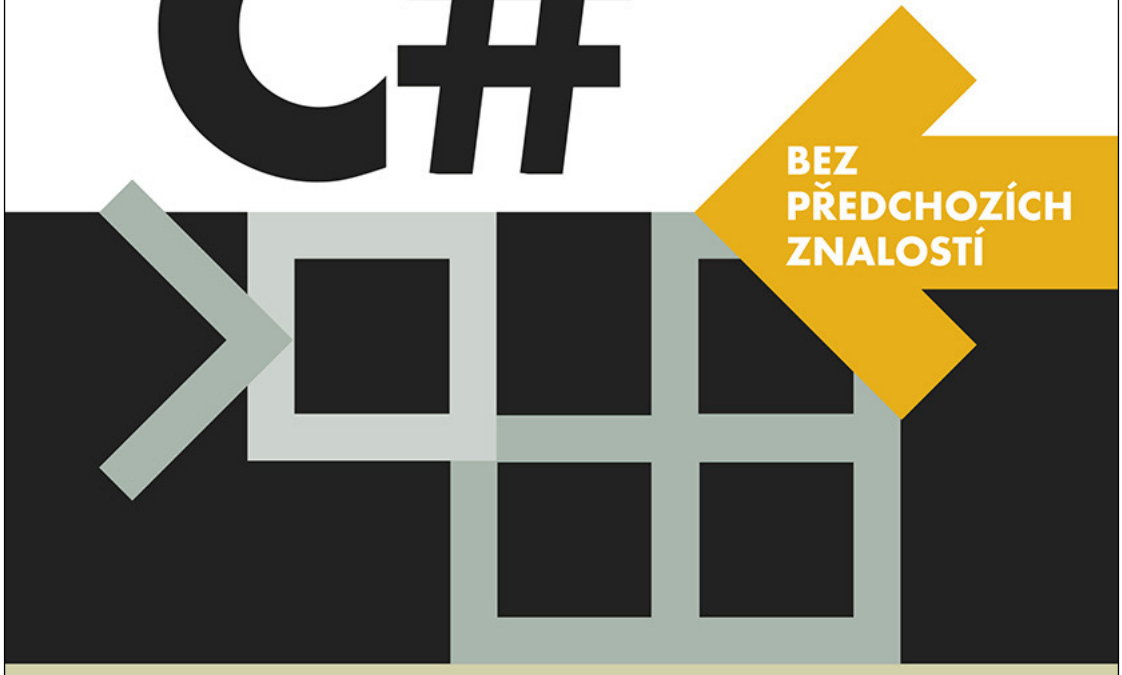
Pavel Bory

C#



BEZ PŘEDCHOZÍCH ZNALOSTÍ

Datové typy, operátory, řídicí struktury

Základy objektově orientovaného programování

Pole, řetězce, práce se soubory

Ladění programu, zpracování výjimek

computer press

C# bez předchozích znalostí

Vyšlo také v tištěné verzi

Objednat můžete na
www.cpress.cz
www.albatrosmedia.cz

Pavel Bory
C# bez předchozích znalostí – e-kniha

Obsah

O autorovi

Pavel Bory je seniorním vývojářem a řadu let pracuje hlavně na projektech společnosti Unicorn, a.s., především v oblasti financí. Kromě samotného vývoje softwaru se věnuje školení a výuce programování, zejména nad platformou .NET na vysoké škole Unicorn College, s.r.o. Tato kniha je jeho první monografií.

Poděkování

Rád bych poděkoval všem blízkým, bez jejichž podpory by tato kniha nevznikla. Dále bych rád poděkoval všem spolupracovníkům, od kterých jsem měl možnost se učit, a v neposlední řadě chci poděkovat svým učitelům, z nichž jsou mi někteří dodnes vzorem. Musím poděkovat i redaktorovi nakladatelství Computer Press panu Martinovi Herodkovi za jeho profesionální přístup a podporu během psaní této knihy.

Úvod

S pomocí této knihy se můžete sami naučit základům programovacího jazyka C# a objektově orientovaného programování. Kniha je koncipována tak, že je probíraná látka hojně ilustrována na příkladech, ke kterým jsou uvedeny jejich zdrojové kódy a výstupy programů. Zdrojové kódy jsou pro lepší srozumitelnost doprovázeny komentáři a u složitějších příkladů je zdrojový kód tvořen postupně a průběžně je jeho tvorba rozebírána. Na závěr každé kapitoly je uvedena sada kontrolních otázek pro ověření získaných znalostí a zároveň jsou zde uvedeny řešené příklady. Kromě kontrolních otázek a řešených příkladů jsou na závěr každé kapitoly uvedeny příklady k samostatnému procvičení.

Komu je tato kniha určena

Tato kniha nepředpokládá žádné znalosti programování. Je tedy určena každému, kdo má zájem se naučit základy programování v moderním objektově orientovaném jazyce C#. Kniha může posloužit jak čtenáři, který chce získat pouze základní přehled o programování, tak i čtenáři, který se chce programování věnovat dlouhodobě a hledá vhodný výchozí bod pro začátek studia.

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:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

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

Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/K2202* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih nějakou najdete, ať už v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2202* po klepnutí na odkaz Soubory ke stažení. (Nejsou-li žádná errata zatím k dispozici, není odkaz Soubory ke stažení dostupný.)

KAPITOLA 1

První program

V této kapitole:

- Instalace potřebného softwaru
- Náš první program v jazyce C#
- Co je to vlastně program a programovací jazyk?
- Základy práce s Microsoft Visual Studiem
- Kontrolní otázky
- Cvičení

V této kapitole se dozvíte, jaký software budeme v rámci studia programování pomocí této knihy používat, jak jej získat, nainstalovat a jak s ním pracovat. Dozvíte se, co je to program, programovací jazyk a vytvoříte si svůj první program v jazyce C#.

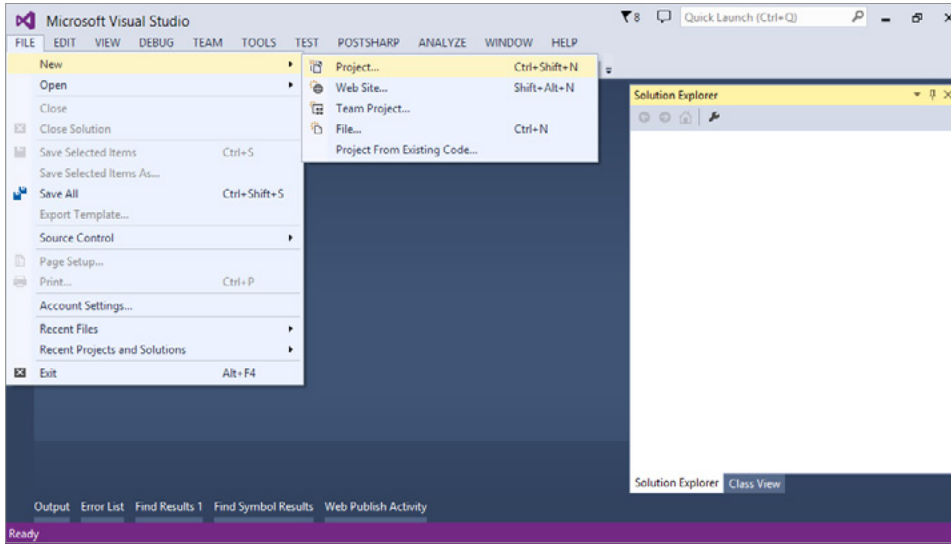
Instalace potřebného softwaru

Než začnete se studiem programování pomocí této knihy, je nezbytné si nainstalovat vývojové prostředí Microsoft Visual Studio, pomocí kterého budeme naše programy vytvářet. V této knize budeme používat verzi Microsoft Visual Studio 2015 Community Edition, která je k dispozici zdarma ke stažení zde: *https://www.visualstudio.com/products/visual-studio-community-vs*.

Vývojové prostředí nainstalujte podle pokynů uvedených na webových stránkách a v instalačním průvodci vývojového prostředí Microsoft Visual Studio.

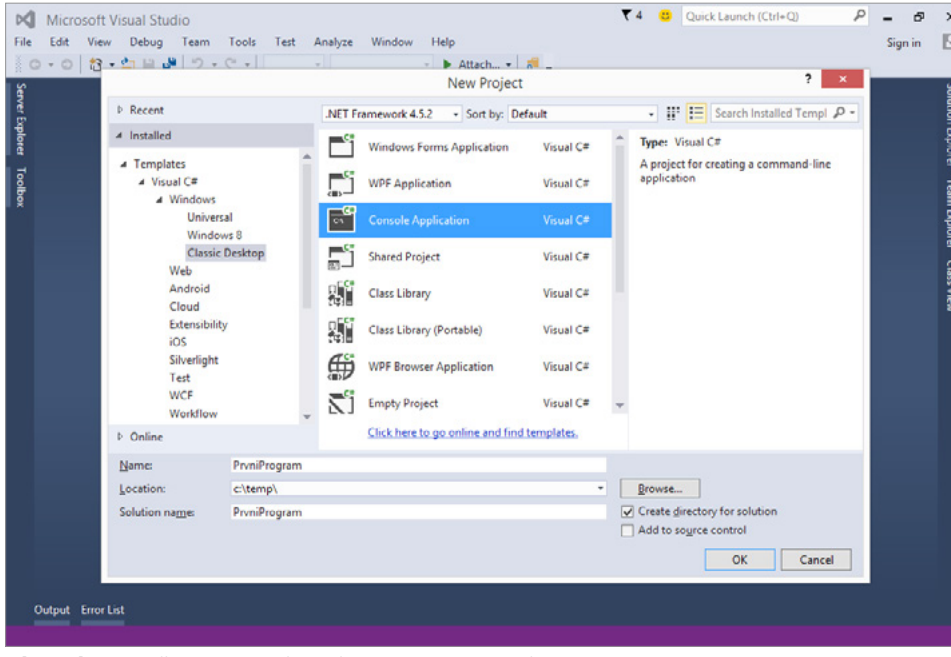
Náš první program v jazyce C#

Pokud máte nainstalované vývojové prostředí Microsoft Visual Studio, se můžete pustit do vytvoření svého prvního programu v jazyce C#. Spusťte vývojové prostředí Microsoft Visual Studio. Zobrazí se vám úvodní obrazovka, na které vyberte z hlavního menu možnost **File → New → Project**.



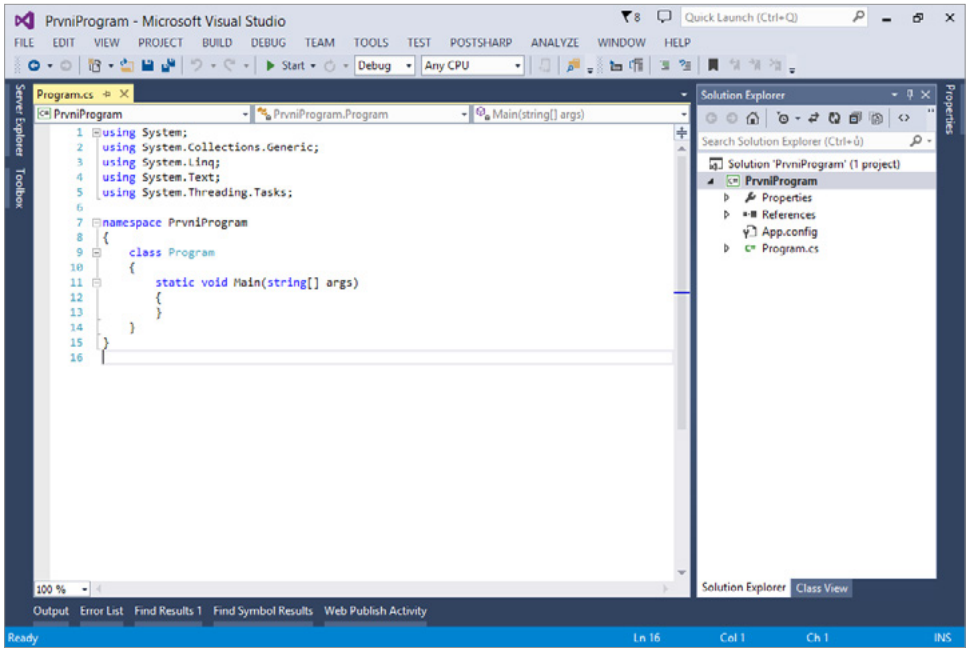
Obrázek 1.1 Vytvoření nového projektu

Zobrazí se poměrně pestrá nabídka, protože Microsoft Visual Studio umožňuje vytvářet celou řadu různých typů programů. Vyberte projekt typu **Visual C# → Windows → Classic Desktop → Console Application**. Do pole **Name** vyplňte název projektu `PrvniProgram`. V poli **Location** vyberte umístění, kde na disku bude uložen vytvořený projekt.



Obrázek 1.2 Volba typu projektu, jeho pojmenování a výběr umístění

Jakmile budete mít vyplněno, stiskněte tlačítko **OK** a Microsoft Visual Studio vytvoří nový projekt.



Obrázek 1.3 Výchozí stav po vytvoření nového projektu

Možná jste očekávali, že bude projekt zcela prázdný, a on přitom obsahuje pro nás zatím neznámý zdrojový kód a několik souborů. To je dáno tím, že na základě vybraného typu projektu dojde k zjednodušeně řečeno vygenerování určitého výchozího stavu, který je pro daný typ programu typický. Zatím do onoho výchozího stavu projektu nezasahujme, ale nebojte se, postupně se při čtení této knihy s významem jednotlivých příkazů seznámíte.

Než se pustíme do vytvoření prvního programu, pojďme si vysvětlit dva pojmy: `Project` a `Solution`.

`Project` obsahuje zdrojové kódy programu, které jsou umístěny v jednom či více souborech. `Solution` může obsahovat jeden či více projektů.

