Petr Roudenský, Mokhtar M. Khorshid

Programujeme hry v jazyce C#

9 plně funkčních projektů

Od úplných základů po pokročilé hry

Snadno srozumitelný výklad na příkladech

Ke stažení zdrojové kódy her z knihy

WWW

CPRESS

Petr Roudenský, Mokhtar M. Khorshid

Programujeme hry v jazyce C#

Computer Press, a. s.
Brno
2011

Programujeme hry v jazyce C#

Petr Roudenský, Mokhtar M. Khorshid

Computer Press, a. s., 2011. Vydání první.

Grafika her: Tomáš Kopecký
Odborná spolupráce a testy: Martin Kačmařík

Jazyková korektura: Alena Láníčková
Sazba: Zuzana Šindlerová
Rejstřík: Daniel Štreit
Obálka: Martin Sodomka
Komentář na zadní straně obálky: Martin Herodek

Technická spolupráce: Jiří Matoušek, Zuzana Šindlerová, Dagmar Hajdajová
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek
Produkce: Petr Baláš

Computer Press, a. s.,
Holandská 3, 639 00 Brno

Objednávky knih:
http://knihy.cpress.cz
distribuce@cpress.cz
tel.: 800 555 513

ISBN 978-80-251-3355-2
Prodejní kód: K1890
Vydalo nakladatelství Computer Press, a. s., jako svou 3983. publikaci.

Obsah

Úvodem

O programování v jazyce C# dnes naleznete na trhu řadu knih, ať jste úplný začátečník, pokročilý nebo jen přecházíte z jiného programovacího jazyka. Zcela pochopitelně jsou tak na místě otázky, čím se tato kniha odlišuje a co vám může nabídnout oproti ostatním. Na to by vám měly odpovědět následující odstavce.

Knihy zaměřené na programování v určitém programovacím jazyce používají obvykle velmi teoretický výklad, doplněný většinou abstraktními příklady. Tento způsob výkladu sice umožňuje seznámit čtenáře s daným programovacím jazykem v požadované míře, avšak často bohužel jen málo v rovině skutečného využití. Některým, především zkušenějším, čtenářům takový výklad vyhovuje, jiným může připadat nedostatečný a další odradí přílišnou abstrakcí ukázek, které možná nejlépe popisují daný problém, ale čtenáři prostě není dané řešení či použití zřejmé.

Tato kniha naopak přistupuje k programování v jazyce C# především tak, aby bylo pro čtenáře zábavné. A protože začínající programátoři rádi vidí výsledky svého snažení, je kniha uspořádána tak, že každá kapitola popisuje tvorbu nezávislé funkční aplikace – většinou hry – od prvotního návrhu přes, popis funkčnosti jednotlivých částí až po kompletní realizaci. Začnete prostými textovými aplikacemi a budete pokračovat přes vlastní editor obrázků až po jednoduché grafické hry v prostředí Windows Forms.

S každou kapitolou se tak budete více seznamovat s jazykem C#, průběžně (v kontextu vyvíjené aplikace) procvičovat získané vědomosti a v neposlední řadě se také naučíte základy řešení různých problémů, které s programováním obecně souvisí. V pozdějších kapitolách bude pozornost věnována také technologii Windows Forms a tvorbě grafického uživatelského rozhraní. Je zřejmé, že kvůli návaznosti výkladu je třeba číst kapitoly v knize tak, jak jsou uspořádány.

Nemusíte se bát, že látka probraná v jedné kapitole bude v kapitole následující brána jako samozřejmost – některé důležité pojmy či postupy jsou použity a příslušně vysvětleny u více příkladů různým způsobem.

Jak už obsah a forma této knihy napovídají, je určena především začátečníkům v jazyce C#, ale také těm, kteří dosud nikdy v žádném programovacím jazyce nepracovali. Z tohoto důvodu mohou pokročilejším programátorům přijít některá řešení použitá v jednotlivých aplikacích jako neefektivní či nepříliš vhodná, ovšem jinak by nebylo možné zachovat koncept knihy. Díky tomuto postupu se kód zlepšuje průběžně spolu s vašimi znalostmi, kapitolu za kapitolou.

Vzhledem k rozsahu a tématu tato kniha samozřejmě nepokrývá zdaleka všechny možnosti a vlastnosti jazyka C#, což ani nebylo jejím cílem. Aplikace v každé kapitole obsahuje vždy některé nové prvky, konstrukce či pojmy, avšak samozřejmě ne vždy bylo možné začlenit je smysluplně do jejího kódu, proto jsou vysvětleny na samostatných, jednoduchých příkladech.

Způsob výkladu je přizpůsoben účelu knihy: snažili jsme se minimalizovat používání technických termínů či odborných názvů a teoretických příkladů a místy jsme si dovolili dopustit se mírných zjednodušení.

Po prostudování této knihy budete mít dobrý přehled o jazyce C# a práci ve Visual Studio 2010 Express. Budete mít již také nezanedbatelné zkušenosti s praktickým využitím jazyka C#. To vše vám poslouží jako dobrý základ pro prohlubování nabytých znalostí za pomoci literatury pro pokročilejší a pokročilé programátory.

Základní znalosti o programovacích jazycích

Ještě před samotným představením jazyka C# je vhodné si v krátkosti něco říct o programovacích jazycích obecně, což vám objasní některé pojmy používané dále v této knize.

Počítačový program je posloupnost určitých příkazů, pomocí níž programátor sděluje počítači, co má provést s daty. Při zpracování je program spolu s potřebnými daty uložen v tzv. *operační paměti*, ke které může procesor rychle přistupovat a jejíž obsah je pouze dočasný.

Informace v operační paměti se udržují ve formě *bitů* – základních jednotek, které mohou nabývat pouze hodnoty 0 a 1. Běžně však budete pracovat s jednotkou, která se označuje jako *byte* (bajt) a která je tvořena uskupením osmi bitů. Jeden byte proto může nabývat hodnoty od 0 do 255, tedy celkem 256 hodnot (2^8).

Procesor jako hlavní výpočetní jednotka provádí s daty různé operace, ovšem jde pouze o čísla, neboť nic jiného nemůže být v paměti uloženo. Aby mohl procesor zpracovávat i příkazy, musí i ty být vyjádřeny čísly, přičemž tomuto způsobu vyjádření instrukcí se říká *strojový kód*.

Protože programovat ve strojovém kódu je velice obtížné, existují tzv. vyšší programovací jazyky, které umožňují zápis kódu pro člověka srozumitelným způsobem, tedy s vyšší mírou abstrakce. Z řady vyšších programovacích jazyků lze jmenovat například C, C++, Javu, Basic a pochopitelně také C#.

Jak bylo uvedeno, procesor zpracovává strojový kód, takže program zapsaný ve vyšším programovacím jazyce samozřejmě nemůže provést. Jednou z možností je tak nechat program přeložit do strojového kódu – *zkompilovat*, což za vás udělá program zvaný *kompilátor* (překladač). Klasickým příkladem kompilovaného jazyka je C++.

Ovšem kromě překladu do strojového kódu existuje také tzv. *interpretace*, kdy je zdrojový kód programu zapsaného v určitém programovacím jazyce vykonáván prostřednictvím jiného programu, který se označuje jako *interpret*. K překladu do strojového kódu tak nedochází, provádění je obecně pomalejší než u zkompilovaného kódu a ke spuštění daného programu je třeba mít navíc daný interpret. Intepretovaným jazykem je například Python či Basic.

U vyšších programovacích jazyků lze dále rozlišovat jazyky procedurální a neprocedurální a v dělení lze pokračovat dále, což je však již mimo rozsah tohoto stručného úvodu.

Jazyk C# a platforma .NET

Koncept jazyka C# byl společností Microsoft představen v roce 2000. O dva roky později byla uvolněna první verze jeho implementace pod názvem Visual C# jako součást vývojového prostředí Visual Studio .NET.

Dle specifikace je C# moderní, jednoduchý, objektově orientovaný a typově bezpečný programovací jazyk založený na základech programovacích jazyků z rodiny C. Syntaxe těchto jazyků je tak velmi podobná, přičemž účel změn v jazyce C# měl za cíl především její zjednodušení.

