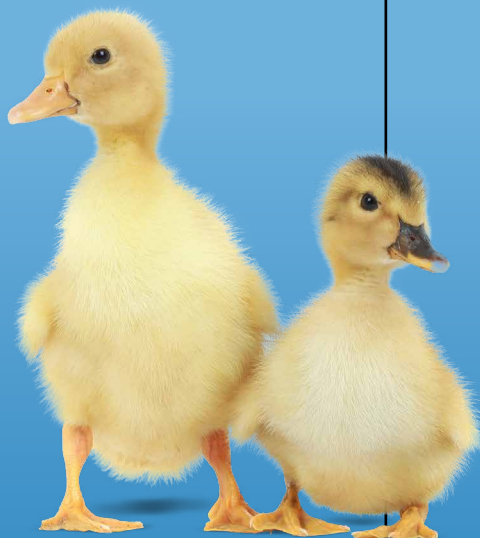


knihovna programátora

- Přehledná a praktická učebnice C# pro začátečníky i uživatele ostatních programovacích jazyků
- Základní programovací konstrukce a potřebné pojmy (program, operační systém, programovací jazyk, objektově orientované programování a další)
- Programovací jazyk C# od základů k pokročilým konstrukcím
- Úvod do nejdůležitějších knihoven, .NET Core, verze .NET 5 a do používání vývojových nástrojů



Programování v

C#

od základů k profesionálnímu použití

MIROSLAV VIRIUS



soubory
ke stažení na

WWW.GRADA.CZ



knihovna programátora

C# Programování v

**od základů
k profesionálnímu
použití**

Miroslav Virius

Upozornění pro čtenáře a uživatele této knihy

Všechna práva vyhrazena. Žádná část této tištěné či elektronické knihy nesmí být reprodukována a šířena v papírové, elektronické či jiné podobě bez předchozího písemného souhlasu nakladatele. Neoprávněné užití této knihy bude **trestně stíháno**.

Miroslav Virius

Programování v C#

Od základů k profesionálnímu použití

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401
jako svou 7866. publikaci

Odborná korektura Jan Hájek, NETWORK
Odpovědný redaktor Petr Somogyi
Sazba Petr Somogyi
Počet stran 424
První vydání, Praha 2021
Vytiskla Tiskárna v Ráji, s.r.o., Pardubice

© Grada Publishing, a.s., 2021
Cover Design © Grada Publishing, a. s., 2021

Názvy produktů, firem apod. použité v knize mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

ISBN 978-80-271-4004-6 (ePub)
ISBN 978-80-271-4003-9 (pdf)
ISBN 978-80-271-1216-6 (print)

Předmluva	15
Programovací jazyk C#	15
Co v této knize najdete	15
Na co se nedostalo	16
Nástroje	16
Terminologie	16
Příklady	16
Na závěr	16
1 Než opravdu začneme	17
1.1 Počítač	17
1.2 Operační paměť	17
1.3 Datové typy a proměnné	18
1.4 Programy a programovací jazyky	19
1.5 Operační systém	20
1.6 Program a algoritmus	20
1.7 Objekty a třídy	23
1.7.1 Zapouzdření	23
1.7.2 Objektový program	24
1.7.3 Modelovací jazyk UML	25
1.7.4 Skládání objektů	25
1.7.5 Dědění	25
1.7.6 Polymorfismus	27
1.7.7 Abstraktní třída	27
1.7.8 Dědění versus skládání	28
1.8 Jazyk C#	28
1.8.1 Prostředí .NET	29
1.8.2 Metadata	30
1.8.3 Společný systém typů	30
1.8.4 C#, Java a C++	31
1.9 Kde získat překladač C#	31
2 První programy	33
2.1 Nápis na obrazovce	33
2.1.1 Zdrojový text	33
2.1.2 Příprava překladu z příkazové řádky	34
2.1.3 Překládáme program z příkazové řádky	34
2.1.4 Jak to zkazit	35
2.2 Co jsme naprogramovali	36
2.3 Ódy žlutého koně: čeština v programu	38
2.4 Sestavení (assembly)	39

2.5	Opět žlutý kůň, tentokrát s Visual Studiem 2019	40
2.5.1	Projekt a řešení	40
2.5.2	Projekt nového programu	40
2.5.3	Visual Studio nám našeptává	44
2.5.4	Nové konstrukce v programu	44

3

	Jednoduché příklady	46
3.1	Drobné úpravy programu	46
3.2	Počítač ať počítá	48
3.3	Metoda, která vypočte hodnotu	50
3.4	Jednoduchý vstup z konzole	52
3.4.1	Vytváříme dynamickou knihovnu	54
3.4.2	Použití dynamické knihovny	55
3.5	Ještě trochu počítání	56
3.5.1	Faktoriál	56
3.5.2	Užitečné operátory	58
3.5.3	Podmínka	59
3.5.4	Indikace chyby	60
3.5.5	Výjimky	61
3.5.6	Magická čísla	62
3.6	Univerzální nápis	63
3.6.1	Třída Text	63
3.6.2	Přetěžování metod	64
3.6.3	Vytvoření instance	65
3.6.4	Volání metod	66
3.6.5	Program	66
3.6.6	Přiřazování odkazů	66
3.6.7	Automatická správa paměti	67

