

# Programování

Hezky pěkně pomalu a od začátku



Starý známý  
se vrací

# Tělo kódu aneb kam napíšeme středník (;)?

- ▶ Středníkem ukončujeme samostatný příkaz:
  - ▶ deklarace proměnné, načtení/výpis vstupů, vložení hodnoty do proměnné, volání funkce, ...
- ▶ Středník naopak nepoužíváme v případě, kdy je použita konstrukce případně funkce ohraničená { }:
  - ▶ - tělo části, podmínky, cykly, funkce, ...

```
Počet odkazů: 0
static void Main(string[] args)
{
    int a, b, c = 5;
    a = int.Parse(Console.ReadLine());
    b = a + c;
    while (a > 0)
    {
        Console.WriteLine("decrement");
        a--;
    }
    if (b > c)
    { Console.WriteLine("B is bigger"); }
    else
    { Console.WriteLine("C is bigger"); }

    for (int i = 0; i < c; i++)
    {
        Console.WriteLine("Ahoj");
    }
    Function("text");
}
Počet odkazů: 1
private static void Function(string v)
{
    Console.WriteLine(v.Length);
}
```

## Kód jako neposlušné ovečky

- Pokud chceme (a to vážně chceme), aby se nám provedl příslušné instrukce je třeba je umístit na správné místo
- Kód, který má vykonat příslušná funkce je potřeba umístit do těla funkce
- Tělo funkce jsou { } a slouží jako pomyslná ohrádka pro naše ovečky (řádky kódu)
- Jakmile nám naše ovečky utečou mimo ohrádku, nejsme dobrými pastýři ani programátory

```
Počet odkazů: 0
class Program
{
    Počet odkazů: 0
    static void Main(string[] args)
    {
        int a, b, c = 5;
        a = int.Parse(Console.ReadLine());
        b = a + c;
        while (a > 0)
        {
            Console.WriteLine("decrement");
            a--;
        }
        if (b > c)
        { Console.WriteLine("B is bigger"); }
        else
        { Console.WriteLine("C is bigger"); }
        Function("text");
    }
    Počet odkazů: 1
    private static void Function(string v)
    {
        Console.WriteLine(v.Length);
    }
}
for (int i = 0; i < c; i++)
{
    Console.WriteLine("Ahoj");
}
}
```

Utekly

# Proměnné v programu

- ▶ Proměnná je pojmenované místo v paměti obsahující nějakou informaci
- ▶ Pomocí datových typů můžeme specifikovat o jakou informaci se jedná
  - ▶ Číselné hodnoty - short, long, byte, **int**, float, **double**
  - ▶ Textové hodnoty - char, **string**
  - ▶ Pravdivostní hodnoty - **bool**
- ▶ Mimo základních datových typů může být proměnná i jiného typu:
  - ▶ Struktura, enumerace, objekt třídy
- ▶ Ať již je proměnná jakéhokoliv typu, vždy musí dodržovat pravidla pro pojmenování
  - ▶ Nesmí začínat číslicí, nesmí obsahovat mezery a diakritiku, nesmí být pojmenovaná jako některé z vyhrazených slov (for, if, while, ...)

