Kapitola 4

Rozhodovací příkazy a jejich použití

Po přečtení této kapitoly budete schopni:

- Deklarovat proměnné typu `bool`
- Používat logické operátory pro vytváření výrazů, jejichž výsledkem je hodnota `true` nebo `false`
- Psát příkazy `if`, které rozhodnou o dalším průběhu provádění kódu na základě výsledku logického výrazu
- Psát příkazy `switch` pro komplikovanější rozhodovací operace

V kapitole 3 jste se naučili seskupovat k sobě patřící příkazy do metod. Dále jste poznali, jak pomocí parametrů předávat metodám informace a jak používat příkaz `return` pro předávání informací ven z metody. Obecně bývá nezbytné rozdělit program na několik samostatných metod, z nichž každá provádí určitou operaci. V mnoha programech je nutné řešit rozsáhlé a složité problémy. Rozdělením celého programu do metod se vylepší logická struktura a návaznost všech problémů a vývojář se může soustředit na jejich postupné řešení. V praxi je však často nutné psát metody, které provádí různé akce na základě aktuálních okolností. V této kapitole uvidíte, jak se takový úkol provádí.

Deklarace logických proměnných

Ve světě programování je (na rozdíl od skutečného světa) vše buď černé, nebo bílé, správné, či špatné, pravdivé, či nepravdivé. Jestliže například vytvoříte celočíselnou proměnnou s názvem `x`, přiřadíte do ní číslo 99 a poté se zeptáte: „Obsahuje proměnná x hodnotu 99?", bude odpověď nepochybně znít ANO. Na otázku „Je x menší než 10?" bude odpověď znít NE. Toto byly ukázky *logických výrazů*. Logický výraz je vždy vyhodnocen jako `true`, či `false` (ano/ne; výraz platí, nebo neplatí).

Poznámka: Ve všech programovacích jazycích nemusí být odpovědi na výše uvedené otázky tak jednoznačné. Neinicializovaná proměnná nemá přiřazenu žádnou hodnotu, a nelze proto říci, zda je, či není menší než 10. Takovéto mezní situace jsou častým zdrojem chyb v programech napsaných v jazycích C a C++. Kompilátor jazyka Microsoft Visual C# řeší tento problém tak, že proměnné musí programátor vždy přiřadit hodnotu, než může použít její obsah. Pokud se pokusíte získat obsah neinicializované proměnné, nebudete moci program zkompilovat.

Jazyk Microsoft Visual C# nabízí programátorům také datový typ `bool`. Proměnná tohoto typu může obsahovat jednu ze dvou hodnot: `true` nebo `false`. Následující tři příkazy deklarují

proměnnou typu `bool` s názvem `jsiPřipraven`, přiřadí jí hodnotu `true` a vypíše její obsah na příkazový řádek:

```
bool jsiPřipraven;
jsiPřipraven = true;
Console.WriteLine(jsiPřipraven); // vypíše True
```

Používání logických operátorů

Logický operátor je takový operátor, který provádí výpočet, jehož výsledkem je hodnota buď `true` nebo `false`. Jazyk C# nabízí několik velmi užitečných logických operátorů: nejjednodušší je operátor logické negace, vyjádřený znakem vykřičník („!“). Operátor `!` neguje logickou hodnotu, což znamená, že vrátí hodnotu opačnou. Pokud proměnná `jsiPřipraven` obsahuje hodnotu `true`, je výsledkem výrazu `!jsiPřipraven` hodnota `false`.

Poznámka odborného korektora: Autor v této podkapitole směšuje dvě skupiny operátorů. Jako logické operátory se zpravidla označují pouze operátory `!`, `&&` a `||`. Tyto operátory pracují s logickými hodnotami (jejich operandy jsou typu `bool`).

Operátory `<`, `<=` a další, které porovnávají hodnoty operandů jiných typů, než je `bool`, se označují jako relační a považují se za samostatnou skupinu. Společným rysem obou těchto skupin ale je, že jejich výsledky jsou typu `bool`.

Operátor rovnosti a relační operátory

K nejčastěji používaným logickým operátorům patří operátor rovnosti („==“) a nerovnosti („!=“). Pomocí těchto binárních operátorů lze zjistit, zda se jedna hodnota rovná druhé hodnotě téhož typu. Následující tabulka nabízí přehled těchto operátorů a jako příklad uvádí proměnnou `věk` typu `int`.

