Datové typy a konverze

Úvod

Svět kolem nás je plný informací. Informace může být tak prostá, jako počet žáků ve třídě vyjádřený číslem nebo naopak velmi komplexní – například školní databáze studentů. Pro zpracování těchto informací v jazyce C# nestačí znát jen konkrétní data (jména a počty studentů), ale zároveň i vědět, jak s těmito daty pracovat. To je hlavním důvodem existence `datových typů` v jazyce C#.

Datové typy

Vzpomínáte si na proměnné z minulé kapitoly? Ke každé proměnné se v jazyce C# váže i tzv. `datový typ` (`data type`). Ten nám říká, jaké povahy proměnná je a zároveň tím i určujeme, jakým způsobem budeme s danými daty uvnitř proměnné pracovat. Datových typů je velké množství a my si pro začátek uvedeme jenom jeden základní - `int`. Datový typ `int` (z anglického integer) slouží k ukládání celých čísel. Nejlepší bude si vše ukázat na jednoduchém příkladu:

Deklarace proměnné

```
int x;
```

Deklarace konstanty

```
const int y = 5;
```

Tímto zápisem jsme založili proměnnou se jménem `x`. Druhá ukázka pak demonstruje založení konstanty s hodnotou `5`. Povšimněte si, že samotnému názvu proměnné `x` ještě předchází i onen datový typ `int` a v případě konstanty i klíčové slovo `const`. Naše první proměnná zatím nedrží žádnou konkrétní hodnotu. Pouze jsme vyjádřili, že tuto proměnnou chceme založit (`inicializovat`) a prozatím ji ponechat „prázdnou“. Tomuto procesu říkáme odborně `deklarace proměnné`. Každý příkaz v jazyce C# je pak vždy ukončen středníkem (`;`). Nyní už pojďme zkusit naší proměnné přiřadit i nějakou hodnotu:

Proměnná s přiřazením

```
int x = 1;
```

Touto změnou jsme kromě deklarace proměnné provedli i přiřazení hodnoty `1`. Je důležité si uvědomit, že když přiřazujeme hodnotu k proměnné, jsme vždy limitováni jejím datovým typem. V našem případě jsme omezeni pouze na celá čísla. Pokoušet se tedy například uložit do proměnné `x` hodnotu `3.1415` povede k chybovému hlášení z důvodu typové chyby. Omezeni jsme rovněž i pamětí počítače. Rozsah datového typu `int` je přibližně od mínus dvou miliard do plus dvou miliard. Pro ukládání větších čísel nebo čísel s desetinným rozvojem nám poslouží jiné datové typy.

> Zatímco v matematice znak „`=`“ vyjadřuje rovnost dvou výrazů (levé a pravé strany), v jazyce C# (a řadě dalších programovacích jazyků) tento znak znamená pokyn k „přiřazení pravé strany do levé“. Po (úspěšném) provedení takovéhoto řádku bude sice skutečně platit ona matematická rovnost mezi levou a pravou stranou, ale to je až přirozený důsledek přiřazení.

Kdybychom nyní chtěli, můžeme hodnotu proměnné opět změnit:

Změna hodnoty proměnné

```
// Deklarace a přiřazení k proměnné
int x = 1;

// Změna hodnoty proměnné
x = 32;
```

Pokud byla proměnná již deklarována, není nutné při práci s ní opětovně zmiňovat její datový typ. Pokud bychom tak učinili ve stejném kontextu (o tom více v dalších kapitolách), kompilátor by náš zápis vnímal jako snahu založit novou stejnojmennou proměnnou, což by opět vedlo k pádu. Po našich posledních úpravách bude tedy konečná hodnota proměnné `32`.