4

	Složitější příklady	69
4.1	Řazení slov	69
4.1.1	Třídy v programu	69
4.1.2	Slovník	71
4.1.3	Analyzátor	72
4.1.4	Program složený z více souborů	75
4.1.5	Zkoušíme program	76
4.2	Seznam	77
4.2.1	Jednosměrně zřetěžený seznam	77
4.2.2	Implementace seznamu	79
4.3	Vylepšujeme vstup	82
4.3.1	Požadavky	83
4.3.2	Úvodní úvahy	83
4.3.3	Čtení řádky	83
4.3.4	Přeskočení mezer	84
4.3.5	Další slovo	84
4.3.6	Čtení řetězce	85
4.3.7	Na závěr	85
4.4	Ladění programu	85
4.4.1	Zdrojový kód	86
4.4.2	Nástroje pro ladění	86

4.4.3	Krokujeme program	87
4.4.4	Ukončujeme krokování	87
4.4.5	Skok na zadané místo programu.....	88
4.4.6	Vstup do metody, výstup z metody.....	88
4.4.7	Zjištění hodnoty proměnné	88
4.4.8	Zarážka	89
4.4.9	Další možnosti.....	89

5	Začínáme naostro.....	91
5.1	Jak budeme C# popisovat.....	91
5.2	Základní pojmy.....	92
5.2.1	Komentář.....	92
5.2.2	Klíčová slova	93
5.2.3	Identifikátor.....	93
5.2.4	Zápis programu.....	94
5.3	Jmenné prostory	94
5.3.1	Deklarace jmenného prostoru	95
5.3.2	Spojování jmenných prostorů	96
5.3.3	Globální jmenný prostor	97
5.3.4	Direktiva using	97
5.3.5	Přejmenování.....	97
5.4	Atributy	98
5.4.1	Některé atributy	99

6	Proměnné a datové typy.....	100
6.1	Proměnné.....	100
6.1.1	Deklarace s klíčovým slovem var.....	100
6.1.2	Povinná inicializace	101
6.1.3	Typy a proměnné	101
6.2	Třída System.Object.....	101
6.2.1	Rovnost objektů.....	102
6.2.2	Řetězcová reprezentace objektu	102
6.2.3	Mělká kopie objektu.....	102
6.2.4	Další metody	102
6.3	Hodnotové typy poprvé: atomické typy	103
6.3.1	Celá čísla	103
6.3.2	Znaky	107
6.3.3	Reálná čísla (typy float a double).....	108
6.3.4	Desítková čísla (typ decimal).....	111
6.3.5	Logické hodnoty.....	111
6.3.6	Prázdný „typ“ void.....	112
6.3.7	Převod řetězce na hodnotu číselného, znakového nebo logického typu	114
6.4	Pole: skupina proměnných stejného typu	115
6.4.1	Počet prvků.....	115
6.4.2	Deklarace pole	115
6.4.3	Vytvoření pole	116
6.4.4	Práce s prvky.....	116
6.4.5	Inicializace pole	117

6.4.6	Vícerozměrná pole	117
6.4.7	Nepravidelná pole	117
6.4.8	Práce s poli.....	118
6.5	Instance tříd.....	121
6.5.1	Automatická správa paměti.....	121
6.6	Hodnotové typy podruhé	122
6.6.1	Výčtové typy.....	122
6.6.2	Struktury	124
6.7	Zabalení a vybalení.....	125
6.8	Uspořádané n-tice.....	125
6.8.1	Funkce vracející n-tice.....	125
6.8.2	Přiřazování pomocí n-tic.....	126
6.8.3	Deklarace skupiny proměnných.....	126
6.8.4	Vynechání jedné složky.....	126
6.8.5	Implicitní dekonstrukce.....	127

7

Příkazy.....	128	
7.1	Blok (složený příkaz)	128
7.2	Elementární příkazy.....	129
7.2.1	Prázdný příkaz	129
7.2.2	Výrazový příkaz	129
7.2.3	Deklarace.....	129
7.3	Podmíněné příkazy.....	130
7.3.1	Příkaz if.....	130
7.3.2	Příkaz switch (přepínač)	131
7.4	Cykly.....	132
7.4.1	Příkaz while	133
7.4.2	Příkaz do-while	133
7.4.3	Příkaz for.....	134
7.4.4	Příkaz foreach.....	136
7.5	Skokové příkazy	137
7.5.1	Příkaz break	137
7.5.2	Příkaz continue	137
7.5.3	Příkaz goto.....	138
7.5.4	Příkaz return.....	139
7.5.5	Příkazy throw, checked, unchecked	139
7.6	Příkaz using.....	140
7.7	Příkazy yield.....	140
7.8	Ještě jednou příkaz switch.....	142
7.8.1	Výběr alternativy podle typu.....	142
7.8.2	Výběr podle typu s klauzulí when	144

8

Výrazy a operátory.....	145	
8.1	Vlastnosti operátorů.....	145
8.1.1	Priorita	145
8.1.2	Asociativita	145
8.2	Aritmetické výrazy.....	145
8.2.1	Unární rozšíření.....	146
8.2.2	Binární rozšíření.....	146

