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

Úkol **A6: Zprávy**

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

601. [**Rozšíření základního displeje**] Stávající funkce základního displeje zachováme beze změny a rozšíříme je zejména o zobrazování jednotlivých písmen anglické abecedy, a to s potlačením jejich velikosti. Nakonec přidáme zastřešující funkci, která přijme jeden libovolný znak a pomocí větvení zařídí zobrazení správných glyfů pro všechny následující situace: písmeno, číslice, mezera pro jakýkoli bílý znak a speciální glyf pro jakýkoli nepodporovaný symbol.

602. [**Rozhraní dílčích funkcí**] Rozhraní funkce na zobrazení konkrétní číslice a funkce na zobrazení konkrétního písmene musí být vymyšlené tak, abychom je mohli používat i samostatně, a tedy byly plnohodnotné a uživatelsky přívětivé.

603. [**Typ parametru symbolu**] Konkrétně u funkce na zobrazení písmene budeme technicky očekávat parametr typu `char` a nikoli ukazatel na `char`, ať už by šlo o samostatný znak, nebo by byl součástí nějakého pole nebo řetězce. Funkce tak bude univerzálnější a navíc i rychlejší.

604. [**Dočasné ladicí glyfy**] Pro účely ladění bude vhodné, abychom dočasně používali jiné než prázdné a hlavně různé speciální glyfy pro mezery a již zmíněné nepodporované znaky. Díky tomu tyto situace dokážeme korektně odlišit, a budeme tedy mít jistotu, že jsme jejich zpracování naprogramovali správně. Teprve před odevzdáním obě konstanty příslušným způsobem upravíme.

605. [**Zachování číslování pozic**] Přestože dává smysl písmena na displeji na rozdíl od číslic zarovnávat doleva, i nadále zachováme konvenci, že pozice na displeji číslujeme od `0` do `3` zprava doleva.

606. [**Reprezentace textového displeje**] Podobně jako u numerického displeje v předcházejícím úkolu, i tentokrát naprogramujeme textový displej pomocí dědičnosti, tedy odvozením od základního displeje. Textový displej pak bude umět zobrazit požadovaný řetězec znaků, a to samozřejmě pomocí metody časového multiplexu.

607. [**Funkce textového displeje**] Ovladač textového displeje si bude pamatovat, a tedy dokáže zobrazit jen čtyři konkrétní symboly. Jinými slovy bude umět zobrazovat řetězce s délkou přesně odpovídající počtu pozic, které na displeji máme. Nikoli delší.

608. [**Nastavení vyžádaného řetězce**] Uvnitř funkce na nastavení nového zobrazovaného řetězce si všechny požadované znaky okopírujeme do své vlastní datové položky. Jinými slovy si nestačí uložit jen ukazatel na tento řetězec, protože nechceme nutit volající, aby nám garantovali, že by takový řetězec zaručeně musel existovat po celou dobu, během které bychom jej chtěli používat.

609. [**Konstantnost řetězce**] Rozhraní této funkce bude očekávat ukazatel na konstantní řetězec, tedy `const char*`. Příznakem konstantnosti nejenom říkáme, že nemáme zájem tento vstupní řetězec měnit, což pak překladač pomůže ohlídat, ale současně je to vhodné i z toho důvodu, abychom tuto funkci vůbec mohli s takovými konstantními řetězci volat. Např. v případě řetězců zadaných přímo literálem.

610. [**Délka vstupního řetězce**] Přestože očekáváme, že délka zadaného řetězce bude přesně odpovídat velikosti našeho displeje, celou funkci přesto musíme naprogramovat tak, abychom zvládli korektně zvládnout i řetězce kratší resp. delší. V prvním případě je zprava doplníme mezerami, ve druhém případě je ořízneme a zbytek jednoduše ignorujeme.

611. [**Kontrola mezí pole**] Současně je potřeba si uvědomit, že korektní ošetření obou předchozích situací je nutné už jen kvůli tomu, abychom zabránili případnému čtení nebo dokonce zápisu za koncem pole. Tyto chyby totiž v lepším případě povedou k okamžitému pádu programu při přístupu do nealokované paměti, v horším případě si přepíšeme nějaká jiná naše data. Problém se pak může projevit kdykoli později a hlavně bude velmi obtížně laditelný.