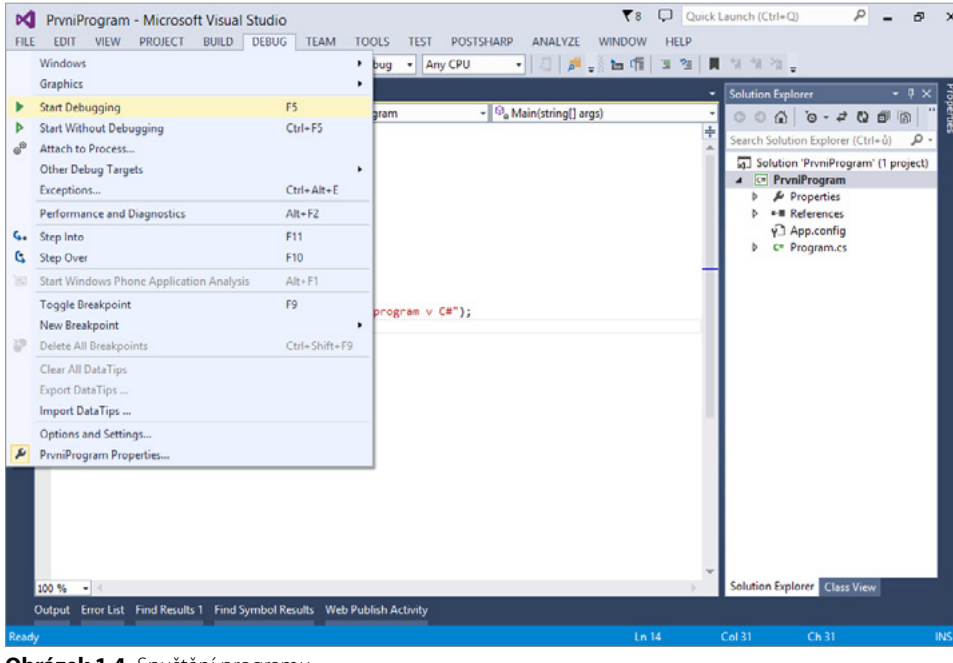
My jsme tedy vytvořili jednu `Solution`, která obsahuje právě jeden `Project`, ve kterém napíšete svůj první program v jazyce C#. V rámci ukázkových příkladů k této knize se můžete setkat s tím, že pro jednu kapitolu existuje jedna `Solution`, která obsahuje více projektů, přičemž jednotlivé projekty obsahují jednotlivé ukázkové programy. Pojďte vytvořit svůj první program v jazyce C#! Doplňte následující dva řádky do souboru `Program.cs`,

```
Console.WriteLine("Muj prvni program v C#");
Console.ReadKey();
```

tak aby přesně odpovídal následujícímu zápisu:

