

Programování

Rekapitulace, co bychom doposud měli znát

Proměnná

- ▶ = pojmenované místo v paměti, které je určitého datového typu
- ▶ Název proměnné se může skládat z písmen, číslic a podtržítek
 - ▶ Název nesmí začínat číslicí **int 1jmenoUzivatele**; !
- ▶ Názvy proměnných jsou case sensitive (citlivé na velká malá písmena)
- ▶ Např.: **nazevpromenne** není stejná proměnná jako **nazevPromenne**
- ▶ Proměnou, kterou deklarujeme (definujeme) je třeba i inicializovat (přiřadit jí hodnotu)
 - ▶ Většina datových typů automaticky inicializuje hodnoty proměnných

Přehled vybraných datových typů

Datový typ	Uchovávaná hodnota	Výchozí hodnota	Velikost (B)
byte	Celočíselná hodnota	0	1
int	Celočíselná hodnota	0	4
long	Celočíselná hodnota	0	8
double	Desetinné číslo	0	8
float	Desetinné číslo	0	4
char	Znak	\0	2
bool	Pravdivostní hodnota (true/false)	false	1
string	Řetězec / pole znaků	null	18 + (2*x)

Není vhodné spoléhat na
obsah výchozích hodnot!

Deklarace a inicializace proměnných

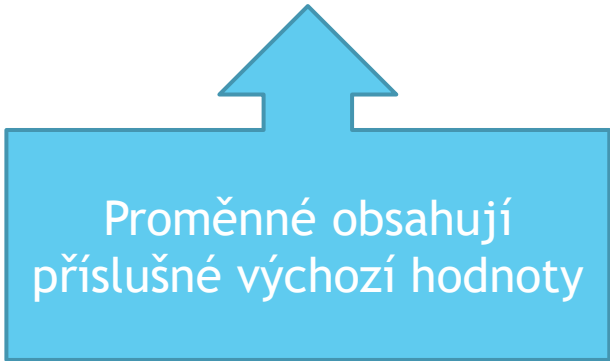
Deklarace s inicializací

```
int novaCiselnaPromenna = 120;  
double novaDesetinnaPromenna = 3.87;  
string retezec = "Ahoj Světe!";  
bool pravdivostniHodnota = false;  
char znak = 'g';
```

Deklarace bez inicializace

```
int novaCiselnaPromenna;  
double novaDesetinnaPromenna;  
string retezec;  
bool pravdivostniHodnota;  
char znak;
```

Proměnné obsahují
příslušné výchozí hodnoty



Proměnná - přiřazení hodnoty

- ▶ Využívá se operátoru **=**
- ▶ Vždy platí, že vlevo je proměnná a vpravo hodnota
 - ▶ **PROMĚNNÁ = HODNOTA** -> **takto ANO**
 - ▶ **HODNOTA = PROMĚNNÁ** -> **takto NE**
- ▶ Pozor ve vývojovém diagramu se používá pro přiřazení symbolu **:=**
- ▶ Operátor **=** se **nepoužívá**, pokud chceme zjistit, zda mají dvě proměnné stejnou hodnotu. Pro porovnání používáme **==**

Přehled vybraných operátorů

Operátor	Význam	Příklad použití v C#
+	Sečtení dvou čísel, případně spojení řetězců	A = 8 + 7; nebo text = "slovo1" + "slovo1";
-	Rozdíl dvou čísel	A = 13,7 - 6,3;
*	Násobení dvou čísel	A = 15 * 3,2;
/	Dělení dvou čísel (při celočíselném dělení se výsledek ořezává)	A = 8 / 4; nebo B = 7,6 / 3,12;
%	Modulo, zbytek po celočíselném dělení	A = 17 % 3;
++ / --	Zvýšení/snížení hodnoty o 1	Pocet++; nebo Pocet2--; nebo ++Pocet;
+=	Přičtení s přiřazením. Zkrácený zápis: promenna = promenna + 3;	A += 3;
-=	Odečtení s přiřazením. Zkrácený zápis: prom = prom - 8;	B -= 8;
&&	Logické násobení pravdivostních tvrzení. Obě musí platit pro <i>true</i>	Result = (a >= 1) && (b < 3);
	Logický součet pravdivostních tvrzení. Alespoň jedno musí platit pro <i>true</i>	Result = (c < -4) (d > 5);

