Proměnné a datové typy

- základní datové typy v jazyce C#
- explicitní a implicitní deklarace proměnných
- konstanta
- pole (velikost pole, rozměr pole, index, asociativní pole)
- datové typy u databází
- označování proměnných
- operace s proměnnými a sestavování výrazů
- priorita operátorů

Základní datové typy

Konkrétní hodnoty rozsahu čísel si pamatovat nemusíte, je to tu, aby jste měli představu o tom, o jakých číslech mluvíte.

VB		**C#**	
Celočíselné			
`Byte`	zabírá 1 byte (0–255)		
`Integer`	zabírá 2 byty, číslo se znaménkem (-32768 až 32767)	`int`	zabírá 4 byty
`Long`	zabírá 4 byty, celočíselná hodnota (-2147483648 až 2147483647)	`long`	zabírá 8 bytů
Reálné			
`Decimal`	zabírá 14 bytů, číslo se znaménkem, lze ho použít jen pomocí Variant, jehož je podtypem		
`Single`	zabírá 4 byty, číselná hodnota (-3.402823E38 až –1.401298E-45 pro záporné hodnoty, 1,401298E-45 až 3,402823E38 pro kladné hodnoty)	`float`	zabírá 4 byty
`Double`	zabírá 8 bytů, obdoba `Single` s dvoujnásobnou přesností	`double`	zabírá 8 bytů
Logické			
`Boolean`	zabírá 2 byty, uchovává hodnotu `True` (pravda), nebo `False` (lež)	`bool`	zabírá 1 byte, logická hodnota – nabývá hodnot `true`, nebo `false`
Řetězcové			
`String`	řetězec libovolných znaků kódovaných pomocí ASCII, délka prakticky neomezena (asi 2 miliardy znaků); dělí se na řetězec pevné a dynamické délky	`string`	posloupnost znaků (řetězec), 2 byty na jeden znak
Nestandardní			

`Object`	zabírá 4 byty, obsahuje odkaz na objekt (adresu objektu v paměti)	`object`	podle přiřazené hodnoty přiřazuje i datový typ
			`object a;`
			`a = 1;`
			proměnná `a` je typu `Int32`

`Variant`	zabírá 16 bytů, obecný datový typ, který může obsahovat jeden z výše uvedených datových typů, proměnná tohoto typu může dynamicky měnit svůj datový typ		
`Date`	zabírá 8 bytů, obsahuje hodnotu odpovídající datumu a času (1.leden 100 až 31.prosinec 9999,00:00:00 až 23:59:59	`DateTime`	uchovává datum a čas

Explicitní a implicitní deklarace proměnných

Explicitní

- Předem určíme, jaké proměnné budeme používat.
- Pro proměnnou rezervujeme prostor v paměti pro uložení její hodnoty za běhu programu
- Explicitní deklarace proměnné se zapisuje před jejím vlastním použitím, zpravidla na začátku procedury události, programu, funkce.
- Ve VisualBasicu začíná deklarace příkazem `Dim` (v podstatě znamená dimenze nebo dimenzovat)
- V deklaraci můžeme sdělit jaký typ dat budeme do proměnné ukládat (`Integer`, `Double`, `String`,...)
- Neuvedeme-li ve Visual Basicu v deklaraci typ proměnné (typ dat) je automaticky použit typ `Variant` - ten může obsahovat data libovolného typu a velikosti
- Ve VisualBasicu můžeme vynutit použití explicitní deklarace pomocí příkazu **`Option Explicit`**

Příklad explicitní deklarace

Visual Basic	C#

`Dim Prijmeni As String`
`Prijmeni = "Karel IV"`
Explicitní deklarace proměnné `Prijmeni`, která je datového typu `String`, ve VisualBasicu.
	`string Prijmeni = "Karel IV";`
	Explicitní deklarace proměnné `Prijmeni`, která je datového typu `string`, v C#.