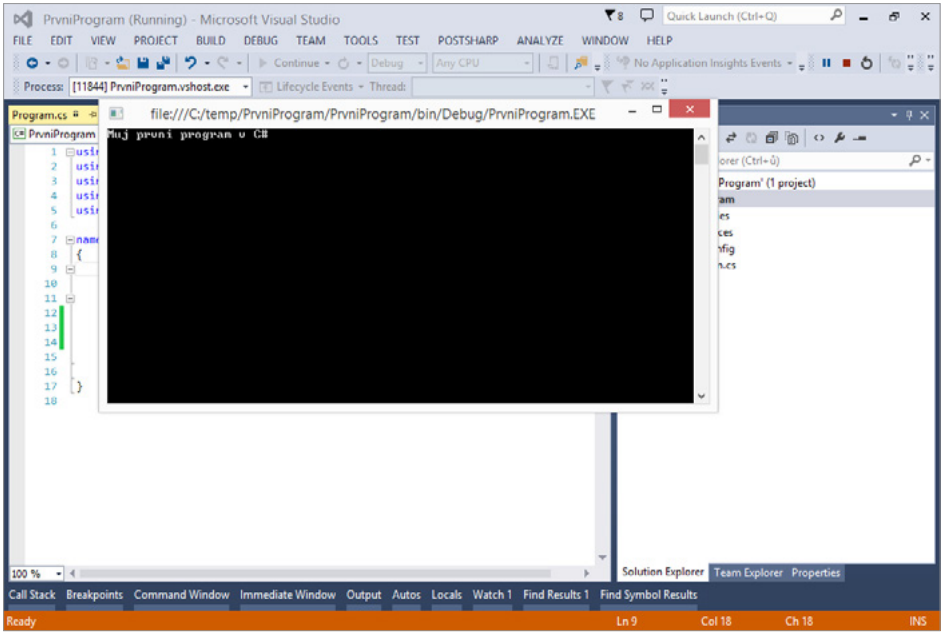
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PrvniProgram
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Muj prvni program v C#");
      Console.ReadKey();
    }
  }
}
```

Nyní program spusťte pomocí možnosti **Start Debugging**, která se nachází v menu **Debug**.



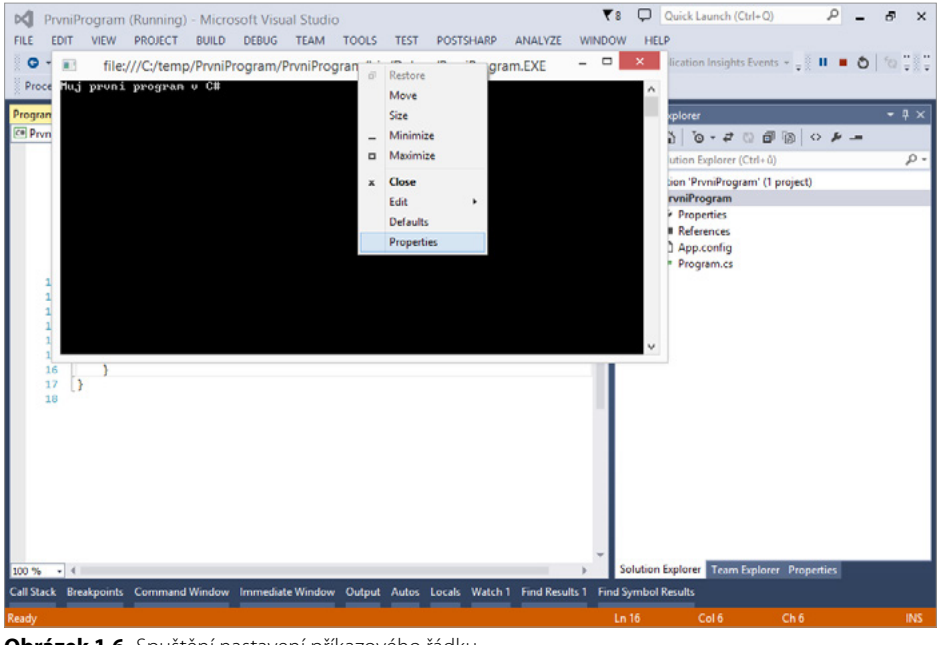
Obrázek 1.4 Spuštění programu

Zobrazí se vám příkazový řádek, ve kterém program vypíše zadaný text, který jste vložili do souboru `Program.cs`.



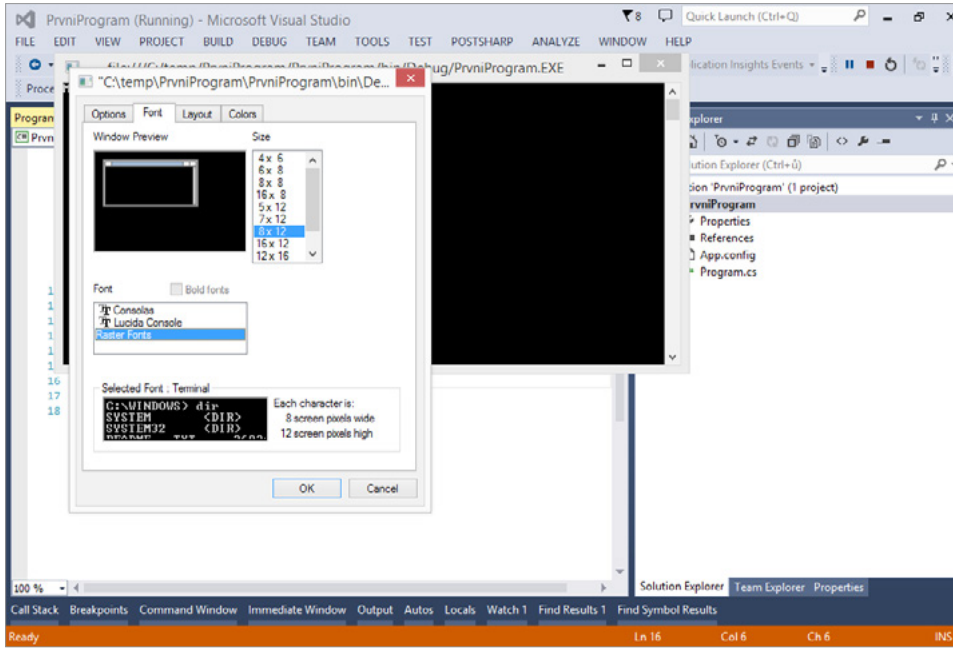
Obrázek 1.5 Příkazový řádek s výstupem vašeho prvního programu

Je možné, že se vám bude velikost písmen v příkazovém řádku zdát příliš malá. Proto si nyní ukažme jak ji zvětšit. Klikněte pravým tlačítkem do vrchní části rámu příkazového řádku.



Obrázek 1.6 Spuštění nastavení příkazového řádku

Z kontextové nabídky vyberte možnost **Properties**. Zobrazí se vám následující obrazovka s nastavením příkazového řádku.



Obrázek 1.7 Nastavení fontu příkazového řádku

Zde můžete změnit velikost písma v příkazovém řádku. Pokud chcete, prozkoumejte i jiné možnosti nastavení příkazového řádku a přizpůsobte si jeho zobrazení tak, aby se vám s ním dobře pracovalo.

Poznámka: Program můžeme spouštět jak pomocí **Debug → Start Debugging**, tak i pomocí stisknutí klávesy **F5**.

Co je to vlastně program a programovací jazyk?

Abychom si mohli vysvětlit, co je to vlastně program, je potřeba si nejprve definovat pojem algoritmus. Pro tento pojem neexistuje jednoznačná, dokonalá a všemi uznávaná definice. My si algoritmus definujme jako určitý návod, postup jak řešit určitý problém. Tento postup musí být zejména jednoznačný a nesmí umožňovat více výkladů. Příkladem neprogramátorského algoritmu může být recept v kuchařce. Ten přesně definuje postup přípravy pokrmu a měl by být jednoznačný a neumožňovat více způsobů výkladu.

Když jsme si přiblížili pojem algoritmus, můžeme si počítačový program popsat jako zápis algoritmu ve zvoleném jazyce, který je srozumitelný počítači. Tyto jazyky se nazývají programovací jazyky a slouží právě k dorozumění mezi člověkem (programátorem) a počítačem. A vy se pro dorozumění s počítačem naučíte právě programovací jazyk C#.

Ve spojitosti s programovacím jazykem C# se často setkáte s pojmem *.NET Framework*. .NET Framework je softwarová platforma poskytující širokou škálu prostředků pro programy. Její popis by zcela jistě vydal na celou řadu knih a není cílem této knihy ji detailně popisovat. Prostředky z této platformy budeme při programování v jazyce C# používat, a dokonce jste je již použili, pokud jste si vytvořili předchozí program. Použili jsme její prostředky např. pro výpis na konzoli. Kromě programovacího jazyka C# se tedy seznámíte i s řadou základních prostředků platformy .NET.

Řekli jsme si, že programovací jazyk je jazyk srozumitelný počítači. K tomuto je potřeba ještě vysvětlit, že než program spustíte, je potřeba spustit tzv. **Build**. Ten zajistí ověření, že program neobsahuje formální chyby, a vytvoří spustitelný soubor. Možná si teď říkáte, že když jsme spouštěli náš první program, **Build** jsme neprováděli. Tím, že jsme zvolili možnost **Start Debugging**, se provedly v podstatě dva kroky zároveň, a to **Build** a následné spuštění programu.

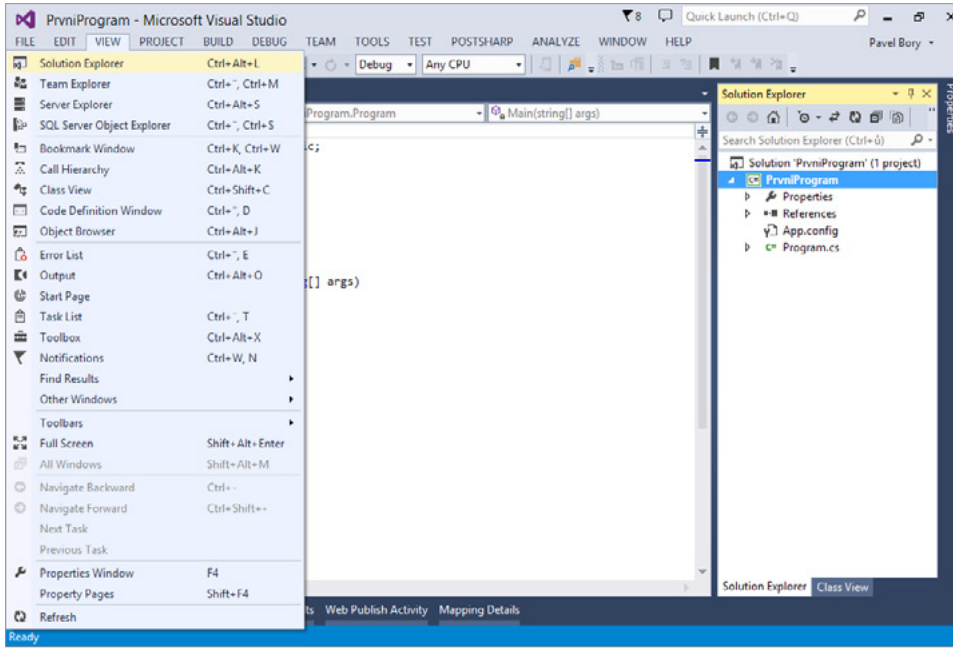
Ve skutečnosti je tato problematika podstatně komplikovanější, ale cílem této knihy není zabíhat do přílišných detailů. Naším cílem je, abyste se naučili základy programování a byli schopni na těchto zejména praktických základech stavět.

Základy práce s Microsoft Visual Studiem

V této kapitole se seznámíte se základy ovládání Microsoft Visual Studia, abyste mohli pohodlně pracovat s touto knihou, zkoušet si ukázkové programy, a zejména tvořit programy vlastní. Většina prvků Microsoft Visual Studia, které budeme používat, je reprezentována okny, která lze zavírat, přesouvat a minimalizovat. Zkrátka s nimi lze pracovat, jako jste zvyklí pracovat s okny v prostředí Microsoft Windows.

Tip: Každý ovládací prvek (okno), který zde budeme popisovat, lze otevřít pomocí menu **View**.

Na tuto kapitolu budeme v průběhu studia v dalších kapitolách volně navazovat a znalosti práce s Visual Studiem postupně dále rozšiřovat. První, co si ukážeme, je postup, jak se orientovat v `Solution`, projektech a souborech těchto projektů. K tomuto slouží tzv. **Solution Explorer**.



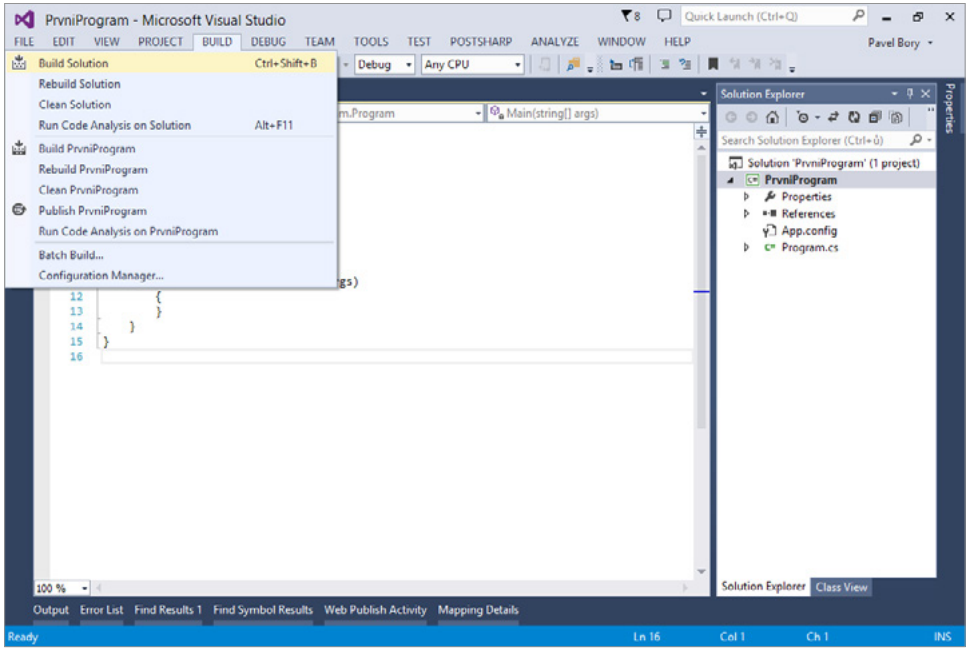
Obrázek 1.8 Solution Explorer

Toto okno vám ukáže, které projekty se v `Solution` nachází, a můžete zde otevírat jednotlivé soubory obsahující zdrojové kódy a ty editovat. Zkuste si v **Solution Explorer** otevřít soubor `Program.cs`, do kterého jsme napsali náš první program.

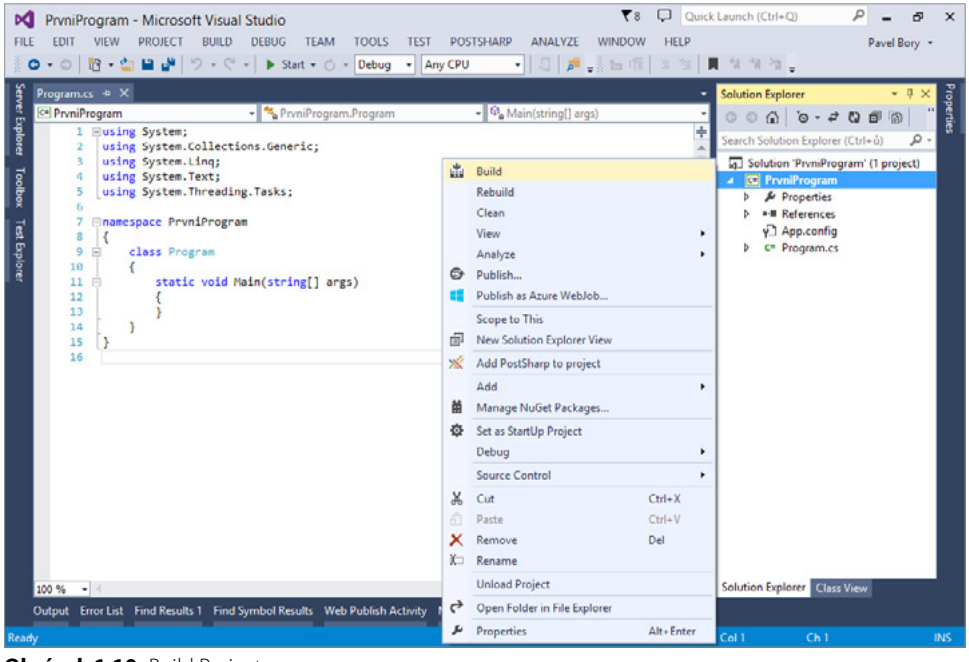
V předchozí kapitole jsme hovořili o tzv. **Build**, který vytvoří spustitelný soubor programu. Máme v základu dvě možnosti jak **Build** spustit. První možnost je použít **Build Solution** z menu **Build**.

Víme, že `Solution` může obsahovat více projektů. Z každého projektu obsaženého v `Solution` se vytvoří jeden spustitelný soubor s příponou `.exe` a několik dalších souborů, které nejsou nutně zapotřebí pro samotné spuštění programu, a nebudeme se jimi proto v této knize zabývat. Pokud máte vytvořenu `Solution` obsahující jeden projekt z kapitoly Náš první program v jazyce C# v umístění `C:\temp\PrvniProgram`, naleznete spustitelný `.exe` soubor programu zde: `C:\temp\PrvniProgram\bin\Debug\PrvniProgram.exe`. Zkuste si tento soubor najít a program spustit. Měla by se vám zobrazit konzole, na kterou se vypíše text stejně, jako když jsme program spouštěli pomocí **Debug → Start Debugging** přímo z Microsoft Visual Studia.

Další možností je provést **Build** nad konkrétním projektem. Můžeme jej spustit buď z menu **Build** pomocí příkazu **Build PrvniProgram** (nebo jiný název projektu), nebo z kontextového menu nad projektem v **Solution Explorer**. Klikněte pravým tlačítkem na název projektu v **Solution Explorer** a vyberte možnost **Build**.



Obrázek 1.9 Build Solution

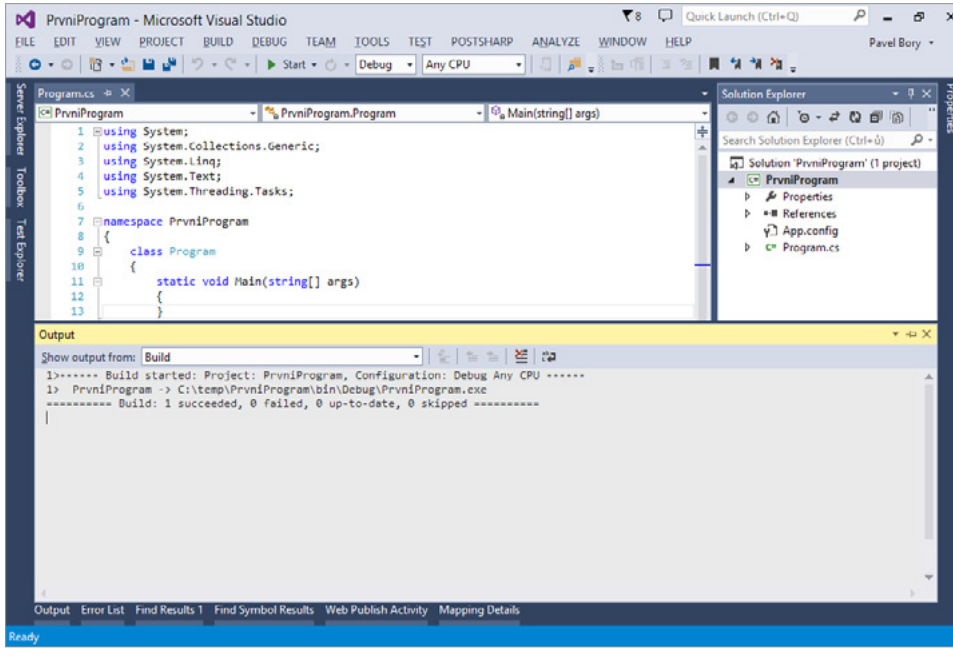


Obrázek 1.10 Build Project

Příkazem **Build** jsme vždy pouze vytvořili spustitelný soubor programu.

Při studiu a samostatném vytváření programů budete nejčastěji potřebovat programy rovnou spouštět z vývojového prostředí. Za tímto účelem můžete použít volbu **Debug → Start Debugging**, stejně jako když jsme spouštěli náš první program.

Někdy se může stát, že v programu uděláte chybu, která zamezí jeho spuštění. Např. se pokusíte použít neexistující příkaz, zapomenete někde napsat závorku apod. V tomto případě se spustitelný soubor programu nevytvoří, program se nespustí a Microsoft Visual Studio bude hlásit chybu. Otevřeme si další okno v nabídky **View → Output**, ve kterém budeme zpravidla sledovat, jak dopadl příkaz **Build**.



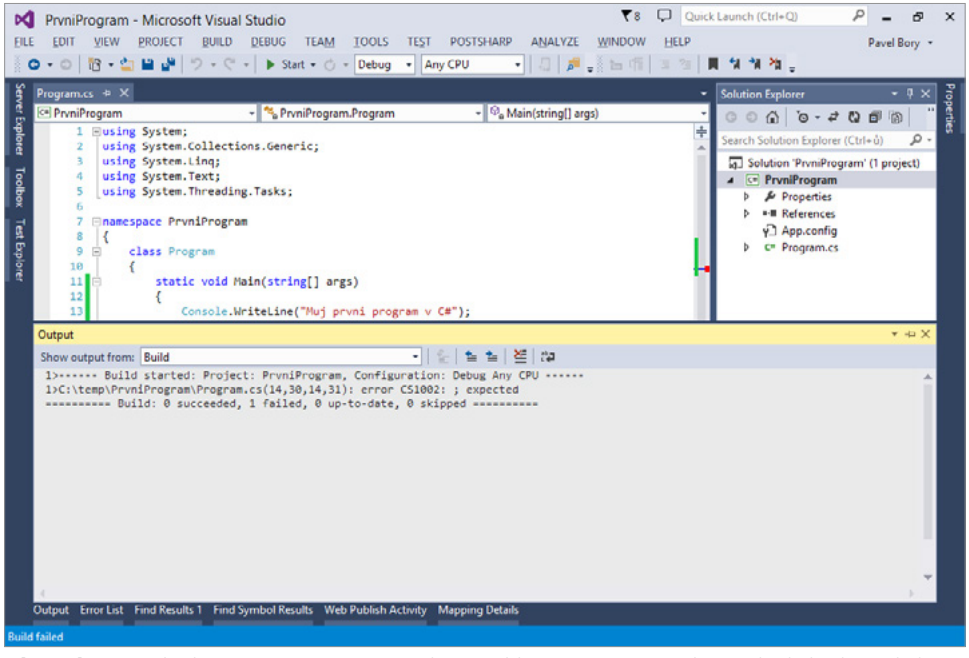
Obrázek 1.11 Ukázka výstupu po spuštění příkazu Build Project

