NSWI170 – **Počítačové systémy** – 2023/24 LS – Martin Svoboda

Úkol **A4: Počítadlo**

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

401. [**Dekompozice do tříd**] Jako obvykle všechen potřebný kód uspořádáme do vhodně navržených tříd, případně univerzálně použitelných globálních funkcí.

402. [**Třída displeje**] Třída segmentového displeje bude reprezentovat displej jako celek. Dekompozice na jeho samostatné pozice by totiž nedávala smysl, protože tyto pozice netvoří samostatné funkční celky. Navíc by to ani nepřineslo nějaké jiné výhody.

403. [**Funkcionalita displeje**] Displej nabídne minimálně funkci na jeho inicializaci a nízkoúrovňovou funkci na zobrazení požadovaného glyfu na vybrané pozici, v obou případech podle zadaných masek. Také nabídne funkci na zobrazení konkrétní číslice na vybrané pozici, tentokráte už ale na logické úrovni, tedy nikoli pomocí masek. Takových funkcí na vyšší úrovni abstrakce budeme v průběhu dalších úkolů postupně přidávat více, abychom co nejvíce zjednodušili použití displeje jeho uživateli.

404. [**Dodržování terminologie**] Abychom se vyhnuli případným nedorozuměním, je při pojmenovávání nutné důsledně dodržovat zavedenou terminologii. Aneb máme 1 *displej*, ten se skládá ze 4 *pozic*, každá obsahuje 8 jednotlivých *segmentů*. Jelikož uvažujeme desítkovou soustavu, *číslice* máme `0` až `9`.

405. [**Logické číslování pozic**] Abychom v tomto duchu zjednodušili uživatelské rozhraní a eventuálně také přenositelnost kódu, budeme předpokládat logické očíslování těchto pozic ve směru zprava doleva. Pozice vpravo tedy má číslo `0`, naopak pozice vlevo `3`. S výjimkou nízkoúrovňového kódu nad maskami pak budeme s těmito pozicemi pracovat všude, kde to je možné.

406. [**Dopočítání hodnot konstant**] Ani tentokráte nechceme nikde v kódu pevně zadrátovat konkrétní počet dostupných pozic displeje, jakkoli je jasné, že přílišnou volnost v tomto smyslu stejně nemáme. Pro masky pozic totiž používáme datový typ `byte`, který více než 8 pozic jednoduše nenabídne. Přesto všechny konstanty, které jsou z ostatních odvoditelné, opět dopočítáme.

407. [**Pole masek glyfů číslic**] Masky glyfů číslic zadefinujeme v rámci pole, které současně budeme používat pro jejich překlad. Tyto masky uvedeme v binární podobě, např. `0b11000000` pro číslici `0`, protože právě hodnoty jednotlivých bitů odpovídají logickému problému, který zde řešíme. Současně dodejme, že v tomto poli nemůžeme ukrývat masky jiných glyfů než právě číslic, to by narušilo jeho sémantický význam.

408. [**Neošetřování nízkoúrovňových chyb**] Snadno si dokážeme představit, že uživatel může naše funkce displeje zavolat s neplatnými hodnotami parametrů, např. neexistující číslicí k zobrazení. Takové situace však záměrně detekovat ani nijak ošetřovat nebudeme. V případě nízkoúrovňového programování je to totiž zejména v případě jazyka C++ z důvodu efektivity běžnou praxí. Podmínky použití funkcí jednoduše pečlivě popíšeme a je na uživatelích, aby je respektovali. Pokud tak neučiní, považuje se to za jejich chybu, chování programu bude nedefinované a klidně může skončit i pádem.

409. [**Inicializace displeje**] Ve funkci `setup` jako obvykle potřebujeme nejprve náš displej inicializovat. To kromě nastavení požadovaných módů pinů také znamená vypnutí všech segmentů na všech pozicích, protože tuto skutečnost analogicky k diodám nemáme garantovanou.

410. [**Nepřesnost vestavěné funkce pow**] V rámci našeho kódu patrně budeme potřebovat počítat nejrůznější mocniny. K tomuto účelu však není možné použít vestavěnou funkci `pow`, protože nepočítá přesně. Jednoduše proto, že pracuje s datovým typem `float`, tedy s čísly s pohyblivou desetinnou čárkou. Nad nimi však nemáme přesnou aritmetiku, ani čísla samotná nejsou reprezentovatelná přesně.

411. [**Vlastní funkce pro mocniny**] Jinými slovy bude potřeba, abychom si naprogramovali svoji vlastní funkci na počítání mocnin. Dodejme, že by měla být univerzální, a tedy umožnit práci s jakýmkoli základem. Na druhou stranu postačí jen nezáporné exponenty.

