NSWI170 – **Počítačové systémy** – 2022/23 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 pomocí konceptu dědičnosti. Navrhneme tedy novou třídu pro numerický displej, ta bude odvozena od základního displeje. To nám pochopitelně nebrání v možnosti stávající funkce základního displeje v případě potřeby upravovat nebo přidávat nové, omezíme se však jen na nízkoúrovňové záležitosti.

502. [**Podporovaná doména čísel**] Teprve numerický displej jako takový získá schopnost zobrazovat celá čísla s využitím všech dostupných pozic segmentového displeje. Konkrétně budeme chtít umět zobrazovat jakákoli celá čísla, tedy včetně záporných, stejně jako desetinná čísla s tečkou umístěnou na jakékoli pozici.

503. [**Zobrazení číslic s tečkou**] Pro vlastní zobrazování číslic pochopitelně využijeme stávající funkci základního displeje, jen ji budeme muset rozšířit právě o podporu rozsvícení segmentu s tečkou pro případ vyžádání nepovinné desetinné tečky. Pro vzájemnou kombinaci masek odpovídajících glyfů použijeme vhodnou bitovou operaci.

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

505. [**Myšlenka časového multiplexu**] Při zobrazování různých glyfů na jednotlivých pozicích displeje musíme aplikovat myšlenku časového multiplexu, tedy že zamýšlené číslo nemůžeme zobrazit hned celé najednou v okamžiku vyžádání. Namísto toho si jej jen zapamatujeme a jeho zobrazování rozložíme v čase přes jednotlivé pozice. To znamená, že budeme neustále cyklicky iterovat přes jednotlivé pozice displeje a v každém běhu funkce `loop` dáme nízkoúrovňový pokyn k zobrazení právě jedné z nich. Tím dosáhneme potřebné optické iluze, kdy nabydeme pocitu, že svítí všechny pozice najednou, přestože tomu tak technicky nebude.

506. [**Reprezentace vyžádaného čísla**] Uvnitř třídy numerického displeje si můžeme pamatovat i další potřebné vnitřní informace, základem ale na logické úrovni musí být právě číslo vyžádané k zobrazení jako takové.

507. [**Vynechání úvodních nul**] Čísla budeme na displeji zobrazovat tak, že vynecháme zbytečné úvodní nuly před první platnou pozicí. Detekci takové pozice musíme korektně vyřešit pro čísla celá i desetinná.

508. [**Zobrazení symbolu mínusu**] V případě záporných čísel zobrazíme i glyf mínusu, ten pak umístíme hned před první platnou pozici čísla, nikoli na začátek displeje úplně vlevo.

509. [**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.

510. [**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 je samozřejmě nerealistické. Na druhou stranu je střídání pozic v iteracích funkce `loop` na bázi jedna ku jedné patrně to nejlepší možné řešení, protože jiné metody by musely zahrnout kontrolu časování, a byly by tedy samy o sobě nadměrně komplikované s ohledem na jejich očekávané přínosy.

511. [**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 celou věc z logického pohledu jen znepřehlednilo a reálného přínosu bychom stejně kvůli nárůstu objemu kódu a větvení nedosáhli. Proto se této myšlence vyhneme.

512. [**Efektivní kódu multiplexu**] Na druhou stranu je zřejmé, že kód obsluhující tento multiplexing bude volaný z každé jednotlivé iterace funkce `loop`, a tedy by neměl obsahovat věci, které nebudou samy o sobě dostatečně efektivní.

513. [**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 funkcí, které pro nastavování takových čísel budeme používat. Určitě pak není možné využívat např. nějakou nevalidní pozici jako `-1` pro situace, kdy tečku vlastně nechceme. Naštěstí můžeme elegantně využít mechanismu přetěžování, kdy můžeme mít více funkcí stejného jména, pokud mají jiný počet nebo jiné typy argumentů.

514. [**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 výrazně zhoršilo přehlednost a snížilo uživatelskou přívětivost.

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

516. [**Možnost vypnutí displeje**] Celý numerický displej bychom měli naprogramovat tak, abychom jej měli možnost explicitně deaktivovat. V opačném případě bychom totiž pořád na displeji museli díky multiplexu integrovanému do funkce `loop` něco zobrazovat, což nám obecně nemusí vyhovovat.

517. [**Třída pro stopky**] Podle očekávání veškerou funkcionalitu stopek zapouzdříme do samostatné třídy, která pochopitelně bude využívat služeb tlačítek a displeje, samotná logika stopek však musí být od těchto ovladačů jako tradičně striktně oddělena.

518. [**Postup návrhu stopek**] Návrh této 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.

519. [**Enumerační třída pro stavy**] Přestože by to šlo i jinak, vnitřní stavy stopek je patrně nejrozumnější reprezentovat pomocí hodnot enumerační třídy. Tu stačí deklarovat pomocí konstrukce `enum class MyEnum { VALUE_1, ... }`, k jednotlivým hodnotám se přistupuje pomocí `MyEnum::VALUE_1`.

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

521. [**Pozice desetinné tečky**] Přestože 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 konfigurovatelná.

522. [**Granularita vnitřního času**] Navíc je potřeba si uvědomit, že pro dosažení očekávané přesnosti měření samozřejmě musíme uvnitř stopek akumulovaný čas evidovat s přesnější granularitou, než jakou očekáváme na výstupu.

523. [**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.

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

525. [**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.

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

527. [**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.

528. [**Plánování událostí časovače**] Efektivnější cesta tedy 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.

529. [**Aktuálnost zobrazené hodnoty**] Ať už zvolíme jakýkoli postup, musíme samozřejmě zajistit, že na displeji budeme vždy zobrazovat aktuální hodnotu naměřeného času, ne nějakou, která by už jakkoli krátce platná nebyla.