# Proměnné v kódu

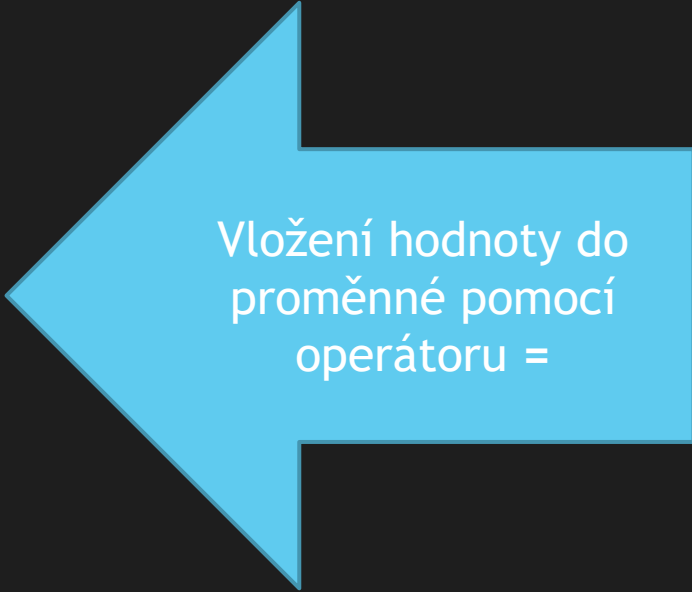
```
// celočíselní hodnoty
short num = 8;
int num1 = 20 * 71;
long num2 = -189165164;

// desetinné číslo
float partial = 8.73f;
double partial1 = 8.73;

// znak
char letter = 'X';

// řetězec
string text = "ahoj světe";
string text1 = "krásný" + " a " + "radostný " + "den!"; // řetězec

// pravdivostní hodnoty
bool resolution = true;
bool resolution1 = 18 > 3;
```



Vložení hodnoty do  
proměnné pomocí  
operátoru =

# Operace se vstupem a výstupem

## Vstup z konzole

- ▶ `Console.ReadLine()` nám přečte celý řádek až do místa, kdy zmáčkne Enter
- ▶ Funkce `ReadLine()` nám vrací data ve formě řetězce (`string`)
- ▶ Pokud potřebujeme jiný datový typ používáme příslušnou metodu `Parse()`

```
// vstup je ve formě řetězce (string)
string vstup = Console.ReadLine();

// očekáváme na vstupu jiný typ, použijem Parse()
int input = int.Parse(Console.ReadLine());
double input1 = double.Parse(Console.ReadLine());
char input2 = char.Parse(Console.ReadLine());
bool input3 = bool.Parse(Console.ReadLine());
```

## Výstup z konzole

- ▶ Vstupním parametrem funkce `Console.WriteLine()` je řetězec nebo proměnná, kterou lze automaticky převést na řetězec
- ▶ `Console.Write()` po vypsání nepřidává odřádkování

```
// vypsání hodnoty proměnné a odřádkování
Console.WriteLine(vstup);

// vypsání hodnoty (dojde k převedení na string)
Console.Write(input);

// možnosti formátování výstupu
Console.WriteLine("P1: {0}, P2: {1}", input1, input2);
Console.Write("Proměnná" + input3 + "na výstup");
```

# Operátory a priorita

Základní operátory	Relační operátory	Logické operátor
+ sčítání / spojení	== rovná se	&& a zároveň (and)
- odčítání	< menší než	nebo (or)
/ dělení	> Větší než	! Negace (not)
* násobení	<=	
% modulo	>=	
^ mocnina	!= nerovná se	

- inkrement (++), dekrement (--), přiřazení (=)
- Priorita operátorů je stejná jako v matematice



# Malá odbočka - logika

A	B	A & B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A   B
True	True	True
True	False	True
False	True	True
False	False	False

A	! A
True	False
False	True

A	B	A => B
True	True	True
True	False	False
False	True	True
False	False	True

IMPLIKACE

A	B	A <=> B
True	True	True
True	False	False
False	True	False
False	False	True

EKVIVALENCE

# Základní konstrukce - rozhodovací blok

- ▶ Zkráceně hovoříme o **if-else** bloku
- ▶ Konstrukce nám umožňuje rozhodnout se, kterou část kódu má vykonat na základě pravdivostní podmínky (bool)
- ▶ If-else blok musí mít vždy blok kódu pokud je podmínka splněna
- ▶ Pokud podmínka splněná není provede se část kódu, která následuje za klíčovým slovem **else** (tato větev není povinná)

```
if(input == 10)
{
    Console.WriteLine("Vstup je roven 10");
}

if(input1 >= 0 )
{
    Console.WriteLine("Číslo je kladné");
}
else
{
    Console.WriteLine("Číslo je záporné");
}

if (true)
{
    Console.WriteLine("Pravda");
}
```