612. [**Použití pointerové aritmetiky**] Abychom dosáhli efektivnější implementace a současně se naučili lépe pracovat s ukazateli, budeme při práci s našimi řetězci namísto operátoru hranatých závorek manuálně používat právě je. Myslí se tím jakýkoli kód, kde budeme iterovat přes jednotlivé znaky řetězce. Namísto cyklu přes čísla pozic a přístupu k prvku pomocí konstrukce `string[i]` tedy použijeme postupně inkrementovaný ukazatel a `*` pro dereferenci. Přístup na jedinou konkrétní pozici mimo cyklus je výjimkou, tam i nadále hranaté závorky používat můžeme. Nic bychom totiž nezískali.

613. [**Složitější úpravy řetězce**] Jelikož třídu textového displeje vnímáme jako ovladač výstupního zařízení, není možné, aby jeho funkce na nastavení řetězce řešila jakoukoli složitější aplikační logiku, třeba přidávání nějakého počtu mezer před začátek apod.

614. [**Plnění vnitřního pole zvenčí**] Na závěr dodejme, že datová položka vnitřního pole pro aktuální znaky je ve vlastnictví textového displeje, takže obsah tohoto pole opravdu musíme naplnit pomocí diskutované funkce, není možné ukazatel na toto pole poskytnout někomu k naplnění zvenčí.

615. [**Možnost vypnutí displeje**] Stejně jako u numerického displeje i ten textový naprogramujeme tak, abychom jej měli možnost vypnout. A to explicitně, tedy ne nějakým trikem, kdy bychom jen vyžádali zobrazení řetězce obsahujícího jen mezery.

616. [**Třída běžících zpráv**] Celou problematiku posouvání a nastavování jednotlivých textových zpráv považujeme za aplikační problém. Musíme jej tedy řešit samostatnou třídou, která bude od ovladače textového displeje zcela oddělena.

617. [**Zákaz rozšíření displeje**] Stejně tak není možné tuto novou třídu chápat jako další odvozenou a rozšířenou verzi ovladače displeje. Pro dosažení potřebné funkcionality posouvání delších řetězců tedy budeme využívat textový displej a dávat mu pokyny, nemůžeme od něj ale sami dědit.

618. [**Skončení posunu zprávy**] Po zobrazení celé aktuální zprávy skončí displej ve stavu, kdy na všech pozicích budou zobrazeny mezery. K vypnutí displeje nebo explicitnímu zobrazení prázdného glyfu na všech pozicích však nedojde, jen prostě přestaneme s další činností.

619. [**Funkce na detekci konce**] Očekáváme totiž, že logika nastavování nových zpráv bude řešena zvenčí. Kromě funkce umožňující nastavení nové zprávy tedy naše třída musí také nabídnout funkci, pomocí které bude možné zjistit, jestli posouvání poslední zprávy již zcela skončilo, nebo naopak ještě stále probíhá.

620. [**Nastavení nové zprávy**] Přestože by bylo vhodnější, abychom si z předloženého řetězce zprávy vyžádané k zobrazení udělali svou vlastní kopii, spokojíme se zde s tím, že si jen uložíme poskytnutý ukazatel. Jinými slovy se tentokráte naopak můžeme spolehnout na to, že tento ukazatel bude po celou dobu platný a původní řetězec bude dostupný.

621. [**Mechanismus dynamické alokace**] Důvodem je, že naše zpráva může mít libovolnou, a tedy předem neznámou délku. Kdybychom ji opravdu chtěli okopírovat, vedlo by to k nutnosti použití takzvané dynamické alokace paměti. Tomu se však chceme v tomto předmětu vyhnout, protože její využití vyžaduje jistou obezřetnost a disciplínu. Bez ní bychom mohli v běžícím programu nekontrolovaně a nevratně ztrácet dostupnou paměť.

622. [**Libovolná délka zprávy**] Délka zprávy vyžádané k zobrazení může být zcela libovolná, takže třeba i nulová. Nikdy se však nestane, že bychom dostali nějaký neplatný ukazatel.

623. [**Nadbytečné změny displeje**] Samozřejmostí je, že pokyny textovému displeji dáváme jen tehdy, když došlo ke změně, tedy v okamžiku posunu zprávy na další pozici.

624. [**Nedovolené systémové funkce**] V rámci tohoto úkolu se vyvarujeme použití dynamické alokace. Není to potřeba, navíc jsme se ji ani neučili. Stejně tak nebudeme používat prostředky nabízené knihovnou `cstring`.