Nyní si představíme ještě jeden důležitý datový typ. Je jím `string`. Ten nám dobře poslouží při ukládání textových řetězců. Opět si vše ukážeme na příkladu:

Datový typ string (1)

```
string pozdrav = "Ahoj, jak se máš?";
```

Datový typ string (2)

```
// Deklarace
string pozdrav;

// Přiřazení
pozdrav = "Ahoj, jak se máš?";
```

Způsob zápisu je pořád stejný. Datový typ předchází názvu proměnné a poté je provedeno samotné přiřazení. Tento příkaz samozřejmě může být také opět rozdělen do dvou samostatných řádků (ukázka 2). Nová je pro nás pouze pravá část přiřazení. V jazyce C# vždy obalujeme textové řetězce do uvozovek (`"`). Dáváme tím najevo, že se nejedná o klasický přeložitelný kód, ale náš psaný text. Do datového typu `string` můžeme rovněž ukládat i čísla, protože ta jsou sama o sobě ostatně také jenom textovým řetězcem. Musíme je ale opět obalit do uvozovek tak, jak je to vidět v následující ukázce:

Číslo jako řetězec

```
// Celé číslo uložené jako textový řetězec
string y = "10";
```

Na ukázce číslo 2 je pak porovnání obou dvou přístupů:

Porovnání

```
int x = 10;
string y = "10";
```

> V budoucích kapitolách si ukážeme, jaký je mezi oběma přístupy rozdíl a jak přesně datové typy ovlivňují práci se samotnými daty uvnitř proměnných.

> V některých jazycích je rovněž povoleno psát textové řetězce mezi apostrofy ('). Jazyk C# toto neumožňuje, protože apostrof je vyhrazen pro práci s datovým typem char, který slouží k uložení pouze jednoho znaku:

Datový typ char

```
char znak = 'A';
```

Použití datových typů

Vhodná volba datového typu je zásadní pro funkci i čitelnost programu. Datový typ vyplývá z charakteru ukládaných informací a musí je co nejlépe vystihovat. Například:

Proměnná `pocatecniPismeno` musí mít typ `char` a nemůže mít typ `string` (ten je vyhrazen pro text a počáteční písmeno je JEDNO písmeno, ne text.

Proměnná `polomer` definuje vzdálenost bodů kružnice od středu, což je fyzikální veličina, která může nabývat desetinných hodnot, například `2.35` centimetru. Musí tedy mít desetinný typ (double, float nebo decimal - podle požadavků na rychlost nebo přesnost) a nemůže být `byte`, `int`

nebo `long`. Totéž se týká času, rychlosti, zrychlení, teploty, tepelné kapacity, proudu, napětí, atd.

Proměnná nadpis musí být typu `string` a nemůže být typu `char` - jedná se o více písmen za sebou.

Proměnná `pocetClenuDomacnosti` nemůže být jiný než byte nebo `int`, nikdy desetinný typ. Lidé nejsou ve zlomcích, stejně tak počet čísel, které jsou větší než `2.5`. Ani ta nemůžeme vyjádřit jinak, než celočíselným typem (`byte`, `int`, `long`, `uint`...)

Další datové typy

V předchozí sekci jsme si představili datové typy `int` a `string`. Nyní si naše portfolio rozšíříme ještě o několik dalších. Prvním takovým bude datový typ `double`. Ten se nám bude hodit při ukládání čísel s desetinným rozvojem. V jazyce C# oddělujeme desetinnou část čísla vždy tečkou (`.`):

Datový typ double

```
double d = 3.14;
```

Je nutné zmínit, že tento datový typ není úplně přesný a při dlouhém desetinném rozvoji nad 15 až 16 míst má tendence zaokrouhlovat. Nehodí se tedy například pro práci s penězi nebo kdekoliv jinde, kde je vyžadována vysoká přesnost. Právě pro tyto případy tu máme datový typ `decimal`:

Datový typ decimal

```
decimal x = 34.58m;
```

Za samotné desetinné číslo ještě v případě decimalu píšeme i suffix (příponu) `m` nebo `M`. Díky této příponě můžeme odlišit desetinné číslo typu `double` od desetinného čísla typu `decimal`. Kdybychom na konec čísla příponu nedoplnili, jazyk by naše počínání vnímal jako snahu přiřadit hodnotu typu `double` do proměnné typu `decimal`, což by vyústilo v chybové hlášení.

