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

Úkol **A5: Stopky**

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

501. [**Třída numerického displeje**] Současný ovladač displeje rozšíříme o nově potřebnou funkcionalitu zobrazování celých čísel pomocí konceptu dědičnosti. Navrhneme tedy novou třídu pro numerický displej, ta bude odvozena od toho základního. V případě potřeby pochopitelně můžeme stávající funkce základního displeje upravit nebo i přidat nové.

502. [**Rozdělení odpovědnosti displejů**] Dodržíme však myšlenku, že základní displej bude odpovídat jen za nízkoúrovňové instrukce k vlastnímu zobrazování nejrůznějších jednotlivých glyfů, zatímco složitější mechanismy zobrazování celých čísel bude s jeho pomocí řešit až právě numerický displej.

503. [**Doména podporovaných čísel**] Teprve odvozený numerický displej tedy získá schopnost zobrazovat celá čísla s využitím všech dostupných pozic segmentového displeje najednou. Konkrétně budeme podporovat nezáporná celá čísla a nezáporná desetinná čísla.

504. [**Zobrazení číslic s tečkou**] Pro zobrazování jednotlivých číslic budeme nadále využívat stávající funkci základního displeje, jen ji rozšíříme o podporu rozsvícení segmentu s tečkou pro případ vyžádání této nepovinné desetinné tečky. Pro vzájemnou kombinaci masek odpovídajících glyfů použijeme vhodnou bitovou operaci.

505. [**Pevně zadrátovaná tečka**] Samozřejmostí je, že implementace displeje bude univerzální, tedy že pozice tečky nebude nijak pevně zadrátovaná, natož svázaná s našimi požadavky konkrétně pro stopky.

506. [**Myšlenka časového multiplexu**] Pro zobrazování celých čísel zjevně potřebujeme, aby náš displej dokázal v jeden okamžik na různých pozicích zobrazovat různé glyfy. Toho však není konstrukčně schopen. Pomůžeme si trikem, kdy si číslo vyžádané k zobrazení v daný okamžik jen zapamatujeme a jeho zobrazování rozložíme v čase přes jednotlivé pozice. To znamená, že jednotlivé pozice displeje budeme neustále cyklicky střídat a v každé jednotlivé iteraci funkce `loop` dáme pokyn k zobrazení jen jedné z nich. To nám umožní dosáhnout optické iluze, kdy nabydeme pocitu, že svítí všechny pozice najednou, přestože tomu tak technicky nebude.

507. [**Reprezentace vyžádaného čísla**] Kromě vlastního čísla vyžádaného k zobrazení si uvnitř třídy numerického displeje můžeme pamatovat i další potřebné informace. Dodejme jen, že není vhodné vymýšlet odlišný způsob reprezentace celého a desetinného čísla. Speciálně pokud jde o ta desetinná, určitě nemůžeme kvůli nepřesnosti použít datový typ s pohyblivou desetinnou tečkou jako `float`, stejně tak není praktické rozdělit taková čísla na dvě části, celou a desetinnou.

508. [**Nastavení nového čísla**] Uvědomme si dále, že celé číslo bez tečky je něco jiného než desetinné číslo s tečkou na nějaké z pozic `0` až `3`. Na to bychom měli pamatovat i při návrhu rozhraní funkce nebo funkcí, které pro nastavování takových čísel budeme používat.

509. [**Zákaz neplatných hodnot**] Pro rozlišení obou situací bychom patrně mohli vymyslet nějakou speciální pozici jako `-1` a využít ji právě pro případ, kdy desetinnou tečku začlenit vlastně nechceme. Už kdysi jsme si ale vysvětlili, že takovéto nevalidní hodnoty obecně používat nemůžeme. Potřebujeme tedy najít jiné řešení.

510. [**Mechanismus přetěžování funkcí**] Stačí jednoduše funkci pro celá čísla oddělit od funkce pro ta desetinná. Můžeme však jít ještě dál a elegantně využít mechanismu přetěžování. Ten nám umožní mít více funkcí stejného jména za podmínky, že mají jiný počet nebo jiné typy parametrů.

511. [**Atomicita nastavení čísla**] Ať už rozhraní těchto nastavovacích funkcí navrhneme jakkoli, měli bychom mít možnost vše potřebné nastavit atomicky jediným pokynem. Neměli bychom tedy být

nuceni postupně volat více funkcí, třeba odděleně na nastavení čísla, aktivaci tečky a určení její pozice. To by totiž zhoršilo přehlednost kódu a snížilo uživatelskou přívětivost. Současně ani nechceme náš displej byť jen dočasně ponechávat v nějakém vnitřně nekonzistentním nebo jinak nevalidním stavu.

512. [**Vynechání úvodních nul**] Čísla budeme na displeji zobrazovat tak, že vynecháme zbytečné úvodní nuly před první platnou pozicí. Dejme si pozor na korektní detekci takové pozice zejména u desetinných čísel, stejně jako se vyvarujme zbytečně opakovaného počítání této pozice.