V okně **Output** se po spuštění příkazu **Build** dozvíme, pro kolik projektů dopadl **Build** úspěšně (succeeded), pro kolik neúspěšně (failed) a pro kolik se nespustil, protože již byl pro ně spuštěn a nebyla v nich provedena žádná úprava (up-to-date). Zkusme se záměrně dopustit chyby v programu. Odstraňte např. středník za příkazem `Console.ReadKey()`. Záměrně chybně zapsaný program by mohl vypadat takto.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

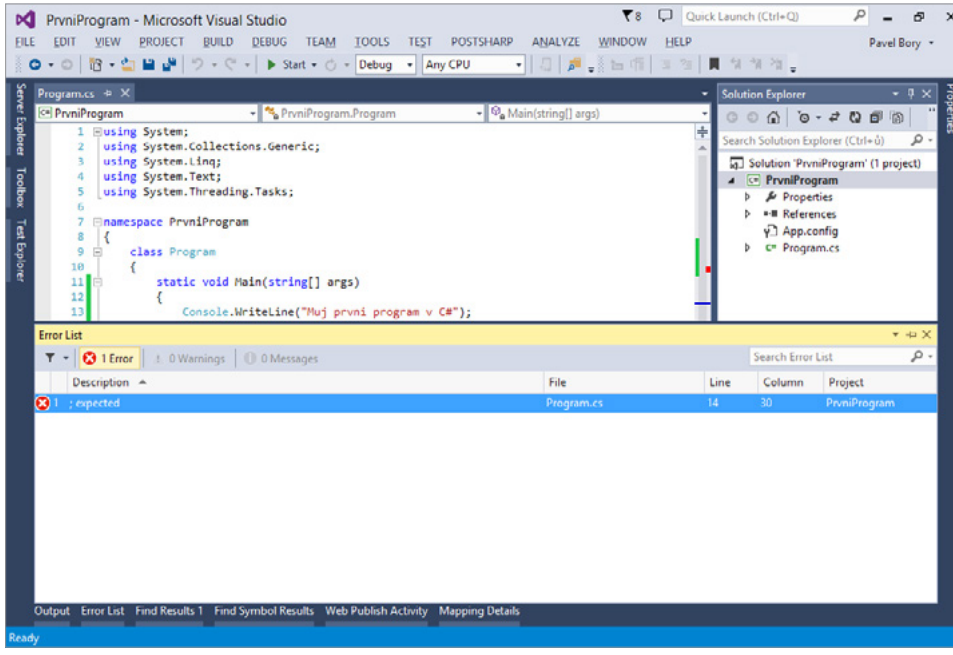
```
namespace PrvniProgram
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Muj prvni program v C#");
      Console.ReadKey()
    }
  }
}
```

Spusťte příkaz **Build** pro projekt a podívejte se do okna **Output**, ve kterém se dozvíme, že byl pro projekt neúspěšný.



Obrázek 1.12 Ukázka výstupu po spuštění příkazu Build Project, přičemž zdrojový kód obsahuje chybu

Pokud se v programu vyskytuje chyba bránící jeho spuštění, je pro nás podstatně zajímavější okno **Error List**, které je opět možné spustit z nabídky **View**. V tomto okně se dozvíte o tom, kde se v programu vyskytují chyby.



Obrázek 1.13 Error List

Je zde uveden tabulkový výpis, kde každý řádek popisuje jednu chybu. Pro každou chybu vás bude v tomto výpisu především zajímat její popis, soubor, ve kterém se vyskytuje, a řádek, na kterém se vyskytuje. Dvojklikem na řádek s chybou vás Visual Studio nasměruje na umístění příslušné chyby. Je zapotřebí říci, že v některých případech nemusí být na první pohled zřejmé, v čem chyba spočívá. Nicméně postupně v tomto ohledu získáte zkušenosti a odhalení příčin chyb pro vás bude jednodušší. V rámci ukázek programů z této knihy máte vždy možnost si otevřít přiložené ukázkové projekty, porovnat je s vlastním řešením, nalézt odlišnosti a odvodit, v čem chyba spočívá. Při vašem studiu může být rovněž přínosné nalézt řešení obdobného problému, který budete řešit, a inspirovat se z něj nebo poznat, v čem jste se dopustili chyby.

Kontrolní otázky

1. K čemu slouží vývojové prostředí Microsoft Visual Studio?
2. Co je to `Project` a `Solution` a jaký je mezi nimi vztah?
3. Co je to program a programovací jazyk?
4. Jakým příkazem vytvoříme spustitelný soubor z programu, aniž bychom jej rovnou spustili? Kde tento soubor naleznete?
5. Jakým příkazem spustíme program přímo z Microsoft Visual Studia?

6. Pokud se program nepodaří kvůli chybám spustit, kde nalezneme informace o výskytu chyb?

Cvičení

1. Vyzkoušejte si založení nové `Solution` a `Project`. Jakmile budete mít projekt vytvořen, zkuste si doplnit kód v souboru `Program.cs`, aby pozdravil uživatele slovy „Ahoj clovece“ obdobně, jako jste učinili ve vašem prvním programu v jazyce C#.
2. Zkuste si záměrně zanést do programu chybu, např. tím že odstraníte středník nebo závorky apod. Následně zkuste program spustit a pomocí **Error List** nalézt informace o chybě a jejím umístění.

KAPITOLA 2

Datové typy a proměnné

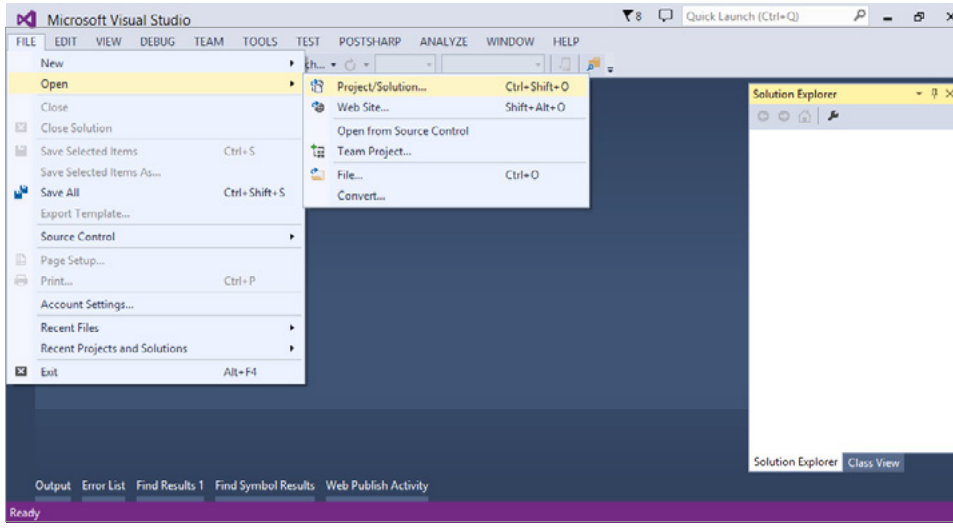
V této kapitole:

- Otevření existujícího projektu
- Základní struktura programu
- Komentáře zdrojového kódu
- Proměnné
- Datové typy
- Konstanty
- Základní operace s čísly
- Výpis na příkazový řádek
- Načtení vstupu od uživatele
- Konverze řetězce na číslo
- Konverze čísla na řetězec
- Intelisense ve Visual Studiu
- Kontrolní otázky
- Řešená cvičení
- Cvičení

V této kapitole si rozšíříte své znalosti práce s vývojovým prostředím Microsoft Visual Studio. Naučíte se, jak otevřít ukázkové příklady, které doprovázejí tuto knihu. Poté se naučíte, jak si v programu uchovávat data, se kterými program může pracovat. Nemalé množství programů potřebuje pro svou činnost komunikovat s jejich uživatelem, proto se dozvíte, jak může program uživateli zobrazovat textové zprávy v příkazovém řádku, a zároveň se naučíte, jak do programu předat data, která uživatel do příkazového řádku zapíše.

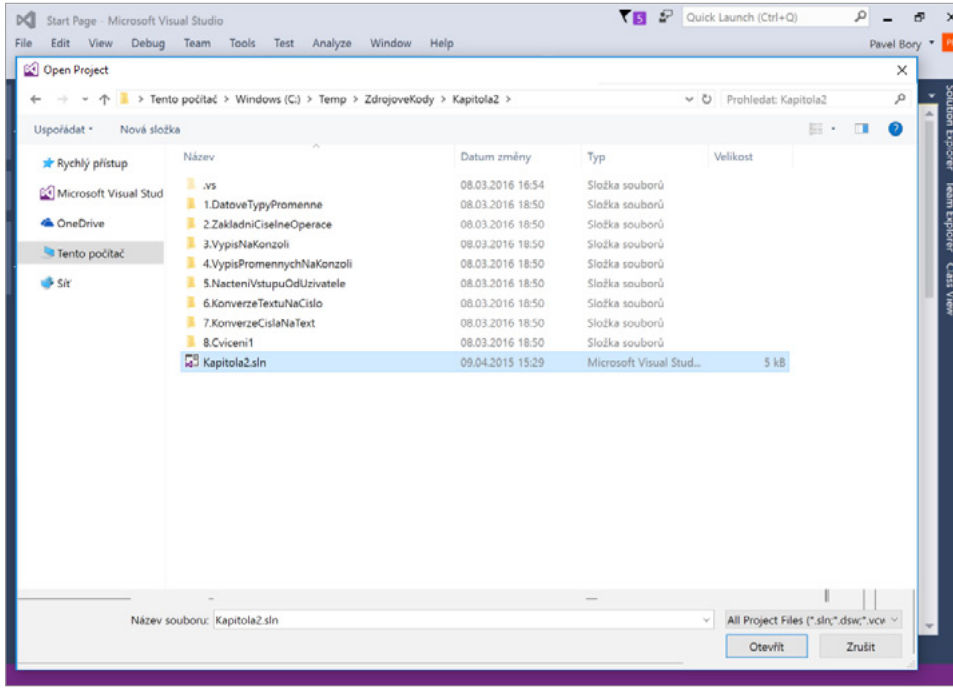
Otevření existujícího projektu

Nyní i v následujících kapitolách budeme při výkladu vycházet z ukázkových projektů. Proto se naučíte, jak nějaký již existující projekt otevřít. Spusťte Visual Studio. Zobrazí se vám úvodní obrazovka, na které vyberte z hlavního menu možnost **File → Open → Project/Solution**.



Obrázek 2.1 Otevření existujícího Project/Solution – krok 1

Zobrazí se vám standardní okno pro výběr souboru. Pomocí něj najděte složku, ve které máte zdrojové kódy k této knize, a otevřete složku `Kapitola2`. V této složce vyberte soubor `Kapitola2.sln` a stiskněte tlačítko **Open**.

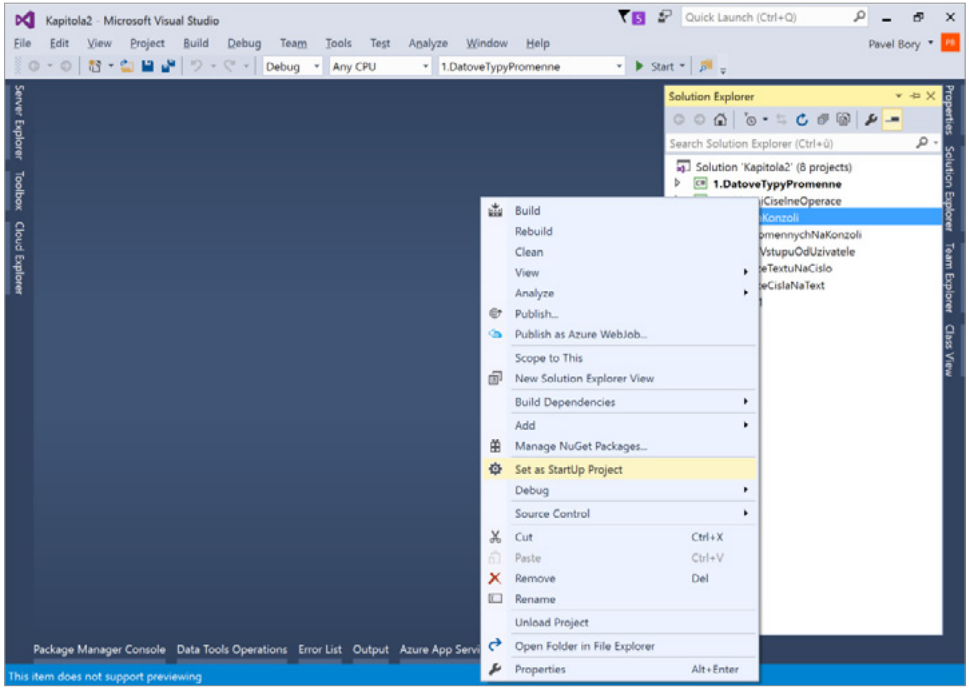


Obrázek 2.2 Otevření existujícího Project/Solution – krok 2

Poznámka: Pro připomenutí si zopakujme, že `Solution` v sobě může obsahovat více projektů, přičemž každý projekt obsahuje program, který může být umístěn v jednom či více souborech.

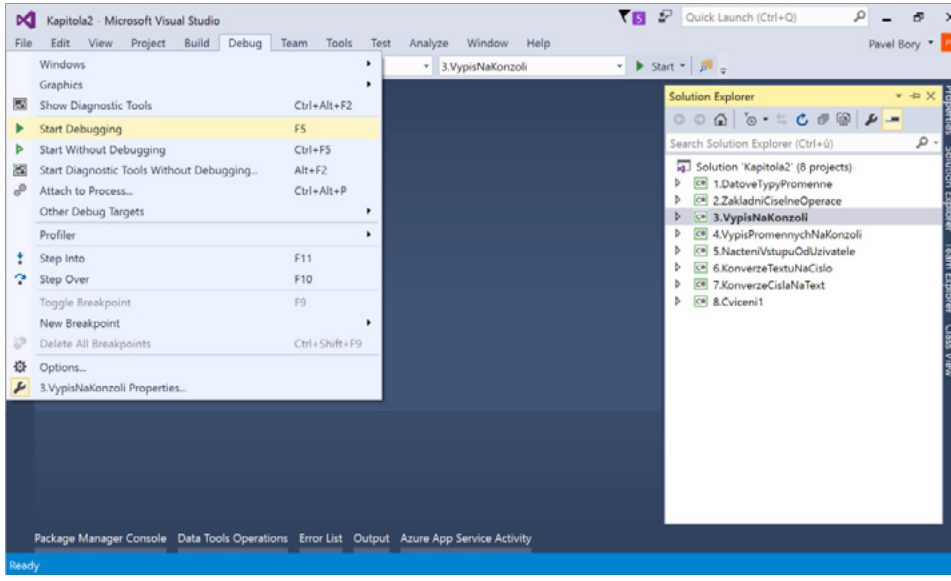
Nyní byste měli mít otevřenou `Solution` pro tuto kapitolu. My zatím pracujeme s tím, že program vytváříme v rámci souboru `Program.cs`. Všimněte si, že `Solution` opravdu obsahuje více projektů, přičemž každý z nich představuje ukázku jedné či více oblastí probíraných v této kapitole.

Musíte se naučit, jak ve vývojovém prostředí nastavíte, který projekt (program) se má spustit. Klikněte pravým tlačítkem na projekt s názvem `3.VypisNaKonzoli` a z kontextového menu vyberte možnost **Set as StartUp Project**. Zkontrolujte, že se nyní název projektu oproti jiným projektům zobrazuje tučným písmem.



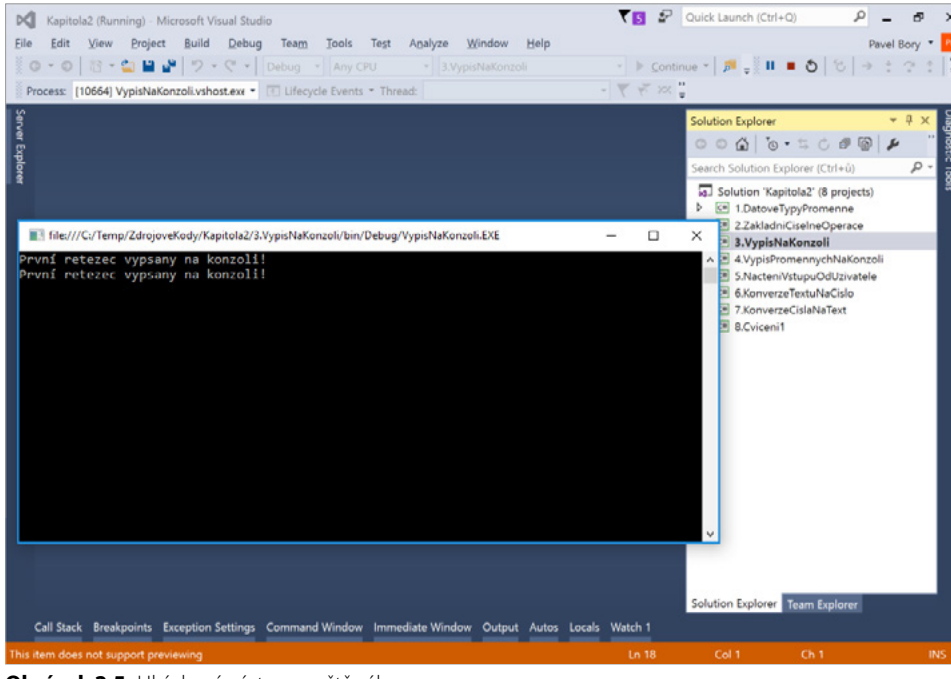
Obrázek 2.3 Nastavení projektu (programu), který se má spustit při provedení příkazu Start Debugging

Nyní máte projekt nastaven jako výchozí projekt pro spuštění. Zkuste program v tomto projektu spustit stejně, jako jste spouštěli svůj první program v jazyce C#. To znamená: klikněte na možnost **Start Debugging**, která se nachází v menu **Debug**.



Obrázek 2.4 Spuštění nastaveného projektu

Měl by se vám zobrazit příkazový řádek, na který program vypíše text jako na následujícím obrázku.



Obrázek 2.5 Ukázkový výstup spuštěného programu

Nyní umíte pracovat se vzorovými programy, které budete využívat při čtení této knihy. Většina kapitol v sobě zahrnuje ukázkové programy a je vhodné, abyste si je při studiu programování prostudovali. V jednotlivých kapitolách jsou zpravidla uvedeny pouze klíčové výseky ze zdrojových kódů, a je proto žádoucí, abyste mimo samotnou knihu viděli kompletní zdrojové kódy programů a zkoušeli si je upravovat. Poslouží vám jako inspirace pro řešení obdobných problémů.

Základní struktura programu

Ve chvíli, kdy jsme vytvářeli náš první program, nebyl po založení projektu soubor `Program.cs` prázdný. Obsahoval již předpřipravený zdrojový kód. Nebudeme si nyní vysvětlovat, co jednotlivé příkazy znamenají, protože vysvětlení by v tuto chvíli bylo značně komplikované a nemělo by pro vás zásadní přínos. Seznámíte se s nimi postupně v průběhu studia této knihy.

Upozornění: Výchozí kód v souboru Program.cs neupravujte s jedinou výjimkou: Na začátku programu se nachází několik příkazů začínajících slovem **`using`**. Ty můžete smazat až na příkaz **`using System;`**. Pokud ostatní příkazy **`using`** nesmažete, pak se také nic neděje. Zatím pro nás nemají žádný význam. Zdrojový kód programů, které budete tvořit, pište mezi složené závorky za **`Main(string[] args)`**, jak je naznačeno v následující ukázce.