S takto deklarovanými proměnnými můžeme následně dále pracovat, např. přiřadit novou hodnotu:
`Prijmeni = "Novotný"`

Implicitní

- V případě, že přímo neuvedeme deklaraci proměnné, ale rovnou proměnnou použijeme (např. v přiřazení hodnoty, ve výrazu apod.), dojde k tzv. **implicitní deklaraci**
- V případě VisualBasicu se jedná o deklaraci proměnné bez příkazu `Dim`
- Implicitní deklarace znamená, že proměnnou přímo použijeme v programu (viz příklad)
- Implicitní deklarace má výhodu **rychlejšího zápisu** programového kódu, ale nese s sebou **určitá rizika**
  - hlavním rizikem je to, že při použití implicitních deklarací nepovažuje kompilátor jazyka případný překlep ve jméně za chybu, ale považuje překlep za deklaraci nové proměnné

- Příklad implicitní deklarace proměnných `Jmeno` a `Prijmeni` - tyto proměnné budou mít datový typ `Variant`:

```
' Implicitni deklarace
Jmeno = "Pepa"
Prijmeni = "Novak"
```

- V případě použití explicitní deklarace by náš příklad vypadal následovně:

```
' Explicitni deklarace
Dim Jmeno As String
Dim Prijmeni As String
Jmeno = "Pepa"
Prijmeni = "Novak"
```

- Implicitní deklarace zavádí do kódu nebezpečí toho, že nedojde k rozpoznání překlepu, např. mějme proměnnou `Jmeno`, ve které je uložena hodnota `"Pepa"`. Budeme chtít tuto hodnotu změnit (na hodnotu `"Karel"`), ale nevšimneme si toho, že jsme ve jméně proměnné udělali překlep:

```
' Nastavime promennou Jmeno na danou hodnotu
Jmeno = "Pepa"

' Chceme hodnotu zmenit, ale omylem udelame preklep a nevsimneme si toho
Jemno = "Karel"

' Misto toho, abychom zmenili hodnotu promenne Jmeno, vytvorili jsme dalsi
promennou Jemno s hodnotou "Karel". Promenna Jmeno zustala nezmenena!
```

- *C# tuto deklaraci proměnné **nepodporuje**. V C# lze použít **pouze explicitní** deklaraci proměnné!*

Konstanta

- Konstanty jsou symboly reprezentující v programu neměnnou hodnotu. Konstantě může být přiřazen řetězec, nebo číslo.
- Příklad konstant v jazyce C#:

```
const int months = 12, weeks = 52, days = 365;
```

- Uživatel si může nadefinovat svou konstantu příkazem `const`. Definice vlastních konstant má tuto syntaxi (ve VisualBasicu):

```
[Public|Private] Const název konstanty [As typ] = výraz

' např.:
Const Months As Integer = 12
Const Weeks As Integer = 52
```

```
Public Const Days As Integer = 365

Private Const MyTitle As String = "HELP"
```

Pole

- kolekce hodnot (čísel, textových řetězců, ...) daného datového typu
- **Velikost pole** - odpovídá počtu jeho prvků. U jednorozměrného pole je tedy dána maximální hodnotou indexu. U pole dvourozměrného odpovídá součinu maximálních hodnot obou indexů.

```
' Jednorozměrné pole - 3 indexy (0-2) => velikost pole = 3; jedna se v
podstate o 'seznam hodnot'
pole1(0) = 18
pole1(1) = 45
pole1(2) = 10

' Dvourozměrné pole - indexy 2x3 (0-1, 0-2) => velikost pole = 2*3 = 6; jedna
se v podstate o 'tabulku hodnot' s dvema radky a tremi sloupci
pole2(0,0) = 18
pole2(0,1) = 45
pole2(0,2) = 12

pole2(1,0) = 16
pole2(1,1) = pole(0,2)   ' tj. nastavi prvku s indexem (1,1) stejnou hodnotu
jakou obsahuje prvek s indexem (0,2) - tj. 12
pole2(1,2) = 60
```

- **Index** (také nazýván jako *klíč*) - jednoznačně identifikuje prvek v poli
  - většinou je indexem celé číslo, indexy většinou začínají číslem 0
  - pomocí indexu lze přistupovat ke konkrétnímu prvku v poli a dále s tímto prvkem pracovat (*"vrať mi **3**. prvek pole"*, *"nastav **60**. prvku v poli hodnotu XYZ"*)
  - jak se s indexy pracuje (ve VisualBasicu) ukazuje příklad u předchozího bodu (*Velikost pole*)
- **Deklarace pole**

VisualBasic	C#
`Dim MePole(10) As Integer` • Deklaruje pole s názvem `MePole` o velikosti 10 prvků, každý jeden prvek pole bude datového typu `Integer` • Pole může být indexováno od 0 nebo od 1. Způsob indexování závisí na nastavení příkazu `Option Base`. Není-li uveden příkaz `Option Base = 1`, začínají se všechna pole indexovat od nuly	`int[] mePole = new int[10];` • Deklaruje pole s názvem `mePole` o velikosti 10 prvků, indexem budou čísla od `0` do `9`, každý jeden prvek bude datového typu `Int32` (celé číslo o velikosti 4 byty)

- **Asociativní pole**
  - je pole jehož prvky nejsou indexovány pomocí posloupnosti celých čísel, ale pomocí *klíčů*. Klíčem může

být číslo (v nesekvenční posloupnosti), textový řetězec a jiné.