8.3	Relační výrazy.....	148
8.4	Logické výrazy.....	148
8.5	Přehled operátorů.....	148
8.5.1	Podmíněný výraz.....	148
8.5.2	Operátor switch (podmíněný výraz s více možnostmi).....	149
8.5.3	Určování typu instance.....	152
8.5.4	Určení velikosti hodnotového typu.....	153
8.5.5	Obrácené indexování.....	153
8.5.6	Definice rozsahu.....	154
8.5.7	Operátor nameof.....	154

9

	Třídy a objekty.....	156
9.1	Deklarace třídy.....	156
9.1.1	Modifikátory v deklaraci třídy.....	156
9.1.2	Specifikace předka a rozhraní.....	157
9.2	Tělo třídy.....	158
9.2.1	Přístupová oprávnění.....	158
9.3	Datové složky.....	159
9.3.1	Nestatické datové složky.....	159
9.3.2	Neměnitelné složky.....	160
9.3.3	Statické datové složky.....	161
9.4	Metody.....	162
9.4.1	Přetěžování.....	162
9.4.2	Deklarace metody.....	162
9.4.3	Parametry metod.....	163
9.4.4	Nestatické metody.....	168
9.4.5	Statické metody.....	169
9.4.6	Lokální proměnné.....	171
9.4.7	Rozšiřující metody.....	171
9.4.8	Rekurze.....	172
9.4.9	Metoda Main().....	173
9.4.10	this.....	175
9.5	Konstruktory a destruktory.....	176
9.5.1	Inicializátor.....	176
9.5.2	Statický konstruktor.....	177
9.5.3	Destruktor.....	177
9.6	Vlastnosti.....	178
9.6.1	Deklarace vlastnosti.....	178
9.7	Dědění.....	180
9.7.1	Konstruktor potomka.....	181
9.7.2	Předefinované metody a vlastnosti.....	182
9.7.3	Nepolymorfní chování předefinovaných metod.....	182
9.7.4	Polymorfní chování metod (překrývání).....	184
9.7.5	Grafické objekty.....	184
9.7.6	Abstraktní metody, abstraktní třídy.....	189
9.7.7	Zapečetěné třídy, zapečetěné metody.....	190
9.8	Struktury.....	190
9.8.1	Neměnné struktury.....	191

9.9	Vnitřní datové typy a anonymní třídy.....	192
9.9.1	Vnitřní datové typy.....	192
9.9.2	Nepojmenované třídy.....	192

10	Ještě jednou objektové typy	193
10.1	Rozhraní.....	193
10.1.1	Rozhraní jako seznam metod a jako typ.....	193
10.1.2	Deklarace rozhraní.....	194
10.1.3	Rozhraní a dědění	195
10.1.4	Implementace rozhraní.....	195
10.1.5	Explicitní implementace metody z rozhraní.....	195
10.1.6	Příklady rozhraní z knihoven prostředí .NET	196
10.1.7	Klonování objektů	197
10.1.8	Další možnosti rozhraní.....	199
10.2	Generické typy a metody.....	200
10.2.1	Deklarace generické třídy	201
10.2.2	Omezení formálních typů.....	201
10.2.3	Inicializace datových složek formálních typů	202
10.2.4	Generické metody	202
10.3	Seznam jako (téměř) standardní kolekce	203
10.3.1	Potřebné pojmy.....	203
10.3.2	Třída Seznam: základní část.....	204
10.3.3	Enumerátor a co s ním souvisí	207
10.4	Delegáty a lambda-výrazy.....	212
10.4.1	Deklarace delegátu.....	213
10.4.2	Vytvoření delegátu	213
10.4.3	Operace s delegátem	213
10.4.4	Anonymní metoda.....	214
10.4.5	Vícenásobné delegáty.....	215
10.4.6	Lambda-výrazy.....	215
10.5	Události	216
10.5.1	Delegát pro událost	216
10.5.2	Vyvolání události.....	217
10.6	Přetěžování operátorů.....	218
10.6.1	Základní pravidla.....	219
10.6.2	Deklarace přetíženého operátoru.....	219
10.6.3	Příklad: komplexní čísla.....	221
10.6.4	Indexování seznamu	226
10.6.5	„Operátory“ true a false.....	228

11	Hodnota null.....	233
11.1	Nulovatelné hodnotové typy	233
11.2	Referenční typy a null.....	235
11.2.1	Deklarace nulovatelného referenčního typu.....	236
11.2.2	Nulovatelnost typů.....	237
11.2.3	Kontexty nulovatelnosti.....	237
11.2.4	Význam kontextů nulovatelnosti.....	238
11.2.5	Podmíněný přístup ke složkám (operátory ?. a ?[])..	239

11.3	Nestandardní třída Optional<T>.....	240
11.3.1	Implementace.....	240
11.3.2	Použití třídy Optional<T>.....	241
11.3.3	Výhody a nevýhody.....	242

12	Ošetřování chyb v programu.....	243
12.1	Co dělat, když dojde k chybě?.....	243
12.2	Výjimka v C#.....	244
12.2.1	Třídy pro přenos informací o výjimkách.....	245
12.2.2	Vznik výjimky.....	245
12.3	Ošetřování výjimek.....	246
12.3.1	Syntax.....	246
12.3.2	Obsluha.....	247
12.3.3	Když vznikne výjimka.....	247
12.4	Pokročilejší možnosti.....	250
12.4.1	Pošli to dál.....	251
12.4.2	Vnitřní výjimka.....	251
12.4.3	Filtr.....	252
12.4.4	Koncovka.....	252
12.5	Složitější příklady.....	254
12.5.1	Ještě jednou vstup.....	254
12.5.2	Rovnost komplexních čísel.....	256
12.6	Aserce.....	258
12.6.1	Metody System.Diagnostics.Debug.Assert().....	258