```
using System;

namespace VypisNaKonzoli
{
    class Program
    {
        static void Main(string[] args)
        {
            // Zde pište program
        }
    }
}
```

Poslední velmi důležitou věcí je, že téměř všechny řádky programu je zapotřebí ukončovat středníkem. Výjimku tvoří komentáře, ty středníkem končit nemusí. Dále výjimku tvoří různé konstrukce jazyka C#, se kterými se seznámíte později. Ty zpravidla obsahují nějaký příkaz, za kterým následuje blok dalších příkazů uzavřený mezi složené závorky. V jednotlivých ukázkách si všímejte i toho, jak jsou jednotlivé příkazy odsazovány a že programy působí přehledným dojmem.

Komentáře zdrojového kódu

Pojďme si na úvod říci, že programy mohou obsahovat kromě příkazů, které má program vykonat, i tzv. komentáře. Smyslem komentářů je usnadnit programátorovi pochopení zdrojového kódu. Pro funkčnost programu nemají naprosto žádný význam.

Do komentářů je vhodné psát stručné a výstižné informace k jednotlivým částem kódu. Jednořádkové komentáře se zapíší tak, že na začátek řádku napíšeme dvě lomítka `//`, za která můžeme napsat naprosto libovolný text.

```
// Toto je jednořádkový komentář
```

Druhým typem jsou tzv. víceřádkové komentáře. Ty začínají symbolem `/*` a končí symbolem `*/`. Text napsaný mezi počáteční a koncový symbol je považován za komentář a pro funkčnost programu nemá žádný význam. Program jej zkrátka ignoruje.

```
/*
 * To je víceřádkový komentář
 * To je víceřádkový komentář
 * To je víceřádkový komentář
 */
```

V ukázkových programech dodaných k této knize se s komentáři setkáte poměrně často a měly by vám pomoci pochopit význam jednotlivých částí programu. Komentáře můžeme použít i tak, že pomocí nich označíme nějaký řádek zdrojového kódu, když nechceme, aby se vykonal, ale nechceme jej smazat. Takovýmto řádkům se mezi programátory často říká „zakomentované řádky". Až budete psát vlastní programy, rozhodně je používejte, abyste si usnadnili orientaci v programu (např. si pomocí nich dělali poznámky k novým příkazům, programovým konstrukcím apod.).

Proměnné

S pojmem proměnná se budete při programování velmi často setkávat a jeho pochopení je velmi důležité. Tento pojem je poměrně obtížně definovatelný a my si jeho vysvětlení pro účel této knihy značně zjednodušíme.

Jedno z možných míst, kde si mohou počítačové programy uchovávat data, je počítačová paměť. Tu si můžete představit jako velkou kartotéku, která se skládá z pojmenovaných šuplíků. V každém šuplíku může být uložena nějaká hodnota.

Proměnnou si představte jako jeden takovýto pojmenovaný šuplík. Proměnné tedy mají vždy svůj název a mohou obsahovat nějakou hodnotu. Při programování v jazyce C# musíte pro každou proměnnou kromě jejího názvu zapsat, jaký typ hodnoty bude obsahovat. Musíte tedy říct, zdali proměnná bude obsahovat celé číslo, desetinné číslo, text nebo nějaký jiný typ, o kterých se dozvíte více později.

Tímto se dostáváme k pojmu datový typ, který velmi úzce souvisí s proměnnými. Budeme si tedy pamatovat, že proměnná je vždy nějakého datového typu a má nějaké jméno. Pokud bychom chtěli v programu evidovat počet studentů, pravděpodobně bychom vytvořili proměnnou, která bude moci obsahovat celé číslo (určíme datový typ), a pojmenovali ji např. `pocetStudentu`. O datových typech, pravidlech a způsobech pojmenování proměnných se dozvíte v následující kapitole.

Datové typy

Datový typ určuje rozsah hodnot, kterých může proměnná nabývat. .NET framework obsahuje mnoho vestavěných datových typů. Vy se s nimi budete seznamovat postupně během čtení této knihy. Na začátek si představme následující často používané datové typy.

Tabulka 2.1 Datové typy

Datový typ	Popis	Rozsah
byte	Kladná celá čísla	0 .. 255
int	Záporná i kladná celá čísla	-2147483648 .. 2147483647
float	Záporná i kladná desetinná čísla	-3.402823e38 .. 3.402823e38
string	Textový řetězec	V podstatě libovolný rozsah

Nyní již znáte pojmy proměnná a datový typ. Pojďme si ukázat, jak můžete v programu vytvořit několik proměnných a pomocí symbolu = do nich přiřadit hodnotu. Všimněte si, že každá proměnná má před svým název definováno, jakého je typu.

```
byte pocetTestu = 7;
int pocetObyvatel = 10580456;
float slevaProPartnery = 11.5f;
string jmenoStudenta = "Pavel";
```

Upozornění: Pokud do proměnné, která je typu **float**, přiřazujete hodnotu, je nutné za číslo napsat písmeno **f**, např. **43.7f**. Pokud do proměnné typu **string** přiřazujete řetězec, je nutné, aby byl uzavřen do uvozovek.

Poznámka: Je praktické proměnné nazývat podle toho, jaká data obsahují. Čím budou vaše programy rozsáhlejší, tím více oceníte vhodné názvy proměnných, které vám usnadní orientaci ve zdrojovém kódu.

Název proměnné se může skládat z písmen, číslic a podtržítek, ale nesmí začínat číslicí. Pokud je název proměnné víceslovný, je vhodné každé další slovo začít velkým písmenem. Doporučuje se v názvech proměnných používat pouze písmena anglické abecedy, a vyhnout se tedy např. používání češtiny s diakritikou. Zároveň platí, že proměnná nesmí mít stejný název jako klíčová slova jazyka C# a nesmí se jmenovat jako již existující datové typy. Nelze tedy

proměnnou pojmenovat `Console`, `int`, `float` apod. Podívejte se na následující ukázku vhodných a nevhodných názvů proměnných.

```
int 1cislo = 7555; // Toto je špatně. Název proměnné nesmí začínat číslicí
byte pocetDeti = 3; // Všimněte si, že druhé slovo začíná velkým písmenem
float celkovemzdovenakladyoddeleni = 180900f; // Toto je špatně čitelné
float celkoveMzdoveNakladyOddeleni = 180900f; // Vhodnější název proměnné
```

Pokud definujeme proměnnou, nemusíme jí ihned přiřadit hodnotu. Zároveň platí, že hodnotu proměnné můžeme během programu měnit, ale vždy musíte respektovat její datový typ. Vytvoříme si proměnnou `pocetZamestnancu`, která bude datového typu `int`. Do té si později přiradíme číslo 36. Na dalším řádku tuto hodnotu přepíšeme hodnotou 39. Proměnná tedy bude obsahovat číslo 39 – s tím, že předchozí hodnota je přepsána. V proměnné je tedy vždy uložena právě jedna hodnota a proměnná si nepamatuje svou historii hodnot, které do ní byly dříve nastaveny.

```
// Definujeme proměnnou, ale zatím jsme jí nepřiřadili hodnotu
int pocetZamestnancu;
// Do již existující proměnné přiřadíme hodnotu
pocetZamestnancu = 36;
// Hodnotu proměnné můžeme v průběhu programu měnit
pocetZamestnancu = 39;
// Proměnná pocetZamestnancu je typu int (celé číslo). Toto je tedy špatně:
// pocetZamestnancu = 35.4f;
```

Řádek `pocetZamestnacu = 35.4f;` je označen jako komentář, protože není správný a program by nebylo možné spustit. Do proměnné typu `int` nemůžeme přiřadit desetinné číslo. Dále je důležité si zapamatovat, že pokud definujeme v programu více proměnných, nemohou mít stejný název, i kdyby měly rozdílné datové typy. Pokud bychom do výše uvedeného programu doplnili následující příkazy, program by nebylo možné spustit.

```
// Proměnná byla již definována s typem int. Toto je tedy špatně:
// float pocetZamestnancu = 39.5f;
```

Příkaz `float pocetZamestnancu = 39.5f;` je tedy záměrně označen jako komentář, protože proměnná `pocetZamestnancu` již byla v programu dříve definována.

Konstanty

Pojem konstanta definujme na základě znalosti pojmu proměnná následujícím způsobem. Pro konstanty platí totéž co pro proměnné s jedinou výjimkou: při definici konstanty jí přiřadíme hodnotu, kterou již nemůžeme nikdy za běhu programu změnit. Ve zdrojovém kódu definujeme konstantu pomocí klíčového slova `const`.

```
// Definujeme konstantu, její hodnotu nelze během programu měnit
const float koeficient = 1.75f;
// Toto je špatně:
```

```
koeficient = 2.1;
```

Představte si situaci, kdy program provádí výpočet ceny zboží s DPH. Výpočet je prováděn pomocí vzorce (Počet kusů * cena za kus) * DPH, kde DPH je např. 0,21.

Takovýto program by mohl fungovat tak, že počet kusů a cenu zas kus bude možné za běhu program změnit a DPH bude po celou dobu běhu programu neměnné. V tom případě je vhodné pro DPH použít konstantu, protože budete mít jistotu, že nikde v programu nedojde ke změně její hodnoty.

Základní operace s čísly

V programovacím jazyku C# můžete používat standardní matematické operace, jako je např. sčítání, odčítání, násobení a dělení. Podívejte se, jaké symboly se pro jednotlivé elementární matematické operace používají:

Tabulka 2.2 Základní matematické operace

Matematická operace	Symbol
Sčítání	+
Odčítání	-
Násobení	*
Dělení	/
Zbytek po celočíselném dělení	%

Nyní si ukažme, jak využijeme výše uvedené operace při práci s proměnnými. Na začátku ukázky deklarujeme dvě celočíselné a jednu desetinnou proměnnou. Poté si do dalších proměnných ukládáme výsledky jednotlivých matematických operací. Dále si všimněte, že platí přednost násobení a dělení před sčítáním a odčítáním. Samozřejmě je možné příslušnou operaci uzavřít do závorek a tím ji upřednostnit.

Upozornění: Věnujte pozornost zejména odlišnosti v situaci, kdy dělíme dvě celá čísla a kdy je alespoň jedno z čísel desetinné. Pokud dělíme dvě celá čísla je provedeno celočíselné dělení. Pokud je při dělení alespoň jedno číslo desetinné, je i výsledek desetinný.

```
int x = 10;
int y = 4;
float z = 4;

int soucet = x + y;
int rozdil = x - y;
int soucin = x * y;

// Pokud dělíme dvě celá čísla, je provedeno celočíselné dělení.
```

```
int podilCeleCislo = x / y;  // Výsledek je 2
int zbytePoCelociselnemDeleni = x % y;// Zbytek po celočíselném dělení je 2

// Pokud je při dělení alespoň jedno číslo desetinné, je i výsledek
// desetinné číslo
float podilDesetinneCislo = x / z; // Výsledek je 2.5
// Toto je špatně:
int podilSpatnyTyp = x / z;
int vysledek1 = 2 * 3 + 5 * 4; // Výsledek je 26
int vysledek2 = 2 * (3 + 5) * 4; // Výsledek je 64
int vysledek3 = 2 + 18 / 2; // Výsledek je 11
int vysledek4 = (2 + 18) / 2; // Výsledek je 10
```

Výpis na příkazový řádek

Nyní již umíte základy práce s proměnnými a je jistě na místě se naučit, jak uživateli vašeho programu něco sdělit. K tomuto účelu budeme využívat příkazový řádek, na který se nyní naučíte vypisovat potřebné informace.

Poznámka: Většina programů, které uživatelé používají, má vzhledné grafické uživatelské rozhraní. Nicméně tvorba takovýchto programů přesahuje rámec této knihy.

Proto budeme pracovat s příkazovým řádkem, který poskytuje velmi jednoduchý způsob, jak komunikovat s uživatelem prostřednictvím psaného textu, a budeme se soustředit především na to, abyste se naučili programovací jazyk C#. Ukažme si, jak by mohl program, který vypíše na příkazový řádek daný řetězec, vypadat.