513. [**Zobrazení symbolu mínusu**] Pokud bychom se dobrovolně rozhodli podporovat i záporná čísla, glyf mínus umístíme hned před první platnou pozici čísla, nikoli na začátek displeje úplně vlevo.

514. [**Rovnoměrné střídání pozic**] Během realizace časového multiplexu musíme každé jednotlivé pozici na displeji věnovat stejnou pozornost, jinak bychom byli schopni detekovat nerovnoměrnou intenzitu rozsvícení. To platí i pro pozice, na kterých ve skutečnosti zrovna nic zobrazovat nechceme.

515. [**Rovnoměrné trvání iterací loopu**] Provedení našeho časového multiplexu se zjevně spoléhá i na dosud nevyřčený předpoklad, že každý běh funkce `loop` bude trvat stejnou dobu. Takové očekávání obecně není úplně realistické, v našem případě ale bude fungovat dobře.

516. [**Slučování stejných pozic**] Mohlo by nás také napadnout, že pokud bychom na dvou nebo i více pozicích chtěli zobrazit stejný glyf, mohli bychom to udělat současně, a tedy v tomto smyslu náš kód optimalizovat. To však ve skutečnosti není dobrý nápad, protože by to celý mechanismus z logického pohledu jen znepřehlednilo a žádného reálného přínosu bychom stejně kvůli nárůstu objemu kódu a větvení nedosáhli. Proto se této myšlence vyhneme.

517. [**Dosažení efektivního kódu**] Na druhou stranu to neznamená, že bychom na optimalizaci kódu časového multiplexu měli rezignovat. Právě naopak, protože jde o mechanismus, který bude realizován v průběhu každé jednotlivé iterace funkce `loop`, tedy neustále pořád dokola.

518. [**Vyvolání mechanismu multiplexu**] Pro úplnost dodejme, že celý mechanismus multiplexu je ze své povahy hardwarovou záležitostí našeho ovladače displeje, takže jej musíme vyvolávat přímo z funkce `loop`, nikoli nepřímo z našich stopek, které mají být jen aplikací náš displej využívající.

519. [**Možnost vypnutí displeje**] Celý numerický displej bychom měli mít možnost explicitně vypnout aneb chceme umět deaktivovat a obejít mechanismus časového multiplexu a vrátit se k manuálnímu ovládání základního displeje skrze přímé pokyny na zobrazení glyfů. V opačném případě bychom totiž pořád na displeji museli nějaké číslo zobrazovat, což nám zjevně nemusí vždy vyhovovat. Přestože takovou funkcionalitu v tomto úkolu potřebovat nebudeme, v dalších se už ale hodit bude.

520. [**Jediná instance displeje**] Díky použití konceptu dědičnosti bude náš numerický displej automaticky umět a obsahovat i všechno to, co uměl ten základní. To znamená, že vytvoříme jen jednu jedinou instanci displeje, a to právě toho numerického. Vytvořit jakési duální instance obou by bylo nejenom technickou chybou. Fyzicky totiž samozřejmě máme jen jedno zařízení displeje, obě logické instance by se navíc navzájem mohly nekontrolovaně ovlivňovat.

521. [**Zřetězení inicializačních funkcí**] Třídy základního i numerického displeje pochopitelně potřebují svoje vlastní inicializační metody. S cílem navrhnout uživatelsky co nejpřívětivější rozhraní obě tyto metody záměrně pojmenujeme stejně. Díky tomu pak můžeme v průběhu funkce `setup` celkovou inicializaci displeje vyvolat unifikovaným způsobem, aniž bychom museli tušit, pro jakou variantu ovladače jsme se rozhodli. Aby však vše fungovalo správně, potřebujeme v rámci inicializační funkce numerického displeje explicitně inicializační funkci toho základního zavolat, a to hned na začátku jako první krok. Jelikož je ale jméno této metody předka překryto, musíme použít plně kvalifikované jméno `Display::initialize()`.

522. [**Názvy metod displeje**] Při hledání vhodných názvů metod displeje bychom měli odlišit funkce, které opravdu realizují aktivní zobrazení nějakého glyfu, od funkcí na nastavení zamýšlených čísel. Ty totiž samy o sobě nic nezobrazují, neměli bychom proto uživatele mást.

523. [**Třída pro stopky**] Veškerou funkcionalitu stopek podle očekávání zapouzdříme do samostatné třídy. Ta bude využívat služeb tlačítek a displeje, samotná logika stopek však bude od těchto ovladačů striktně oddělena.

524. [**Postup návrhu stopek**] Návrh třídy pro stopky se vyplatí nepodcenit. Zamyslet bychom se měli minimálně nad rozsahem uchovávaných informací, vnitřními stavy a přechody mezi nimi, detekcí událostí vyvolaných tlačítky, realizací tomu odpovídajících akcí, korektností měření času, přípravou hodnoty k zobrazení a samotným ovládáním numerického displeje. Důsledné promyšlení a oddělení všech těchto aspektů povede k lepšímu kódu a ušetříme i čas při ladění.