13	13 Dotazovací jazyk LINQ.....	260
13.1	Základy jazyka LINQ.....	260
13.1.1	Zdroj dat.....	260
13.1.2	První dotaz.....	261
13.1.3	Složitější dotazy.....	263
13.2	Zdroje dat pro LINQ.....	265
13.2.1	Kolekce a pole jako zdroj dat.....	265
13.2.2	Vlastní třída jako zdroj dat.....	265
13.3	Dotazy v LINQ.....	266
13.3.1	Struktura dotazu.....	266
13.3.2	Spojování zdrojů dat.....	267
13.3.3	Pomocná proměnná.....	270
13.3.4	Řazení a seskupování.....	271
13.3.5	Agregace, transformace a další možnosti.....	276
13.3.6	Přístup k prvkům.....	277
13.3.7	Vytváření posloupností.....	279
13.3.8	Transformace posloupnosti.....	280
13.3.9	Množinové a související operace.....	281
13.3.10	Paralelní zpracování.....	281
13.4	Příklad: Úloha N dam.....	281
13.4.1	O co jde.....	282
13.4.2	Postup řešení.....	282
13.4.3	Třída Řešitel.....	283
13.4.4	Úloha N dam a jazyk LINQ.....	287

14	Práce se znakovými řetězci	290
14.1	Třída string a navazující nástroje.....	290
14.1.1	Vytvoření řetězce	290
14.1.2	Řetězcové literály.....	291
14.1.3	Operace se znakovými řetězci.....	292
14.1.4	Formátování.....	294
14.1.5	Další možnosti formátování	298
14.1.6	Standardní formátování data a času.....	299
14.1.7	Vlastní formát data a času	300
14.1.8	Normalizace znakových řetězců	301
14.2	Regulární výrazy.....	303
14.2.1	Nástroje pro práci s regulárními výrazy v C#	304
14.2.2	Regulární výraz.....	305
14.2.3	Ladění regulárních výrazů.....	310
15	Soubory, vstupy a výstupy.....	311
15.1	Třídy pro práci se soubory a proudy.....	311
15.2	Práce se soubory a adresáři	312
15.2.1	Práce se soubory.....	312
15.2.2	Práce s adresáři.....	312
15.3	Vstupy a výstupy.....	314
15.3.1	Čtení a zápis binárních dat	315
15.3.2	Čtení a zápis textových dat	319
15.3.3	Paměťové proudy.....	323
15.4	Serializace	325
15.4.1	Serializace našich vlastních datových typů.....	326
15.4.2	Serializace neserializovatelných objektů.....	327
16	Grafické uživatelské rozhraní (Windows Forms).....	330
16.1	První okno	330
16.1.1	Projekt okenní aplikace	330
16.1.2	Komponenty, jejich vlastnosti a události	333
16.2	Aplikace založená na knihovně Windows Forms	337
16.2.1	Nejdůležitější třídy	337
16.2.2	Okenní aplikace pro Windows	337
16.2.3	Předdefinované události.....	340
16.3	Úloha Nám v okně	342
16.3.1	Co budeme od programu požadovat	342
16.3.2	Základní třídy programu	343
16.3.3	Okno.....	343
16.3.4	Obsluha událostí.....	345
16.3.5	Řešení a jeho zobrazení.....	347
16.3.6	Odstraňujeme problémy	351
16.3.7	Prostředky	354
16.4	Nastavení (vlastní dialog)	356
16.4.1	Stránka Settings.settings	357
16.4.2	Nová nabídka	358
16.4.3	Dialog Nastavení: vizuální návrh	359

16.4.4	Dialog Nastavení: funkčnost.....	360
16.4.5	Úprava třídy Okno.....	363
17	Grafické uživatelské rozhraní (WPF).....	366
17.1	Některé důležité třídy.....	366
17.2	Obvyklá struktura programu s WPF.....	367
17.2.1	První program.....	367
17.2.2	Zdrojové soubory prvního programu.....	367
17.2.3	Jazyk XAML.....	368
17.2.4	Další soubory.....	371
17.3	Práce s oknem.....	371
17.3.1	Komponenty určující rozložení ovládacích prvků.....	372
17.3.2	Prostředky pro speciální grafické efekty.....	372
17.3.3	Styly.....	373
17.3.4	Spouště.....	374
17.4	Některé další možnosti.....	375
17.4.1	Vazba mezi ovládacími prvky.....	375
17.4.2	Transformace.....	377
17.5	Program bez XAML.....	378
18	Souběžné výpočty.....	380
18.1	Podproces.....	380
18.1.1	Třída Thread.....	380
18.1.2	Základní operace s podprocesy.....	382
18.1.3	Spánek a čekání na dokončení podprocesu.....	386
18.1.4	Pozastavení a obnovení podprocesu.....	387
18.1.5	Přerušování podprocesu.....	388
18.1.6	Násilné ukončení podprocesu.....	388
18.1.7	Podprocesy na pozadí.....	390
18.1.8	Priorita podprocesu.....	391
18.2	Synchronizace podprocesů.....	391
18.2.1	Problémy při sdílení prostředků.....	391
18.2.2	Zámek.....	392
18.2.3	Zámek v prostředí .NET.....	393
18.2.4	Atomické operace.....	398
18.3	Komunikace mezi podprocesy.....	398
18.4	Podprocesy a grafické uživatelské rozhraní.....	402
18.4.1	Přístup ke komponentám GUI z podprocesu.....	403
18.4.2	Výpočet na pozadí.....	405
18.5	Některé další nástroje.....	409
18.5.1	Fond podprocesů.....	409
18.5.2	Statická datová složka třídy vlastní podprocesu.....	412
18.5.3	Synchronizace celé metody.....	412
18.5.4	Třída Parallel.....	413
18.5.5	Asynchronní metody, asynchronní úlohy.....	415
	Literatura.....	419
	Rejstřík.....	420