412. [**Efektivní počítání mocnin**] Zaměřit bychom se také měli na nalezení rozumné implementace této funkce, určitě bychom se měli vyvarovat rekurzivního řešení. V závislosti na zvoleném přístupu se případně pokusíme obejít i bez `if` podmínek, je-li to možné. Jakkoli to není nutné, můžeme se dokonce pokusit navrhnout řešení s lepší časovou složitostí než lineární ve vztahu k hodnotě exponentu.

413. [**Počítání mocnin dvojky**] Samozřejmě platí, že počítat mocniny `2` je výhodnější pomocí operace bitového posunu, nikoli námi představené obecné funkce.

414. [**Změny hodnot vstupních parametrů**] Obecně není dobrou praxí měnit hodnoty vstupních parametrů funkce, protože by to mohlo vyvolávat dojem, že jsou výstupní. Tím spíše, že nám takový mechanismus jazyk opravdu nabízí. My jsme se s ním jen záměrně neseznámili, nebylo to potřeba.

415. [**Recyklace lokálních proměnných**] Pokud uvnitř funkce začneme za nějakým účelem používat lokální proměnnou, neměli bychom ji později najednou začít používat pro úplně jiný záměr jenom proto, že už onu původní proměnnou fakticky nepotřebujeme a vyhovuje nám i shoda datových typů. Aneb v takovém případě jednoduše začneme používat další novou proměnnou. Vliv na rychlost kódu to nebude mít žádný, optimalizátor překladače si s takovými situacemi snadno poradí.

416. [**Kontext platnosti proměnných**] Lokální proměnné budeme vždy deklarovat až v místě, kde s nimi opravdu potřebujeme začít pracovat, nikoli automaticky všechny na začátku těla funkce. Kromě toho je také deklarujeme jen uvnitř toho nejspecifičtějšího vnořeného bloku, kde jsou potřeba, abychom zbytečně nerozšiřovali kontext a dobu jejich života.

417. [**Třída počítadla**] Podle očekávání celou logiku a datové položky týkající se funkcionality počítadla zapouzdříme do samostatné třídy. Ta samozřejmě dle potřeby bude využívat služeb tlačítek a displeje, ty však samy o sobě nesmí řešit žádný aspekt počítadla, ani o jeho existenci vůbec vědět.

418. [**Aktuálně vybraná pozice**] To znamená například i to, že displej jako takový nesmí být zodpovědný za pamatování si aktuálně vybrané pozice, protože ta nepochybně patří do logiky naší aplikace.

419. [**Inicializace displeje a počítadla**] Přestože v rámci inicializace našeho počítadla musíme zařídit zobrazení jeho výchozí hodnoty na displeji, tato skutečnost opět ze stejného důvodu nemůže mít žádný vliv na nutnost provedení úplné a nepozměněné inicializace displeje samotného, tedy včetně již zmíněného vypnutí všech jeho pozic.

420. [**Maximální hodnota počítadla**] Vzhledem ke způsobu odvození maximální povolené hodnoty počítadla a používané verzi jazyka už nebudeme moci použít konstantní výraz, stále ale můžeme použít alespoň konstantní proměnnou.

421. [**Extrakce vybrané číslice**] Získat číslici v daném řádu aktuální hodnoty počítadla lze nepochybně různými způsoby, ať už to ale uděláme jakkoli, opět bychom se měli vyvarovat neefektivního řešení, tedy naši funkci na počítání mocnin můžeme volat nejvýše jednou.

422. [**Pole předpočítaných řádů**] Vhodným řešením nebude ani předpočítání všech relevantních mocnin a jejich uložení v paměti. A to už jen kvůli tomu, že takový přístup by nebyl realisticky rozšiřitelný na větší displeje.

423. [**Extrakce číslice displejem**] Ve vztahu k této extrakci ještě dodejme, že by ji určitě neměla řešit třída displeje, protože ten přeci chápeme jako ovladač výstupního zobrazovacího zařízení, které ani takovým věcem rozumět nemá.

424. [**Zbytečné aktualizace displeje**] Samozřejmostí je, že měnit zobrazené glyfy na displeji budeme jen tehdy, pokud je to nezbytně nutné, tedy pokud opravdu došlo k nějakým změnám v hodnotě počítadla nebo vybrané pozici. Čistě pro orientační představu, jedna taková změna je totiž řádově 30× pomalejší než volání funkce `digitalWrite` používané pro práci s diodami.

425. [**Používání ladících výpisů**] Nezapomeňme, že pokud při implementaci úkolu narazíme na problémy, můžeme při hledání a odstraňování chyb používat vlastní ladící výpisy posílané z Arduina do počítače přes sériovou linku.

426. [**Závěrečné pročištění kódu**] Dodejme však, že ve finálním kódu by takové fragmenty měly být zakomentovány nebo jinak deaktivovány. Speciálně pak není možné v kódu ponechávat žádné části, které nejsou relevantní, dokončené nebo korektní, a to ani zakomentované.

427. [**Nedovolené systémové funkce**] Kromě již zakázaných systémových funkcí nebudeme používat ani funkce `bitRead` nebo `pow`.