```
using System;

namespace VypisNaKonzoli
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("První řetězec vypsaný na konzoli!");
      Console.ReadKey();
    }
  }
}
```

V ukázce vidíte příkaz `Console.WriteLine` obsahující v závorkách řetězec, který chceme na konzoli vypsat. Připomeňme si, že řetězce vždy uzavíráme do uvozovek. Kdybychom se měli vyjádřit přesně, řekli bychom, že na třídě `Console` voláme metodu `WriteLine`, které předáváme parametr typu `string`.

Následuje příkaz `Console.Readkey`, který zajistí, že program počká, dokud uživatel nestiskne libovolnou klávesu. Pojmy metoda a třída zatím neznáme. Prozatím si je popišme zjednodušeným způsobem. Později jim budou věnovány samostatné kapitoly.

Poznámka: Metoda je část programu, kterou někdo naprogramoval a která vykoná určitou činnost, v našem případě vypíše řetězec na konzoli.

Metoda může pro svou činnost požadovat parametry. V našem případě musíme metodě předat parametr typu `string`, kterým říkáme, co chceme na konzoli vypsat.

Poznámka: Třídu si zatím definujme jako část programu, která obsahuje metody.

Třída je zpravidla definována v tzv. jmenném prostoru. Třída `Console` je definována ve jmenném prostoru `System`. Jmenný prostor může obsahovat více tříd a budeme se mu věnovat v dalších kapitolách. Abychom mohli metodu `WriteLine` ze třídy `Console` zavolat, máme dvě možnosti. Buď můžeme napsat na začátek programu příkaz

```
using System;
```

kterým sdělíme, že v programu budeme používat třídy ze jmenného prostoru `System`. Nebo můžeme název třídy `Console` psát včetně jejího jmenného prostoru.

```
System.Console.WriteLine("První řetězec vypsaný na konzoli!");
```

Poznámka: V praxi se setkáte spíše s tím, že programátor použije příkaz **`using`**, než aby vypisoval název třídy včetně jejího jmenném prostoru. V této knize budeme preferovat právě používání onoho příkazu **`using`**.

Nyní již umíte vypsat libovolný řetězec na konzoli. Metoda `Console.WriteLine` umí kromě řetězce vypsat na konzoli i hodnotu proměnné, která je v podstatě libovolného typu.

Protože již znáte alespoň přibližnou definici třídy, je vhodné doplnit několik informací k základním datovým typům uvedeným v tabulce 2.1. Názvy, které byly uvedeny v tabulce 2.1, jsou takzvanými zkratkami pro názvy tříd jednotlivých datových typů uvedených v tabulce 2.3. V praxi se setkáte s tím, že se často používají právě ony zkratky namísto celého názvu třídy. Důvod je zřejmý: zápis pomocí zkratek je kratší a jednodušší.

Tabulka 2.3 Zkratky a názvy tříd základních datových typů

Datový typ (zkratka)	Třída	Rozsah
`byte`	`Byte`	0 .. 255
`int`	`Int32`	-2147483648 .. 2147483647
`float`	`Single`	-3.402823e38 .. 3.402823e38
`string`	`String`	V podstatě libovolný textový řetězec

V podkapitole Základní operace s čísly jsme vytvořili program, který provedl základní matematické operace a do proměnných ukládal jejich výsledky. Nyní tento program rozšíříme tak, aby výsledky výpočtů vypsal na konzoli.

```
using System;

namespace VypisPromennychNaKonzoli
{
  class Program
  {
    static void Main(string[] args)
    {
      int x = 10;
      int y = 4;
      float z = 4;

      int soucet = x + y;
      Console.WriteLine("10 + 4 =");
      Console.WriteLine(soucet);

      int rozdil = x - y;
      Console.WriteLine("10 - 4 =");
      Console.WriteLine(rozdil);

      int soucin = x * y;
      Console.WriteLine("10 * 4 =");
      Console.WriteLine(soucin);

      int podilCeleCislo = x / y;
      Console.WriteLine("10 / 4 =");
      Console.WriteLine(podilCeleCislo);

      int zbytekPoCelociselnemDeleni = x % y;
      Console.WriteLine("10 % 4 =");
      Console.WriteLine(zbytekPoCelociselnemDeleni);

      float podilDesetinneCislo = x / z;
      Console.WriteLine("10 / 4.0 =");
      Console.WriteLine(podilDesetinneCislo);

      // Na závěr počkáme, až uživatel stiskne klávesu
      Console.ReadKey();
    }
  }
}
```

```
10 + 4 =
14
10 - 4 =
6
10 * 4 =
40
10 / 4 =
2
10 % 4 =
2
10 / 4.0 =
2,5
```

Připomeňme si význam metody `Console.ReadKey`. Tato způsobí, že program čeká, dokud uživatel nestiskne libovolnou klávesu. Jinak by se pravděpodobně stalo to, že by uživatel ani nestihl zaregistrovat, to co bylo vypsáno na konzoli, předtím než se konzole zavřela, protože program skončil.

Dále si všimněte si, že výpis na konzoli není příliš přehledný. Metoda `Console.WriteLine` totiž po vypsání odřádkuje. To znamená, že další volání metody `Console.WriteLine` bude vypisovat na následující řádek konzole. Pokud chceme provést výpis na konzoli a neodřádkovat, použijeme metodu `Console.Write`. Takto by mohla vypadat část programu vypisující na konzoli sčítání dvou proměnných bez odřádkování před vypsáním výsledku.

```
// Výpis bez odřádkování
int prvniCislo = 7;
int druheCislo = 3;
int soucetDvouCisel = prvniCislo + druheCislo;
Console.Write("7 + 3 = ");
// Po vypsání výsledku odřádkujeme pro případ,
// že bychom vypisovali další informace
Console.WriteLine(soucetDvouCisel);
7 + 3 = 10
```

Zamysleme se nyní nad tím, co by se stalo, kdybychom změnili hodnoty proměnných `prvniCislo` a `druheCislo`. Museli bychom upravit i řetězec `"7 + 3 = "`. To není příliš praktické. Ukažme si, jak podobné situace řešit lépe. Využijeme toho, že metody `Console.Write` i `Console.WriteLine` umožňují v řetězci definovat místa, do kterých dosadíme potřebnou hodnotu. Připravíme si teď v řetězci místa, do kterých dosadíme potřebnou hodnotu.

```
int cislo1 = 7;
int cislo2 = 3;
int soucetCisel = cislo1 + cislo2;
Console.WriteLine("{0} + {1} = {2}", cislo1, cislo2, soucetCisel);
Console.ReadKey();
7 + 3 = 10
```

Všimněte si, že v řetězci jsou ve složených závorkách čísla, která určují, kolikátý z parametrů oddělených čárkami za řetězcem se má na dané místo dosadit. Abychom měli srovnání a dokázali ocenit přínos této možnosti dosazení hodnot do řetězce, zkusme výše uvedený program zapsat bez použití onoho dosazení.

```
int a = 7;
int b = 3;
int soucetAB = a + b;
Console.Write(a);
Console.Write(" + ");
Console.Write(b);
Console.Write(" = ");
// Poslední výpis uděláme pomoci WriteLine,
// abychom odřádkovali pro další možné výpisy
Console.WriteLine(soucetAB);
Console.ReadKey();
7 + 3 = 10
```

Ještě je zapotřebí si říct, že existují speciální symboly, které můžeme v řetězcích používat. Představte si situaci, kdy chcete v rámci vypisovaného řetězce vypsat i uvozovky. Mohli bychom zkusit napsat následující příkaz.

```
Console.WriteLine("Pavel: "Ahoj jak se mas?" ");
```

Program nebude možné spustit, protože řetězec musí být uzavřen do páru uvozovek. V tomto řetězci jsou ale uvozovky čtyřikrát a jazyk C# tomuto zápisu nerozumí. Pokud chcete v rámci řetězce použít uvozovky, je zapotřebí před ně umístit znak lomítka \. Správný zápis by tedy byl:

```
Console.WriteLine("Pavel: \"Ahoj jak se mas?\" ");
```

V následující tabulce uvádíme stručný výčet nejčastěji používaných speciálních symbolů v řetězcích.

Tabulka 2.4 Speciální symboly pro zápis v řetězcích

Symbol	Význam	Příklad
\\	Slouží k zápisu zpětného lomítka	`"Soubor je umisten na ceste C:\\temp\\soubor.txt"`
\"	Slouží k zápisu uvozovek	`"Pavel: \"Ahoj jak se mas?\""`
\n	Slouží k odřádkování	`"Jmeno: Karel\nPrijmeni: Prochazka"`
\t	Slouží k zápisu jednoho tabulátoru	`"Petr\tSoucek\t15.5.1976"`

Vyzkoušejte si použití těchto symbolů v řetězcích, které se budou vypisovat na konzoli.

Načtení vstupu od uživatele

Nedílnou součástí řady programů je získávání vstupu od uživatele. Jinými slovy: Programy mnohdy pro svou činnost potřebují informace, které jim uživatel v průběhu práce s programem zadá. Např. program, který by měl za úkol sečíst dvě čísla, by nejdříve uživatele vyzval k jejich zadání a následně by vypsal jejich součet na konzoli. Vy se nyní naučíte, jak získat informace od uživatele prostřednictvím konzole.

Využijeme metodu `Console.ReadLine`, která je definována na třídě `Console`. Opět mírně předběhneme a zjednodušeně se seznámíme s jednou vlastností metod.

Metody mohou po svém vykonání vracet výsledek, který budeme nazývat návratová hodnota metody. Každá metoda má jednoznačně určeno, jakého datového typu je její návratová hodnota. Metoda `Console.ReadLine` má návratovou hodnotu typu `string`. Pojďme si nyní ukázat program, který od uživatele načte jeho jméno a poté jej použije při výpisu věty na konzoli.

```
Console.Write("Zadejte jmeno: ");
string jmeno = Console.ReadLine();
Console.WriteLine("Zadane jmeno bylo: {0}", jmeno);
Console.ReadKey();
```

Protože jde o první program, kde očekáváme zadání hodnoty od uživatele, projdeme si příkazy programu krok za krokem. Nejdříve se provede metoda Console.Write, která do příkazového řádku vypíše řetězec „Zadejte jmeno“ a neodřádkuje. Uživatel tedy bude moci psát na ten samý řádek.

```
Zadejte jmeno:
```

Program počká na to, až uživatel napíše libovolný text a stiskne Enter. Toto je zajištěno zavoláním metody `Console.ReadLine`. Její návratovou hodnotou je tedy uživatelem napsaný text a tento se uloží do proměnné `jmeno`. Poté program vypíše na konzoli řetězec, do kterého vsadí hodnotu z proměnné `jmeno`. Na posledním řádku programu je volání metody `Console.ReadKey`. Tím je docíleno toho, že program po provedení posledního výpisu neskončí, příkazový řádek se automaticky nezavře a my můžeme vidět vypsaný řetězec.

```
Zadejte jmeno: Pavel
Zadane jmeno bylo: Pavel
```

Konverze řetězce na číslo

Nyní již umíte načíst řetězec od uživatele a uložit si jej do proměnné. Nicméně v řadě případů budete potřebovat získat od uživatele např. celé nebo desetinné číslo a dále s ním pracovat. Řekněme, že bychom měli napsat program, který od uživatele načte dvě celá čísla a vypíše jejich součin.

Následující program by nedával smysl, protože bychom násobili řetězce, a nikoliv celá čísla. To je způsobeno tím, že metoda `Console.ReadLine` vrací datový typ `string`. To je poměrně

pochopitelné. Tato metoda zkrátka vrátí přesně to, co uživatel napíše do konzole, když pak stiskne Enter. Může tedy napsat např. „Ahoj jak se mas?“ a „Mam se dobre“. My bychom si tyto dva řetězce uložili do proměnných a pokusili se je vynásobit. Násobit řetězce „Ahoj jak se mas?“ a „Mam se dobre“ nedává smysl. Takže i když uživatel zadá do konzole text „10“ a „20“, vrátí metoda `Console.ReadLine` datový typ `string`. Nyní si popíšeme postup, jak převést uživatelem zadaný řetězec na datový typ, který potřebujeme.

```
Console.Write("Zadejte prvni cislo: ");
string cislo1 = Console.ReadLine();
Console.Write("Zadejte druhe cislo: ");
string cislo2 = Console.ReadLine();
// Toto je špatně. Násobení řetězců nedává smysl.
// Je zapotřebí převod na čísla
string vysledek = cislo1 * cislo2;
```

Je zapotřebí převést řetězec, který uživatel zadá, na celé číslo. Převod provedeme pomocí metody `Parse` definované na třídě `Int32`, která přijímá parametr typu `string` obsahující řetězec, který chceme převést na celé číslo. Ukažme si program, který tento převod provede a již úspěšně vypíše onen součin dvou celých čísel.

```
Console.Write("Zadejte prvni cele cislo: ");
string retezecCislo1 = Console.ReadLine();
int cislo1 = Int32.Parse(retezecCislo1);
// Návratovou hodnotu metody ReadLine si nemusíme ukládat do proměnné,
// ale můžeme ji rovnou předat metodě Parse
Console.Write("Zadejte druhe cele cislo: ");
int cislo2 = Int32.Parse(Console.ReadLine());
int soucin = cislo1 * cislo2;
Console.WriteLine("{0} * {1} = {2}", cislo1, cislo2, soucin);
Console.ReadKey();
```

Pokud by uživatel zadal namísto čísla nějaký text např. „abcd“, program by skončil chybou, protože by se řetězec nepodařilo na číslo převést. V tuto chvíli se s tímto typem problémů neumíme vypořádat, ale nezoufejte, budeme se tímto zabývat v kapitole věnované tzv. výjimkám.

