Kapitola 3. Pole

Jednoduché úlohy

ÚLOHA 3-J-00

Projděte si kapitolu 3 knihy *Moderní programování – učebnice pro středně pokročilé* a snažte se všemu porozumět.

ÚLOHA 3-J-01 (VŠECHNA ÁČKA)

Program *Počítání áček* z učebnice upravte tak, aby počítal všechna áčka – kromě malých také velká, případně i dlouhá.

ÚLOHA 3-J-02 (POČET DISKOVÝCH JEDNOTEK)

Program *Diskové jednotky* z učebnice upravte tak, aby namísto výpisu všech jednotek uživateli pouze zobrazil jejich počet.

ÚLOHA 3-J-03 (V OPAČNÉM POŘADÍ)

Program *Diskové jednotky* z učebnice vypisuje seznam jednotek v abecedním pořadí (A až Z), tj. přesně tak, jak jej vrací metoda `Environment.GetLogicalDrives`. Upravte jej, aby seznam vypisoval v obráceném pořadí (Z až A).

ÚLOHA 3-J-04 (ÚPRAVY *EVIDENCE NÁJEMNÍKŮ*)

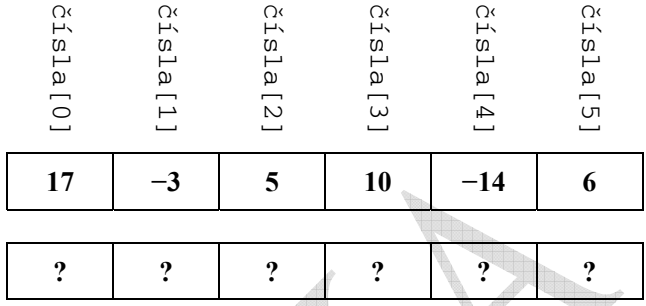
V programu *Evidence nájemníků* z učebnice proveďte následující úpravy:

- Přidejte tlačítko „Vystěhuj přízemí“, po jehož stisku se vystěhuje nájemník bytu č. 0.
- Při nastěhování kontrolujte, zda byt již není obsazen. Pokud zjistíte, že je, oznamte to uživateli a nastěhování stornujte.
- Tlačítko „Vystěhovat“ bude funkční pouze tehdy, když je v příslušném textovém poli zapsáno číslo z rozmezí 0 až 12. V opačném případě bude zablokováno.
- Pokud se po stisku tlačítka „Zobrazit“ zjistí, že daný byt je prázdný, nevypíše se „V bytě XX bydlí:“ a nic. Namísto toho se vypíše „V bytě XX nikdo nebydlí.“.
- Přidejte tlačítko „Zdvojnásob nájem“, po jehož stisku se vystěhují nájemníci ze všech bytů.

Kontrolní otázky

ÚLOHA 3-K-00 (CO TO DĚLÁ?)

Pole `čísla` celých čísel je naplněno hodnotami podle obrázku.

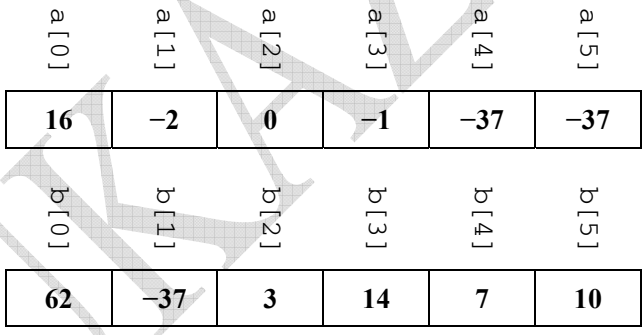


Zapište, jaké budou v poli hodnoty po provedení následujícího kódu:

```
for (int index = 1; index < čísla.Length; index += 2)
    čísla[index] *= 3;
```

ÚLOHA 3-K-01 (INDEX A HODNOTA)

Celočíselná pole `a`, `b` jsou zaplněna hodnotami podle obrázku.



Jaké budou hodnoty čtyřech výsledkových proměnných po provedení následujícího kódu?

```
int výsledek1 = b[4]*a[1];
int výsledek2 = a[1]*b[4];
int výsledek3 = a[b[2]];
int výsledek4 = b[a[2]];
```

ÚLOHA 3-K-02 (KDE JE CHYBA?)