Předmluva

Kniha, kterou jste právě otevřeli, vás seznámí se základy programování v jazyce C#. Je určena jak čtenářům, kteří ještě vůbec neprogramovali, tak i čtenářům, kteří již v nějakém jazyce programovali a chtějí se s tímto programovacím jazykem seznámit. Přitom předpokládám, že jste sice začátečníky v programování, ale umíte s počítačem zacházet uživatelsky, tedy že umíte spustit program, zkopírovat nebo smazat soubor atd. V průběhu osmnácti kapitol této knihy se vás pokusím dovést na úroveň lehce pokročilého programátora.

Programovací jazyk C#

Programovací jazyk C# uveřejnila firma Microsoft v roce 2002. I když byl původně určen pouze k programování pro Windows, v současné době se používá k vytváření aplikací i pro řadu dalších platforem, jako je Linux, macOS, Android a další a lze v něm vytvářet cloudové aplikace.

Co v této knize najdete

Jediným způsobem, jak se naučit nějaký programovací jazyk, je psát v něm programy, a proto už ve druhé kapitole začneme programovat. Cílem druhé, třetí a čtvrté kapitoly je uvést na scénu vybrané konstrukce jazyka C# a umožnit vám zkusit si vše, o čem si v následujících kapitolách povíme, na vlastních programech – jednoduchých a možná nešikovných, ale fungujících.

Než se ovšem pustíme do programování, musíme si ujasnit některé pojmy a sjednotit terminologii. Proto se v první kapitole seznámíme s pojmy, které budeme používat. Povíme si, co je třeba vědět o počítači, o programovacích jazycích, algoritmech, objektově orientovaném programování a některých dalších věcech, bez nichž se při programování neobejdeme.

Ve druhé kapitole napíšeme svůj první program, ukážeme si, jak ho přeložit – neboť jazyk, v němž ho píšeme, je jiný než jazyk, v němž „myslí“ počítač – a jak ho spustit. Seznámíme se s integrovaným vývojovým prostředím Visual Studio 2019, které budeme v této knize používat.

Ve třetí kapitole napíšeme několik verzí jednoduchých programů, které přečtou číslo, vypočtou z něj jistou hodnotu a výsledek vypíšou. To nám umožní seznámit se alespoň povrchně s některými základními programovacími konstrukcemi, s rozdělením programu do několika souborů, s pojmem projektu apod. Napíšeme také jednoduchou vlastní programovou knihovnu pro čtení dat z klávesnice. Se znalostmi z této kapitoly byste měli být schopni zkusit si vše, co se v dalších kapitolách naučíte, na vlastních programech.

Ve čtvrté kapitole se seznámíme s kolekce – nástroji, které umožňují uchovávat v programu libovolné množství dat –, seznámíme se velmi povrchně s generickými třídami a ukážeme si, jak ladit program – tedy jak zjistit, proč dělá něco jiného, než co jsme chtěli naprogramovat.

Od páté kapitoly začíná výklad „naostro“. Po seznámení se základními pojmy postupně poznáme vestavěné datové typy, příkazy jazyka C#, s výrazy a operátory, které máme k dispozici, naučíme se používat tzv. pole. Pak přijdou na řadu kapitoly věnované programátorem definovaným negenerickým objektovým typům, rozhraním, genericitě, delegátům a lambda-výrazům (to vše jsou zajímavé a užitečné konstrukce, které jazyk C# nabízí) a tzv. výjimkám (nástrojům pro ošetřování chyb za běhu). Samostatná kapitola je věnována tzv. nulovatelným typům. Nelze samozřejmě pominout nástroj označovaný zkratkou LINQ, který umožňuje získávat vybraná data z různých zdrojů – mohou to být už zmíněné kolekce, soubory a mnohé další.

Pak přijdou na řadu kapitoly věnované použití vybraných programových knihoven. Začneme nástroji pro práci se znakovými řetězci, tedy s textem. Pak se podíváme na vstupní a výstupní operace, na vytváření grafického uživatelského rozhraní pomocí knihovny Windows Forms a pomocí knihovny Windows Presentation Foundation a na nástroje pro paralelní počítání.