Pro programování a spouštění aplikací napsaných v jazyce Visual C# musí mít cílový počítač nainstalováno prostředí Microsoft .NET Framework, což je zjednodušeně řečeno platforma obsahující základní knihovnu tříd a rozhraní sloužící k provádění programů napsaných v jazycích, které .NET podporuje – například C# či Visual Basic .NET.

Ať použijete pro psaní programů na platformě .NET jakýkoli podporovaný programovací jazyk, výsledný kód bude vždy přeložen do mezijazyka *CIL* (Common Intermediate Languge), který je na platformě nezávislý. To znamená, že může být proveden v jakémkoli prostředí, které podporuje *CLI* (Common Language Interface), což je klasicky .NET *CLR* (viz níže) pro Windows, ovšem nikoli výhradně, neboť existují i implementace CLI pro jiné operační systémy.

CLR (Common Language Runtime – společné běhové prostředí) je jádrem prostředí .NET, zajišťujícím vše potřebné pro běh programů přeložených do mezijazyka CIL.

Při spuštění aplikace CLR zajistí, že CIL kód je překládán do strojového kódu specifického pro systém cílového počítače ve chvíli, kdy je to třeba (*JIT* - Just In Time), což příznivě ovlivňuje rychlost provádění programu.

Kód běžící pod CLR se označuje jako *řízený* (managed code), neboť využívá služeb CLR, jako je automatická správa paměti či řízení výjimek, přičemž o tyto úlohy se nadále programátor nemusí starat.

Poznámka

V prostředí .NET je možné pracovat i s neřízeným kódem, tato problematika je však mimo rozsah této knihy.

Prostředí .NET Framework ve verzi 3.0 je součástí operačního systému Windows Vista, Windows 7 obsahuje již verzi 3.5. Do obou zmíněných, ale i do starších Windows XP (po nainstalování potřebného servisního balíčku) lze doinstalovat případně verzi 4.0.

Visual C# 2010 Express

Ačkoli k samotnému napsání programu byste v podstatě mohli použít jakýkoli textový editor, tato kniha popisuje práci s aplikací Visual C# 2010 Express, což je funkčně omezená, neplacená verze tzv. *integrovaného vývojového prostředí* (IDE – Intergrated Development Environment), které obsahuje vše potřebné pro vývoj, překlad a ladění aplikací.

Visual C# 2010 Express je produktem z řady Microsoft Visual Studio 2010 Express, které jsou určeny pro studenty, začínající vývojáře a ostatní neprofesionální uživatele.

Po 30 dnech je nutno produkt bezplatně registrovat, aby bylo možné dál pokračovat v jeho užívání.

Instalace platformy .NET 4.0 a Visual C# 2010 Express

Instalaci Visual C# 2010 Express a platformy .NET 4.0 lze provést dohromady – stačí spustit instalační soubor **vcssetup.exe, k**terý stáhnete z adresy *http://www.microsoft.com/express/Downloads/* (odkaz **Visual C# 2010 Express**), přičemž všechny potřebné části budou staženy automaticky z Internetu.

Pokud nechcete nebo nemůžete využít instalaci s připojením k Internetu, je možné stáhnout celý instalační balík a instalaci spustit na počítači bez připojení k Internetu. Příslušný soubor **VS2010Express1.iso** najdete také na adrese *http://www.microsoft.com/express/Downloads/* pod odkazem **All – Offline Install ISO image file.** Jedná se o soubor formátu ISO, takže je třeba jej buď vypálit na CD/DVD, nebo využít software emulující CD/DVD-ROM mechaniky, tzv. virtuální mechaniku. Tento soubor obsahuje všechny produkty z řady Visual Studio 2010 Express, vám však samozřejmě postačí zvolit instalaci Visual C# 2010 Express.

Prostředí .NET 4.0 je možné přímo a samostatně nainstalovat ze souboru **dotNetFx40_Full_x86_x64.exe**, který naleznete na adrese *http://www.microsoft.com/net/* pod odkazem **Get the .NET Framework** (v rámečku **Get Started**).

Poznámka

V závislosti na verzi vašeho operačního systému Windows může být potřeba nainstalovat příslušné servisní balíčky (Service Packs).

Po dokončení instalace platformy .NET 4.0 a vývojového prostředí Visual C# 2010 Express budete mít připraveno vše, co je nutné k programování podle této knihy.

Zdrojový kód aplikací

Přestože v každé kapitole je vývoj dané aplikace popsán od počátku a obsahuje všechny části zdrojového kódu popsaného v textu, vždy máte možnost svůj kód zkontrolovat podle hotových projektů, které jsou uloženy ve složkách pojmenovaných podle jednotlivých kapitol.

Z adresy *http://knihy.cpress.cz/K1890* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

redakce PC literatury
Computer Press
Spielberk Office Centre
Holandská 3
639 00 Brno
nebo
sefredaktor.pc@cpress.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nedá. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji nahlásíte. Ostatní uživatele tak můžete ušetřit frustrace a nám můžete pomoci zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K1890* po klepnutí na odkaz Errata.

KAPITOLA 1

Karetních „21“

Jakou hru budete tvořit

V této kapitole společně vytvoříme jednoduchou textovou hru založenou na karetní hře známé u nás především jako Oko bere či Jednadvacet. Názorně a snadno se tak naučíte následující:

- Co je to konzolová aplikace a jak ji založit
- Jak se deklarují a používají proměnné
- Co jsou datové typy `int`, `short`, `long` a `string` a jak je použít
- Jak lze rozhodovat pomocí příkazu `if`
- Co jsou operátory relační a logické
- Co jsou operátory složeného přiřazení
- Jak se používá příkaz `goto`
- Jak pracovat s režimem ladění
- Jak pracovat s vestavěnou třídou `Console` a `Random` (Třídám obecně se budete věnovat v pozdější kapitole, pro tento příklad není nutné jim rozumět).

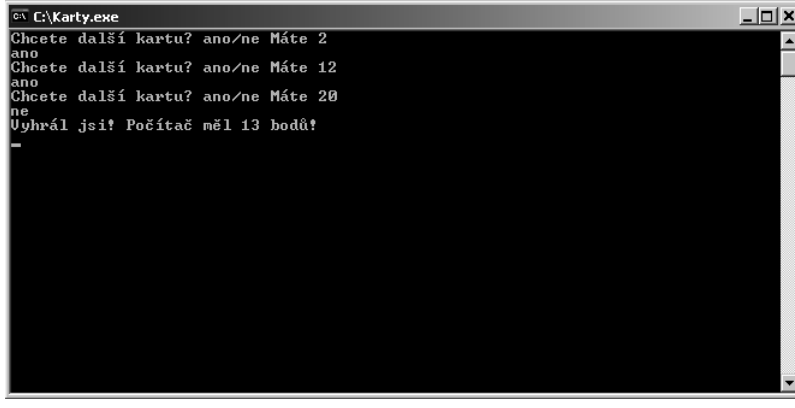
Nejprve je třeba si říct, jak má zamýšlená aplikace fungovat. Pointa karetní hry Oko bere spočívá v tom, že hráči snímají karty z balíčku a snaží se, aby byl jejich součet co nejblíže nebo přímo roven číslu 21. Pokud hráč tuto hodnotu přesáhne, prohrává. Záleží tedy pouze na náhodě a ochotě hráče riskovat, protože hodnotu příští karty nelze nijak ovlivnit ani předvídat.

Jednotlivé karty budou představovat čísla od 1 do 11. V plánované hře budou hráči pouze dva – uživatel a jeho protihráč počítač. Každý z nich bude začínat s první kartou náhodné hodnoty.

Poté se hra dotáže hráče (uživatele), zda má zájem vytáhnout si další kartu. Pokud odpoví kladně, bude karta náhodně vybrána a její hodnota přičtena k celkovému součtu jeho karet. Pak přijde na řadu protihráč (počítač) a také se rozhodne, zda chce další kartu či nikoli. Takto se hráči střídají do té doby, než uživatel zvolí, že již další kartu nechce. V takovém případě program porovná celkové součty karet obou hráčů a určí vítěze podle toho, který z nich dosáhl čísla nebo se k němu alespoň více přiblížil.

Pro úplnost je vhodné zmínit, že protihráč je znevýhodněn tím, že skončit hru (resp. přistoupit k určení vítěze) může pouze uživatel. Tedy když ten kartu odmítne, hra skončí i v případě, že protihráč by další kartu ještě riskovat chtěl.