Přetypování hodnot

- ▶ Přetypování by se dalo zjednodušit na pojem převedení, ale není to přesné!
- ▶ Přetypováním z jednoho datového typu na druhý může docházet k nežádoucí ztrátě informací v případě, že převádíme z přesnějšího na méně přesné
- ▶ Implicitní vs Explicitní přetypování
 - ▶ **Implicitní** - provedeno automaticky kompilátorem
 - ▶ **Explicitní** - vynucené převedení - může dojít ke ztrátě informace

```
// implicitní na proměnnou se kouká jako na desetinné číslo  
double a = 10;
```

```
// explicitní vynucené ořezání desetinného čísla b = 11;  
int b = (int)11.73;
```

Kolekce, řetězce

- ▶ Pokročilé datové struktury, které nám pomáhají jednodušeji operovat s daty
- ▶ **Kolekce** = struktury, které nám umožňují ukládat větší množství jednoduchých datových typů
 - ▶ **Pole** - statická kolekce, obsahující položky stejného datového typu
 - ▶ **List** (seznam) - dynamická kolekce, obsahující položky stejného datového typu
- ▶ **Řetězec** - struktura obsahující text, lze na něj pohlížet jako na pole znaků

```
// možnosti deklarace a inicializace pole

int[] pole1 = new int[10]; // prázdné pole o velikosti 10
int[] pole2 = { 1, 3, 5, 7, 9 };
int[] pole3 = new int[3];
pole3 = new[] { 1, 2, 3 };
int[] pole4 = new[] { 2, 4, 6, 8 };

// deklarace a inicializace řetězce
string text1 = "Text!!!";
```


Práce s konzolí

- ▶ Rozhraní konzole lze využít pro komunikace uživatel s aplikací
- ▶ Pomocí konzole můžeme vypisovat potřebné hodnoty případně získat vstupy

```
// načtení textu z konzole
string jmeno = Console.ReadLine();

// načtení čísla z konzole (nutné parsovat hodnotu)
int num = int.Parse(Console.ReadLine());

// vypsání textu pomocí konzole
Console.WriteLine("Text na výstup");

// vypsání čísla pomocí konzole
int cislo = 20;
Console.WriteLine(cislo);
```

Rozhodování o pravdivosti tvrzení

- ▶ V mnoha programovacích konstrukcích potřebujeme rozhodnout konkrétní výraz (podmínku) zda je pravdivá či nikoliv (*true/false*)
- ▶ Mimo operandů logického součtu (||) a součinu (&&) lze použít pro rozhodnutí o pravdivosti operátory <, >, <=, >=.
- ▶ Další možností rozhodnutí pravdivosti o nějakém tvrzení je porovnání hodnot pomocí == případně rozhodnutí o jejich nerovnosti !=
- ▶ Tvrzení lze negovat (obrátit pravdivostní hodnotu) pomocí operandu !

```
bool a0 = (10 == 10) && (3 < 4); // true
```

```
bool a1 = 10 < 7; // false
```

```
bool a2 = 10 > 7; // true
```

```
bool a3 = 13 == 4; // false
```

```
bool a4 = 1 != 0; // true
```

```
bool a5 = !false; // true
```

Rozhodovací podmínka - if else

- ▶ Základní konstrukcí pro řešení daného algoritmu je využít **if else**
- ▶ Podmínka může být úplná nebo částečná (není nutné implementovat else)
- ▶ Pro rozhodnutí, která část programu se má vykonat se rozhoduje na základě pravdivostní podmínky

```
// úplná podmínka, využíváme if i else část
if (num > 10)
{
    Console.WriteLine("num je větší než 10");
}
else
{
    Console.WriteLine("num je menší nebo rovno než 10");
}

// neúplná podmínka, využíváme pouze povinou if část
if (num == 10)
{
    Console.WriteLine("num je rovno 10");
}
```

