Programování

Letem světem programkem

Deklarace a inicializace proměnné

- Proměnná je pojmenované místo v paměti
- Určením datového typu proměnné víme, jak s ní pracovat
- Pokud pomocí přiřazovacího operátoru nenastavíme počáteční hodnotu, nelze na 100% určit, jaká hodnota se zde nachází

```
// 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!"; // ř

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

Načtení a výpis hodnoty z konzole

- Musíme mít na paměti, že konzole pracuje s textovým řetězcem
- Načtená hodnota je datového typu string
- Pro získání hodnoty z daného řetězce lze využít příslušné funkce Parse() nebo konverze pomocí ConvertToX()
- Pří výpisu do konzole kompilátor sám převádí netextové hodnoty na řetězec

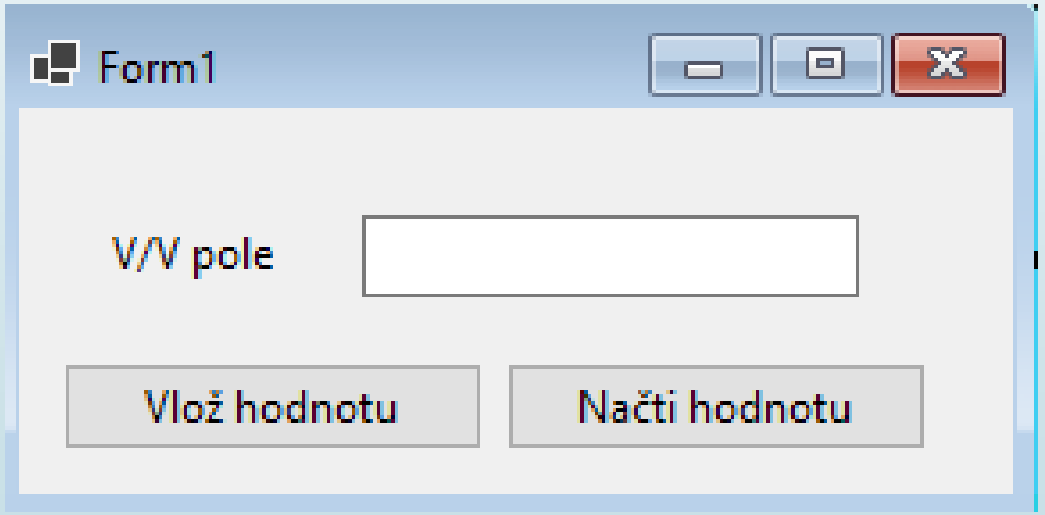
```
// vstup je ve formě řetězce (string)
string vstup = Console.ReadLine();

// očekáváme na vstupu jiný typ, použíjem 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());
```

```
// 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");
```

Načtení a výpis hodnoty ve formulářové aplikaci

- Pro zápis a čtení využíváme parametru **Text** příslušné komponenty
- Většina komponent také pracuje s textem jako hodnotou a je nutné převedení na potřebný datový typ
- Pro rychlé převedení hodnoty na řetězec využíváme metody ToString()

```
Počet odkazů: 1
private void button1_Click(object sender, EventArgs e)
{
    Random rnd = new Random();
    textBox1.Text = rnd.Next(1, 100).ToString();
}

Počet odkazů: 1
private void button2_Click(object sender, EventArgs e)
{
    string vstup = textBox1.Text;
    MessageBox.Show("Načtená hodnota " + vstup);
}
```



Základní programovací konstrukce
if-else 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)
- Podmínka musí mít podobu výrazu, u kterého lze rozhodnout, zda je pravdivý či nikoliv

```
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 programovací 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;
}
```

Základní programovací 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 programovací konstrukce
cykly s pevným počtem opakování

- 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 řídící proměnné
- Cyklus **for** musí obsahovat řídící proměnnou, kterou můžeme uvnitř těla využívat

```
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("-----------------");
```

Datové struktury - pole

- Statická homogenní datová struktura
  - umožňuje uchovávat více hodnot stejného datového typu
  - Po stanovení velikosti pole již nelze v průběhu měnit počet prvků
- Dle datového typu, jednotlivých položek pole lze využívat speciální funkce – řazení, suma, průměr, vyhledávání, …
- Pro práci s konkrétní položkou musíme uvést její index (začínáme od 0)

```
double[] novePole = new double[4];

bool[] pravdivostniPole = { true, false, true, true, false };

string[] poleSlov = { "ahoj", "svete", "jsem", "tady" };

int[] poleCisel = new int[3];
poleCisel[0] = 1;
poleCisel[2] = 4;
```

Datové struktury - list

- Dynamická homogenní datová struktura
- Oproti poli nedefinujeme jeho velikost – ta se mění podle počtu prvků
- Pro práci s listem využíváme funkce **Add()** pro přidání a **Remove()** případně **RemoveAt()** pro odebrání – nová položka se přidává vždy na konec
- Pro editaci prvku na příslušném poli využíváme indexace stejně jako u pole
- Pokud nám stačí zjistit hodnotu na příslušném indexu, lze využít funkce **ElementAt()**

```
List<int> listCisel = new List<int>();

listCisel.Add(1);
listCisel.Add(17);
listCisel.Add(1);
listCisel.Add(3);

listCisel.Remove(1);
listCisel.RemoveAt(2);

listCisel[1] = 500;
```

Ošetření výjimek a chyb v chodu programu

- Pro kritickou sekci kódu je vhodné mít vymyšleno, co se má stát pokud se vyskytne problém
- **try-catch** funguje jako pískoviště, kde si v sekci **try** necháme proběhnout kód, kde se může objevit výjimka (Exception) v části **catch** pak říkáme, co se má stát
- Mimo výjimky, které již existují lze vyhazovat i výjimky vlastní pomocí **throw**

```
int[] array = { 1, 2, 3, 4 };

try
{
    Console.WriteLine(array[5]);
}
catch (IndexOutOfRangeException e)
{
    Console.WriteLine("Nedostupný index pole");
}

throw new Exception("Stejně vytvoříme chybu");
```