bychom pro bezproblémovou práci s tlačítky mohli potřebovat, a to včetně nezbytných vnitřních stavů nebo časování. Jeho implementace musí být dostatečně univerzální a robustní, protože nemůžeme předjímat, jak konkrétně budou tlačítka našimi aplikacemi využívána.

308. [**Zakrytí vnitřní implementace**] Rozhraní veřejných metod tlačítka bude navrženo tak, aby pro uživatele bylo maximálně jednoduché, intuitivní a elegantní. Jinými slovy opět chceme záměrně schovat veškeré vnitřní implementační nebo technické detaily, abychom si jich navenek vůbec nemuseli být vědomi, natož jim museli rozumět nebo byli nuceni je nějak řešit či hlídat.

309. [**Typy událostí tlačítka**] Přestože nás v rámci tohoto úkolu zajímá jen událost stisknutí tlačítka, v budoucnu se nám bude hodit i možnost detekce uvolnění tlačítka. Pokud jde konkrétně o stisknutí, užitečné by dokonce mohlo být rozlišení konkrétní varianty, tedy zda jde o prvotní resp. opakované události při delším držení tlačítka.

310. [**Sloučení událostí stisknutí**] Na druhou stranu by se také hodilo mít i možnost dotázat se na událost stisknutí tlačítka, aniž by se takové varianty prvotního resp. opakovaného stisknutí rozlišovaly, protože je jednoduše chceme obsluhovat naprosto stejně (jako právě my teď).

311. [**Detekce vzniku událostí**] Detekce událostí závisí nejenom na návratové hodnotě získané z funkce `digitalRead`, ale také na korektní práci s naší vnitřní stavovou logikou tlačítka. To ale znamená, že taková detekce není opakovatelná. Kdybychom se o ní v jedné iteraci funkce `loop` pokusili vícekrát, mohli bychom dostat neočekávané výsledky. Buď se tedy bez jakékoli záruky spolehneme na disciplínu uživatele, že k opakovanému volání nedojde, nebo jednoduše detekci vzniku událostí od dotazování nad nimi zcela oddělíme.

312. [**Opakovaná detekce stisknutí**] Ať už se ale rozhodneme jakkoli, detekci stisknutí pomocí funkce `digitalRead` jako takové i tak není možné realizovat opakovaně. Jednak její vyhodnocení je velmi pomalé, současně ale každé volání může dát jiné hodnoty (pokud se už skutečný stav změnil).

313. [**Veřejné rozhraní tlačítka**] Kromě metody na inicializaci tlačítka tedy nabídneme aktualizační funkci (pro vlastní detekci událostí) a dotazovací funkce (pro jednotlivé typy událostí). Aktualizační funkci zavoláme pro každé tlačítko vždy povinně na začátku funkce `loop`, zjištěné závěry si interně zapamatujeme a prohlásíme je za platné pro celou danou iteraci funkce `loop`. Díky tomu pak bude možné následné dotazování provádět i opakovaně.

314. [**Aktivace opakovaných stisknutí**] Pokud bychom opakované události tlačítka nechtěli obsluhovat, nejspíše by je stačilo jen ignorovat. Lepší ovšem bude, pokud bychom při inicializaci tlačítka mohli takovou funkcionalitu explicitně zapnout nebo naopak vypnout.

315. [**Problém odskakování tlačítka**] Součástí vnitřní logiky tlačítka by měla být i schopnost odfiltrovat krátké výkyvy stavů způsobené mechanickými aspekty a nedokonalostí tlačítek. Konkrétně budeme pracovat s myšlenkou, že změna stavu (jak stisknutí, tak uvolnění) musí trvat souvisle alespoň např. 10 ms, abychom ji zaregistrovali. Pokud v této době dojde k narušení započatého záměru, k žádné změně vůbec nedojde. Naopak dokud k úspěšnému přechodu opravdu nedojde, budeme i nadále fungovat beze změny, tedy např. nepřestaneme vyvolávat případné události opakovaného stisknutí.

316. [**Využití třídy časovače**] Pro kontrolu časování uvnitř tlačítek samozřejmě využijeme instance třídy časovače, kterou už máme k dispozici z předchozího úkolu.

317. [**Komentování kódu**] Přinejmenším komplikovanější části kódu by měly být doprovázeny dostatečně vysvětlujícími komentáři, abychom usnadnili jejich porozumění. V našem případě se můžeme zaměřit např. na řešení stavové logiky tlačítka a vysvětlení prováděných akcí.

318. [**Používání vybraných tlačítek**] Způsob, jakým budeme programátorsky pracovat s instancemi jednotlivých tlačítek, nemůže být ovlivněn skutečností, jestli reálně chceme pracovat se všemi, nebo jen některými vybranými. Musíme mít tedy k dispozici všechna, teprve aplikace samotná (tedy např. naše počítadlo) bude určovat, se kterými pracovat chce a se kterými nikoli.

319. [**Třída pro reprezentaci počítadla**] Veškerá logika našeho počítadla bude opět řešena pomocí vhodně navržené třídy. Z hlediska rozhraní jejích veřejných metod vyčleníme minimálně inicializaci počítadla, vlastní operace měnící jeho hodnotu, obsluhu událostí vyvolaných tlačítky nebo zobrazení jeho hodnoty na diodách.

320. [**Oddělení metod počítadla**] Konkrétně metody inkrementace a dekrementace počítadla nebudou samy od sebe vyvolávat zobrazení jeho hodnoty na diodách, abychom zajistili důsledné oddělení manipulace s daty od jejich vizualizace aneb nezávislost těchto operací.

