

## Úkol A1: Teploměr

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

101. [**Dekompozice kódu**] Každou netriviální úlohu je obecně nutné dekomponovat do vhodně navržených menších a jednodušších celků, přičemž každý z nich by měl řešit jeden dílčí a dobře logicky vymezený a ohraničený úkol. Jde o jeden z klíčových aspektů psaní dobrého kódu, a proto jej budeme důsledně dodržovat už od samotného začátku.
102. [**Dekompozice pomocí funkcí**] V rámci tohoto konkrétního úkolu si při dekompozici vystačíme jen s návrhem jednotlivých funkcí. Pokud bychom řešili složitější úlohy, obdobně bychom postupovali i pro větší celky na úrovni celých souborů, modulů atp.
103. [**Indikátor dobrého návrhu**] Prvotním ukazatelem dobrého návrhu často bývá i schopnost nalezení vhodných názvů, v našem případě tedy názvů funkcí a vlastně i jejich parametrů. Naopak podivné, neurčité nebo až příliš dlouhé názvy by měly být varováním.
104. [**Důležitost kvalitních názvů**] To nás přivádí k obecnému principu, kdy názvy všech funkcí, jejich parametrů, lokálních i globálních proměnných, konstant a vlastně i čehokoli dalšího by měly být rozumně stručné, přesto výstižné a vysvětlující z hlediska jejich významu a očekávaného použití.
105. [**Pojmenovávací konvence**] Není až tak důležité, jakou konkrétní konvenci pojmenovávání budeme používat, vždy však musíme být v celém našem kódu konzistentní. Funkce, proměnné a parametry se nicméně obvykle píšou s malými počátečními písmeny. Také je potřeba promyslet způsob zápisu víceslovných názvů.
106. [**Použití angličtiny**] Jelikož v dnešní době často na vývoji aplikací pracujeme v týmech, navíc mezinárodních, je více než vhodné veškerý kód psát automaticky v angličtině. A nejde jenom o již diskutované názvy, ale také o komentáře.
107. [**Struktura zdrojového souboru**] Na začátku zdrojového souboru uvedeme potřebné deklarace na načtení hlavičkových souborů, následně případné konstanty, teprve pak jednotlivé funkce, a to v takovém pořadí, abychom vždy měli definované všechny ty funkce, které již aktivně chceme používat. Funkce `main` tedy bude na úplném konci.
108. [**Obsah funkce main**] Funkce `main` jako taková by obecně neměla obsahovat žádný složitý nebo z pohledu logiky aplikace příliš nízkourovňový nebo technický kód. V našem případě se v podstatě můžeme omezit jen na volání jediné funkce, která bude reprezentovat a která zapouzdří veškerou funkcionalitu našeho nespolehlivého teploměru.
109. [**Globální proměnné**] V rámci tohoto úkolu se zcela obejdeme bez globálních proměnných, nebudeme je potřebovat. Jinými slovy rozhraní všech funkcí a případné návratové hodnoty navrhne tak, abychom nad tokem a změnami dat měli plnou kontrolu.
110. [**Alternativní vstupy**] Abychom měli technicky co nejjednodušší situaci, máme vstupní teploty zadané formou globálního konstantního pole. Je ale zřejmé, že bychom je mohli získat i jinak, třeba z argumentů předaných při spuštění programu, ze standardního vstupu nebo i nějakého vstupního souboru na disku. Očekává se tedy, že náš teploměr bude naprogramovaný tak, abychom jej případně mohli použít i nad jinými vstupy, a to i opakovaně. Jinými slovy není možné naše vstupní pole teplot jakkoli ve výkonném kódu napevno zadrátovat.
111. [**Úpravy vstupního pole**] S polem zadaných teplot budeme pracovat jen v režimu čtení, určitě tedy naši úlohu nebudeme řešit tak, že bychom si nejprve vyrobili kopii vstupního pole a nějakým způsobem v něm jednotlivé hodnoty upravovali. Jinými slovy obsah a význam vstupních dat měnit nechceme, jen je chceme konkrétním způsobem použít a zpracovat.