Pro účely zachování jednoduchosti se však tímto nedostatkem zabývat nebudete. Po spuštění a chvíli hraní bude aplikace vypadat takto:



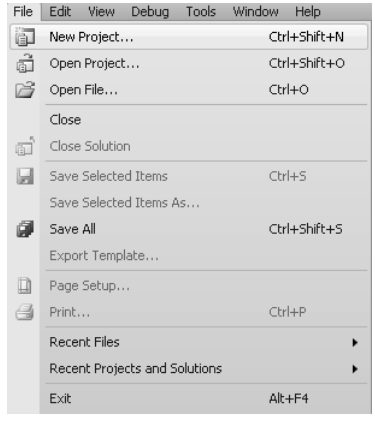
Obrázek 1.1 Dokončená a spuštěná aplikace Karty

Založení konzolové aplikace

Aplikace bude pouze textová bez grafického rozhraní – systém bude zobrazovat informace pouze ve formě textu a hráč bude reagovat vstupem z klávesnice. Takovým aplikacím, které běží v okně příkazového řádku, se říká *konzolové*.

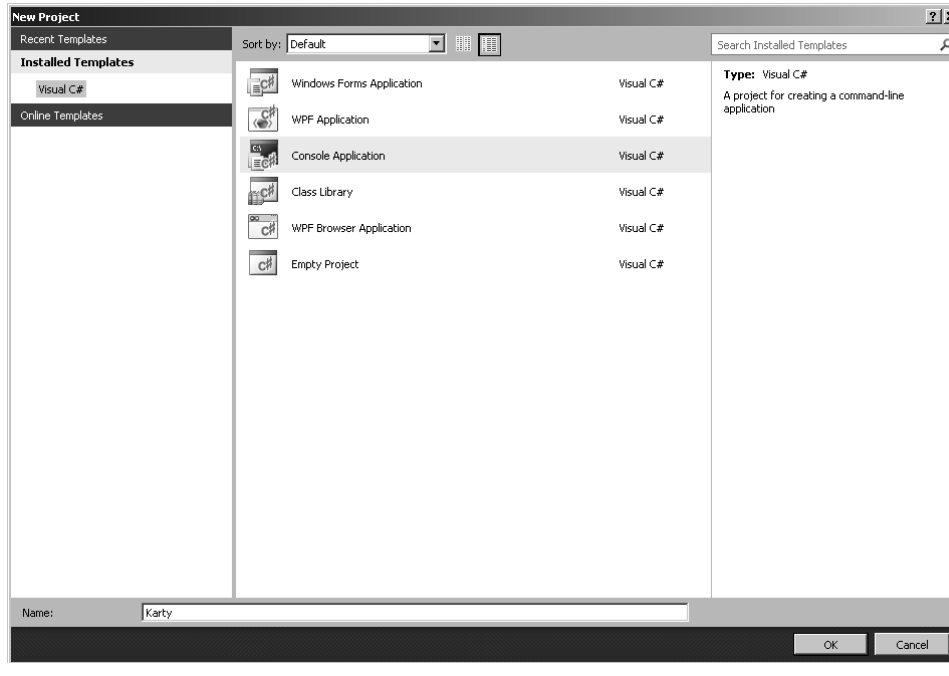
Vaším prvním krokem tak bude založení konzolové aplikace v prostředí Visual Studia Express 2010:

1. Spusťte Visual Studio Express 2010.
2. Z panelu nabídek vyberte **File → New Project.**



Obrázek 1.2 Vytvoření nového projektu

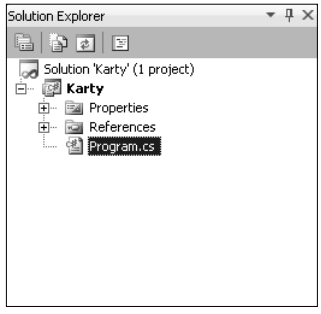
3. Zvolte typ **Console application**.
4. Do kolonky **Name** dole zadejte název aplikace, v tomto případě „Karty“, a potvrďte tlačítkem **OK.**



Obrázek 1.3 Založení konzolové aplikace

Po vytvoření projektu bude jeho struktura zobrazena v okně s názvem **Solution Explorer**. Pokud toto okno nemáte zobrazeno, můžete jej vyvolat pomocí nabídky **View → Other Windows → Solution Explorer** z panelu nabídek Visual Studia.

Kromě složek `Properties` a `References`, kterými se zatím vůbec nemusíte zabývat, se zde nachází i soubor `Program.cs`. Jeho zdrojový kód by měl být zobrazen v otevřeném okně se stejným názvem. Pokud tomu tak není, jednoduše na soubor poklepejte levým tlačítkem myši.



Obrázek 1.4 Struktura projektu a soubor Program.cs

Ve zdrojovém kódu souboru `Program.cs` se nachází mimo jiné i statická (co to znamená se dozvíte až později) metoda `Main`, jejíž kód je uveden spolu s komentářem níže. Tato metoda je vstupním bodem programu – po jeho spuštění dojde k zavolání této metody a provedení příkazů v jejím těle. Její přítomnost je proto povinná, čili každá aplikace v C# musí tuto metodu obsahovat. Pokud byste ji přejmenovali či smazali, program by jednoduše nevěděl, kde začít.

Prozatím tedy budete zapisovat kód hry přímo do těla této metody, tedy mezi složené závorky, tak, jak vidíte níže.

```
static void Main(string[] args)
{
  //sem budete zapisovat celý kód vaší hry
}
```

Zde je vhodné ještě zmínit, že cokoli zapsané za dvě lomítka slouží v kódu jako komentář, který je jednoduše při sestavování programu ignorován. Pokud chcete zapsat delší poznámku, zapište nejprve `/*` a poté normálně pokračujte svým textem. K ukončení použijte znaky `*/`.

```
/* Takto je možné zapsat delší poznámku
 * na více řádků.
 */
```

Kdo má jaké karty aneb proměnné

Z návrhu i samostné podstaty hry vyplývá, že program bude muset pro každého z hráčů uchovávat jednu číselnou hodnotu – ta bude představovat hodnotu součtu jeho karet, a bude se tak měnit (zvyšovat) s každou další vytaženou kartou.

Je proto zřejmé, že budete potřebovat dvě proměnné, z nichž jedna bude představovat součet karet uživatele a druhá součet karet protihráče. Každá tak bude obsahovat číselnou hodnotu, což je vlastně konkrétní typ dat. Jiným typem dat je například textový řetězec.

Proměnné v jazyce C# musí být vždy určitého datového typu, který tak určuje, jaké hodnoty mohou uchovávat a jak s nimi lze pracovat.

Vaše dvě proměnné proto budou muset být takového datového typu, aby v nich mohlo být uloženo číslo. Takových (celočíselných) datových typů je v jazyce C# několik. Liší se tím, jak velká čísla mohou uchovávat:

- Typ `short`, do kterého lze uložit číslo o délce 16 bitů
- Typ `int`, do kterého lze uložit číslo o délce 32 bitů
- Typ `long`, do kterého lze uložit číslo o délce 64 bitů

Poznámka

Typ `short` je ve skutečnosti jen přezdívkou (tzv. alias) pro `Int16`, `int` pro `Int32` a `long` pro `Int64`. Int je zkratkou anglického slova „integer" – celé číslo. Maximální hodnoty jednotlivých typů jsou uvedeny ve shrnutí na konci kapitoly. Uvedené tři typy nejsou jedinými celočíselnými typy, které jazyk C# nabízí.

Když si uvědomíte, že cílem obou hráčů je mít méně nebo právě 21 bodů, je vcelku zřejmé, že pro tento případ by bohatě postačil typ `short`. Ovšem z určitých důvodů, které by vám příklad zkomplikovaly, použijete datový typ `int`.

Abyste mohli s proměnnou začít pracovat, je nejprve nutné ji deklarovat. Tím v podstatě oznámíte, že je potřeba rezervovat místo v paměti, kde bude uložena hodnota daného datového typu.

U deklarace proměnných je třeba uvést nejdříve datový typ a poté zvolený název, který se také označuje jako *identifikátor*. Ten by měl v tomto případě začínat malým písmenem, a pokud je složen z více slov, je první písmeno každého dalšího slova velké. Tento zápis se označuje jako *velbloudí* (Camel case).