321. [**Reprezentace hodnoty počítadla**] Přestože pro zobrazení hodnoty počítadla na diodách budeme muset získat její binární rozklad, počítadlo jako takové by si ale informaci o této hodnotě mělo pamatovat na logické úrovni, tedy jako obyčejné celé číslo. Určitě tedy není vhodné použít pole binárních číslic, natož snad pole logických stavů diod.

322. [**Kontrola přetečení počítadla**] Hodnota počítadla jako taková musí vždy spadat do povoleného intervalu, takže při každé operaci inkrementace nebo dekrementace musíme současně ohlídat případné přetečení. To je navíc nutné ošetřit bez zbytečného odkladu, abychom tuto konzistenci zajistili na očekávané úrovni atomicity.

323. [**Korekce přetečených hodnot**] Pokud k přetečení dojde, jistě bychom novou hodnotu dokázali upravit pomocí podmíněného větvení. Jde to však i obyčejným výpočtem bez větvení, což je určitě preferovaná varianta. Pozor jen na to, že operace získání zbytku při celočíselném dělení může vracet záporná čísla (to je důležité pro korektní implementaci dekrementace).

324. [**Maximální hodnota počítadla**] Při ošetření přetečení se určitě neobejdeme bez stanovení a následného použití maximální povolené hodnoty počítadla. Tuto hodnotu pochopitelně zavedeme pomocí pojmenované konstanty. Pozor však na to, že je odvoditelná z jiných již známých informací, a tedy je nutné ji spočítat a nedefinovat fixně.

325. [**Počáteční hodnota počítadla**] Součástí inicializace počítadla na konci funkce `setup` by mělo být nastavení jeho aktuální, tedy výchozí hodnoty. Jakkoli je totiž v našem případě konkrétně rovná `0`, obecně by nemusela. Aneb klidně bychom mohli začínat s nějakou jinou počáteční hodnotou.

326. [**Zobrazení počáteční hodnoty**] A nejde jen o nastavení této hodnoty, stejně tak musíme zajistit její zobrazení na diodách.

327. [**Pořadí inicializačních akcí**] Inicializační akce vykonávané v průběhu funkce `setup` by z povahy věci měly být uspořádány tak, abychom se nejdříve věnovali hardwarovým aspektům a teprve následně těm aplikačním.

328. [**Počítání binárního rozkladu**] Při počítání binárního rozkladu hodnoty počítadla bychom se měli zaměřit na jeho efektivitu. Konkrétně bychom se měli vyhnout funkcím počítajícím obecné mocniny. Specificky u mocnin dvojky se totiž bez nich snadno obejdeme. Stačí jen elegantně využít některé základní bitové operace a masky.

329. [**Konverze logických hodnot**] Pokud někde v kódu očekáváme logickou hodnotu, neměli bychom se při jejím určení spoléhat na automatické konverze dané jazykem, kdy `0` znamená `false` a cokoli jiného `true`. Například výsledkem bitových operací je číslo, to tedy na potřebnou logickou hodnotu teprve potřebujeme nějak explicitně převést, třeba porovnáváním.

330. [**Zbytečné aktualizace diod**] Jelikož stav rozsvícení diody zůstává platný až do jeho další změny a volání funkce `digitalWrite` je navíc velmi pomalé, je vhodné vyvarovat se přenastavování diod v každém jednotlivém běhu funkce `loop`. Jinými slovy k němu přistoupíme jen tehdy, když to bude nutné, tedy jen při změně hodnoty počítadla. A i v takovém případě budeme dávat instrukce jen těm konkrétním diodám, jejichž stav změnit opravdu potřebujeme.

331. [**Přiřazení akcí tlačítkům**] Přiřazení uživatelských funkcí obsluhujících příslušné události našich jednotlivých tlačítek nesmí být v kódu pevně zadrátováno, musíme jej tedy mít možnost snadno změnit. V tomto smyslu plně postačí použít vhodné pojmenované konstanty.

332. [**Špatně použitý cyklus**] Cykly je obecně vhodné použít v situacích, kdy očekáváme, že každá jejich iterace bude vypadat řekněme podobně. Pokud tedy máme tlačítka a každé z nich má vyvolávat jinou akci, nedává smysl obsluhu jimi vyvolaných událostí řešit `for` cyklem, abychom obratem pomocí konstrukce `switch` nebo jinak zase řešili větvení na jednotlivé případy. Pak by totiž takový cyklus zcela postrádal smysl.

333. [**Nezávislost jednotlivých tlačítek**] Jednotlivá tlačítka jsou na sobě nezávislá, musíme tedy být schopni obsloužit i situace, kdy v rámci jednoho běhu funkce `loop` vyvolá událost více z nich najednou.

334. [**Jednoduchý obsah loopu**] Podobně jako funkce `main` u normálního programu, ani funkce `loop` by neměla obsahovat žádný složitý nebo nízkoúrovňový kód. Na druhou stranu ale stejně tak nedává smysl všechny zamýšlené akce jen vzít a zabalit do nějaké jedné umělé funkce, kterou bychom zde jako jedinou volali.

335. [**Nepovolené systémové funkce**] Kromě již zakázaných systémových funkcí nebudeme používat ani funkce `bitRead` (dokážeme se totiž bez ní snadno obejít) a `pow` (nepočítá přesně).