Programátor chtěl deklarovat inicializované pole řetězců. Zapsal tento řádek:

```
string[] jména = new string {"Anna", "Dana", "Hana", "Jana"};
```

Program se ale kvůli syntaktické chybě nesestavil. V čem byla chyba? Jak je to správně?

ÚLOHA 3-K-03 (KDE JE DALŠÍ CHYBA?)

Po odstranění syntaktické chyby programátor z Úlohy 3-K-02 (Kde je chyba?) pokračoval následujícím kódem ve snaze obsah pole `jména` zobrazit uživateli.

```
string zpráva = null;
for (int index = 0; index <= jména.Length; index++)
    zpráva += jména[index] + Environment.NewLine;
MessageBox.Show(zpráva);
```

Program se sestavil, ale po jeho spuštění došlo k běhové chybě. Proč? Jak to má být správně?

ÚLOHA 3-K-04 (V ČEM JE ROZDÍL?)

Pro celočíselné pole `čísla` a celočíselnou proměnnou `index` vysvětlete rozdíl mezi příkazem

```
čísla[index] = index + 1;
```

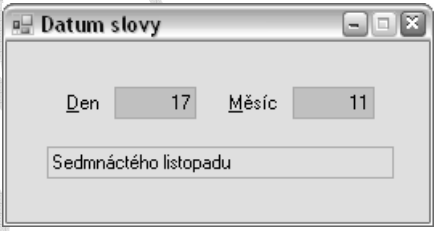
a příkazem

```
čísla[index] = čísla[index + 1];
```

Aplikační úlohy

ÚLOHA 3-A-00 (DATUM SLOVY)

Připravte program, který slovy vypíše datum podle zadaného čísla dne a měsíce. S neexistujícími daty, jako je třeba 30. února, si pro jednoduchost starosti nedělejte.



ÚLOHA 3-A-01 (KULOVÝ BLESK)

Do programu *Evidence nájemníků* z učebnice přidejte tlačítko „Kulový blesk“, které bude simulovat hromadnou výměnu bytů ze stejnojmenného filmu. Po stisku tlačítka se nájemník bytu 1 přestěhuje do bytu 0, nájemník bytu 2 do bytu 1, nájemník bytu 3 do bytu 2, …, nájemník bytu 12 do bytu 11 a nakonec nájemník bytu 0 do bytu 12.

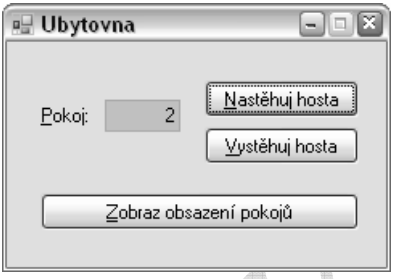
ÚLOHA 3-A-02 (ZPĚTNÝ KULOVÝ BLESK)

V řešení Úlohy 3-A-01 (Kulový blesk) obraťte směr stěhování. Nyní se bude stěhovat z 0 do 1, z 1 do 2, z 2 do 3, …, z 11 do 12 a z 12 do 0.

ÚLOHA 3-A-03 (UBYTOVNA)

Připravte program *Ubytovna* evidující obsazení ubytovny s deseti třílůžkovými pokoji. Pomocí příslušných ovládacích prvků bude možno hosty na pokoje stěhovat a vystěhovávat. Na vyžádání program vypíše momentální obsazení všech pokojů. Je-li nějaký pokoj plný, není možno do něj nastěhovat dalšího hosta.

Upřesnění: V tomto programu nejde o jména hostů, pouze o jejich počty.



Náročnější varianta: Uvažujte různé počty lůžek pro různé pokoje, např. podle uvedené tabulky.

Číslo pokoje	*Počet lůžek*
0–3, 8	3
4–6	4
7	6
9	8

ÚLOHA 3-A-04 (DOSTIHY PRO DESET KONÍ)

Ve dvanácté kapitole učebnice pro začátečníky se vytvářela hra Dostihy, v níž soupeřili dva koně. Program této hry upravte pro deset koní. Pro jednoduchost není třeba zadávat jejich jména.