Proměnná musí mít vždy před použitím přiřazenu hodnotu. To lze provést přímo během deklarace nebo později prostým přiřazením, jak vidíte na ilustračním kódu níže:

```
//Ukázka! Tento kód není součástí hry!
int ukazka;  //deklarace proměnné typu int
ukazka=5;    //přiřazení hodnoty 5
int promenna=5; //deklarace s inicializací na hodnotu 5
```

Poznámka

Přiřazení počáteční hodnoty proměnné se běžně označuje jako její inicializace.

Začněte tedy vytvořením dvou proměnných typu `int` se smysluplnými názvy, které se vám později nebudou plést, a přiřaďte oběma počáteční hodnotou 0. Protože jde o deklarační příkazy, nesmíte zapomenout zakončit je středníkem:

```
static void Main(string[] args)
{
  int kartyHrace=0;
  int kartyPC=0;
}
```

Jak je specifikováno v návrhu programu, každý hráč má začínat s jinou hodnotou (každý si totiž vytáhne jinou počáteční kartu). Zatím však začínají oba s nulou. Potřebujete tedy, aby byla hodnota proměnné nastavena místo na 0 na náhodné číslo od 1 do 11.

K tomu můžete využít metodu `Next` třídy `Random` a upravit kód do této podoby:

```
static void Main(string[] args)
{
  Random nahodnaCisla=new Random(); //zatím jen opíšeme
  int kartyHrace=nahodnaCisla.Next(1,12);
  int kartyPC=nahodnaCisla.Next(1,12);
}
```

Zatím tento kód jednoduše opište a přijměte vysvětlení, že příkaz (volání metody) `nahodnaCisla.Next(1,12)` vrátí náhodné číslo z intervalu 1 až 11 a výsledná hodnota je přiřazena proměnné. Protože hodnota horní hranice intervalu je při použití metody `nahodnaCisla.Next` vyloučena, je třeba zde použít číslo 12.

Poznámka

Náhodná čísla generovaná třídou Random ve skutečnosti úplně náhodná nejsou, jsou pseudo náhodná, neboť k jejich generování je použit určitý algoritmus. Ač se to může zdát zvláštní, ve skutečnosti není generování náhodných čísel jednoduché.

Výpis textu na obrazovku

Nyní, když jsou karty rozdány, by měl systém zobrazit součet hodnot karet hráče (uživatele) a prostřednictvím textové zprávy se jej zeptat, zda si přeje vytáhnout další. Zobrazení textu provedete snadno pomocí třídy `Console` a její metody `WriteLine`, která zadaný text vypíše na nový řádek obrazovky:

```
Console.WriteLine("Chcete další kartu? ano/ne Máte " + kartyHrace);
```

Obsah závorky je ve skutečnosti parametr metody (zde typu `string` – viz níže), o čemž se více dozvíte až ve třetí kapitole. Celý příkaz samozřejmě zakončíte středníkem.

Řetězcová konstanta (předem vepsaný textový řetězec) musí vždy začínat a končit dvojitými uvozovkami `"`. Následuje operátor `+`, který slouží ke spojování řetězců a k zadanému textu vám připojí hodnotu proměnné `kartyHrace`, respektive její textovou reprezentaci.

Aby se hodnota této proměnné nevypsala hned za poslední písmeno textového řetězce, je na jeho konci záměrně umístěna mezera.

Čtení vstupu od uživatele

Po vypsání otázky bude třeba, aby program umožnil uživateli reagovat zadáním vstupu – tedy textového řetězce „ano“ či „ne“. Na základě toho se program bude moci později rozhodnout, jak dál pokračovat.

Abyste ale mohli se vstupem od uživatele pracovat, bude třeba jej uložit do proměnné, kterou si k tomuto účelu založíte. Jak jste se již dočetli, každá proměnná musí být takového datového typu, jaká data bude uchovávat. Pro čísla jste použili typ `int`, nyní bude třeba pracovat s textovým řetězcem a pro ten existuje datový typ nazvaný `string`.

Proměnnou tohoto typu a s názvem `volba` deklarujete stejně jako všechny ostatní proměnné: uvedením typu a názvu. Přiřazení na druhém řádku je v ukázce níže uvedeno jen pro doplnění, a tento řádek tak do kódu hry nepište.

```
string volba;
volba="Ukázkový řetězec"; //Tak lze přiřadit hodnotu proměnné typu string
```

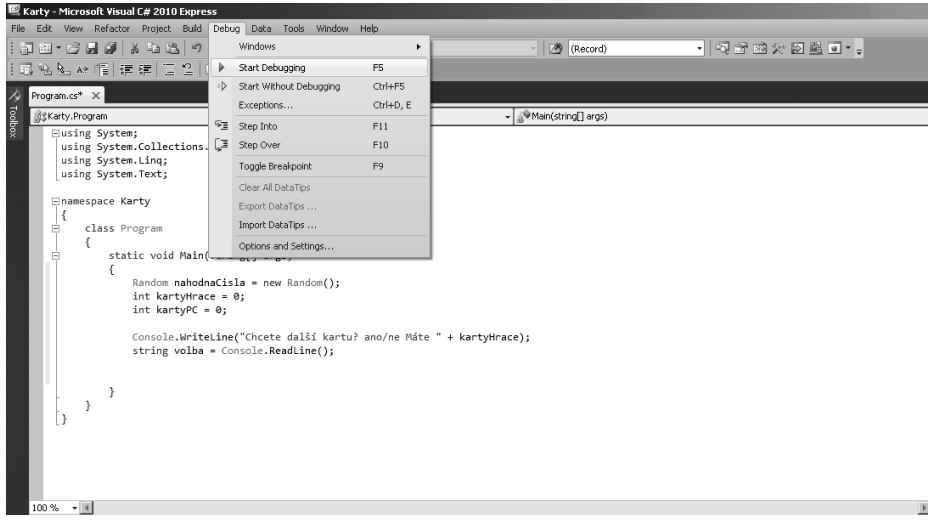
Pro zachycení vstupu uživatele použijete metodu `Console.ReadLine`. Po jejím zavolání bude uživatel moci psát do příkazového řádku programu a potvrdit svůj vstup stiskem klávesy Enter. Upravte tedy předchozí deklaraci proměnné takto:

```
string volba=Console.ReadLine();
```

Tak vlastně říkáte, že proměnné `volba` bude přiřazena taková hodnota, jakou vrátí volaná metoda, a `Console.ReadLine` vrací text zadaný uživatelem, tedy textový řetězec typu `string`. I kdyby uživatel zadal pouze číslo, výsledek bude vrácen jako textový řetězec, a tak jej můžete bezpečně uložit do proměnné typu `string`.

Ještě než se pustíte do další části, můžete si aplikaci spustit a tak si ozkoušet, že skutečně zobrazí zadaný text a hodnotu karet a čeká na vstup od uživatele, po kterém skončí.

Sestavení a spuštění provedete stiskem klávesy **F5** nebo klepnutím na položku **Debug → Start Debugging** v panelu nabídek.



Obrázek 1.5 Spuštění programu

Vaše dosavadní metoda `Main` by měla vypadat takto:

```
static void Main(string[] args)
{
  Random nahodnaCisla=new Random(); //zatím jen opíšeme
  int kartyHrace=nahodnaCisla.Next(1,12);
  int kartyPC=nahodnaCisla.Next(1,12);
  Console.WriteLine("Chcete další kartu? ano/ne Máte " +
  kartyHrace);
  string volba=Console.ReadLine();
}
```

Rozhodování příkazem if

Když máte zajištěno, že proměnná `volba` bude obsahovat textový vstup zadaný uživatelem („ano“ či „ne“), využijete ji k rozhodnutí, jak má program dále pokračovat. Jestliže uživatel odpověděl kladně, pak má dostat další kartu. Pokud ne, hra má skončit a vyhodnotit výsledky. V každém případě je třeba, aby program provedl různé činnosti podle toho, jak uživatel odpověděl, a tedy jaká je hodnota proměnné `volba`.