Cykly s neznámým počtem opakování

- ▶ V případě, že potřebujeme opakovat nějakou část kódu, ale nevíme, kolikrát by se měla opakovat používáme konstrukce **while** nebo **do-while**

```
// konstrukce do-while, kód se alespoň jednou vykoná
do
{
    Console.WriteLine("Info");
} while (num > 10); // pokud je podmínka splněná, cyklus se opakuje

// konstrukce while, kód se nemusí ani jednou vykonat
while(num < 10) // pokud je podmínka splněna, cyklus se opakuje
{
    Console.WriteLine("Info 1");
}
```

Cyklus s pevným počtem opakování

- Pokud potřebujeme určitou část kódu vykonávat opakovaně a víme, kolikrát se má opakovat je vhodné využít konstrukci **for**, později si ještě řekneme o **foreach**, která je vhodná pro použití s kolekcemi

```
// pole prvních 5 lichých čísel
int[] pole = { 1, 3, 5, 7, 9 };

// pro deklarovanou řídicí proměnnou i, kontrolujeme zda je menší než 5
// pokud ano, provede se kód výpisu a dojde ke zvětšení i
for(int i = 0; i < 5; i++)
{
    Console.WriteLine(pole[i]);
}
```

Rozhodování podle hodnoty proměnné

- V momentě, kdy chceme na základě hodnoty v proměnné rozhodnout, která část kódu by se měla vykonat používáme konstrukci **switch-case**

```
int znamka = int.Parse(Console.ReadLine());  
// známka je sledovaná proměnná  
switch (znamka)  
{  
    case 1:  
        Console.WriteLine("Student prospěl - výborný");  
        break;  
    case 2:  
        Console.WriteLine("Student prospěl - chvalitebný");  
        break;  
    case 3:  
        Console.WriteLine("Student prospěl - dobrý");  
        break;  
    case 4:  
        Console.WriteLine("Student prospěl - dostatečný");  
        break;  
    case 5:  
        Console.WriteLine("Student neprospěl - nedostatečný");  
        break;  
    default:  
        Console.WriteLine("Neznámé hodnocení");  
        break;  
}
```

Funkce

- ▶ Každá funkce, ať již předdefinovaná nebo námi vytvořená musí mít:
 - ▶ Návratový typ (void, int, bool, double[], ...)
 - ▶ Název
 - ▶ Viditelnost
 - ▶ Seznam vstupních argumentů - může být prázdný
- ▶ Pokud není návratový typ void (nic nevracíme) funkce musí obsahovat instrukci **return** pro vrácení hodnoty, kterou pomocí funkce získáme
- ▶ Funkce používáme pro členění aplikace do menších lépe testovatelných částí, pro zpřehlednění a pro omezení opakování kódu
- ▶ Duplicita kódu je nežádoucí!

// funkce je všude viditelná, vrací int, má dva vstupní celočíselné argumenty

Počet odkazů: 0

```
public int SoucetCisel(int a, int b)
{
    return a + b;
}
```

// funkce je privátní (viditelná pouze v tomto souboru) a nevrací žádnou hodnotu

Počet odkazů: 0

```
private void Info()
{
    Console.WriteLine("Informace");
}
```

// privátní metoda vracející pravdivostní hodnotu

// má jeden číselný a jeden řetězcový argument

Počet odkazů: 0

```
private bool Adult(int vek, string jmeno)
{
    if(vek > 18)
    {
        Console.WriteLine("{0} je dospělý člověk", jmeno);
        return true;
    }
    else
    {
        Console.WriteLine("{0} není dospělý člověk", jmeno);
        return false;
    }
}
```




Prostor pro dotazy