Na co se nedostalo

Víc se mi bohužel do této knihy nevešlo. Nedostalo se na tvorbu databázových aplikací, na tvorbu webových aplikací, na tvorbu mobilních aplikací, na práci s datovými typy, které nebyly v době překladu k dispozici (tzv. reflexi), na použití nespravovaného kódu a dynamických knihoven napsaných v jiných programovacích jazycích, na práci s místním nastavením (kulturou) a na mnoho dalších zajímavých a užitečných témat.

Nástroje

Výklad v této knize je založen na jazyce C# verze 8 publikované v roce 2019. Pro psaní, překlad, sestavení a ladění programů používám vývojové prostředí Visual Studio 2019 Community Edition, které lze zdarma získat na stránkách firmy Microsoft.

Není to ovšem jediná možnost: Dobrou alternativou je nástroj Visual Studio Code, který je také zdarma a který lze instalovat pod operačními systémy Windows 7, 8 a 10, Debian, Ubuntu, Red Hat, Fedora, Suse a macOS 10.10 a pozdějších. Jeho ovládání se poněkud liší od ovládání Visual Studia; to ale nic nemění na platnosti výkladu o jazyce C# a jeho knihovnách, které v této knize najdete.

Terminologie

V celé knize používám důsledně českou terminologii. Jsem přesvědčen, že použití vhodných českých názvů výrazně usnadní pochopení, oč jde. Anglické termíny samozřejmě uvádím ale spouň při prvním výskytu také.

Příklady

Výklad v této knize doprovází velké množství příkladů. Jejich zdrojové texty si můžete stáhnout z webových stránek nakladatelství Grada Publishing (vyhledejte na www.grada.cz stránku této knihy), nebo z mých osobních stránek, jejichž adresu najdete dále. V příkladech důsledně používám české identifikátory, a to včetně háčeků a čárek. Víím, že profesionální programátoři se tomu vyhýbají a že v mezinárodních týmech jsou anglické identifikátory samozřejmostí. Mám ale dlouholetou zkušenost, že při výkladu určeném začátečníkům mohou české identifikátory výrazně usnadnit orientaci v ukázkách zdrojového kódu – a to je můj hlavní cíl.

Na závěr

I přes veškerou péči, kterou jsem této knize věnoval, se do ní mohly vloudit chyby. Jestliže při studiu nějakou najdete, pošlete mi prosím zprávu na níže uvedenou adresu; bude-li to možné, uveřejním na svých webových stránkách opravu.

Miroslav Virius, Praha, 21. srpna 2020
miroslav.virius@fjfi.cvut.cz
<http://people.fjfi.cvut.cz/virius>

1 Než opravdu začneme

Než začneme programovat, měli bychom se seznámit s pojmy, které budeme později potřebovat. Povíme si krátce něco o nejen o počítačích a programovacích jazycích, ale také o algoritmech a objektově orientovaném programování. Mnohé z toho jistě znáte; přesto vám doporučuji tuto kapitolu alespoň zběžně přečíst, abychom si sjednotili terminologii.

1.1 Počítač

Kdesi – myslím, že šlo o návod, jak vyplnit daňové přiznání – jsem četl, že počítač je „stroj na zpracování informací“. To je samozřejmě pravda, ale člověk, který se chce naučit programovat, o něm musí vědět přece jen víc.

S trochou zjednodušení můžeme říci, že běžný počítač se skládá ze čtyř základních částí:

- Součást, která opravdu „počítá“, tedy zpracovává informace, a která také řídí činnost všech ostatních částí počítače, se nazývá *procesor*.
- *Operační paměť* slouží k přechodnému ukládání dat (informací), která počítač zpracovává, a programu, tedy příkazů, které určují, co má dělat. Používá se pro ni také anglická zkratka *RAM* (*Random Access Memory*, doslova „paměť s náhodným přístupem“). Vše, co je v této paměti, se při vypnutí počítače ztratí, „zapomene“.
- *Vstupní a výstupní zařízení* slouží počítači k výměně informací s okolím. Nejběžnější vstupní zařízení jsou klávesnice, myš, skener, zařízení na čtení CD/DVD atd. Nejobvyklejší výstupní zařízení osobních počítačů je obrazovka monitoru nebo tiskárna. Pro vstupní a výstupní zařízení se používá zkratka *V/V* nebo *I/O* (z anglického *input/output*).
- *Trvalá paměť* slouží – jak název napovídá – k trvalému ukládání dat a programů; data v ní se při vypnutí počítače neztrácejí. Má zpravidla mnohonásobně větší kapacitu než operační paměť, ale práce s ní je také mnohonásobně pomalejší než práce s operační pamětí (přibližně tisíckrát). Jako vnější paměť se používají převážně magnetické disky (tzv. pevný disk) nebo polovodičové disky (obvykle označovaný zkratkou *SSD* z anglického *Solid State Drive*). Vedle trvalé paměti napevno zabudované v počítači se často používá i externí trvalá paměť, kterou k počítači připojujeme pouze v případě potřeby, pro účely zálohování dat atd.
- Součást, jež propojuje všechny ostatní části počítače, se nazývá *sběrnice* (anglicky *bus*).

1.2 Operační paměť

Bity

Operační paměť je tvořena elektronickými obvody, které mohou mít dva dobře rozlišitelné stavy – např. vypnuto nebo zapnuto. Jeden z těchto stavů obvykle odpovídá číslici 0, druhý číslici 1. Jakékoli údaje do paměti proto můžeme zapisovat jen pomocí nul a jedniček, v tzv.

dvojkové soustavě. Každé místo, na které můžeme zapsat jednu číslici 0 nebo 1, označujeme jako *bit*. Operační paměť je tedy dlouhá řada bitů.