525. [**Globální proměnné ovladačů**] Připomeňme, že na způsobu práce s globálními proměnnými pro instance tlačítek nebo displeje nic neměníme a ani v budoucnu měnit nebudeme. Nadále k nim tedy přistupujeme přímo a nepředáváme je přes parametry funkcí. Nově dodejme, že je ani nemůžeme uchovávat jako datové položky uvnitř třídy stopek, protože žádný aspekt našeho zacházení s hardwarem nemůže být závislý na konkrétních aplikacích jako jsou například stopky.

526. [**Enumerační třída pro stavy**] Přestože existují i jiné způsoby, vnitřní stavy stopek je patrně nejrozumnější reprezentovat pomocí enumerační třídy. Tu stačí deklarovat přes `enum class MyEnum { VALUE_1, ... }`, k jednotlivým hodnotám se přistupuje pomocí `MyEnum::VALUE_1`.

527. [**Opakování stejného kódu**] Jako tradičně bychom neměli zapomenout na vhodnou dekompozici vnitřní funkcionality stopek, abychom zbytečně neopakovali podobné nebo stejné fragmenty kódu.

528. [**Logické akce stopek**] Jako nejrozumnější se v tomto smyslu jeví úplné oddělení implementace nejrůznějších logických akcí stopek od obsluhy událostí jednotlivých tlačítek aneb vyvolání těchto akcí v návaznosti právě na detekované události a také aktuální vnitřní stav stopek.

529. [**Umístění obsluhy tlačítek**] Vzhledem k netriviálnímu rozsahu a nutnosti správně reflektovat zmíněný vnitřní stav stopek je vhodné celý kód obsluhy tlačítek umístit do nějaké metody stopek a nadále jej už neponechávat přímo ve funkci `loop`, kde bychom se neměli věnovat nízkoúrovňovým detailům jednotlivých aplikací.

530. [**Pozice desetinné tečky**] Byť v zadání předpokládáme počítání času s přesností na desetiny sekundy, a tedy umístění desetinné tečky na pozici `1`, kód naprogramujeme tak, aby pozice této tečky (tedy řekněme přesnost stopek) byla v rámci smysluplných mezí snadno konfigurovatelná.

531. [**Granularita vnitřního času**] Uvědomme si navíc, že pro dosažení očekávané přesnosti a korektnosti měření musíme uvnitř stopek akumulovaný čas evidovat s výrazně přesnější granularitou, než jakou očekáváme na výstupu.

532. [**Kontrola přetečení časů**] Připomeňme také, že bychom při počítání uplynulého času měli řešit i přetečení, ať už z hlediska zobrazitelné hodnoty na displeji, tak i systémového času jako takového.

533. [**Výpočet výstupního času**] Při přepočtu vnitřního času do výstupního formátu jen provedeme jednoduchou operaci dělení. Samozřejmostí ale je, abychom za tímto účelem zavedli pojmenovanou konstantu, jejíž hodnotu navíc odvodíme. A zdůrazněme, že korektně.

534. [**Průběžné přičítání času**] Vnitřně uchovávaný údaj o celkově naměřeném času bychom neměli aktualizovat v každém běhu funkce `loop`, abychom se vyhnuli případné akumulované chybě vzniklé z drobných nepřesností. To jinými slovy znamená oddělení naměřeného času od průběžných hodnot určených k zobrazení.

535. [**Zbytečné aktualizace displeje**] Pokyn displeji k zobrazení požadované hodnoty času samozřejmě budeme dávat jen v případě nutnosti, tedy při změně této hodnoty.

536. [**Efektivita detekce změny**] To platí zejména v situaci, kdy jsou stopky aktuálně zapnuté. Nejenom že tedy nemůžeme zbytečně nastavovat displej v každém jednotlivém běhu funkce `loop`, současně ale musíme zajistit i to, aby detekce onoho správného momentu k aktualizaci displeje nebyla výpočetně příliš náročná. Jako ne zcela vhodný se v tomto smyslu jeví i takový postup, kdy bychom v každém běhu funkce `loop` počítali aktuální výstupní hodnotu a jen ji porovnávali s tou předcházející.

537. [**Plánování aktualizace displeje**] Efektivnější cesta povede přes nám důvěrně známý mechanismus kontroly časování, který už jsme v průběhu předcházejících úkolů několikrát použili. Jen nebudeme uvažovat fixní interval, ale další událost vhodně naplánujeme, tedy specificky podle aktuální situace.

538. [**Platnost zobrazené hodnoty**] Ať už zvolíme jakýkoli postup, na displeji musíme vždy zobrazovat aktuální hodnotu naměřeného času, ne nějakou, která by už platná nebyla, byť jakkoli krátce.

539. [**Vizuální kontrola správnosti**] Na závěr ve vztahu k procesu ladění celé aplikace dodejme, že lidské oko není ani zdaleka schopné detekovat cokoli na úrovni jednotlivých milisekund, takže pouhé vizuální ověření správného fungování naší aplikace nemůže její skutečnou korektnost aneb přesnost měření stopek zaručit.