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

Úkol **A2: Kulička**

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

201. [**Dekompozice do funkcí a tříd**] Veškerý kód dekomponujeme do vhodně navržených globálních funkcí, tříd a jejich členských funkcí (metod), a to na různých úrovních abstrakce od uživatelsky přívětivého ovládání jednotlivých diod až po řešení logiky samotné běhající kuličky.

202. [**Třída pro reprezentaci diod**] Jednotlivé diody reprezentujeme jako instance třídy, která zapouzdří potřebné datové položky i funkcionalitu. To minimálně znamená dvě funkce, jedna na inicializaci dané diody a druhá na změnu jejího stavu rozsvícení. Pomocí těchto funkcí zabalíme příslušná nízkoúrovňová systémová volání Arduina, čímž uživatelům nabídneme přívětivější možnost ovládání diod na vyšší úrovni abstrakce.

203. [**Privátní datové položky**] Dobrou praxí při návrhu tříd našeho druhu je, že všechny jejich datové položky učiníme privátními. Manipulace s nimi tedy nebude možná přímo zvenku, ale jen nepřímo prostřednictvím veřejných metod, které za tímto účelem navrhneme. Díky tomu budeme mít jejich obsah pod kontrolou, jinak by je totiž mohl měnit kdokoli a odkudkoli bez našeho vědomí.

204. [**Logická čísla diod**] Abychom opravdu vyšší abstrakce mohli dosáhnout, budeme všude v kódu používat výhradně logická čísla diod `0` až `3` namísto nízkoúrovňových pinů. Tato čísla pak v tomto pořadí budou odpovídat diodám s označeními `D1` až `D4`, a tedy pinům `led1_pin` až `led4_pin`. Jedinou výjimkou, kde můžeme pracovat přímo s piny, budou právě dvě výše zmíněné funkce, které jako jediné budou interně provádět potřebný překlad logických čísel na odpovídající piny.

205. [**Používání hodnot čísel pinů**] Nahlédnutím do přiloženého hlavičkového souboru se sice můžeme snadno dopátrat, že čísla pinů jednotlivých diod mají hodnoty `13` až `10`, určitě s těmito konkrétními čísly ale nemůžeme jakkoli přímo nebo i nepřímo pracovat. Nemůžeme ani předpokládat, že jsou případně nějak uspořádaná nebo že tvoří souvislý interval.

206. [**Nezávislost na hodnotách konstant**] Tuto myšlenku můžeme zobecnit a rozšířit i na práci s jakýmikoli jinými konstantami, které nemáme plně pod kontrolou. Pokud nám jejich autor explicitně nepřislíbil nějaké speciální záruky, nemůžeme jednoduše vůči jejich konkrétním hodnotám uplatňovat žádné předpoklady. Musíme tedy na nich být zcela nezávislí.

207. [**Variabilní počet diod**] Celá aplikace musí být naprogramována tak, abychom nikde nevynucovali ani nepředpokládali nějaký konkrétní fixní počet diod. Jinými slovy vše musí být univerzální a fungovat i při jiném počtu diod než zrovna `4`. Konkrétní počet dostupných diod pak odvodíme z velikosti námi vytvořeného pole, které budeme používat na překlad logických čísel diod na jejich piny.

208. [**Počáteční vypnutí diod**] V rámci inicializace Arduina je nutné kromě nastavení módů pinů zařídit i vypnutí všech diod, protože obecně nemáme zaručeno, v jakém aktuálním stavu se mohou vyskytovat.

209. [**Předčasná systémová volání**] Přestože bychom se do takové situace s našimi prostředky vlastně ani neměli dostat, zdůrazněme, že nesmíme žádné systémové funkce Arduina volat před funkcí `setup`, tedy před jeho inicializací. Zatímco na reálném Arduinu by taková volání obvykle prošla bez jakéhokoli efektu, emulátor v ReCodExu takové situace hlídá striktně.