K takovému účelu, kdy je potřeba spustit příkaz či příkazy, jen pokud je splněna určitá podmínka, slouží příkaz `if`. Následující kód je podrobně vysvětlen níže:

```
if (volba=="ano")
{
  //první blok
}
else if (volba=="ne")
{
  //druhý blok
}
else
{
  //třetí blok
}
```

Při použití příkazu `if` nejprve zapíšete do závorek logický výraz, který lze vyhodnotit jako pravdivý nebo nepravdivý. Prostě řečeno jde o podmínku, která buď platí, nebo ne. Pokud má být v případě jejího splnění provedeno více příkazů než jeden, musí být umístěny ve složených závorkách. To je zde předpokládáno, a proto jsou závorky v kódu již předepsány.

Zde jde nejprve o výraz `volba=="ano"`. V něm se porovnává hodnota vpravo, tj. textový řetězec „ano“, s hodnotou vlevo, uloženou v proměnné `volba`. Aby mohly být hodnoty vůbec porovnány, musí být samozřejmě stejného typu. Zde je to splněno, neboť textový řetězec porovnáváte s proměnnou stejného typu – `string`.

Způsob, jakým se porovnávají, určuje operátor umístěný mezi nimi, v tomto případě operátor rovnosti ==. To znamená, že pokud je textový řetězec v proměnné `volba` roven textovému řetězci „ano“, výraz je vyhodnocen jako pravdivý a dojde ke spuštění bloku kódu označeného jako *první blok*.

Pro úplnost je třeba uvést, že u porovnávání textových řetězců se samozřejmě rozlišují velká a malá písmena.

Poznámka

Pravá a levá hodnota ve výrazu se správně nazývají operandy. Operátor tedy s operandy provádí operaci. Pro ilustraci lze uvést sčítání: 1+2. 1 a 2 jsou operandy, + je operátor a provádí operaci sčítání. Operátor nerovnosti se zapisuje jako `!=`. Např. výraz `volba!=„ano“` bude pravdivý vždy, když bude hodnota proměnné `volba` odlišná od „ano“.

Pokud výraz v `if` pravdivý není (hodnota proměnné `volba` není „ano”), přejde program automaticky ke klauzuli `else`, za kterou může následovat libovolný příkaz – zde `if(volba=="ne")`. Hodnota proměnné `volba` tak bude opět porovnána, tentokrát s textovým řetězcem „ne“. Bude-li tento výraz vyhodnocen jako pravdivý, program spustí blok kódu označený jako *druhý blok*.

A konečně, pokud byly předchozí výrazy vyhodnoceny jako nepravdivé, přejde program k poslední klauzuli `else`. Například v případě, kdy uživatel nenapíše ani „ano“, ani „ne“, spustí program blok kódu označený jako *třetí blok*.

Poznámka

Pro správnost je třeba uvést, že klauzule `else` není povinná. Zapisujete ji tehdy, pokud chcete provést konkrétní příkazy jen v případě, kdy podmínka není splněna.

Po vykonání příkazů v daném bloku program přejde na konec příkazu `if` a pokračuje dalším příkazem – a tak pokud by v uvažovaném případě byla hodnota proměnné `volba` „ano“, vykoná se první blok kódu a program následně pokračuje až za poslední klauzulí `else`. Druhý a třetí blok tak samozřejmě nebude spuštěn.

Jako první zde napíšete kód pro případ, kdy uživatel odpoví „ano“ – přeje si tedy vytáhnout další kartu a tím zvýšit svůj celkový součet o její hodnotu. Z pohledu programu nejde o nic jiného než o vygenerování náhodného čísla (stejně jako při rozdávání úvodní karty) a jeho přičtení k proměnné `kartyHrace`, kterou jste vytvořili právě pro uchovávání hodnot hráčových karet.

Vygenerujete proto opět náhodné číslo (samozřejmě od 1 do 11) a přičtete jej k proměnné `kartyHrace`:

```
if (volba=="ano")
{
  kartyHrace+=nahodnaCisla.Next(1,12);
}
```

Operátor += je zkráceným zápisem a znamená totéž, jako byste napsali `kartyHrace=kartyHrace+nahodnaCisla.Next(1,12)`. Jednoduše tedy k hodnotě vlevo přičítáte hodnotu vpravo. Tento operátor se nazývá *operátor složeného přiřazení* a rozhodně byste si jeho používání měli osvojit, protože šetří čas a zlepšuje přehlednost kódu.

Poznámka

Stejně tak lze samozřejmě zapsat `-=` pro odečítání, `*=` pro násobení či `\=` pro dělení. Např. tedy (uvažujte o A jako o proměnné typu int) `A-=2` má stejný význam jako `A=A-2`.

Hráč již svou kartu má, nyní je tak řada na počítači. Jeho rozhodování o tom, zda riskovat, bude založeno na jednoduchém pravidle: pokud bude součet jeho karet (tedy hodnota proměnné `kartyPC`) nižší než 15, vezme si další kartu.

Přidejte proto tuto podmínku spolu s příkazem, který zajistí přičtení náhodného čísla, k proměnné `kartyPC`:

```
if (kartyPC<15)
{
  kartyPC+=nahodnaCisla.Next(1,12);
}
```

První blok v příkazu `if` tak bude nyní vypadat takto:

```
if (volba=="ano")
{
  kartyHrace+=nahodnaCisla.Next(1,12);
```

```
  if (kartyPC<15)
  {
    kartyPC+=nahodnaCisla.Next(1,12);
  }
}
```

Kód by vám měl být již zřejmý, snad kromě relačního operátoru `<`. Ten v daném výrazu ověřuje, zda je hodnota levého operandu **menší než** hodnota operandu pravého. Pokud je tedy hodnota proměnné `kartyPC` nižší než 15, je k ní přičteno náhodné číslo od 1 do 11. Přehled relačních operátorů je uveden v tabulce na konci této kapitoly.

Ačkoli `if` zde podmiňuje pouze jeden příkaz, a složené závorky by tak bylo možné vynechat, je přesto dobré zapisovat je.

Klauzule `else` není použita, protože nepotřebujete vykonat žádnou akci pouze v případě, kdy podmínka neplatí. Program tak jednoduše pokračuje dál.

Řízení toku programu příkazem goto

Nyní když oba hráči učinili svá rozhodnutí, program by se měl dostat zpět do bodu, kdy zobrazí hodnotu karet hráči a opět se jej dotáže, zda si přeje vytáhnout další kartu. Měl by tedy pokračovat v určité smyčce, což vyplývá ze samotného návrhu hry v úvodu této kapitoly.

K zajištění toho, že zpracování programu se přesune do určitého bodu, použijete příkaz `goto`. Do kódu jednoduše umístíte pojmenovaná návěští (jakési „značky“ v kódu), na něž poté příkazem `goto` ukážete. Ilustrační příklad může vypadat takto:

```
TestovaciNavesti:        //návěští
Console.WriteLine("Ukázka");
goto TestovaciNavesti;
```

Pokud by byl takový kód spuštěn, opakoval by se donekonečna – program by vypsal „Ukázka“ , příkazem `goto` by se přesunul na návěští `TestovaciNavesti` a tak stále dokola.

Poznámka

Příkaz `goto` se ve skutečnosti příliš nepoužívá (ačkoli jsou situace, kdy je jeho použití vhodné) a je zde uvedem především pro snazší pochopení toku programu.

Zpět ale k naší aplikaci. Nejprve umístíte návěští pojmenované `DalsiKarta` nad řádek, kterým vypisujete otázku a hodnotu součtu karet hráče. Pokud byste ale návěští umístili ještě výše, nad deklaraci a přiřazení hodnot proměnným, došlo by znovu k jejich nastavení, a tedy ke ztrátě dosud přičtených hodnot.

Váš kód by tedy měl nyní vypadat takto:

```
int kartyHrace=nahodnaCisla.Next(1,12);
int kartyPC=nahodnaCisla.Next(1,12);
DalsiKarta:
Console.WriteLine("Chcete další kartu? ano/ne Máte " +
  kartyHrace);
```

Poté na toto návěští jen odkážete příkazem `goto`. Ten samozřejmě umístíte až za příkaz `if`, kterým se počítač rozhodoval, zda si vytáhne další kartu:

```
if (volba=="ano")
{
  kartyHrace+=nahodnaCisla.Next(1,12);
  if (kartyPC<15)
  {
    kartyPC+=nahodnaCisla.Next(1,12);
  }
  goto DalsiKarta;
}
```

Tím jste zajistili, že program po spuštění bude tak dlouho nabízet hráči další kartu a zobrazovat aktuální součet bodů, dokud bude uživatel odpovídat „ano".