```
Zadejte prvni cele cislo: 21
Zadejte druhe cele cislo: 4
21 * 4 = 84
```

Jak bylo řečeno v podkapitole Výpis na příkazový řádek, `int` je zkratkou pro třídu `Int32`. Mohli bychom tedy namísto `Int32.Parse` napsat `int.Parse`. V praxi i v tomto textu se často setkáte s oběma způsoby zápisu.

Obdobně jako jsme vytvořili program pro celá čísla, vytvoříme program, který od uživatele načte dvě desetinná čísla a vypíše jejich součet. Pro převod na desetinné číslo použijeme metodu `Parse` definovanou na třídě `Single`. V programu vidíte, že je možné použít jak zkratku `float.Parse`, tak název třídy `Single.Parse`.

```
Console.WriteLine("Zadejte prvni desetinne cislo: ");
string retezecDesCislo1 = Console.ReadLine();
float desCislo1 = float.Parse(retezecDesCislo1);
Console.WriteLine("Zadejte druhe desetinne cislo: ");
// Návratovou hodnotu metody ReadLine si nemusíme ukládat do proměnné,
// ale můžeme ji rovnou předat metodě Parse
float desCislo2 = Single.Parse(Console.ReadLine());
float soucetDesCisel = desCislo1 + desCislo2;
Console.WriteLine("{0} + {1} = {2}", desCislo1, desCislo2, soucetDesCisel);
Console.ReadKey();
Zadejte prvni desetinne cislo:
5,25
Zadejte druhe desetinne cislo:
1,35
5,25 + 1,35 = 6,6
```

Všimněte si, že můžete metodě předat jako parametr přímo návratovou hodnotu jiné metody. V programu předáváme metodě `Parse` návratovou hodnotu metody `ReadLine`, aniž bychom si návratovou hodnotu ukládali do proměnné. Tento postup je běžný, pokud dále v programu nepotřebujete s návratovou hodnotou metody pracovat.

Konverze čísla na řetězec

Kromě konverze z řetězce na číslo je mnohdy zapotřebí provést konverzi opačným směrem. Pokud chceme číslo převést na řetězec, použijeme k tomu metodu `ToString`. Tuto metodu budeme používat následujícím způsobem. Napíšeme název proměnné, jejíž hodnotu chceme převést, a na ni zavoláme metodu `ToString`. Metodu `ToString` lze zavolat na libovolné proměnné libovolného datového typu. Důležité je, že metoda nezmění hodnotu v proměnné, na které ji zavoláme, ale vrátí hodnotu příslušné proměnné převedenou na datový typ `String`. Tuto návratovou hodnotu si můžeme uložit do jiné proměnné nebo ji rovnou použít v jiném příkazu. Dále je důležité, že řetězce je možné sčítat. Podívejte se na následující program, který demonstruje rozdíly mezi sčítáním čísel a sčítáním řetězců.

```
int cislo1 = 5;
int cislo2 = 10;
int soucetCisel = cislo1 + cislo2;
// Výsledek metody ToString si můžeme uložit do proměnné
string retezec1 = cislo1.ToString();
string retezec2 = cislo2.ToString();
string soucetRetezcu = retezec1 + retezec2;
// Výsledek metody ToString můžeme rovnou použít
// v dalších příkazech a metodách
string soucetRetezcu2 = cislo1.ToString() + cislo2.ToString();
Console.WriteLine("Soucet cisel = {0}", soucetCisel);
Console.WriteLine("Soucet retezcu = {0}", soucetRetezcu);
Console.ReadKey();
```

```
Soucet cisel = 15
Soucet retezcu = 510
```

Všimněte si, že pokud sčítáme čísla, je výsledkem číslo 15, a pokud sčítáme řetězce, je výsledkem řetězec `"510"`, který vznikl spojením řetězců `"5"` a `"10"`. K čemu je sčítání řetězců dobré? My jej zatím budeme nejčastěji využívat právě pro výpisy na konzoli. Nejde tedy o nějakou okrajovou záležitost, ale o poměrně často používaný postup, se kterým se často při programování setkáte.

Intelisense ve Visual Studiu

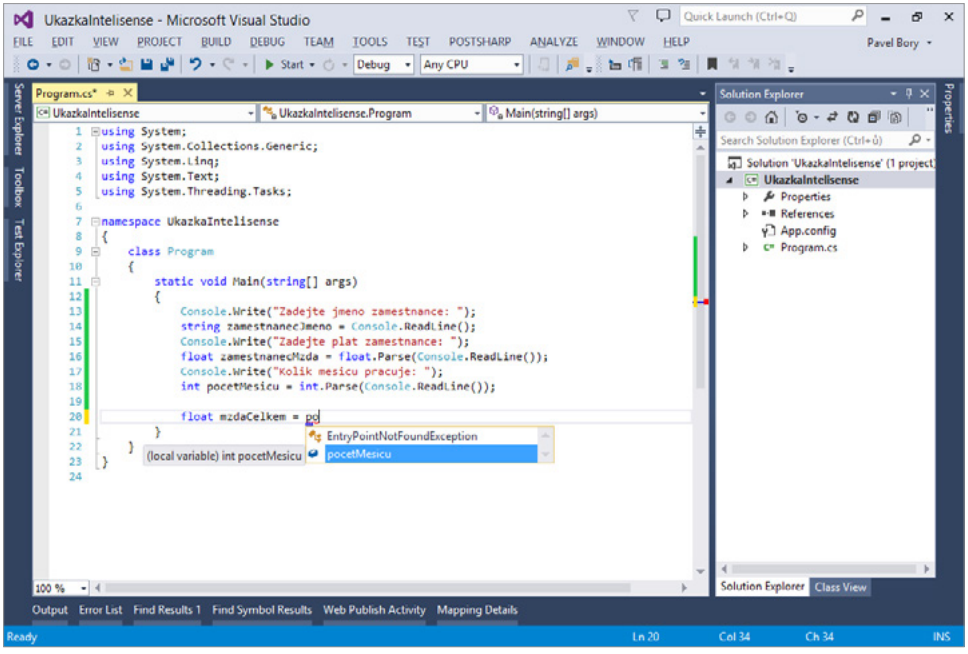
Pravděpodobně jste si všimli, že při psaní výše uvedených programů vám Visual Studio nabízelo jakousi nápovědu při psaní programu. Tomuto inteligentnímu našeptávání říkáme Intelisense. Jde o velmi užitečnou pomůcku při tvorbě programů ve Visual Studiu. Ukážeme si několik nyní několik základních příkladů, jak ji můžete využívat. Prvním možným využitím je nechat si napovědět názvy již definovaných proměnných. Mějme následující program:

```
class Program
{
  static void Main(string[] args)
  {
    Console.Write("Zadejte jmeno zamestnance: ");
    string zamestnanecJmeno = Console.ReadLine();
    Console.Write("Zadejte mzdu zamestnance: ");
    float zamestnanecMzda = float.Parse(Console.ReadLine());
    Console.Write("Kolik mesicu pracuje: ");
    int pocetMesicu = int.Parse(Console.ReadLine());
  }
}
```

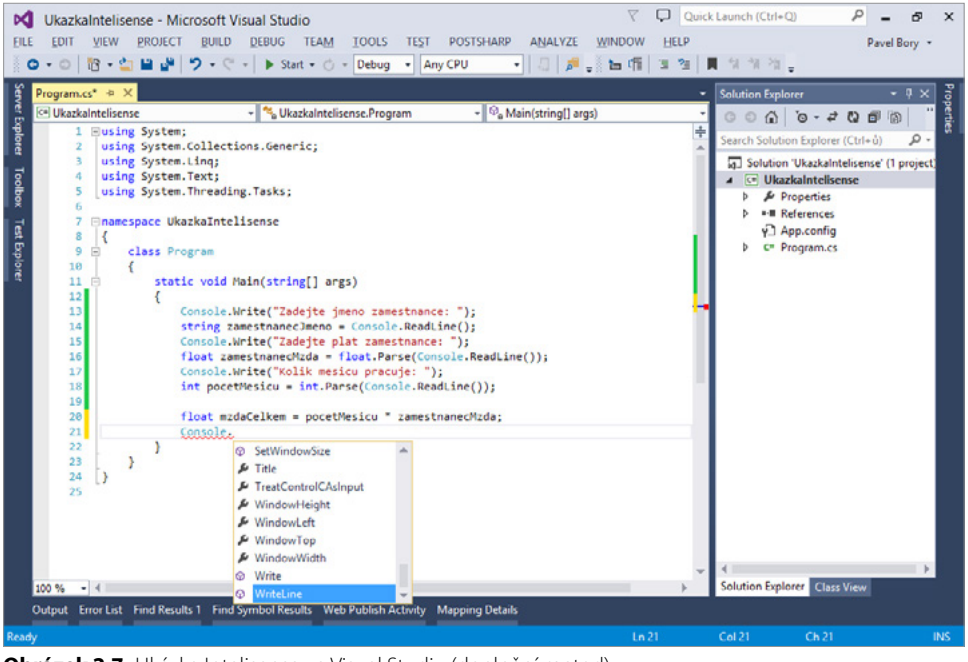
Budeme chtít pokračovat v programování a spočítat, kolik si zaměstnanec za daný počet měsíců celkem vydělal. Víme, že potřebujeme udělat následující výpočet: `pocetMesicu * zamestnanecMzda`. Na další řádek začneme tedy výpočet psát. Do proměnné `mzdaCelkem` budeme chtít výsledek uložit. Napišme část řádku a začněme psát název proměnné `pocetMesicu`. Všimněte si, jak nám Visual Studio začne napovídat.

Tip: Z kontextové nápovědy můžeme buď myší vybrat požadované doplnění, nebo se v nabídce pohybovat pomocí šipek na klávesnici a stiskem klávesy Enter požadované doplnění vybrat.

Další často používanou funkcí Intelisense je nechat si napovědět metody, které lze použít. Budeme chtít v program vypsat výsledek výpočtu na konzoli. Řekněme, že si pamatujeme název třídy `Console` a víme, že je na ní definovaná metoda pro výpis na konzoli. Napišme tedy název třídy a za něj tečku a nechme si napovědět, jaké metody lze na třídě `Console` zavolat.



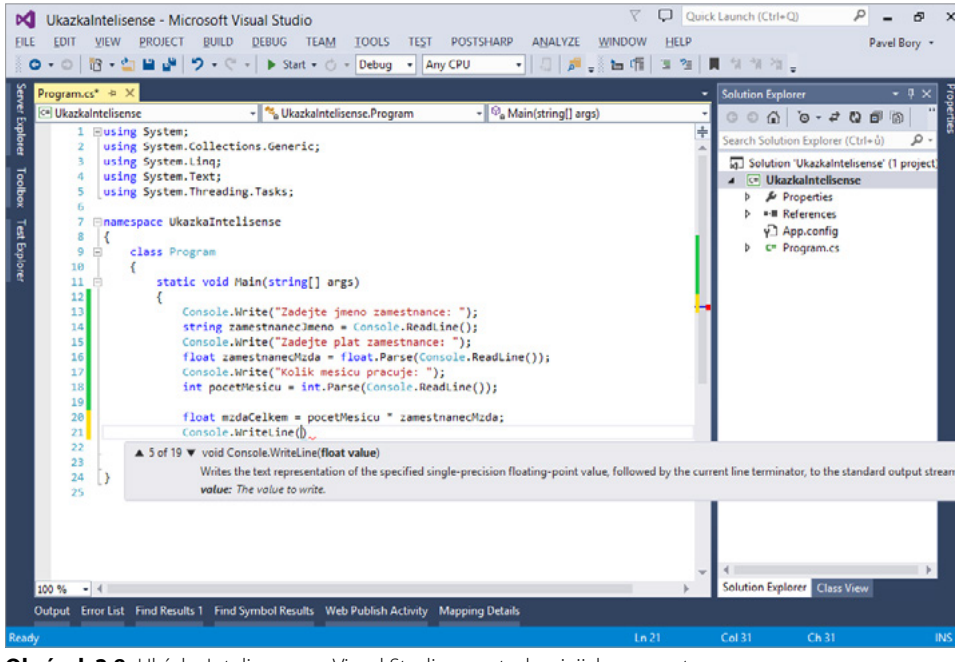
Obrázek 2.6 Ukázka Intelisense ve Visual Studiu (doplnění názvu proměnné)



Obrázek 2.7 Ukázka Intelisense ve Visual Studiu (doplnění metod)

Vidíme, že na třídě `Console` je definované velké množství metod. My si v kontextové nápovědě najdeme metodu `WriteLine` a můžeme si ji nechat do zdrojového kódu doplnit, stejně jako jsme to udělali s názvem proměnné `pocetMesicu`.

Tip: Ve chvíli, kdy napíšete název metody a první závorku nebo si název metody doplníte pomocí Intelisense, začne vám Intelisense napovídat, s jakými parametry lze metodu zavolat. Vzpomeňte si, jak jsme si říkali, že metoda `WriteLine` umí na konzoli vypsat řetězec, celé číslo, desetinné číslo a spoustu dalších datových typů. Právě toto nám Intelisense napoví. Zobrazí se vám kontextová nabídka, ve které se můžete pomocí šipek pohybovat a vidět, jaké parametry lze metodě předat.



Obrázek 2.8 Ukázka Intelisense ve Visual Studiu – metody a jejich parametry

Na obrázku je vidět, že jsme si v kontextové nápovědě zobrazili možnost předání jednoho parametru typu `float`. Zatím jste na samém začátku studia programování v C#, proto je naprosto pochopitelné, že značnou část metod, datových typů a dalších položek z kontextových nápověd neznáte. Seznámíte se s nimi postupně během čtení této knihy.

Kontrolní otázky

1. Co znamená pojem proměnná?
2. Jaké znáte základní datové typy?
3. Pomocí jakých metod můžeme vypisovat řetězce na konzoli?

4. Pomocí jakého speciálního znaku zapíšete v řetězci uvozovky? Jaké další speciální znaky pro použití v řetězcích znáte?
5. Jakou metodou můžeme načíst řetězec od uživatele z konzole?
6. Jak převedete řetězec na celé číslo?
7. Co je to Intelisense? K čemu ji můžeme využít?

Řešená cvičení

1. Napište program, který uživatele vyzve k zadání jména a věku ve formě celého čísla. Obdobně jej vyzve k zadání jména a věku jeho kamaráda nebo kamarádky. Poté program vypíše, jaký je mezi nimi věkový rozdíl (počet let).

Začneme tím, že uživatele vyzveme k zadání jména a věku. V programu očekáváme, že uživatel zadá věk jako celé číslo.