Operátor	Význam	Ukázka	Výsledek v případě, kdy proměnná věk je 42
`==`	rovná se	`věk == 100`	`False`
`!=`	nerovná se	`věk != 0`	`True`

S těmito operátory úzce souvisí relační operátory. S jejich pomocí se dá zjistit, zda je nějaká hodnota větší, nebo menší než jiná hodnota stejného typu. Způsob použití relačních operátorů opět shrnuje níže uvedená tabulka:

Operátor	Význam	Ukázka	Výsledek v případě, kdy proměnná věk je 42
`<`	menší než	`věk < 21`	`False`
`<=`	menší nebo rovno	`věk <= 18`	`False`
`>`	větší než	`věk > 16`	`True`
`>=`	větší nebo rovno	`věk >= 30`	`True`

Poznámka: Nezaměňujte operátor rovnosti `==` s operátorem přiřazení `=`. Výraz `x == y` porovná hodnoty proměnných `x` a `y` a jeho hodnotou je `true`, pokud obě uchovávají tutéž hodnotu. Ovšem výraz `x = y` uloží hodnotu proměnné `y` do proměnné `x`.

Podmíněné logické operátory

Jazyk C# poskytuje ještě dva další logické operátory: operátor logického součinu (a zároveň), vyjádřený symbolem `&&`, a operátor logického součtu (nebo), vyjádřený symbolem `||`. Oba patří mezi *podmíněné logické operátory*. Jejich účelem je zkombinovat dva logické výrazy či hodnoty do jediného logického výsledku. Oba binární operátory se podobají operátorům rovnosti a relačním operátorům v tom, že hodnota výrazů, v nichž se objevují, je buď `true`, nebo `false`, liší se však v tom, že jejich operandy musí také samy o sobě nabývat hodnoty `true` nebo `false`.

Výsledkem operátoru `&&` je `true`, právě tehdy, když mají oba logické výrazy (operandy), které vyhodnocuje, hodnotu `true`. V následující ukázce bude do proměnné `platnéProcento` uložena hodnota `true`, právě tehdy, když má proměnná `procento` hodnotu větší nebo rovnou 0 a zároveň menší nebo rovnou 100.

```
bool platnéProcento;
platnéProcento = (procento >= 0) && (procento <= 100);
```

Tip: Běžnou chybou začátečníků je uvedení názvu proměnné v takovém výrazu jen jednou, asi takto:

```
procento >= 0 && <= 100 // Tento příkaz nebude zkompilován
```

Budete-li používat závorky, snížíte pravděpodobnost těchto chyb a zároveň tak celý zápis zpřehledníte. Zkuste například porovnat následující výrazy:

```
platnéProcento = procento >= 0 && procento <= 100
platnéProcento = (procento >= 0) && (procento <= 100)
```

Oba výrazy vrací stejnou hodnotu, protože priorita operátoru `&&` je nižší než priorita relačních operátorů `>=` a `<=`. Druhý výraz se však rozhodně lépe čte.

Výsledkem operátoru `||` je `true`, je-li výsledek alespoň jednoho z jeho dvou logických operandů roven hodnotě `true`. Pomocí operátoru `||` je tedy možné zjistit, zda platí (= `true`) alespoň jeden z předkládaných logických výrazů. V následující ukázce bude proměnné `neplatnéProcento` přiřazena hodnota `true` v případě, že hodnota proměnné `procento` je menší než 0 nebo větší než 100.

```
bool neplatnéProcento;
neplatnéProcento = (procento < 0) || (procento > 100);
```

Zkrácené vyhodnocování

Operátory `&&` a `||` jsou zajímavé ještě jednou svou vlastností, která je označována jako *zkrácené vyhodnocování* (short circuiting). Někdy není nutné vyhodnocovat oba operandy. Pokud má například levý operand operátoru `&&` hodnotu `false`, je výsledek celého výrazu `false` bez ohledu na hodnotu pravého operandu. Obdobně platí, že má-li levý operandu operátoru `||` hodnotu `true`, je výsledek celého výrazu `true` nezávisle na hodnotě pravého operand. V těchto situacích již operátory `&&` a `||` pravý operand nevyhodnocují. Podívejme se na několik ukázek:

```
(procento >= 0) && (procento <= 100)
```

Je-li hodnota proměnné `procento` v tomto výrazu menší než 0, bude logický výraz na levé straně operátoru `&&` vyhodnocen jako `false`. To znamená, že výsledek celého výrazu musí být `false` a logický výraz na pravé straně operátoru `||` se tak nevyhodnotí

```
(procento < 0) || (procento > 100)
```

Pokud je v tomto výrazu hodnota proměnné `procento` menší než 0, je logický výraz na levé straně operátoru `||` vyhodnocen jako `true`. To znamená, že výsledek celého výrazu musí být `true`, a výraz na pravé straně proto již není vyhodnocován.