Bajty

S jednotlivými bity však pracujeme jen výjimečně, protože je to nepohodlné, a to jak pro člověka, tak pro počítač. Bity v operační paměti se v dnešních počítačích sdružují do skupin velikosti 8 bitů a tyto skupiny se nazývají *bajty* (anglicky *byte*, tj. slabika).

Adresa

Aby mohl počítač s pamětí snadno pracovat, jsou jednotlivé bajty, které paměť tvoří, očíslovány. Počáteční bajt má číslo 0, následující má číslo 1 atd. Toto pořadové číslo se nazývá *adresa* bajtu. (Na některých počítačích, včetně PC, je záležitost s adresami ve skutečnosti trochu složitější, ale to můžeme v souvislosti s jazykem C# ponechat stranou. Pro pochopení dalšího výkladu stačí, co jsme si řekli.)

1.3 Datové typy a proměnné

Není těžké zjistit, že nejmenší číslo, které může jeden bajt obsahovat, se skládá z osmi nul a představuje ve dvojkové i v desítkové soustavě nulu. Největší takové číslo se bude skládat z osmi jedniček a v desítkové soustavě vyjadřuje 255. To je pro téměř jakékoli počítání málo; v běžných programech se obvykle objevují daleko větší čísla. Proto se pro ukládání dat zpravidla používají různě velké skupiny za sebou následujících bajtů.

Ani to ovšem nestačí. Vezmeme-li např. dva za sebou následující bajty, můžeme do nich uložit celá čísla v rozmezí od 0 do 65 535 (od 0 do 1111 1111 1111 1111 ve dvojkové soustavě). Pokud nám ani to nevystačí, můžeme vzít skupinu 4 nebo třeba 8 bajtů. Jenže ani to neřeší problém, co dělat, když budeme potřebovat záporná čísla, reálná čísla, znaky nebo třeba logické hodnoty (jež vyjadřují, že nějaké tvrzení platí nebo neplatí).

Musíme tedy najít nějaký způsob, který nám umožní reprezentovat data různých „druhů“ v paměti počítače. Jinými slovy, musíme najít způsob, jakým určité hodnotě jednoznačně přiřadíme skupinu bitů, která bude tuto hodnotu představovat – jak tuto hodnotu v počítači *zakódovat*.

Datový typ

Můžeme se např. dohodnout, že bajt s hodnotou 65 bude představovat znak **A**. Táž skupina bitů může za jiných okolností také představovat celé číslo, které má v desítkové soustavě hodnotu 65. Bajt se stejnou hodnotou ale může být i součástí většího celku s naprosto jiným významem.

Odtud je vidět, že počítači nestačí znát adresu bajtu nebo bajtů, s nimiž pracuje. Vedle adresy musí vědět, jak velký úsek – kolik bajtů – má vzít a jak má jeho obsah interpretovat. Jinými slovy, musí znát *datový typ* hodnoty, která je tam uložena – musí vědět, zda jde o celé číslo, znak, logickou hodnotu atd.

Datový typ také určuje operace, které lze s danou hodnotou provádět. Celá čísla lze například sčítat a odečítat, znaky lze spojovat do *řetězců*, tedy do souvislého textu.

Proměnná

Až dosud jsme se hodnotami, uloženými v paměti, zabývali z hlediska počítače; nyní se na ně podíváme z hlediska programátora. Hodnotu, s níž budeme chtít ve svém programu pracovat, potřebujeme uložit do paměti: Musíme si vyhradit místo a říci, jakého typu budou údaje, které bude obsahovat. Takovéto místo pro ukládání hodnoty budeme nazývat *proměnná*.

Počítač bude s proměnnou zacházet pomocí její adresy (pořadového čísla jejího počátečního bajtu). Pro programátora by ale práce s adresou byla nepohodlná, a proto ji pojmenujeme, dáme jí *identifikátor*. Tomu se v programování říká *deklarace proměnné*.

1.4 Programy a programovací jazyky

Budeme-li od počítače chtít, aby zpracovával data, která mu předložíme, musíme mu také říci, co s nimi má vlastně dělat – musíme mu dát *program*. Program, podobně jako data, uložíme do operační paměti.

Procesor umí s daty provádět různé jednoduché operace. Umí např. sečíst, odečíst, vynásobit nebo vydělit dvě čísla, zadáme-li mu jejich adresy, umí vzít znak a zobrazit ho na monitoru atd. Protože však do jeho paměti nelze uložit nic jiného než čísla, musí být tyto příkazy vyjádřeny – zakódovány – také čísly. Číselné vyjádření instrukcí (příkazů) pro procesor se nazývá *strojový kód* (v programátorštině *stroják*) a je to jediná věc, které procesor rozumí.

Aby nebyl život příliš jednoduchý, používají různé procesory různé strojové kódy, takže programy ve strojovém kódu nejsou obvykle přenositelné mezi počítači s různými procesory.

Vyšší programovací jazyky

Je asi jasné, že programování ve strojovém kódu je velice namáhavé a nepřehledné. Také to téměř nikdo nedělá; místo toho se používají tzv. vyšší programovací jazyky, jako je Ada, Basic, C, C++, ... a také C#.