> Jakékoliv desetinné číslo bez přípony je díky `implicitní konverzi` bráno jako `double`. Kdybychom ale chtěli, můžeme i přesto příponu `d` nebo `D` doplnit. O implicitních konverzích a jejich dopadu na náš kód se dozvíme více později.

Double - přípona

```
double y = 16.73d;
```

Mezi datové typy s plovoucí desetinnou čárkou ještě řadíme typ jménem `float`. Proces založení je stejný jako u obou předchozích, mění se opět pouze přípona:

Datový typ float

```
float z = 8.763f;
```

Nyní by někoho mohlo zajímat, proč pro práci s desetinnými čísli máme 3 odlišné datové typy. Důvodem jsou různé přesnosti a rychlosti těchto typů. Jako nejpřesnější se osvědčil datový typ `decimal` s 28 až 29 desetinnými řády přesnosti. Daní za tuto přesnost je ale podle některých měření až 20x nižší rychlost oproti zbylým dvěma datovým typům. Zlatou střední cestou je typ `double`, který zachovává přesnost na 15 až 16 míst s přiměřenou rychlostí. Nejrychlejší je pak datový typ `float` s garantovanou přesností pouze na 7 řádů. Posledním naším datovým typem je `char`. Ten slouží k uložení pouze jednoho znaku. Ukládaný znak pak obalujeme vždy mezi dva apostrofy, jak bylo avizováno výše:

Datový typ char

```
char c = 'a';
```

> Tímto pochopitelně náš výčet datových typů nekončí. Další si ukážeme v budoucích kapitolách.

Typový systém

Jazyk C# používá `statický typový systém`, který byl popsán v předchozích sekcích. V praxi to znamená, že proměnné vždy vyžadují definovat datový typ (například tedy `int` nebo `string`), který je dále neměnný. Jakmile jednou proměnnou deklarujeme, není možné její datový typ změnit. Když bychom například chtěli do proměnné typu `int` uložit desetinné číslo `10.67`, dostaneme chybové hlášení.

Opakem je `dynamický typový systém`, který nás plně odstiňuje od toho, že proměnná nějaký datový typ vůbec má. Dynamické typování jde mnohdy tak daleko, že proměnné nemusíme ani deklarovat. Jakmile do nějaké proměnné uložíme hodnotu a jazyk zjistí, že nebyla nikdy deklarována, sám ji založí. Do jedné proměnné můžeme ukládat text a poté ji třeba přepsat celým číslem. Interpreter nebo kompilátor daného jazyka se s tím sám popere a vnitřně automaticky mění datový typ. Zástupci dynamicky typovaných jazyků jsou například `JavaScript` nebo `Ruby`.

Dynamické typování sice nyní vypadá jako velmi výhodné, ale zdrojový kód pak není možné automaticky kontrolovat proti typovým chybám. Pokud někde očekáváme textový řetězec a přijde nám tam místo toho číslo, odhalí se chyba až za běhu a v lepším případě program spadne a v tom horším nebude pracovat správně (mnohdy bez našeho vědomí). Ačkoliv tedy statické typování klade větší nároky na programátora, ochrání nás před nepříjemnými chybami, jejichž hledání a následné řešení často může zabrat i hodiny. Dynamické typování rovněž ubírá na celkové výkonnosti jazyka, protože automatická úprava typů za běhu je výpočetně náročná.

Na závěr je potřeba říct, že každý typový systém má své pro a proti. Jazyk C# je staticky typovaný, a proto pro nás bude klíčové, abychom problematiku datových typů dokonale chápali.