Budete-li pečlivě navrhovat výrazy, v nichž se používají podmíněné logické operátory, můžete urychlit provádění kódu a vyhnout se nepotřebným operacím. Jednoduché, rychle vyhodnotitelné logické výrazy umisťujte na levou stranu podmíněných logických operátorů, složitější výrazy nechte na jejich pravé straně. V mnoha případech tak program nebude muset vyhodnocovat složité výrazy.

Shrnutí priority a asociativity operátorů

V následující tabulce je uveden přehled priority a asociativity všech doposud probraných operátorů. Operátory patřící do stejné kategorie mají stejnou prioritu. Operátory na vyšším místě tabulky mají přednost před operátory na místech nižších.

Kategorie	Operátory	Popis	Asociativita
Primární	`( )`	Potlačení priority	zleva doprava
	`++`	Postfixový inkrement	
	`--`	Postfixový dekrement	
Unární	`!`	Logická negace (NOT)	zleva doprava
	`+`	Unární plus (identita)	
	`-`	Unární minus (obrácení znaménka)	
	`++`	Prefixový inkrement	
	`--`	Prefixový dekrement	
Multiplikativní	`*`	Násobení	zleva doprava
	`/`	Dělení	
	`%`	Zbytek po dělení (modulo)	
Aditivní	`+`	Sčítání	zleva doprava
	`-`	Odčítání	
Relační	`<`	Menší než	zleva doprava
	`<=`	Menší nebo rovno	
	`>`	Větší než	
	`>=`	Větší nebo rovno	
Rovnost	`==`	Rovná se	zleva doprava
	`!=`	Nerovná se	
Logické „a zároveň“	`&&`	Logický součin (konjunkce)	zleva doprava
Logické „nebo“	`||`	Logický součet (disjunkce)	zleva doprava
Přiřazení	`=`		zprava doleva

Rozhodování pomocí příkazů if

Příkaz `if` můžete použít v případě, kdy potřebujete provést jeden ze dvou různých bloků kódu, přičemž výběr bloku je dán nějakou logickou podmínkou.

Syntaxe příkazu if

Syntaxe příkazu `if` vypadá takto (`if` a `else` jsou klíčová slova jazyka C#):

```
if (logickýVýraz)
   příkaz1;
else
   příkaz2;
```

Je-li `logickýVýraz` vyhodnocen jako `true`, bude spuštěn `příkaz1`. V opačném případě (kdy je `logickýVýraz` vyhodnocen jako `false`) bude spuštěn `příkaz2`. Klíčové slovo `else` a za ním následující `příkaz2` jsou nepovinné. Není-li klauzule `else` uvedena, a má-li `logickýVýraz` hodnotu `false`, pak provádění programu pokračuje jakýmikoli příkazy, jež následují po příkazu `if`.

Podívejme se na ukázku příkazu `if`, který proměnnou představující sekundovou ručičku na stopkách inkrementuje vždy o jednu sekundu. Jestliže má proměnná `sekundy` hodnotu 59, je vynulována, jinak se její hodnota zvýší pomocí operátoru ++:

```
int sekundy;
...
if (sekundy == 59)
   sekundy = 0;
else
   sekundy++;
```

Jen logické výrazy, prosím!

Výraz za příkazem `if` musí být uzavřen do závorek. Dále platí, že se musí jednat o logický výraz. V některých jiných programovacích jazycích (konkrétně v jazycích C a C++) lze na tomto místě zapsat celočíselný výraz, který si kompilátor potichu převede na `true` (v případě nenulového čísla) nebo `false` (v případě nuly). Jazyk C# tyto konverze nepodporuje a kompilátor ohlásí chybu, pokud takový výraz napíšete.

Také v případě, že omylem zapíšete v příkazu `if` namísto operátoru rovnosti („==“) operátor přiřazení („=“), kompilátor tuto chybu pozná a ohlásí ji:

```
int sekundy;
...
if (sekundy = 59) // chyba při kompilaci
...
if (sekundy == 59) // v pořádku
```

Tato chybná přiřazení byla dalším zdrojem chyb v programech napsaných v jazycích C a C++, protože v nich dochází k implicitnímu převodu přiřazené hodnoty (59) na logický výraz (jakékoli nenulové číslo bylo považováno za `true`), takže se vždy provedl kód následující za příkazem `if`.

