NSWI170 – **Počítačové systémy** – 2023/24 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 a dle potřeby přidáme nové. Minimálně to bude funkce umožňující zobrazení libovolného písmene anglické abecedy a následně také funkce pro zobrazení libovolného znaku.

602. [**Doména podporovaných symbolů**] Tato druhá zastřešující funkce přijme jeden libovolný znak v podobě datového typu `char` a pomocí větvení zařídí zobrazení správného glyfu pro všechny následující situace: již zmíněná písmena anglické abecedy, dekadické číslice, prázdný glyf pro jakýkoli bílý znak (mezera, tabulátor atp.) a rovněž prázdný glyf pro jakýkoli jiný, řekněme nepodporovaný symbol.

603. [**Potlačení velikosti písmen**] Pokud jde konkrétně o písmena anglické abecedy, potlačíme jejich velikost. To znamená, že na vstupu budeme podporovat písmena malá i velká, pro každou odpovídající dvojici však budeme mít jen jeden jediný glyf.

604. [**Předání symbolů hodnotou**] Obě zmíněné funkce budou očekávat parametr typu `char` a nikoli ukazatel na takový symbol. To povede k rychlejší a univerzálnější implementaci, protože je pak budeme moci zavolat nad samostatnými znaky stejně jako jednotlivými znaky patřícími do řetězců nebo polí.

605. [**Dočasné ladicí glyfy**] Pro účely ladění se vyplatí namísto očekávaných prázdných glyfů pro mezery a nepodporované znaky dočasně používat nějaké jiné, neprázdné a hlavně navzájem různé speciální glyfy. Díky tomu tyto situace (včetně explicitně vyžádaného prázdného glyfu) dokážeme navzájem vizuálně rozlišit, a tedy zvýšíme šanci na vytvoření správně fungujícího kódu. Teprve před odevzdáním hodnoty obou konstant příslušným způsobem upravíme.

606. [**Zachování číslování pozic**] Byť by na displeji dávalo smysl písmena na rozdíl od číslic zarovnávat spíše doleva, i nadále zachováme konvenci, že pozice displeje číslujeme od `0` do `3` zprava doleva.

607. [**Třída textového displeje**] Podobně jako u numerického displeje v předcházejícím úkolu, i tentokrát naprogramujeme textový displej pomocí dědičnosti odvozením od základního displeje. Textový displej pak bude pomocí mechanismu časového multiplexu umět zobrazovat řetězec požadovaných znaků.

608. [**Oddělení ovladače a aplikace**] Cílem tohoto úkolu je naprogramovat zobrazování běžících zpráv. To ovšem neznamená, že by takovou funkcionalitu měl v plném rozsahu implementovat přímo textový displej jako takový. Nezapomeňme totiž, že jej chápeme jako ovladač hardwarového zařízení. Jeho rozhraní a schopnosti tedy sice mohou být netriviální, musí však být univerzálně použitelný a limitovaný jen na rozumně očekávatelnou obecnou funkcionalitu. Nemůže tedy předjímat a řešit specifika aplikací, které jej možná v budoucnu budou chtít využívat a které hlavně zatím ani neexistují.

609. [**Rozhraní textového displeje**] Ovladač textového displeje si proto bude pamatovat a dokáže zobrazit pouze a jenom čtyři konkrétní symboly. Tedy pole znaků s délkou přesně odpovídající počtu pozic, které na displeji máme.

610. [**Alternativní reprezentace řetězce**] Namísto pole znaků bychom alternativně nebo i doplňkově mohli uchovávat jemu odpovídající pole už přeložených glyfů. To na jednu stranu umožní zobrazovat libovolné speciální glyfy, ovšem potlačí logickou podstatu celého textového displeje.

611. [**Nastavení vyžádaného řetězce**] Uvnitř funkce na nastavení nového řetězce je potřeba si všechny znaky do naší vnitřní datové položky překopírovat. Jinými slovy si nestačí jen zapamatovat ukazatel na tento řetězec. Nechceme totiž volajícího nutit, aby nám neměnnost a vůbec samotnou existenci takového řetězce zaručoval po celou dobu, během které jej budeme používat.

612. [**Konstantnost vstupního řetězce**] Z hlediska rozhraní budeme očekávat ukazatel na konstantní řetězec, tedy `const char*`. Příznakem konstantnosti totiž říkáme, že nemáme zájem obsah tohoto řetězce měnit, což pak překladač pomůže vynutit. Důležitější však je, že bychom jinak tuto funkci s konstantními řetězci ani nedokázali zavolat. A tím se myslí i anonymní řetězce zapsané literálem.

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

614. [**Ošetřování nízkoúrovňových chyb**] Pokud bychom zmíněné situace nechtěli detekovat, jistě by to bylo v souladu se zvyklostmi, které jsme si ohledně ošetřování obdobných nízkoúrovňových chyb vysvětlili už v předchozím úkolu. Abychom se však s řetězci naučili lépe pracovat a důsledky takových chyb pochopili, tentokrát je pečlivě ošetříme.

615. [**Kontrola mezí pole**] Konkrétně potřebujeme zabránit 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. Samotný problém se pak může projevit podivným chováním programu kdykoli později a bude jej velmi obtížné odladit.