Nebudete se už zabývat situací, že po překročení hodnoty 21 by již hráč neměl mít možnost žádat další karty. Protože jde o drobnou úpravu, můžete se o ni pokusit sami po dokončení této kapitoly.

Podmínky pro určení vítěze

Nyní je třeba naprogramovat chování programu v případě, kdy uživatel již nechce další kartu (odpoví tedy „ne"). Je zřejmé, že tím hra končí a v tomto okamžiku musí dojít k porovnání součtu hodnot obou hráčů a určení vítěze. K tomu použijete již známé příkazy `if` (budete porovnávat hodnoty), ale naučíte se i něco nového.

Nejprve zapíšete ideální případ výhry uživatele, kdy součet jeho karet je menší nebo roven číslu 21 a zároveň protihráč toho číslo přesáhl. Ačkoli byste mohli obě podmínky zapsat pomocí více příkazů `if`, šlo by o zbytečně komplikovaný zápis. Namísto toho proto zapíšete obě podmínky do jednoho výrazu a mezi nimi použijete logický operátor `&&` s významem „a zároveň" – operátor logického součinu. Tak v podstatě říkáte, že musí platit první a zároveň druhá podmínka:

```
else if (volba=="ne")
{
  if (kartyHrace<=21 && kartyPC>21)
  {
  }
}
```

Jen pokud budou oba výrazy vyhodnoceny jako pravdivé, bude i výraz jako celek vyhodnocen jako pravdivý.

Tento výraz říká: hodnota proměnné `kartyHrace` musí být menší nebo rovna 21 **a zároveň** hodnota proměnné `kartyPC` musí být větší než 21.

Pokud se však zamyslíte, uživatel vyhraje i v případě, kdy je jednoduše blíže číslu 21 než protihráč – má tedy méně nebo právě 21 bodů, ale stále má více než protihráč.

V tom případě můžete přeformulovat podmínku: uživatel vyhraje v případě, že má méně nebo právě 21 bodů **a zároveň** protihráč má více než 21 bodů **nebo** méně než uživatel.

V takovém případě, kdy stačí, aby platila alespoň jedna ze dvou podmínek, použijete logický operátor `||` s významem „nebo" – operátor logického součtu. Předchozí výraz v příkazu `if` tedy upravte do tohoto tvaru:

```
if (kartyHrace<=21 && ( kartyPC>21 || kartyPC<kartyHrace ) )
{
}
```

Je-li tedy hodnota proměnné `kartyHrace` menší nebo rovna 21 **a zároveň** je hodnota proměnné `kartyPC` větší než 21 **nebo** menší než `kartyHrace`, bude celý výraz pravdivý.

Ještě je třeba si vysvětlit, proč jsou oba výrazy vpravo od operátoru `&&` uzavřené v závorce. Je tomu tak proto, že operátor `&&` má přednost před operátorem `||`, stejně jako má například násobení přednost před sčítáním (a tedy operátor + má přednost před operátorem * aneb 2*(1+2) není totéž jako 2*1+2). Pokud byste závorky nezapsali, výraz jako celek by byl vyhodnocen jinak, než zamýšlíte – říkal by, že musí platit první **a zároveň** druhá podmínka, **nebo** třetí podmínka – stačilo by tedy, aby platila jen třetí. Zde je však nutné, aby vždy platila první **a zároveň** ještě buď druhá, **nebo** třetí.

Poznámka

Ve skutečnosti jde opravdu o přednost násobení před sčítáním, protože operátor && provádí logický součin, zatímco operátor || logický součet.

Pokud jsou podmínky splněny a výraz je vyhodnocen jako pravdivý, program by měl informovat uživatele o vítězství a zobrazit i celkovou hodnotu karet, které dosáhl počítač. K tomu vám opět poslouží metoda `Console.WriteLine`:

```
Console.WriteLine("Vyhrál jsi! Počítač měl "+kartyPC+" bodů! ");
```

V takovém případě jste se dostali k jednomu z možných konců hry, a váš program tak může úspěšně skončit.

Můžete tedy rovnou přejít k další variantě, jak může porovnání výsledků skončit – výhrou počítače nad uživatelem.

Jak víte, není-li výraz v příkazu `if` vyhodnocen jako pravdivý, pokračuje program ke klauzuli `else`, pokud existuje. Protože prohra uživatele automaticky neznamená, že by vyhrál počítač (mohou prohrát oba či situace může být nerozhodná), použijte klauzuli `else` a vnořený příkaz `if`, kde specifikujte podmínky pro výhru protihráče:

```
else if (kartyPC<=21 && (kartyHrace>21 || kartyPC>kartyHrace))
{
}
```

Skladbu výrazů i operátory již znáte – příkaz říká, že protihráč musí mít méně nebo právě 21 bodů a zároveň uživatel musí mít více než 21 nebo méně, než má protihráč. V případě, že je výraz pravdivý, vypíšete tuto informaci a i zde zobrazíte body protihráče:

```
Console.WriteLine("Prohrál jsi! Počítač měl "+kartyPC+" bodů! ");
```

Jestliže ani tento výraz nebyl vyhodnocen jako pravdivý, nevyhrál nikdo a v dalších klauzulích `else` s vnořeným přkazem `if` jen zjistíte, zda oba hráči prohráli či je situace nerozhodná.

Nejprve zapište podmínku pro prohru obou hráčů (kdy mají oba nad 21 bodů) a vypsání dané informace:

```
else if (kartyPC>21 && kartyHrace>21)
{
  Console.WriteLine("Oba hráči prohráli! Počítač měl "
    +kartyPC+" bodů! ");
}
```

A nakonec ošetřete situaci, kdy se hodnoty obou hráčů rovnají, a vítěze tak nelze určit:

```
else if (kartyPC==kartyHrace)
{
  Console.WriteLine("Nikdo nevyhrál!");
}
```

Poznámka

Zde by mohla být použita prostá klauzule `else` bez vnořeného příkazu `if`, protože všechny ostatní možné situace jsou již pokryty podmínkami výše.

Pro přehlednost si lze níže zkontrolovat, jak má kód v této části (sekce příkazu `if` podmíněná pravdivostí `volba=="ne"`) programu vypadat:

```
else if (volba=="ne")
{
  if (kartyHrace<=21 && ( kartyPC>21 || kartyPC<kartyHrace ) )
  {
    Console.WriteLine("Vyhrál jsi! Počítač měl "
      +kartyPC+" bodů! ");
  }
  else if (kartyPC<=21 && (kartyHrace>21 || kartyPC>kartyHrace))
  {
    Console.WriteLine("Prohrál jsi! Počítač měl "
      +kartyPC+" bodů!");
  }
  else if (kartyPC>21 && kartyHrace>21)
  {
    Console.WriteLine("Oba hráči prohráli! Počítač měl "
      +kartyPC+" bodů!");
  }
  else if (kartyPC==kartyHrace)
  {
    Console.WriteLine("Nikdo nevyhrál!");
  }
}
```

K úplnému dokončení této sekce kódu schází už jen zastavení programu, protože ten by jinak po vypsání textu okamžitě skončil, a uživatel by tak neměl šanci přečíst výsledek.

K prostému zastavení a pokračování stiskem klávesy můžete použít metodu `Console.ReadKey`. Výsledek volání této metody vás nemusí zajímat, a tak ji postačí jednoduše zavolat, aniž byste výslednou hodnotu přiřazovali nějaké proměnné. Protože je třeba zastavit program vždy po jakémkoli výsledku hry, zapište volání až na konec, za poslední `else` klauzuli.

```
Console.ReadKey();
```

Tímto příkazem jste tedy sekci kódu podmíněnou `volba=="ne"` dokončili, a vaše hra je tak téměř hotová.

Ošetření chybného vstupu

Když je uživatel dotázán na další kartu a odpoví „ano“ nebo „ne“, dojde ke splnění jedné z podmínek ve vnořených příkazech `if` a spuštění příslušného kódu. Pokud však uživatel zadá cokoli jiného, program se v daném příkazu `if` přesune do zatím prázdné sekce za poslední klauzulí `else`.