V příkazu `if` můžete jako výraz použít také libovolnou logickou proměnnou, nesmíte ji však zapomenout uzavřít do závorek:

```
bool veSlově;
...
if (veSlově == true) // v pořádku, ale tento zápis se nepoužívá
...
if (veSlově) // lepší zápis
```

Seskupování příkazů do bloků

Všimněte si, že výše uvedená syntaxe příkazu `if` stanoví, že za příkazem `if` a klíčovým slovem `else` je pouze jediný příkaz. Někdy je třeba v případě platnosti určitého logického výrazu spustit několik příkazů. To se dá řešit tak, že tyto příkazy seskupíte do nové metody a za příkazem `if` tuto novou metodu zavoláte. Jednodušší je však uzavřít všechny spouštěné příkazy do jednoho bloku. *Blok* je posloupnost příkazů uzavřená mezi otevírací a uzavírací složenou závorku. Blok také zahajuje nový obor platnosti. Uvnitř bloku můžete definovat proměnné, které však po jeho opuštění zmizí. V následujícím příkladu jsou do bloku uzavřeny dva příkazy: první vynuluje hodnotu proměnné `sekundy`, druhý inkrementuje hodnotu proměnné `minuty`. Celý blok bude spuštěn, obsahuje-li proměnná `sekundy` hodnotu 59:

```
int sekundy = 0;
int minuty = 0;
...
if (sekundy == 59)
{
   sekundy = 0;
   minuty++;
}
else
   sekundy++;
```

Důležité: Kdybyste v předchozím kódu vynechali složené závorky, kompilátor by k příkazu `if` přidružil jen první příkaz (`sekundy = 0`). Následující příkaz (`minuty++`) by již kompilátor během kompilace programu nebral jako součást příkazu `if`. Jakmile by pak kompilátor narazil na klíčové slovo `else`, neuměl by si ho sdružit s předchozím příkazem `if` a ohlásil by syntaktickou chybu.

Kaskádové příkazy if

Příkazy `if` se dají vnořovat do jiných příkazů `if`. Takto je možné zřetězit posloupnost logických výrazů, které budou postupně vyhodnocovány, dokud jeden z nich nebude mít hodnotu `true`. Jestliže má v následujícím příkladu proměnná `den` hodnotu 0, bude první testovací podmínka vyhodnocena jako `true` a proměnné `názevDne` bude přiřazen řetězec `"Neděle"`. Není-li hodnotou proměnné den 0, skončí první test neúspěchem a řízení přejde na klauzuli `else`, která spustí druhý příkaz `if`, v němž je porovnávána hodnota proměnné den a číslo 1. Druhý příkaz `if` se provede jen v případě, že první test skončí výsledkem `false`. Obdobně třetí příkaz `if` se provede jen v případě, že první i druhý test skončí výsledkem `false`.

```
if (den == 0)
   názevDne = "Neděle";
else if (den == 1)
   názevDne = "Pondělí";
else if (den == 2)
   názevDne = "Úterý";
else if (den == 3)
   názevDne = "Středa";
else if (den == 4)
   názevDne = "Čtvrtek";
else if (den == 5)
```

```
    názevDne = "Pátek";
else if (den == 6)
    názevDne = "Sobota";
else
    názevDne = "neznámý";
```

V následujícím cvičení napíšete metodu, která pomocí kaskádových příkazů if porovná dvě kalendářní data.

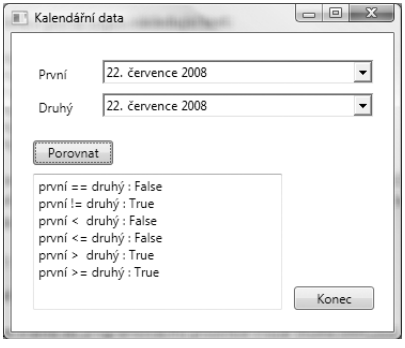
Psaní příkazů if

1. Spusťte Microsoft Visual Studio 2008, pokud již neběží.
2. Otevřete projekt Výběr, umístěný ve složce *Dokumenty\Visual CSharp 2008 Krok za Krokem\Kapitola 4\Výběr*.
3. V nabídce *Debug* klepněte na příkaz *Start Without Debugging*.

   Visual Studio 2008 sestaví a spustí aplikaci. Na formuláři jsou dva ovládací prvky typu `DateTimePicker` s názvy `první` a `druhý`. (Tyto ovládací prvky zobrazí kalendář umožňující po klepnutí na rozvírací šipku výběr konkrétního data.) Oba ovládací prvky jsou na začátku nastaveny na aktuální datum.
4. Klepněte na tlačítko *Porovnat*.