# Základní konstrukce - cykly s neznámým počtem opakování

- ▶ Cykly while a do-while
- ▶ Používáme v momentě, kdy není jasné kolikrát se má určitá část kódu opakovat
- ▶ O tom, zda se má tělo vykonat rozhoduje pravdivostní podmínka na konci nebo začátku cyklu
- ▶ Rozdíl mezi těmito cykly je v tom, kdy se kontroluje podmínka pro další pokračování

```
Random randomNumber = new Random();
int result = randomNumber.Next(1, 11);

while ((result%3)!=0)
{
    Console.WriteLine("{0} není dělitelné 3", result);
    result = randomNumber.Next(1, 11);
}

do
{
    Console.WriteLine("Číslo 7 nenalezeno");
    result = randomNumber.Next(1, 11);
} while (result != 7); // zde ovšem středník na konci bude
```

# Základní konstrukce - cykly s pevným počtem opakování

```
string[] pole = { "ahoj", "jak", "se", "mas" };  
Console.WriteLine("-----");  
foreach (string s in pole)  
{  
    Console.WriteLine("Výpis: {0}", s);  
}  
Console.WriteLine("-----");  
for (int i = pole.Length - 1; i >= 0; i--)  
{  
    Console.WriteLine("Výpis: {0}", pole[i]);  
}  
Console.WriteLine("-----");
```

```
-----  
Výpis: ahoj  
Výpis: jak  
Výpis: se  
Výpis: mas  
-----  
Výpis: mas  
Výpis: se  
Výpis: jak  
Výpis: ahoj  
-----
```

- ▶ Cykly **for** a **foreach**
- ▶ Nejvhodnější použití v momentech, kdy víme, kolikrát se má příslušný kód opakovat
- ▶ Počet kroků je hlídán pomocí tzv. indexu, který iteruje (zvyšuje/zmenšuje se)
- ▶ Cyklus **foreach** nám umožňuje iterovat skrze jednotlivé prvky v datové struktuře (pole, list, ...) bez řídicí proměnné
- ▶ Cyklus **for** musí obsahovat řídicí proměnnou, kterou můžeme uvnitř těla využívat

# Základní konstrukce -rozhodovací blok s více možnostmi

- ▶ Zkráceně hovoříme o **switch-case** bloku
- ▶ Používáme v momentě, kdy máme více než dvě možnosti jak se zachovat, pokud proměnná obsahuje konkrétní hodnotu
- ▶ Sledované proměnná může být libovolného typu - číslo, řetězec, enumerace, objekt, ...
- ▶ Jednotlivé případy jsou označené klíčovým slovem **case**
- ▶ Každý případ je ukončen řídícím příkazem **break**, který ukončí další hledání možností a opustí switch-case blok - bez něj by vykonal kód následujícího případu

```
Random randomNumber = new Random();
int result = randomNumber.Next(1, 5);

switch (result)
{
    case 1:
        Console.WriteLine("Instrukce pro číslo 1");
        break;
    case 2:
        Console.WriteLine("Instrukce pro číslo 2");
        break;
    case 3:
    case 4:
    case 5:
        Console.WriteLine("Instrukce pro číslo 3,4,5");
        break;
}
```

# Funkce - vytváření a volání

- ▶ Knihovny C# obsahují spoustu již definovaných funkcí
- ▶ Přesto lze vytvářet funkce vlastní
- ▶ Obecná funkce musí mít viditelnost, návratový typ, název a volitelně vstupní proměnné
- ▶ Pokud chceme vytvořit funkci, kterou chceme volat z hlavní funkce, je třeba ještě uvést modifikátor **static** za viditelnost
- ▶ Funkce prozatím píšeme mimo hlavní funkci Main
- ▶ Později budou vytvářené funkce součástí vytvářené třídy nebo struktury

```
Počet odkazů: 0
private static void Function1(string v)
{
    Console.WriteLine(v.Length);
}

Počet odkazů: 0
private int Function2(string v)
{
    return v.Length;
}
```

Nelze volat z Main()

```
Počet odkazů: 0
static void Main(string[] args)
{
    Function1("Vstupní text");

    int delka = Function2("Vstupní text");
}
```