210. [**Konstruktory objektů diod**] Výše popsaná komplikace by specificky mohla nastat v situaci, kdy bychom pro naše objekty diod chtěli definovat explicitní konstruktory, v rámci nichž bychom např. chtěli nastavovat módy pinů. Instance objektů globálních proměnných se totiž vytváří už na začátku běhu programu, tedy před funkcí `setup`. Jelikož se ale bez konstruktorů v našich třídách snadno obejdeme, vytvářet je nebudeme. Pokud přesto ano, musíme si být tohoto chování vědomi.

211. [**Vnitřní stav diod**] Pokud se rozhodneme pamatovat si aktuální vnitřní stav diod, určitě je potřeba jej reprezentovat na logické úrovni, nikoli pomocí nízkoúrovňových konstant `LOW` a `HIGH`.

212. [**Pojmenované konstanty**] Všechny konstanty mající nějaký logický význam ve vztahu k našemu programu je nutné deklarovat a vhodně pojmenovat. Je-li to navíc možné s ohledem na jiné již definované konstanty, je současně nutné jejich hodnotu pomocí takových konstant i spočítat či jinak odvodit.

213. [**Příznak konstantnosti**] Také nezapomeňme na to, abychom samotný příznak konstantnosti jako takové uvedli u každé datové položky, která takový charakter má.

214. [**Globální proměnné**] S ohledem na programovací model nabízený Arduinem je nutné, abychom si řadu informací pamatovali prostřednictvím globálních proměnných. Jinak bychom je totiž nemohli mezi funkcí `setup` a jednotlivými voláními funkce `loop` přenášet. Počet takových proměnných však musíme držet na nezbytném minimu, zejména nesmíme používat globální proměnné v situacích, kde by ve skutečnosti stačily jen obyčejné lokální.

215. [**Jednoduchý obsah loopu**] Ať už logiku naší pohybující se kuličky dekomponujeme jakkoli, je nutné zajistit, že kód uvedený přímo v těle funkce `loop` nebude příliš rozsáhlý, komplikovaný nebo technický. V ideálním případě bychom zde měli řešit jen nějaké základní operace na co nejvyšší úrovni abstrakce, podobně jako tomu bylo u funkce `main`.

216. [**Efektivní implementace loopu**] Funkce `loop` tvoří jádro našeho programovacího modelu, má tedy klíčový dopad na efektivitu celé naší aplikace. A nejde samozřejmě jen o funkci samotnou, ale i všechny další funkce, které z ní budeme přímo či nepřímo volat. Samotná funkce `loop` je totiž vykonávána neustále pořád dokola, a to přibližně 1000× do sekundy. V závislosti na složitosti programu to však může být i o řád častěji nebo naopak méně často.

217. [**Třída pro reprezentaci časovače**] Funkcionalitu pro časování událostí také vyřešíme pomocí vhodně navržené samostatné třídy. Bude se nám to totiž hodit v budoucnu, kdy budeme potřebovat využívat dokonce více paralelně běžících a navzájem nezávislých časovačů najednou.

218. [**Detekce uplynutí času**] Samotnou kontrolu uplynutí požadovaného časového intervalu je potřeba vyřešit obezřetně. Jak už jsme diskutovali, jedna iterace funkce `loop` sice může trvat podstatně kratší dobu než 1 ms, stejně tak ale i delší. Tím pádem nemáme garantováno, že budeme schopni zamýšlený časový okamžik další události detektovat s absolutní přesností, tedy na úrovni jednotlivých milisekund.

219. [**Čas předchozí události**] Pro účely detekce další události si zjevně budeme muset zapamatovat čas té předcházející, tedy poslední provedené. Pokud ale nejsme schopni časové okamžiky detekovat přesně, bylo by chybou uložit si skutečný aktuální čas, kdy k této události došlo. Kdybychom to totiž udělali, systematicky by mohlo docházet k opakovanému zpožďování, které by se nám postupně akumulovalo a narušilo by celkovou požadovanou pravidelnost událostí.