   V textovém poli se objeví následující text:

   ```
   první == druhý : False
   první != druhý : True
   první < druhý : False
   první <= druhý : False
   první > druhý : True
   první >= druhý : True
   ```

   Výraz `první == druhý` by se ovšem měl vyhodnotit na `true`, protože oba ovládací prvky jsou nastaveny na stejné datum. Vypadá to tak, že správně funguje jen operátor menší než, větší než a operátor rovnosti!
5. Klepnutím na tlačítko *Konec* se vraťte do programovacího prostředí Visual Studia 2008.
6. Otevřete v okně editoru kód souboru *Window1.xaml.cs*.
7. Vyhledejte metodu `compareClick`, která vypadá takto:

   ```
   private void compareClick(object sender, System.EventArgs e)
   {
       int rozdíl = porovnejData(první.Value, druhý.Value);
       info.Text = "";
       ukaž("první == druhý", rozdíl == 0);
       ukaž("první != druhý", rozdíl != 0);
       ukaž("první < druhý", rozdíl < 0);
       ukaž("první <= druhý", rozdíl <= 0);
       ukaž("první > druhý", rozdíl > 0);
       ukaž("první >= druhý", rozdíl >= 0);
   }
   ```

   Tato metoda je spuštěna při každém klepnutí na tlačítko *Porovnat*, umístěné ve formuláři. Kód metody načte hodnoty dat zobrazené v ovládacích prvcích typu `DateTimePicker` `první`

a `druhý` ve formuláři a zavolá metodu `porovnejData`, která porovná jejich hodnoty. Kód metody `porovnejData` budeme zkoumat v následujícím kroku.

Metoda `ukaž` shrne výsledky porovnání do textového pole `info` ve formuláři.

8. Vyhledejte metodu `porovnejData`, která vypadá takto:

```
private int porovnejData(DateTime levý, DateTime pravý)
{
   // DODĚLAT
   return 42;
}
```

Tato metoda v současné chvíli vrací při každém svém volání stejnou hodnotu 42, nikoli 0, -1 nebo +1 (podle výsledku porovnání parametrů). To vysvětluje, proč aplikace nefunguje podle očekávání!

Účelem této metody je porovnat hodnoty argumentů a podle výsledku tohoto srovnání vrátit celé číslo: 0 (pokud jsou obě hodnoty stejné), -1 (pokud je hodnota prvního argumentu menší než hodnota druhého), nebo +1 (pokud je hodnota prvního argumentu větší než hodnota druhého). Datum je bráno jako větší než druhé, pokud následuje chronologicky po něm. Nyní bude třeba tuto logiku implementovat, aby metoda porovnávala data správným způsobem.

9. Vymažte v těle metody `porovnejData` komentář `// DODĚLAT` a příkaz `return`.

10. Zapište do těla metody `porovnejData` následující příkazy:

```
int výsledek;

if (levý.Year < pravý.Year)
   výsledek = -1;
else if (levý.Year > pravý.Year)
   výsledek = +1;
výsledekvýsledekvýsledekvýsledekvýsledekvýsledek
```

Je-li výraz `levý.Year < pravý.Year` vyhodnocen jako `true`, pak datum na levé straně musí být menší než datum na pravé, takže program nastaví proměnnou `výsledek` na hodnotu -1. V opačném případě, pokud má výraz `levý.Year > pravý.Year` hodnotu true, pak datum na levé straně musí být větší než datum na pravé straně a program nastaví proměnnou výsledek na hodnotu +1.

Jsou-li výrazy `levý.Year < pravý.Year` a `levý.Year > pravý.Year` vyhodnoceny jako `false`, musí být rok obou výrazů stejný, takže je nutné porovnat jejich měsíce.

11. Nyní přidejte níže uvedené příkazy do metody `porovnejData` pod kód, který jste vložili v předchozím kroku:

```
else if (levý.Month < pravý.Month)
   výsledek = -1;
else if (levý.Month > pravý.Month)
   výsledek = +1;
```

Tyto příkazy fungují při porovnávání měsíců podobně jako porovnání roků v předchozím kroku.

Je-li výraz `levý.Month < pravý.Month` vyhodnocen na `false`, a výraz `levý.Month > pravý.Month` je také `false`, pak musí být vlastnost `Month` obou dat stejná, takže program musí nakonec porovnat jejich dny.

12. Přidejte následující příkazy do metody `porovnejData` pod kód, který jste vložili v předchozím kroku:

```
else if (levý.Day < pravý.Day)
   výsledek = -1;
else if (levý.Day > pravý.Day)
   výsledek = +1;
else
   výsledek = 0;
return výsledek;
```

Nyní byste již měli mít ohledně logiky porovnání kalendářních dat úplně jasno.

Jsou-li tedy oba výrazy `levý.Day < pravý.Day` a `levý.Day > pravý.Day` vyhodnoceny na `false`, pak je hodnota vlastnosti `Day` obou proměnných stejná. Hodnoty vlastností `Month` a `Year` musejí být v tomto místě programu již také identické, takže daná dvě kalendářní data musejí být totožná a program nastaví hodnotu proměnné `výsledek` na 0.

Poslední příkaz vrátí hodnotu uloženou v proměnné `výsledek`.