Zde je třeba jednoduše vypsat informaci o nesprávném vstupu a pomocí příkazu `goto` vrátit zpracování programu zpět k vypsání otázky, aby mohl uživatel zadat vstup znovu:

```
else
{
  Console.WriteLine("Nesprávný vstup.");
  goto DalsiKarta;
}
```

Hra bude tedy zadání odpovědi vyžadovat tak dlouho, dokud uživatel nenapíše buď „ano“, nebo „ne“.

Kompletní kód

Zde si můžete zkontrolovat kód, který by měla obsahovat metoda `Main` po dokončení této kapitoly:

```
static void Main(string[] args)
{
  Random nahodnaCisla = new Random();
  int kartyHrace = nahodnaCisla.Next(1, 12);
  int kartyPC = nahodnaCisla.Next(1, 12);
  DalsiKarta:
  Console.WriteLine("Chcete další kartu? ano/ne Máte "
    + kartyHrace);
  string volba = Console.ReadLine();
  if (volba == "ano")
  {
    kartyHrace += nahodnaCisla.Next(1, 12);
    if (kartyPC < 15)
    {
      kartyPC += nahodnaCisla.Next(1, 12);
    }
    goto DalsiKarta;
  }
  else if (volba == "ne")
  {
    if (kartyHrace <= 21 && (kartyPC > 21 || kartyPC
      < kartyHrace))
    {
      Console.WriteLine("Vyhrál jsi! Počítač měl "
        + kartyPC + " bodů!");
    }
    else if (kartyPC <= 21 && (kartyHrace > 21 || kartyPC
      > kartyHrace))
    {
      Console.WriteLine("Prohrál jsi! Počítač měl "
        + kartyPC + " bodů!");
    }
    else if (kartyPC > 21 && kartyHrace > 21)
    {
      Console.WriteLine("Oba hráči prohráli! Počítač měl "
        + kartyPC + " bodů!");
    }
```

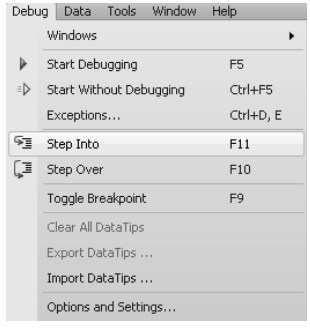
```
    else if (kartyPC == kartyHrace)
    {
      Console.WriteLine("Nikdo nevyhrál!");
    }
    Console.ReadKey();
  }
  else
  {
    Console.WriteLine("Nesprávný vstup.");
    goto DalsiKarta;
  }
}
```

Ladění aplikace

Může se stát, že vám z nějakého důvodu aplikace nefunguje přesně tak, jak si představujete. V takové chvíli vás bude často zajímat, které řádky kódu program právě zpracovává či jakou hodnotu má která proměnná. A právě k tomu slouží režim ladění (*Debugging mode*), který vám umožní krok po kroku sledovat tok programu, prohlížet aktuální hodnoty proměnných a další.

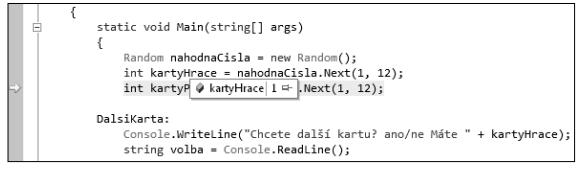
V tomto režimu jste již aplikaci spouštěli, nyní však uvidíte, jak sledovat zpracování programu řádek po řádku:

1. Pro sestavení a spuštění aplikace zvolte z panelu nabídek **Debug → Step Into** nebo stiskněte **F11**.



Obrázek 1.6 Krokování aplikace příkaz po příkazu

2. V okně zdrojového kódu dojde k vyznačení řádku, který má být zpracován. Program je pozastaven v režimu přerušení (*break mode*).
3. Pro zpracování následujícího řádku opět zvolte z panelu nabídek **Debug → Step Into** nebo stiskněte **F11**.
4. Pokračujte až za deklaraci a přiřazení proměnných a poté jednoduše podržte kurzor myši nad jednou z nich (např. `kartyHrace`), abyste zjistili její hodnotu a případně i další informace, což velmi oceníte později.

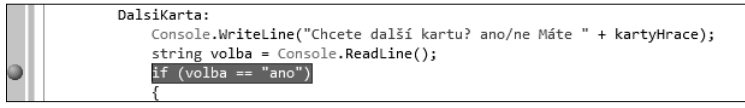


Obrázek 1.7 Zobrazení hodnoty proměnné v režimu ladění

Většinou vás ale bude zajímat jen určitá oblast kódu, a proto nebudete chtít procházet vždy řádek po řádku celým programem, ale budete jej chtít zastavit až v určitém bodě, kde máte například podezření na chybu či vás zajímá stav nějakého objektu. K tomuto účelu slouží zarážky (*breakpoints*).

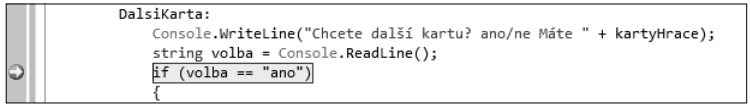
Na příkladu naší hry si můžete představit, že chcete, aby aplikace běžela normálně až do zadávání textu uživatelem – zde chcete prozkoumat, zda následující příkaz `if` funguje správně, a tak potřebujete, aby na tomto místě došlo k přerušení programu:

1. V okně zdrojového kódu klepněte myší na levý okraj okna vedle řádku s kódem `if (volba=="ano")`, čímž umístíte zarážku.



Obrázek 1.8 Umístění zarážky

2. Sestavte a spusťte aplikaci stiskem klávesy **F5** či z menu **Debug → Start Debugging** v panelu nabídek.
3. Zadejte textový vstup očekávaný programem.
4. Zpracování programu došlo k řádku, který je označený zarážkou. Program je nyní pozastaven, ve zdrojovém kódu vidíte daný řádek vyznačený a můžete zkontrolovat hodnotu proměnné.

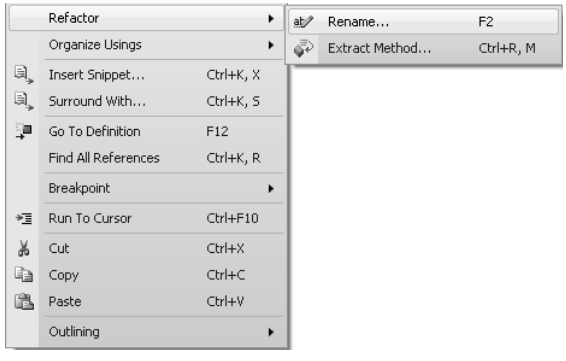


Obrázek 1.9 Zpracování programu došlo k příkazu označenému zarážkou

5. Pokud chcete, aby program pokračoval dál (respektive do další zarážky), stiskněte **F5** či použijte volbu **Debug → Continue**, pro krokování příkaz po příkazu pak **F11** nebo **Debug → Step Into**.

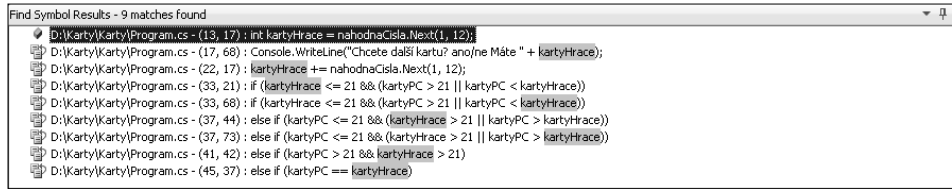
Užitečné tipy

- Pokud potřebujete změnit název identifikátoru, například chcete přejmenovat proměnnou, klepněte na ni pravým tlačítkem a z kontextové nabídky zvolte **Refactor → Rename**, kde zadejte nový název a potvrďte.



Obrázek 1.10 Změna názvu identifikátoru

- Chcete-li vidět, kde všude se v kódu vyskytuje například určitá proměnná, klepněte na ni pravým tlačítkem a z kontextové nabídky zvolte **Find All References**. V okně **Find Symbol Results** budou zobrazeny nalezené řádky kódu; poklepáním na jednotlivý řádek se editor kódu přesune na dané místo.