112. [ **Velikost pole** ] Nezapomeňme také, že pole samo o sobě nemůže znát svoji velikost, takže tento údaj musíme vhodně předávat společně s polem samotným. Současně platí, že oba tyto údaje ze své podstaty logicky patří k sobě, takže jeden bez druhého úplně nedává smysl. Měli bychom k nim tedy přistupovat podobně i z technického pohledu na věc.
113. [ **Minimální teplota** ] Při hledání minimální teploty za účelem zarovnání grafu teplot je vhodné dobře si rozmyslet, jakou výchozí (nebo jinými slovy nejhorší možnou) hodnotu předpokládá samotné zadání. V žádném případě není možné si jen tak vymýšlet nějaké konstanty, které v zadání nejsou předpokládány, jakkoli by se nám mohly zdát dostatečně vhodné, zejména velké nebo malé.
114. [ **Uniformní zpracování teplot** ] Je také vhodné přistupovat ke každému prvku v poli teplot stejně, žádná z platných teplot přeci nemá nějaké výjimečné postavení. Speciálně např. jen proto, že je uvedena jako první v pořadí. Zpracování všech hodnot by tedy mělo být z hlediska kódu rovnocenné.
115. [ **Minimální vs. nejmenší teplota** ] Konečně poznamenejme, že minimální prvek je matematicky něco jiného než nejmenší prvek, takže bychom na to měli myslet při pojmenovávání.
116. [ **Vypisování symbolů** ] Je zřejmé, že při vypisování grafu teplot budeme často potřebovat vypisovat nejrůznější počty určitých symbolů. Přímo se tedy nabízí, abychom si za tímto účelem připravili nějakou pomocnou, dostatečně univerzální funkci a tu pak jen opakovaně používali.
117. [ **Opakování podobného kódu** ] Předchozí myšlenka vlastně není o ničem jiném, než že bychom se vždy měli snažit vyvarovat se zbytečného opakování podobných nebo dokonce stejných fragmentů kódu. Tedy pokud k tomu není nějaký zvláštní důvod, což v našem případě není.
118. [ **Přehnané větvení a zanořování kódu** ] Při vypisování jedné konkrétní teploty je vhodné vyvarovat se přílišného nadužívání podmíněných výrazů, natož komplikovaných nebo zanořovaných. Zvláště když to v tomto případě není potřeba. Lepší je tedy snažit se namísto větvení hlavního kódu očekávaný počet symbolů spíše spočítat pomocí nejrůznějších operací nebo třeba i ternárního operátoru.
119. [ **Volání funkce s různými parametry** ] Pokud chceme volat nějakou funkci s různými hodnotami jednoho nebo dokonce více parametrů podle vzniklé situace, určitě problém nebudeme řešit pomocí větvení hlavního kódu a volání této funkce v každé větvi. Jinými slovy zachováme celkově jen jedno volání této funkce a pomocí větvení nebo třeba ternárního operátoru budeme řešit jen parametry samotné.
120. [ **Zapamatování si poslední platné teploty** ] Při vypisování grafu teplot bude zjevně nutné vyrovnat se situací, kdy aktuální hodnota není platná (tedy že jde o `no_value`). V takovém případě by rozhodně nebylo dobrým nápadem pokoušet se polem pohybovat v opačném směru a poslední předcházející platnou teplotu nějak zpětně zkoušet dohledávat. A tím spíše ne pomocí nějaké rekurzivní funkce.
121. [ **Přenášení teploty přes iterace cyklu** ] Pokud potřebujeme přes jednotlivé iterace cyklu přenášet nějaké hodnoty, omezíme se jen na nezbytně nutnou informaci odpovídající logické podstatě řešeného problému. Nebudeme tedy přenášet žádné dopočítané, technické nebo pomocné údaje, ale opravdu jen teplotu jako takovou.
122. [ **Obecné zásady kvality** ] Případné komplikovanější části kódu budou doplněny o stručné komentáře. Samozřejmostí je dodržování všech obecných zásad kvalitního kódu. To například znamená, že bychom kód neměli psát pro sebe, ale pro někoho jiného. Tedy vytvářet takový kód, který bude srozumitelný i pro ostatní. Včetně nás samotných za několik let, kdy si už nepochybně nic pamatovat nebudeme.
123. [ **Nedovolené prostředky jazyka** ] V rámci úkolu se vyvarujeme použití jakýchkoli pokročilých prostředků, které jsme se neučili. Především nebudeme používat řetězce `std::string`, knihovnu `iostream` apod. Nechceme se totiž vzdálit od našeho záměru nízkoúrovňového programování. Navíc v dalších úkolech budou naše hardwarové prostředky při práci s Arduinem omezené a také se vždy budeme snažit o co nejvíce efektivní kód.