NSWI170 – **Počítačové systémy** – 2022/23 LS – Martin Svoboda

Úkol **A3: Počítadlo**

Závazné pokyny | Dobře míněné rady | Náměty k zamyšlení | Časté chyby

301. [**Dekompozice do tříd**] Veškerý kód týkající se činnosti diod, tlačítek a časovačů, stejně jako i kód samotného počítadla 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. [**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, …).

304. [**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.

305. [**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`.

306. [**Třída pro reprezentaci tlačítka**] Třída tlačítka musí zapouzdřit veškeré datové položky a metody, které pro práci s daným tlačítkem potřebujeme, včetně řešení nezbytných vnitřních stavů nebo časování. Konkrétně pro časování samozřejmě využijeme třídu časovače, kterou už máme k dispozici.

307. [**Zakrytí vnitřní implementace**] Rozhraní veřejných metod tlačítka musí být navrženo tak, aby pro uživatele bylo maximálně jednoduché, intuitivní a elegantní. Jinými slovy chceme záměrně schovat veškeré vnitřní implementační nebo technické detaily, abychom jim navenek nemuseli rozumět, natož je nějak řešit nebo hlídat.

308. [**Veřejné metody tlačítka**] Tlačítko nabídne minimálně metodu na jeho inicializaci a metody, prostřednictvím kterých se budeme schopni dotazovat na vznik událostí stisknutí a uvolnění tlačítka. Pokud jde konkrétně o stisknutí, umožníme dokonce dotazování i na konkrétní variantu, tedy prvotní resp. opakované události při delším držení tlačítka, stejně jako obecnou událost stisknutí pokrývající obě varianty bez rozlišení.

309. [**Možnost opakovaného dotazování**] Dotazovací funkce musí být naprogramovány tak, abychom je v rámci jednoho běhu funkce `loop` mohli korektně volat i opakovaně. Vnitřní logiku detekce změny stavu tlačítka tak patrně budeme muset vyčlenit a zcela ji oddělit od dotazování, a tedy ji řešit ve speciální aktualizační funkci, kterou vždy jednou jedinkrát zavoláme na začátku funkce `loop`.

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

311. [**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.

312. [**Komentování logiky tlačítka**] Komplikovanější aspekty týkající se zejména stavové logiky tlačítka si určitě zaslouží přímo v kódu pečlivě komentovat a vysvětlovat význam prováděných operací.

313. [**Používání jen 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.

314. [**Přiřazení funkcí 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.

315. [**Třída pro reprezentaci počítadla**] Veškerá logika našeho počítadla bude opět řešena vhodně navrženou třídou. Z hlediska rozhraní jejích veřejných metod musíme oddělit minimálně inicializaci počítadla, vlastní operace měnící jeho hodnotu, obsluhu událostí vyvolaných tlačítky nebo zobrazení jeho hodnoty na diodách.

316. [**Oddělení aplikace od ovladačů**] Současně je potřeba dodat, že 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 neřeší žádný aspekt počítadla, dokonce si ani nesmí uvědomovat jeho existenci.

317. [**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.

318. [**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.

319. [**Korekce přetečené hodnoty**] 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.

320. [**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.

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

322. [**Odlišení inicializace a resetování**] Přestože se v obecné rovině máme snažit o neopakování podobných nebo dokonce stejných fragmentů kódu, stejně tak je ale potřeba preferovat zachování očekávaného logického významu kódu nad jeho technickou formou. Konkrétně obsah našich funkcí na inicializaci a resetování počítadla sice tedy může být analogický, jde ale o sémanticky jiné operace, a tedy je do jedné funkce sloučit nemůžeme.

323. [**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 operace nad bitovými maskami.

324. [**Nevhodná optimalizace diod**] Na druhou stranu dodejme, že některé na první pohled přímočaré optimalizace naopak mohou uškodit. Např. nastavení nového stavu diody bez ohledu na ten aktuální bude ve skutečnosti rychlejší než dotazování se na aktuální stav a vyvolání jeho změny jen v případě nutnosti.

325. [**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.

326. [**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 vzít a zabalit do nějaké jedné umělé funkce, kterou bychom zde jako jedinou volali.

327. [**Pseudounifikovaný 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 situace. Pak by totiž takový cyklus zcela postrádal smysl.

328. [**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.

329. [**Zbytečné aktualizace diod**] Jelikož stav rozsvícení diody zůstává platný až do jeho další změny, určitě je vhodné vyvarovat se nastavování diod podle hodnoty počítadla v každém běhu funkce `loop`. Jinými slovy k němu přistoupíme jen tehdy, když to bude opravdu nutné.

330. [**Nepovolené systémové funkce**] Kromě již zakázaných systémových funkcí nebudeme používat ani funkci `bitRead`, dokážeme se totiž bez ní snadno obejít.