Program ve vyšším programovacím jazyce je textový soubor, který obsahuje popis řešené úlohy vyjádřený pomocí vybraných anglických slov a pomocí výrazů zapsaných podobně jako v matematice. Napsat program ve vyšším programovacím jazyce je samozřejmě daleko snazší než napsat odpovídající program ve strojovém kódu. Ovšem nic není zadarmo: Program ve vyšším programovacím jazyce nelze na počítači přímo spustit, neboť počítač mu nerozumí. Takovýto program se proto musí buď *přeložit* do strojového kódu, nebo *interpretovat*. V obou případech k tomu potřebujeme další program, který to za nás udělá.

Příklad

Textový soubor, který obsahuje zápis programu ve vyšším programovacím jazyce, se zpravidla nazývá *zdrojový kód* nebo *zdrojový program*, v programátorské hantýrce „*zdroják*“. K *překladu* do strojového kódu (hovoříme také o *kompilaci*), slouží program zvaný *překladač* neboli *kompilátor*. Často s ním spolupracuje ještě *sestavovací program* neboli *linker*, který dokáže spojit několik nezávisle přeložených částí programu do jednoho celku. Linker také připojí knihovny – části programu, které už někdo naprogramoval předem a které můžeme už jen používat. Dnes je však sestavovací program zpravidla součástí překladače.

Příkladem (a následným sestavením) programu vznikne soubor obsahující strojový kód, který již lze na cílovém počítači spustit. Mezi typické překládané programovací jazyky patří např. C, C++ nebo Pascal.

Interpretace

Místo překladačů však můžeme v některých případech použít speciální program, který bude číst zdrojový text a interpretovat ho, tj. provádět příkazy, které v něm najde. Typickým interpretovaným jazykem je klasický Basic.¹ Interpretované programy obvykle běží podstatně pomaleji

1 To se netýká současných verzí Visual Basicu; s programem v tomto jazyce se zachází stejně jako s programem v C#, jak o tom budeme hovořit dále.

než překládané programy. Vedle toho musíme na cílový počítač spolu s naším programem instalovat také interpretační program.

Jazyk C#

C# je tak trochu zvláštní případ. Zdrojový kód programu napsaného v tomto jazyce se přeloží, ovšem nikoli do strojového kódu počítače, ale do univerzálního pomocného jazyka označovaného *Common Intermediate Language (CIL*, dříve označovaný *Microsoft Intermediate Language, MSIL*, nebo jen IL). Ten se pomocí dalšího překladače převede do strojového kódu cílového počítače – ale zpravidla až v okamžiku, kdy program spustíme.

To znamená, že na cílovém počítači musí být překladač, který překládá z IL do strojového kódu. Tento počítač se nazývá JIT (z anglického *Just In Time*, neboť překládá právě v době, kdy je to potřeba). Věc je ale ještě složitější: Na cílovém počítači musí instalováno prostředí .NET, které kromě překladačů JIT obsahuje i další součásti potřebné pro běh programů vytvořených v jazyce C#.

Na závěr ještě doplníme, že překladač libovolného programovacího jazyka (i C#) zároveň kontroluje syntaktickou správnost programu – tedy zda je program napsán podle jistých formálních pravidel, která zaručují, že mu počítač porozumí. (Syntaktická správnost programu bohužel nezaručuje věcnou správnost programu, tj. nezaručuje, že program bude dělat to, co si přejeme.)

1.5 Operační systém

Počítač bez programového vybavení by nám nebyl mnoho platný. Proto se s ním zpravidla dodává alespoň jeden základní program, který se rozeběhne automaticky hned po spuštění počítače a běží po celou dobu jeho provozu. Tento program se nazývá *operační systém* a „oživuje“ počítač, tj. přijímá pokyny uživatele, stará se o jejich provedení a informuje uživatele o výsledcích. Pokyny mohou být vyjádřeny příkazem zapsaným v příkazové řádce, kliknutím myši na ikoně nebo jiným způsobem.

Operační systém má ovšem ještě řadu dalších úloh, z nichž pro nás nejdůležitější je, že poskytuje služby dalším programům. Stará se o jejich spouštění, o přidělování paměti, poskytuje nástroje pro práci se soubory atd. Když např. chceme v programu otevřít soubor, program předá odpovídající požadavek operačnímu systému a ten se postará o vše potřebné.

Mezi nejznámější operační systémy na osobních počítačích patří různé verze Windows a Linuxu, macOS, IOS a Android. Poznamenejme, že pro všechny tyto systémy lze v jazyce C# vytvářet programy, to znamená, že je pro ně k dispozici prostředí .NET.

1.6 Program a algoritmus

Jazyk C#, podobně jako ostatní programovací jazyky, slouží k zápisu programu. Už víme, že program je nějaký soubor instrukcí (příkazů), které počítači říkají, co má dělat.

Každý program představuje návod k řešení nějakého problému. Při programování si ale musíme uvědomit, že počítač neumí najít způsob, jak problém vyřešit: To musíme udělat my. Když už známe cestu, jak najít řešení, můžeme to naučit také počítač (musíme mu napsat program). Počítač umí jen to, co ho my nebo někdo jiný naučí, na co má program.