Příklad asociativního pole v jazyce PHP:

```
$foo = array(
    "a" => "1",
    "b" => "10",
    "c" => "100",
    1254 => "apple item",
    "name" => "John Doe"
);
echo $foo["c"] . $foo["b"] . $foo["a"] . $foo[1254] . $foo["name"]; // vypise
text 100101apple itemJohn Doe
```

Datové typy u databází

- **MySQL datové typy**

Číselné typy	
Název	**Rozsah čísel**
TINYINT	−128 až 127 nebo 0 až 255
SMALLINT	−32768 až 32767 nebo 0 až 65535
MEDIUMINT	−8388608 až 8388607 nebo 0 až 16777215
INT	−2147483648 až 2147483647 nebo 0 až 4294967295
DOUBLE	nejmenší nenulové hodnoty jsou ±2,2250738585072014E-308; největší nenulové hodnoty jsou −±1,17976931348623157E+308. Je-li sloupec UNSIGNED, jsou záporné hodnoty zakázané.
Řetězcové typy	
Název	**Velikost**
VARCHAR	řetězec s pevně danou délkou – od 0 do 255 znaků
TEXT	řetězec o velké velikosti
a spousta dalších řetězcových typů...	
Typy pro datum a čas	
DATE, TIME, DATETIME, TIMESTAMP, YEAR	typy pro reprezentaci datumu a času

Označování proměnných

- To jak proměnnou označíme (pojmenujeme) závisí na použitém programovacím jazyce. Obecně lze pojmenovat proměnnou podle následujících pravidel (tato pravidla většinou platí i pro pojmenování funkcí a procedur):
  - název proměnné (**identifikátor**) může být sestaven z libovolné kombinace písmen a číslic a případně dalších znaků (většinou se používá ještě podtržítko _)
  - název nesmí začínat číslicí

- rozlišují se malá a velká písmena
- jako identifikátor nesmí být použito klíčové (vyhrazené) slovo daného programovacího jazyka (`if`, `then`, `for`, `else`, ...)
- název by neměl obsahovat háčky a čárky (může způsobovat problémy)

Operace s proměnnými a sestavování výrazů

VisualBasic	C#	Popis	Příklad
Aritmetické operátory			
`+`	`+`	sčítání	`1 + 2`
`-`	`-`	odčítání	`2 - 1`
`/`	`/`	dělení	`30/10`
`*`	`*`	násobení	`3 * 3`
`Mod`	`%`	modulo – zbytek po celočíselném dělení	`7 Mod 3` výsledek bude `1`
`\`		celočíselné dělení	`7 \ 3` výsledek bude `2`
Relační (porovnávací) operátory			
Relační operátory vrací hodnotu `TRUE` , pokud vztah platí, jinak vrací `FALSE` .			
`<`	`<`	menší než	`1 < 2`
`>`	`>`	větší než	`2 > 1`
`<=`	`<=`	menší nebo rovno	`10 <= 10` , `20 <= 30` ,...
`>=`	`>=`	větší nebo rovno	`3 >= 3` , `10 >= 5` , ...
`<>`	`!=`	není rovno	`7 <> 3` ☐ výsledek bude `TRUE` , `5 <> 5` ☐ výsledek bude `FALSE`
`=`	`==`	je rovno	`10 = 10` ☐ výsledek bude `TRUE` , `5 = 2` ☐ výsledek bude `FALSE`
`Like`		prověří , zda se v řetězci nachází určitý vzorek	
Logické operátory			
`AND`	`&&`	logický součin	
`OR`	`||`	logický součet	

`XOR`	`^`	exkluzivní `OR` (*buď ..., anebo ...*)	`0 XOR 0` ☐ `0`
			`0 XOR 1` ☐ `1`
			`1 XOR 0` ☐ `1`
			`1 XOR 1` ☐ `0`

`NOT`	`!`	negace argumentu	`NOT TRUE` ☐ `FALSE` `NOT FALSE` ☐ `TRUE`

Priorita operátorů

Určuje pořadí vyhodnocování částí výrazu. Nejvyšší prioritu mají operátory aritmetické, pak následují relační a nejnižší prioritu u kombinovaných výrazů mají logické operátory. V rámci aritmetických operátorů mají vyšší prioritu * a `/` před + a `-`.