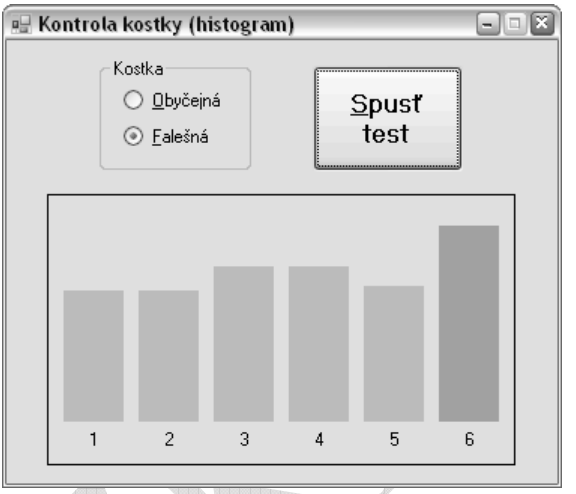
ÚLOHA 3-A-05 (ODHALENÍ FALEŠNÉ KOSTKY)

V desáté kapitole sbírky úloh pro začátečníky se vytvářela falešná hrací kostka, na níž šestka padala stejně často jako všechna ostatní čísla dohromady. Neférovost této kostky by okamžitě byla zřejmá každému uživateli. Pokud by však šestka padala jen o něco častěji, tak by to hned tak poznat nebylo. V takovém případě však může napomoci tabulka četností.

Připravte program, jehož součástí bude falešná kostka, u níž šestka padne o 20 % častěji než třeba pětka nebo jiné číslo. Program sám svoji nepoctivost odhalí zobrazením tabulky četností např. pro 600 opakovaných hodů.

ÚLOHA 3-A-06 (HISTOGRAM)

Výsledné tabulky četností z programu *Kontrola kostky* z učebnice a z Úlohy 3-A-05 (Odhalení falešné kostky) zobrazte graficky jako histogram – sloupcový diagram četností.



Tip: Chcete-li u sloupečků zobrazit také popisky 1 až 6, použijte volání `DrawString`.

Tip: Chcete-li v uživatelském rozhraní umožnit výběr mezi obyčejnou a falešnou kostkou, jak je tomu na obrázku, můžete použít přepínač tvořený knoflíky „RadioButton" z Toolboxu.

Technické úlohy

ÚLOHA 3-T-00 (TABULKA ČETNOSTÍ OD NULY)

Program *Kontrola kostky* z učebnice přepracujte do druhé varianty diskutované v učebnici, tj. s polem indexovaným [0] až [5] a odečítáním jedničky.

ÚLOHA 3-T-01 (DRAWLINE A POINT)

Metodu `DrawLine` používáme vždy s pěti parametry:

```
kp.DrawLine(pero, xPoč, yPoč, xKonc, yKonc);
```

Někdy však může být šikovnější pracovat přímo s body, proto existuje i ve variantě se třemi parametry

```
kp.DrawLine(pero, bodA, bodB);
```

v níž se koncové body `bodA` a `bodB` zadávají jako objekty `Point`. Tuto variantu si nyní ozkoušejte.

ÚLOHA 3-T-02 (POLE LOGICKÝCH HODNOT)

Program *Evidence nájemníků* z učebnice upravte tak, že u každého bytu se bude zaznamenávat pouze údaj typu ano/ne, neboli jestli je byt obsazen či nikoli. Pro reprezentaci těchto údajů použijte pole logických hodnot `bool[]`.

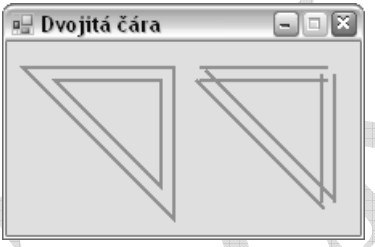
Rozšiřující úlohy

ÚLOHA 3-R-00 (DVOJITÁ ČÁRA)

Pole se dají využívat pro specifikaci per kreslících dvojité či vícenásobné čáry. V takovém případě se do vlastnosti `CompoundArray` pera přiřadí pole desetinných čísel (ale pozor, typu `float`, nikoli `double`!) rostoucích od nuly do jedničky. Tato čísla představují „souřadnice“ začátků a konců čar při příčném řezu čárou s tím, že nula odpovídá levému okraji dvojité (vícenásobné) čáry a jednička okraji pravému.