Konverze

Při programování velmi často vzniká potřeba převést proměnnou na jiný datový typ než ten stávající. Příkladem může být situace, kdy uživatelem zadaný řetězec (`string`) chceme převést na desetinné číslo (například `float`). Jak již bylo napsáno výše, po deklaraci proměnné je v jazyce C# nemožné její datový typ měnit. Pro účely převodu proměnné na jiný datový typ tedy musíme založit úplně novou pomocnou proměnnou a samotné `přetypování` (převod datového typu) poté provést za pomocí `implicitní konverze`, `explicitní konverze` nebo za pomocí `podpůrných tříd`. Vše si nyní pokusíme co nejjednodušeji vysvětlit.

 # Implicitní konverze

Tento druh konverzí se děje automaticky podle předdefinovaných pravidel. Pro demonstraci této látky trošku předběhneme tematický plán a předvedeme si tuto problematiku v kombinaci s operátorem součtu, kde se tento typ převodu děje nejčastěji:

Implicitní konverze

```
double x = 1 + 2.5; // x => 3.5
```

Na této ukázce sčítáme číslo 1 (datový typ `int`) s číslem 2.5 (typ `double`) a výsledek ukládáme do proměnné typu `double`. Za normálních okolností by nebylo možné provádět operaci součtu na dvou odlišných datových typech (`int` a `double`), ale právě díky `implicitní konverzi` se o převod proměnných na společný datový typ nemusíme starat a vše je vyřízeno za nás. Implicitní konverze dokonce probíhá i na samotném číslu 2.5, neboť není doplněno o žádnou z možných přípon (d - `double`, f - `float` nebo m - `decimal`). Výsledkem takovéto implictní konverze pak bude `double`, protože jakékoliv desetinné číslo bez přípony je převedeno právě na tento datový typ. Implicitní konverze jsou předem dány a tabulky těchto konverzí lze dohledat v dokumentaci jazyka C#.

V jazyce C# je dokonce možné psát i vlastní implicitní konverze. Touto problematikou se ale nebudeme v tomto materiále zabývat.

 # Explicitní konverze

`Explicitní konverze` se na rozdíl od té implicitní děje z naší vůle za použití kulatých závorek a námi požadavaného datového typu tak, jako je to vidno na ukázce níže:

Explicitní konverze

```
// Proměnná typu int
int i = 1;


// Explicitní konverze z intu do doublu
double d = (double)i; // d => 1.0d
```

> Explicitní konverze stejně jako ta implicitní musí být podporována daným datovým typem! Pokud se nám povede napsat konverzi, která není podporovaná, budeme na tuto skutečnost upozorněni chybovým hlášením.

 # Konverze za pomocí podpůrných tříd

Ke konverzi datových typů můžeme využít i tzv. `podpůrných tříd` a jejich metod. Třída a metoda jsou pro nás novým pojmem a budeme se jim podrobně věnovat v budoucích kapitolách. Prozatím si ukážeme, jak takováto konverze vypadá a přesné vysvětlení jednotlivých znaků ponecháme na později:

Podpůrné třídy

```
// Podpůrná třída Convert:
int i = Convert.ToInt32("45");
double d = Convert.ToDouble(11);

// Podpůrná třída Int32:
int i = Int32.Parse("14");
```

Tímto pochopitelně náš list podpůrných tříd nekončí. Další si ukážeme v budoucích kapitolách až se podrobněji seznámíme s metodami a třídami.

Shrnutí

Při založení (deklaraci) proměnné musíme vždy definovat i její datový typ, který je poté už neměnný. Datový typ určuje, jaký druh hodnoty můžeme do proměnné uložit. Zatím jsme si představili datové typy `int`, `string`, `double`, `float`, `decimal` a `char`. V další kapitole si představíme operátory a ukážeme si základní operace s proměnnými.

Revision #2
Created 2025-05-19 10:48:07 UTC by Magdalena Dobešová
Updated 2025-05-21 06:46:21 UTC by Magdalena Dobešová