13. V nabídce *Debug* klepněte na příkaz *Start Without Debugging*.

Aplikace bude znovu sestavena a spuštěna. I nyní budou oba ovládací prvky typu `DateTimePicker` implicitně nastaveny na aktuální datum.

14. Klepněte na tlačítko *Porovnat*.

V textovém poli formuláře se objeví následující text:

```
první == druhý : True
první != druhý : False
první < druhý : False
první <= druhý : True
první > druhý : False
první >= druhý : True
```

Tyto výsledky jsou již pro dvě shodná kalendářní data naprosto správné.

15. Ve druhém ovládacím prvku typu `DateTimePicker` vyberte zítřejší datum.

16. Klepněte na tlačítko *Porovnat*.

V textovém poli formuláře se objeví následující text:

```
první == druhý : False
první != druhý : True
první < druhý : True
první <= druhý : True
první > druhý : False
první >= druhý : False
```

I tyto výsledky jsou v pořádku.

17. Otestujte ještě další data a zkontrolujte, že výsledky jsou vždy takové, jak byste očekávali. Nakonec klepněte na tlačítko *Konec*.

Porovnávání kalendářních dat ve skutečné aplikaci

Nyní, když jste viděli použití poměrně dlouhé a komplikované série příkazů `if` a `else`, by bylo vhodné uvést, že se nejedná o postup, který byste použili pro porovnání kalendářních dat ve skutečné aplikaci. V knihovně tříd rozhraní .NET Framework jsou data uchovávána pomocí speciálního typu nazvaného `DateTime`. Když se podíváte na metodu `porovnejData`, kterou

jste napsali v předchozím cvičení, uvidíte, že její logika porovnává pouze datové části proměnných, ovšem k dispozici je také časový prvek. Dvě hodnoty jsou považovány za sobě rovné, pokud se shodují nejen datem, ale také časem. Porovnání data a času bývá tak běžnou operací, že pro tento účel obsahuje typ `DateTime` vestavěnou metodu nazvanou `Compare`. Metoda `Compare` přijímá dva argumenty typu `DateTime`, porovná je a vrací hodnotu signalizující, zda je první argument menší než druhý (výsledek bude záporný), zda je první argument větší než druhý (výsledek bude kladný), nebo zda oba argumenty představují totéž datum a čas (výsledek bude 0).

Jak používat příkazy switch

V některých kaskádových příkazech `if` jsou si všechny příkazy podobné, protože všechny vyhodnocují stejný výraz. Jediným rozdílem je, že každý z příkazů `if` porovnává výsledek výrazu s jinou hodnotou. Uvažte například následující blok kódu, který pomocí příkazu `if` odvozuje od hodnoty v proměnné `den` název dne v týdnu:

```
if (den == 0)
   názevDne = "Neděle";
else if (den == 1)
   názevDne = "Pondělí";
else if (den == 2)
   názevDne = "Úterý";
else if (den == 3)
   ...
else
   názevDne = "neznámý";
```

V těchto situacích je často možné vnořené příkazy `if` nahradit příkazem `switch`, který zlepší čitelnost kódu a zvýší jeho efektivitu.

Syntaxe příkazu switch

Syntaxe příkazu `switch` vypadá takto (`switch`, `case` a `default` jsou klíčová slova):

```
switch (řídicíVýraz)
{
case konstantníVýraz:
   příkazy
   break;
case konstantníVýraz:
   příkazy
   break;
...
default:
   příkazy
   break;
}
```

Výraz `řídicíVýraz` bude vyhodnocen pouze jednou a pak budou spuštěny příkazy za klíčovým slovem `case`, jehož `konstantníVýraz` má hodnotu rovnou výsledku výrazu `řídicíVýraz`, a to až po příkaz `break`. Příkaz `switch` poté skončí a program pokračuje prvním řádkem za jeho uzavírací koncovou závorkou příkazu.Pokud hodnota žádného z výrazů `konstantníVýraz` není rovná hodnotě výrazu `řídicíVýraz`, budou spuštěny příkazy za nepovinným návěštím `default` (pokud ho příkaz `switch` obsahuje).

Poznámka: Hodnota každého výrazu `konstantníVýraz` musí být jedinečná, aby tak `řídicíVýraz` mohl odpovídat nejvýše jedné z nich. Jestliže se hodnota výrazu `řídicíVýraz` nerovná žádné z hodnot výrazů `konstantníVýraz` a v příkazu `switch` chybí návěští `default`, pokračuje program prvním řádkem za uzavírací koncovou závorkou příkazu `switch`.