220. [**Zjišťování aktuálního času**] Volání funkce `millis` sice není časově přespříliš náročné, stejně tak ale ani rychlé. Z tohoto důvodu je vhodné zajistit, že ji v rámci jedné iterace funkce `loop` zavoláme za účelem zjištění aktuálního času pouze a jen jednou. To nám navíc přinese i další výhody. Jakkoli to totiž zatím nepotřebujeme, pokud bychom v jedné iteraci zjišťovali aktuální čas vícekrát, mohlo by se stát, že dostaneme jiné hodnoty. To by pak v závislosti na našem kódu mohlo přinést i velmi nepříjemné chybové situace způsobené takovou nekonzistencí.

221. [**Přetečení hodnoty času**] Pro reprezentaci systémového času se používá datový typ `unsigned long`. Ten nám umožní uchovat čas v milisekundách odpovídající až necelým 50 dnům, i tak ale dříve nebo později k přetečení jeho hodnoty nevyhnutelně dojde. I takovou situaci tudíž musíme umět korektně ošetřit.

222. [**Aritmetika nad čísly diod**] S proměnnými bychom vždy měli pracovat v souladu s jejich datovým typem, ale také očekáváními vyplývajícími z jejich logické podstaty. Pokud například pozice diod reprezentujeme pomocí jejich logických čísel, nemůžeme nad nimi řešit násobení nebo počítat absolutní hodnotu, jakkoli z technického pohledu na věc takové operace celá čísla jako taková připouští.

223. [**Zákaz neplatných hodnot**] Analogicky také nemůžeme pracovat s pozicemi jako `-1`, které ani samy o sobě nejsou validní. Tuto myšlenku pak můžeme zobecnit tak, že ani v jiných situacích nikdy nebudeme vyčleňovat a používat nějaké jinak normální hodnoty pro reprezentaci chybných, okrajových nebo třeba neočekávaných situací, hodnot apod.

224. [**Realizace prvotní akce**] V rámci funkce `setup` se očekávají nejrůznější inicializační akce, není tedy vhodné zde začít realizovat operace, které jsou součástí až standardní běhové logiky naší aplikace jako takové. Jinými slovy zde konkrétně ještě nemůžeme rozsvítit diodu odpovídající výchozí pozici pohybující se kuličky.

225. [**Zombie akce v loopu**] V návaznosti na požadavek efektivity funkce `loop` je potřeba se vždy velmi pečlivě zamyslet nad tím, jaké akce v rámci ní chceme provádět. Zcela nepřijatelné by například bylo, pokud bychom do ní vložili nějakou podmíněnou akci, u které bychom ale ve skutečnosti už předem věděli, že bude provedena jen jednou jedinkrát hned v první iteraci a pak už nikdy.

226. [**Význam aktuální pozice**] Pro zvládnutí logiky pohybující se kuličky budeme patrně muset nějak řešit její aktuální polohu. Měli bychom se tudíž zamyslet nad tím, co přesně má v tomto kontextu slovo *aktuální* vlastně znamenat. Jinými slovy bychom se měli ujistit, že onou aktuální polohou ve skutečnosti nemyslíme například polohu předcházející nebo naopak následující.

227. [**Šablona pozic kuličky**] Přestože by vlastně šlo o hezký nápad, určitě náš problém pohybu kuličky nechceme řešit tak, že bychom si vytvořili jakousi šablonu posloupnosti očekávaných poloh naší kuličky, přes které bychom následně iterovali. Neodpovídá to totiž logice našeho zadání, které pochopitelně musíme vždy respektovat.

228. [**Logické pořadí akcí**] Při realizaci vlastního pohybu kuličky je více než vhodné respektovat intuitivní pořadí jednotlivých akcí, tedy nejprve vypnout stávající diodu, následně provést přesun na další pozici a teprve pak novou diodu zapnout. Odlišná uspořádání by byla zavádějící.

229. [**Nepovolené systémové funkce**] Celá naše aplikace musí být naprogramována takovým způsobem, abychom se obešli bez použití funkcí `delay` nebo `delayMicroseconds`, stejně jako nesmíme ani žádným jiným způsobem blokovat průběh funkce `loop`. Jednoduše proto, že by takovéto snahy šly přímo proti logice celého našeho programovacího modelu.