```
Console.Write("Zadejte vase jmeno: ");
string jmeno = Console.ReadLine();
Console.Write("Zadejte vas vek: ");
int vek = int.Parse(Console.ReadLine());
```

Pro výpis na konzoli použijeme metodu `Console.Write`, které předáme jako parametr řetězec, který chceme vypsat na konzoli. Tato metoda po výpisu neodřádkuje, a uživatel tedy píše na konzoli na stejný řádek, na kterém byl vypsán řetězec. Dále zavoláme metodu `Console.ReadLine`, která vrátí uživateli zadaný text. Tato metoda vrací výsledek jako datový typ `string`. Výsledek volání metody si uložíme do proměnné `jmeno`. Obdobně postupujeme při získání věku – s tím rozdílem, že zadaný řetězec převedeme na celé číslo pomocí metody `int.Parse`. Dále v programu budeme chtít počítat rozdíl věku, a proto musíme mít možnost pracovat s věkem jako s číslem, a nikoliv jako s řetězcem. Stejným způsobem postupujeme při získání údajů o kamarádce.

```
Console.Write("Zadejte jmeno kamaradky: ");
string jmenoKamaradky = Console.ReadLine();
Console.Write("Zadejte vek kamaradky: ");
int vekKamaradky = int.Parse(Console.ReadLine());
```

Nyní máme všechny potřebné údaje. Provedeme výpočet rozdílu věku a výsledek vypíšeme na konzoli. Pro procvičení sčítání řetězců jsme výslednou zprávu pro výpis na konzoli sestavili níže uvedeným způsobem. Bylo by samozřejmě možné použít metodu `Console.WriteLine` s doplněním parametrů na definovaná místa – `Console.WriteLine("Rozdil ve veku {0} je ...", jmeno, ...)`

```
int rozdilVeku = vek - vekKamaradky;
string zpravaRozdil = "Rozdil ve veku " + jmeno + " a " +
  jmenoKamaradky + " je " + rozdilVeku.ToString();
Console.WriteLine(zpravaRozdil);
Console.ReadKey();
```

Na závěr si ukážeme kompletní řešení včetně komentářů ve zdrojovém kódu.

```
using System;

namespace Cviceni1
{
  class Program
  {
    static void Main(string[] args)
    {
      // Vyzveme uživatele, aby zadal své jméno a věk
      Console.Write("Zadejte vase jmeno: ");
      string jmeno = Console.ReadLine();
      Console.Write("Zadejte vas vek: ");

      // Nezapomeneme provést konverzi z řetězce na číslo.
      // Mohli bychom také použít zápis Int32.Parse(Console.ReadLine())
      int vek = int.Parse(Console.ReadLine());

      // Vyzveme uživatele, aby zadal jméno své kamarádky a její věk
      Console.Write("Zadejte jmeno kamaradky: ");
      string jmenoKamaradky = Console.ReadLine();
      Console.Write("Zadejte vek kamaradky: ");
      int vekKamaradky = int.Parse(Console.ReadLine());

      // Pro procvičení sčítání řetězců si připravíme výsledek, který vypíšeme
      // jako řetězec, který vytvoříme postupným sčítáním jednotlivých částí
      int rozdilVeku = vek - vekKamaradky;
      string zpravaRozdil = "Rozdil ve veku " + jmeno + " a " +
        jmenoKamaradky + " je " + rozdilVeku.ToString();
      Console.WriteLine(zpravaRozdil);
      Console.ReadKey();
    }
  }
}
```

```
Zadejte své jmeno: Pavel
Zadejte svůj vek: 27
Zadejte jmeno kamaradky: Zuzka
Zadejte vek kamaradky: 28
Rozdil ve veku Pavel a Zuzka je 1
```

Cvičení

1. Napište program, který uživatele vyzve k zadání dvou celých čísel. Poté program vypíše jejich součet, rozdíl, součin a podíl.
2. Modifikujte program ze cvičení 1 tak, aby pracoval s desetinnými čísly.

KAPITOLA 3

Řízení toku programu

V této kapitole:

- Logický datový typ a logický výraz
- Logické operátory
- Příkaz if
- Příkaz else a vnořování podmínek
- Příkaz else if
- Příkaz switch
- Kontrolní otázky
- Řešená cvičení
- Cvičení

Nyní umíte vytvořit program, který postupně vykoná všechny příkazy, které jste zapsali. Představte si ale situaci, že budete chtít napsat program, který od uživatele načte dvě čísla a poté podle volby uživatele vypíše jejich součet nebo rozdíl. Jinými slovy budete chtít napsat takový program, který vykoná jednu sadu příkazů, pokud uživatel zvolí možnost součtu, a druhou sadu příkazů v případě, že uživatel zvolí možnost rozdílu. Toto se právě v této kapitole naučíte.

Logický datový typ a logický výraz

Než se pustíte do řízení toho, které části programu budou vykonány na základě různých podmínek, je zapotřebí, abyste se seznámili s novým datovým typem. Jde o datový typ `Boolean`, který může nabývat pouze dvou hodnot – `true` a `false` neboli česky pravda – nepravda. Obdobně jako u datového typu `Int32` má i `Boolean` svou zkratku, a to `bool`. Ukažme si, jak definovat proměnnou, která bude logického datového typu `Boolean`.

```
Boolean promenna1 = true;
bool promenna2 = false;
```

S logickým datovým typem úzce souvisí pojem logický výraz. Jde o výraz, jehož výsledkem je právě hodnota `true` nebo `false` výše zmíněného logického datového typu `Boolean`. Logický výraz musí splňovat pravidlo, že je jednoznačně vyhodnotitelné, zdali je pravdivý. Příkladem logického výrazu, u nějž můžeme prohlásit, že je pravdivý, může být výraz 5 > 3. Naopak

nepravdivým výrazem může být např. výraz 1 > 100. Pojďme si nyní představit nejčastěji používané logické výrazy, které budete používat.

Tabulka 3.1 Logické výrazy

Výraz	Význam
x == y	x je rovno y
x < y	x je menší než y
x > y	x je větší než y
x <= y	x je menší nebo rovno y
x >= y	x je větší nebo rovno y
x != y	x není rovno y

Níže uvedený program obsahuje ukázku několika logických výrazů. Všimněte si, že pomocí metody `Console.WriteLine` můžeme vypsat jak hodnotu proměnné typu `Boolean`, tak přímo výsledek logického výrazu.

```
int x1 = 5;
int y1 = 3;
bool vyraz1 = x1 > y1;
Console.WriteLine(vyraz1);
Console.WriteLine(x1 > y1);
Console.WriteLine("{0} > {1} ... {2}",x1,y1,x1 > y1);

int x2 = 4;
int y2 = 4;
Console.WriteLine(x2 > y2);
Console.WriteLine("{0} > {1} ... {2}", x2, y2, x2 > y2);
Console.WriteLine("{0} >= {1} ... {2}", x2, y2, x2 >= y2);
Console.WriteLine("{0} == {1} ... {2}", x2, y2, x2 == y2);
Console.ReadKey();
```

```
False
True
True
True
5 > 3 ... True
False
4 > 4 ... False
4 >= 4 ... True
4 == 4 ... True
```

Tip: Je důležité zmínit, že pomocí logického výrazu `x == y` nemusíme porovnávat pouze celá čísla. Můžeme porovnat v podstatě jakékoliv datové typy. Podívejte se na následující program, který obsahuje ukázku porovnání desetinných čísel, řetězců a logických hodnot.

```
float desCislo1 = 32.1f;
float desCislo2 = 32.2f;
Console.WriteLine("{0} > {1} ... {2}", desCislo1, desCislo2,
  desCislo1 > desCislo2);
Console.WriteLine("{0} <= {1} ... {2}", desCislo1, desCislo2,
  desCislo1 <= desCislo2);

string jmeno1 = "Pavel";
string jmeno2 = "Karel";
string jmeno3 = "Pavel";

Console.WriteLine("{0} == {1} ... {2}", jmeno1, jmeno2, jmeno1 == jmeno2);
Console.WriteLine("{0} == {1} ... {2}", jmeno1, jmeno3, jmeno1 == jmeno3);

bool hodnota1 = true;
bool hodnota2 = false;
bool hodnota3 = true;
Console.WriteLine("{0} == {1} ... {2}", hodnota1, hodnota2,
  hodnota1 == hodnota2);
Console.WriteLine("{0} == {1} ... {2}", hodnota1, hodnota3,
  hodnota1 == hodnota3);
```

```
32,1 > 32,2 ... False
32,1 <= 32,2 ... True
Pavel == Karel ... False
Pavel == Pavel ... True
True == False ... False
True == True ... True
```

Logické operátory

Velice často se setkáte s tím, že budete muset zapsat komplexnější logický výraz. Představte si situaci, kdy bude chtít zapsat výraz, kterým budete moci vyhodnotit, zdali je možné si objednat donášku pizzy. Podmínku pro objednání si zjednodušme tak, že objednávku bude možné provést pouze tehdy, pokud máte u sebe dostatek peněz a zároveň má pizzerie otevřeno. Abyste mohli tento výraz zapsat, musíte použít logický operátor „a zároveň“, který v jazyce C# zapíšeme jako `&&`. Kromě operátoru `&&` existují i operátory `||` (nebo) a `!` (negace). Podívejte se na následující přehled logických operátorů, kde je uvedeno jejich vyhodnocení pro jednotlivé logické hodnoty.

Tabulka 3.2 Logické operátory

x	y	!x	!y	x && y	x || y
true	true	false	false	true	true
true	false	false	true	false	true

x	y	!x	!y	x && y	x || y
false	true	true	false	false	true
false	false	true	true	false	false

Prostudujte následující program a ověřte si, že rozumíte tomu, jaké budou výsledky jednotlivých logických výrazů, ve kterých jsou použity logické operátory.

```
bool vyraz1 = !(5 >= 4);
Console.WriteLine("!(5 >= 4) : {0}", vyraz1);
bool vyraz2 = !(5 < 4);
Console.WriteLine("!(5 < 4) : {0}", vyraz2);

int cenaPizzy = 150;
bool otevreno = true;
int penize = 500;
bool vyraz3 = ((cenaPizzy < penize) && (otevreno == true));
Console.WriteLine("((cenaPizzy < penize) && (otevreno == true)) : {0}",
  vyraz3);

bool vyraz4 = ((penize > cenaPizzy) && (otevreno != false));
Console.WriteLine("((penize > cenaPizzy) && (otevreno != false)) : {0}",
  vyraz4);

bool vyraz5 = ((5 > 4) && (3 < 2));
Console.WriteLine("((5 > 4) && (3 < 2)) : {0}", vyraz5);

bool vyraz6 = ((3 > 4) && (3 < 2));
Console.WriteLine("((3 > 4) && (3 < 2)) : {0}", vyraz6);

bool vyraz7 = ((5 > 4) || (3 < 2));
Console.WriteLine("((5 > 4) || (3 < 2)) : {0}", vyraz7);

bool vyraz8 = ((5 > 4) || (1 < 2));
Console.WriteLine("((5 > 4) || (1 < 2)) : {0}", vyraz8);

bool vyraz9 = ((1 > 4) || (3 < 2));
Console.WriteLine("((1 > 4) || (3 < 2)) : {0}", vyraz9);
```

```
!(5 >= 4) : False
!(5 < 4) : True
((cenaPizzy < penize) && (otevreno == true)) : True
((penize > cenaPizzy) && (otevreno != false)) : True
((5 > 4) && (3 < 2)) : False
((3 > 4) && (3 < 2)) : False
((5 > 4) || (3 < 2)) : True
((5 > 4) || (1 < 2)) : True
((1 > 4) || (3 < 2)) : False
```

Příkaz if

Když nyní umíte psát logické výrazy a používat logické operátory, přišel čas se naučit jeden z nejdůležitějších postupů při programování – vykonat určitou část programu pouze tehdy, když platí nějaká podmínka. Za tímto účelem se v jazyce C# používá příkaz `if`, za který se do závorek zapíše podmínka, která určí, zdali se provede část programu, která je za příkazem `if` uvedena ve složených závorkách.

Z tohoto slovního popisu může být poměrně obtížné si představit, jak výše popsanou konstrukci zapsat v jazyce C#. Proto začneme používat ukázky tzv. syntaxe, zjednodušeně řečeno ukázky pravidel, jak formálně zapsat příslušný příkaz nebo konstrukci v jazyce C#. Ukázky syntaxe nejsou spustitelnými programy. Mají za účel pomoci definovat formální pravidla pro zápis příslušného příkazu nebo konstrukce.

Pro příkaz `if` můžeme definovat následující syntaxi.

```
if (logický výraz)
{
  Kód, který se vykoná, pokud je logický výraz
  vyhodnocen jako pravdivý
}
```

Nyní se podívejte na následující program, který demonstruje použití příkazu `if` na základě výše popsané syntaxe.

```
int x = 4;
int y = 5;

if (x > y)
{
  Console.WriteLine("Tento text nebude vypsan");
}

if (x < y)
{
  Console.WriteLine("Tento text bude vypsan");
}
```

Upozornění: Zejména si zapamatujte, že je nutné podmíněně vykonanou část programu zapsat do složených závorek.

Zkusme si nyní napsat program, který vyzve uživatele, aby zadal svůj věk, a pokud je starší 18 let, vypíše, že je dospělý, jinak vypíše, že není dospělý.

```
Console.Write("Zadejte svuj vek: ");
int vek = Int32.Parse(Console.ReadLine());
if (vek >= 18)
{
```