Předchozí kaskádový příkaz `if` lze tedy přepsat pomocí následujícího příkazu `switch`:

```
switch (den)
{
case 0 :
   názevDne = "Neděle";
   break;
case 1 :
   názevDne = "Pondělí";
   break;
case 2 :
   názevDne = "Úterý";
   break;
...
default :
   názevDne = "neznámý";
   break;
}
```

Pravidla pro používání příkazu switch

Příkaz `switch` je opravdu užitečný, naneštěstí ho však někdy nelze použít, ač by se vám hodil. Každý příkaz `switch` musí splňovat následující pravidla:

- V řídicím výrazu v příkazu `switch` lze použít jen primitivní datové typy, například `int` nebo `string`. U jiných typů (včetně typů `float` a `double`) je nutné použít příkaz `if`.
- Návěští `case` musí obsahovat konstantní výrazy, například 42 nebo „42". Hodnoty pro návěští `case` nelze počítat dynamicky za běhu programu; v takovém případě musíte použít příkaz `if`.
- Hodnoty návěští `case` musí být jedinečné. Dvě návěští v jednom příkazu `switch` nesmí obsahovat stejné hodnoty.
- V případě potřeby lze spustit jednu skupinu příkazů pro několik konstantních výrazů (několik návěští), a to tak, že uvedete několik návěští pod sebou. (Mezi návěštími v tom případě nesmějí být žádné příkazy.) Kód uvedený za posledním z těchto návěští bude spuštěn, bude-li mít `řídicíVýraz` hodnotu libovolného návěští z celého uvedeného seznamu. Následuje-li však za návěštím alespoň jeden příkaz, a není-li tato skupina ukončena příkazem `break` (nebo některým jiným z příkazů pro přenos řízení), ohlásí kompilátor chybu:

```
switch (trumf)
{
case Srdce:
case Káry: // pokračování na další návěští dovoleno
   barva = "Červená"; // kód pro návěští Srdce a Káry
   break;
case Kříže:
```

```
    barva = "Černá";
case Piky: // chyba - neukončený kód mezi návěštími
    barva = "Černá";
    break;
}
```

Poznámka: Příkaz `break` je nejčastěji používaným prostředkem pro předčasné ukončení průchodu příkazem `switch`, dá se však použít i příkaz `return`, `throw` nebo jiný z příkazů, které způsobují přenos řízení. Příkaz `throw` si vysvětlíme v kapitole 6.

Předčasné ukončení příkazu switch

Díky pravidlu o ukončení příkazu `switch` ve větvi obsahující prováděné příkazy je možné jednotlivé sekce tohoto příkazu zapisovat v libovolném pořadí, aniž by to mělo vliv na funkčnost. (To se týká i návěští `default`, které je podle zažité konvence dáváno na poslední místo, ale může být uvedeno i jinde.)

Programátoři v jazycích C a C++ si musí zapamatovat, že příkaz musí být uveden v každé větvi příkazu `switch` (dokonce i ve větvi `default`). Tento požadavek je ku prospěchu věci, neboť v programech napsaných v jazyku C nebo C++ se často na příkaz `break` zapomíná, takže kód pak může pokračovat dalším návěštím, a dochází tak k těžko odhalitelným chybám.

Je-li to opravdu nezbytné, pak se dá „propadávání" kódu do následující větve nebo do větve `default` v jazyku C# napodobit pomocí příkazu `goto`. Tento způsob programování však nelze obecně doporučit, a proto se v této knize ani nedozvíte, jak se tyto konstrukce píší.

V následujícím cvičení dokončíte program, který čte znaky řetězce a každý znak převede na jeho vyjádření v jazyku XML. Například znak `'<'` má v jazyku XML zvláštní význam (tvoří počátek elementu), a musí být proto převeden na sekvenci `'<'`, aby rutina zpracovávající kód XML věděla, že se jedná o data a ne část instrukce XML. Podobná pravidla platí pro pravou lomenou závorku (`>`), ampersand (`&`), apostrof (`'`) a uvozovky (`"`). Ve cvičení napíšete příkaz `switch`, který otestuje hodnotu znaku a znaky s takovýmto speciálním významem převede v jednotlivých větvích tohoto příkazu na odpovídající entity.

Psaní příkazů switch

1. Spusťte Visual Studio 2008, pokud již neběží.
2. Otevřete projekt Příkaz Switch, umístěný ve složce *Dokumenty\Visual CSharp 2008 Krok za Krokem\Kapitola 4\Příkaz Switch*.
3. V nabídce *Debug* klepněte na příkaz *Start Without Debugging*.

   Visual Studio 2008 sestaví a spustí aplikaci. V jejím formuláři jsou dvě textová pole oddělená tlačítkem *Kopírovat*.

4. Do horního textového pole zapište následující ukázkový text:

   ```
   vOblasti = (spodni <= cislo) && (cislo <= horni);
   ```

5. Klepněte na tlačítko *Kopírovat*.

Příkaz bude doslovně okopírován do spodního textového pole, znaky `<` , `&` a `>` zůstanou beze změny.