616. [**Řetězce a pole znaků**] Tradiční řetězec je v jazyce C++ namodelován jako obyčejné pole znaků, které však na svém konci obsahuje ještě jeden speciální ukončovací znak `'\0'`. Právě díky němu konec řetěze poznáme, protože řetězec sám od sebe svou délku nezná. Naše nastavovací funkce by měla korektně zvládnout pracovat jak s těmito řetězci, tak i s obecnými poli znaků bez uvedeného ukončení.

617. [**Použití pointerové aritmetiky**] Pokud chceme sekvenčně zpracovat celý řetězec nebo alespoň nějakou jeho větší souvislou část, je rychlejší namísto operátoru hranatých závorek manuálně používat ukazatele. Namísto cyklu přes čísla pozic a přístupu k prvku pomocí konstrukce `string[i]` tedy v takových situacích použijeme postupně inkrementovaný ukazatel a operátor `*` pro dereferencování.

618. [**Omezení počtu průchodů**] Vstupní řetězec zpracujeme jen na jeden průchod, více jich není potřeba. Stejně tak nemusíme zjišťovat ani délku tohoto řetězce.

619. [**Plnění vnitřního pole zvenčí**] Na závěr dodejme, že datová položka vnitřního pole pro aktuální znaky je ve vlastnictví textového displeje, takže obsah tohoto pole musíme naplnit pomocí diskutované nastavovací funkce zevnitř. Jinými slovy není možné ukazatel na toto vnitřní pole poskytnout někomu k naplnění zvenčí, tím bychom nad ním ztratili kontrolu.

620. [**Rozšířené ovládací funkce**] Jak už jsme konstatovali, textový displej nemůže řešit vlastní logiku běžících zpráv. To ale na druhou stranu neznamená, že vedle základní funkce na nastavení celého řetězce nemůžeme nabídnout i nějaké další, například na nastavení znaků na konkrétních pozicích nebo pro základní logické úpravy stávajícího řetězce, třeba ve smyslu jeho posunu nebo doplnění.

621. [**Možnost vypnutí displeje**] Stejně jako u numerického displeje i ten textový naprogramujeme tak, abychom jej měli možnost explicitně vypnout.

622. [**Třída běžících zpráv**] Veškerou funkcionalitu běžících zpráv zapouzdříme do samostatné třídy. Ta bude pro dosažení potřebného chování samozřejmě využívat a ovládat textový displej.

623. [**Zákaz dalšího rozšiřování displeje**] Třída zpráv musí být od našeho textového displeje zcela oddělena, stejně tak ji nemůžeme implementovat jako jeho další rozšíření pomocí dědičnosti. Jednoduše proto, že nejde o žádný ovladač, ale o konkrétní aplikaci.

624. [**Dočasné datové položky**] Každá třída by měla obsahovat jenom takové datové položky, které mají dlouhodobý charakter. Neměli bychom je proto používat pro ukládání čistě dočasných nebo pomocných údajů, u kterých by stačily obyčejné lokální proměnné uvnitř funkcí.

625. [**Zakrývání interních metod**] Podobně jako běžně používáme privátní datové položky, měli bychom za privátní označovat i metody, které jsou čistě interní povahy a nejsou určeny pro přímé použití uživatelem, zejména pokud by jejich neuvážené vykonání dokonce mohlo způsobit nějaké nekonzistence nebo jiné komplikace.

626. [**Získávání zpráv k zobrazení**] Za účelem získávání zpráv určených k zobrazení si vytvoříme instanci předpřipravené třídy `SerialInputHandler` z přiloženého hlavičkového souboru `input.h`. Ta implementuje přijímání zpráv posílaných z počítače do Arduina přes sériovou linku.

627. [**Alternativní zdroje zpráv**] Přestože v rámci tohoto úkolu budeme používat výhradně uvedený mechanismus získávání zpráv, chceme podporovat i alternativní přístupy. To znamená, že naše třída běžících zpráv nemůže náš výchozí zdroj jakkoli pevně zadrátovat, a tedy umožní ovládání celého procesu zvenku prostřednictvím vhodně navržených metod.

628. [**Veřejné rozhraní třídy zpráv**] To zahrnuje přinejmenším metodu umožňující nastavení nové zprávy k zobrazení, hodit by se však mohla i možnost detekce úspěšně dokončeného běhu zprávy nebo případně i další metody v závislosti na zvoleném způsobu implementace.

629. [**Ovládání celého procesu**] Po zobrazení celé aktuální zprávy je nejrozumnější vnitřní mechanismus posunu zpráv zastavit, nic dalšího nedělat a jednoduše jen čekat na případný pokyn k zobrazení další zprávy. To elegantně dovolí jednorázové použití, stejně tak opakované použití téže zprávy, nebo i postupné střídání různých zpráv.

630. [**Nastavení nové zprávy**] Přestože by bylo vhodnější, abychom si z předaného řetězce zprávy udělali svou vlastní kopii, spokojíme se jen s uložením ukazatele na něj. Jinými slovy se tentokráte naopak spolehneme na to, že tento ukazatel bude po celou dobu platný a původní řetězec dostupný.

631. [**Dynamická alokace paměti**] Důvodem je, že naše zpráva může mít libovolnou a předem neznámou délku. Kdybychom ji opravdu chtěli okopírovat, museli bychom použít takzvanou dynamickou alokaci paměti. Tomu se však chceme v tomto předmětu vyhnout, protože její využití vyžaduje jistou úroveň obezřetnosti a disciplíny. V opačném případě bychom mohli v běžícím programu nekontrolovaně a nevratně ztrácet dostupnou paměť.

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

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

634. [**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`.