Například pero vytvořené následujícími příkazy je deset pixelů silné a levá čára tvoří 20 % šířky (0,2 − 0,0 = 0,2) a stejně tak pravá čára (1,0 − 0,8 = 0,2).

```
Pen pero = new Pen(Color.CornflowerBlue, 10);
pero.CompoundArray = new float[] { 0.0f, 0.2f, 0.8f, 1.0f };
```



Aby obrazce kreslené dvojitým perem měly pěkné rohy, je třeba je kreslit najednou (levý trojúhelník na obrázku) s pomocí metod pracujících s poli `Point[]`, jako jsou `DrawLines`, `DrawPolygon`, `DrawCurve` ap., a nikoli po čarách (pravý trojúhelník na obrázku) vícenásobnými voláními `DrawLine`. Vyzkoušejte!

Poznámka: Písmeno `f` za desetinným číslem, např. `0.2f`, znamená, že jde o číslo typu `float`. Pokud by nebylo uvedeno, považoval by program uvedená desetinná čísla automaticky za hodnoty `double`, ty by ale do pole `float[]` nešly vložit.

ÚLOHA 3-R-01 (STRINGBUILDER)

Již mnohokrát jsme byli v situaci, kdy se opakovaně pracovalo s jedním a tímtéž řetězcem. Typickým případem je sestavení nějaké víceřádkové zprávy, např. výpis všech nájemníků v programu *Evidence nájemníků* z učebnice. K řetězci zprávy se postupně v každé obrátce cyklu přidává jeden řádek. V případě velkého množství manipulací s jednou řetězcovou hodnotou se ale může začít projevovat časová neefektivita těchto manipulací. Při každé aktualizaci řetězcové hodnoty se totiž alokuje nové místo v paměti, do něj se překopíruje stávající obsah, přidá se nový řádek a odkaz v proměnné se „překlopí“ na toto nové místo.

Za účelem redukování této neefektivity existuje na platformě .NET pro často měněné řetězce třída `StringBuilder` ze jmenného prostoru `System.Text`. K jejím nejzajímavějším složkám patří:

- Konstruktor, který v bezparametrické variantě vytvoří prázdnou instanci `StringBuilder` a ve variantě jednoparametrické instanci inicializovanou předaným řetězcem;
- Metody `Append`, resp. `AppendLine`, které na konec stávajícího řetězce `StringBuilder`u přidají řetězec předaný jako parametr;
- Metoda `ToString`, která řetězec `StringBuilder`u převede na `string`;
- Indexování – stejně jako u hodnoty `string` lze u instancí `StringBuilder` pracovat s jednotlivými znaky, jako by se jednalo o pole znaků. V případě `StringBuilder`u lze navíc znaky měnit.

Přepracujte výpis nájemníků s použitím třídy `StringBuilder`. Na začátku zdrojového textu nezapomeňte přidat odkaz na jmenný prostor `System.Text`.

ÚLOHA 3-R-02 (PARAMETRY PROGRAMU)

Při spuštění jakéhokoli programu může uživatel do programu předávat upřesňující informace ve formě ***parametrů programu***. Ty se s mezerami zadávají za název programu, ať už jde o spouštění z textového rozhraní (*Příkazový řádek*, příkazové dávky, skripty), nebo z rozhraní grafického (ikona zástupce). Například *Word* může převzít jako první parametr název dokumentu, který se má ihned po spuštění programu otevřít.

Programy na platformě .NET mají k parametrům programu přístup prostřednictvím pole řetězců vraceného statickou metodou `Environment.GetCommandLineArgs`. Na indexu 0 tohoto pole je jméno .EXE souboru i s cestou, na dalších indexech již následují všechny předané parametry.

Program *Pohybující se panáček* z druhé kapitoly učebnice upravte tak, aby přebíral potenciálně dva parametry upřesňující jeho činnost:

- Prvním parametrem bude cesta k obrázku panáčka, který se má použít;
- Druhým parametrem bude velikost kroku, neboli o kolik pixelů panáček poskočí na jeden stisk klávesy se šipkou.

Pokud budou parametry chybět, příp. pokud budou mít nekorektní hodnoty, použije se vestavěný obrázek, resp. základní délka kroku 5 pixelů.

Ještě poznamenám, že pro účely testování programu spouštěním z vývojového prostředí lze parametry zadat v poli „Command line arguments“ záložky „Debug“ v obrazovce, která se objeví po volbě nabídky *Project > (název projektu) Properties*.