6. Zavřete formulář a vraťte se do prostředí Visual Studia 2008.
7. Zobrazte kód souboru *Window1.xaml.cs* v okně editoru a vyhledejte metodu `zkopírujJeden`.

   Metoda `zkopírujJeden` zkopíruje jeden znak z horního textového pole do pole spodního. V současné chvíli obsahuje příkaz `switch` s jedinou větví (`default`). V několika následujících krocích upravíte tento příkaz `switch` tak, aby převáděl znaky, které mají v jazyku XML speciální význam, na odpovídající entity jazyka XML. Například znak `'<'` bude převeden na řetězec „`<` „.
8. Přidejte nad návěští `default` v příkazu `switch` následující příkazy:

```
case '<':
   cíl.Text += "&lt;";
   break;
```

   Bude-li aktuálně kopírovaným znakem znak `'<'`, výše uvedený kód jej v cílovém textu nahradí řetězcem `"<"`.
9. Přidejte následující příkazy do příkazu `switch` pod příkaz `break`, který jste právě přidali nad návěští `default`:

```
case '>':
   cíl.Text += "&gt;";
   break;
case '&':
   cíl.Text += "&amp;";
   break;
case '\"':
   cíl.Text += "&#34;";
   break;
case '\'':
   cíl.Text += "&#39;";
   break;
```

Poznámka: Apostrof a uvozovky mají v jazyku C# I XML speciální význam (slouží k ohraničení znakových a řetězcových konstant). Obrácené lomítko (`\`) v posledních dvou větvích je řídicí znak (mění význam následujících znaků), který zajišťuje, že následující znak uvozovky nebo apostrof (`"` nebo `'`) bude chápán doslovně, nikoli jako textový či znakový omezovač ohraničující řetězcový či znakový literál.

10. V nabídce *Debug* klepněte na příkaz *Start Without Debugging*.
11. Do horního textového pole zapište následující příkaz:

```
vOblasti = (spodni <= cislo) && (cislo <= horni);
```

12. Klepněte na tlačítko *Kopírovat*.

    Příkaz se zkopíruje do spodního textového pole, tentokrát však bude díky příkazu `switch` každý znak zkontrolován a v případě potřeby převeden na příslušnou entitu jazyka XML.

```
vOblasti = (spodni &lt;= cislo) &amp;&amp; (cislo &lt;= horni);
```

13. Experimentujte i s dalšími řetězci a zkontrolujte, že budou všechny zvláštní znaky (<, >, &, " a ') správně ošetřeny.
14. Zavřete formulář.

- **Pokud chcete pokračovat další kapitolou**, nechte Visual Studio 2008 běžet a nalistujte kapitolu 5.
- **Pokud chcete Visual Studio 2008 nyní ukončit**, v nabídce *File* klepněte na příkaz *Exit*. Pokud se objeví dialogové okno s dotazem na uložení změn, klepněte na tlačítko *Yes* (pokud používáte Visual Studio 2008) nebo *Save* (pokud používáte Visual C# Express Edition) a uložte projekt.

Stručné shrnutí čtvrté kapitoly

Pro	učiňte následující	Příklad
Zjištění, zda jsou dvě hodnoty shodné	Použijte operátor == nebo !=.	`odpověď == 42`
Porovnání hodnot dvou výrazů	Použijte operátor <, <=, > nebo >=.	`věk >= 21`
Deklarování logické proměnné	Jako typ proměnné uveďte klíčové slovo `bool`.	`bool vOblasti;`
Vytvoření logického výrazu, který platí (= `true`) jen v případě, kdy platí dvě různé podmínky	Použijte operátor &&.	`vOblasti = (cíl <= číslo) && (číslo <= horní);`
Vytvoření logického výrazu, který platí (= `true`), pokud je splněna alespoň jedna ze dvou různých podmínek	Použijte operátor ||.	`mimoOblast = (číslo < spodníMez) || (horníMez < číslo);`
Spuštění příkazu v případě splnění podmínky	Použijte příkaz `if`.	`if (vOblasti)` `  zpracuj();`

Spuštění více než jednoho příkazu v případě splnění podmínky	Použijte příkaz `if` a blok.	`if (sekundy == 59)`
		`{`
		`  sekundy = 0;`
		`  minuty++;`
		`}`

Spojení různých příkazů s různými výslednými hodnotami řídicího výrazu	Použijte příkaz `switch`.	`switch (aktuální)`
		`{`
		`case '<' :`
		`  ...`
		`  break;`
		`default :`
		`  ...`
		`  break;`
		`}`