Obrázek 1.11 Vyhledané reference na proměnnou kartyHrace

Shrnutí

Probraná látka	Příklad
Proměnné deklarujete zápisem ve formě: *Datový_typ název středník*	`int ciselnaPromenna;` `string Test;`

Proměnným musíte před použitím přiřadit hodnotu odpovídající datovému typu dané proměnné. Můžete tak učinit již při deklaraci či kdykoli později prostým přiřazením.	`int cislo=4; //přirazení součástí deklarace`
	`int pocitadlo; //deklarace`
	`pocitadlo=10; //přiřazení`

V této kapitole byly zmíněny či použity následující datové typy pro celá čísla (vpravo jejich nejvyšší možné kladné hodnoty):
`short`
`int`
`long`
	`short maleCislo=32767;`
	`int celkeCislo=2147483647;`
	`long nej=9223372036854775807;`

Pro textové řetězce slouží typ `string`. Při nastavování hodnoty proměnné tohoto typu je třeba použít dvojí uvozovky.	Deklarace a přiřazení proměnné typu `string`: `string text="test";`

Pro výpis textu na obrazovku v konzolové aplikaci lze použít metodu `Console.WriteLine`. Jako parametr v závorce předejte přímo text (zde ale nezapomeňte na dvojité uvozovky) nebo proměnnou typu `string`.	`Console.WriteLine("Ahoj");`
	`//vypsání proměnné`
	`string pozdrav="Nazdar";`
	`Console.WriteLine(pozdrav);`

Pro získání textového vstupu zadaného uživatelem lze zavolat `Console.ReadLine`. Tato metoda vrací typ `string`, a tak ji můžete přímo použít při porovnávání textového řetězce.	`string Vstup=Console.ReadLine();`
	`//použití volání v podmínce`
	`if (Console.ReadLine()!="ano")`

Probraná látka	Příklad

Příkazem `if` můžete na základě splnění podmínek rozhodovat o tom, jaké příkazy či celé bloky kódu spustit.
Pomocí klauzule `else` lze vnořit další příkaz `if`, který je spuštěn v případě nepravdivosti předchozího výrazu. Klauzule `else` je pak spuštěna vždy, když žádný výraz nebyl pravdivý.
	`int A=3;`
	`string Zprava;`
	`if (A==1)`
	`Zprava="A je 1";`
	`else if (A==2)`
	`Zprava="A je 2";`
	`else`
	`Zprava="A není 1 ani 2";`

V podmínkách můžete používat operátory rovnosti, relační i logické, pokud to umožňují datové typy porovnávaných proměnných. Přehled operátorů je uveden v tabulkách níže.	Proměnná A i B je pro tuto ukázku typu `int`, proměnná C typu `string`.
	A je rovno B **nebo** B je větší než 10:
	`if (A==B || B>10)`
	A je menší než B **a zároveň** C je rovno 0:
	`if (A<B && C==0)`
	Hodnota C se nerovná „Ahoj“
	`if (C !="Ahoj")`

Příkaz `goto` umožňuje kontrolovat tok programu pomocí značek (návěští), na které poté odkazuje. Zpracovaní programu se tak přesune na toto místo.
V příkladu vpravo by šlo o nekonečnou smyčku.
	`Zapis:  //návěští`
	`Console.WriteLine("Dokola...");`
	`goto Zapis;`

Pro kratší a snazší zápis lze použít operátory složeného přiřazení. Při jejich zápisu jednoduše zkombinujte příšlušný (aritmetický) operátor s operátorem přiřazení.	`A+=5;  //A=A + 5`
	`A-=5;  //A=A - 5`
	`A*=5;  //A=A * 5`
	`A/=5;  //A=A / 5`
	`A%=5;  //A=A % 5`

Tabulka 1.1 Operátory relační a rovnosti

Operátor	Význam
==	Je rovno
!=	Není rovno
<	Menší než
<=	Menší nebo rovno
>	Větší než
>=	Větší nebo rovno

Tabulka 1.2 Logické operátory

Operátor	Význam
&&	A zároveň
||	Nebo

KAPITOLA 2

Hádání slov („šibenice“)

Jakou hru budete tvořit

Obsahem této kapitoly je vytvoření hry, ve které se hráč snaží uhodnout slovo pomocí hádání jednotlivých písmen během omezeného počtu pokusů. U nás je taková hra, klasicky hraná s papírem a tužkou, známá jako Šibenice, ovšem v textové variantě samozřejmě žádné kreslení oběšence nebude.

Kromě základních principů řešení daných problémů byste po dokončení této kapitoly měli:

- Znát a umět použít proměnné typu `bool` a `char`
- Vědět, co je pole a jak je vytvořit a také s ním umět pracovat
- Dobře ovládat práci s cyklem `for`
- Umět použít cyklus `while`
- Vědět, co je postfixový inkrement a dekrement

Před návrhem aplikace je třeba ještě zmínit, že v zájmu zachování vyšší přehlednosti a jednoduchosti budete programovat hru ve dvou fázích. V první fázi bude hráč hádat pouze jedno předem zadané slovo, aby nebylo použití vnořených cyklů příliš matoucí. Ve druhé fázi pak aplikaci upravíte – především zajistíte, že hra bude obsahovat více slov k hádání, aby bylo možné hrát ji delší dobu.

Nyní je třeba navrhnout funkční stránku první fáze hry. Stejně jako v předchozím příkladu půjde i zde o konzolovou aplikaci, jejíž specifika již znáte, a proto zde nebudou dále rozebírána.

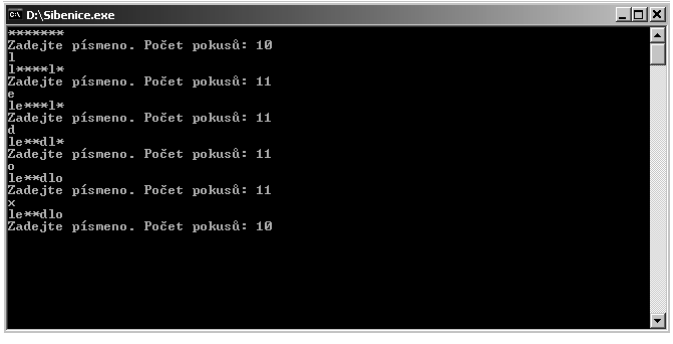
Po startu vybere program slovo (to bude zatím pevně zadané), které má hráč pomocí tipování písmen uhodnout, a zobrazí jej zamaskované – jednoduše nahradí každé písmeno v daném slově určitým znakem, například „*“. Uživatel tak bude zpočátku pouze vědět, kolik písmen hádané slovo obsahuje. A jak to v této hře bývá, hádáním správných písmen dojde k postupnému odkrytí celého slova.

Při každém tahu program zobrazí zprávu, ve které požádá uživatele, aby zadal tipované písmeno, a zároveň vypíše počet zbývajících pokusů. Když tak uživatel učiní a písmeno se ve slově skutečně vyskytuje, systém jej při dalším výpisu hádaného slova zobrazí – tedy nebude již nadále místo něj zobrazovat znak „*“. Za každé správně uhodnuté písmeno také přičte hráči další pokus.

Pokud však uživatel tipuje takové písmeno, které se ve slově nevyskytuje, je mu odečten pokus a v případě, že mu ještě nějaké pokusy zbývají, jej hra vyzve k opětovnému zadání písmene. Pokud má ale uživatel již všechny pokusy vyčerpány, dojde k vypsání zprávy o prohře a ukončení hry.

K opačné situaci, tedy výhře uživatele, dojde samozřejmě v případě, kdy jsou všechna písmena odkryta a počet zbývajících pokusů je vyšší než 0.

Když máte nyní představu o tom, jak bude hra asi fungovat, můžete rovnou založit konzolovou aplikaci pojmenovanou nejlépe „Sibenice“. I v tomto případě ještě budete veškerý kód hry psát do těla metody `Main`.



Obrázek 2.1 Hádání ukrytého slova

Proměnné pro počítadla a hádané slovo

Zamyslete se nyní nad návrhem hry, který je popsán výše. Jaké proměnné budete asi potřebovat? Začněte třeba tou hlavní, tedy slovem, které má hráč uhodnout. Slovo jako takové je obyčejný textový řetězec, takže příslušná proměnná musí být typu `string`. Deklarujte tedy tuto proměnnou s výstižným názvem (např. `hadaneSlovo`) a rovnou ji inicializujte na hodnotu, která tak bude představovat hádané slovo. Jak vidíte níže, v příkladu bude uživatel hádat slovo „automobil“.