Jazyk C# 1

2. seminář

Jakub Večeřa

Univerzita Palackého v Olomouci

4. 10. 2024

Reakce na úkoly

- Boje se systémem na odevzdání?
- Chat
- Metoda s chybami

Základní datové typy

Klíčové slovo	Rozsah	Popis
bool	`true, false`	Pravdivostní hodnoty
sbyte	$\langle -128, 127 \rangle$	8bitové číslo se znaménkem
byte	$\langle 0, 255 \rangle$	8bitové číslo bez znaménka
short	$\langle -32768, 32767 \rangle$	16bitové číslo se znaménkem
ushort	$\langle 0, 65535 \rangle$	16bitové číslo bez znaménka
int	$\langle -2147483648, 2147483647 \rangle$	32bitové číslo se znaménkem
uint	$\langle 0, 4294967295 \rangle$	32bitové číslo bez znaménka
long	$\langle -9223372036854775808, 92 \ldots 7 \rangle$	64bitové číslo se znaménkem
ulong	$\langle 0, 18446744073709551615 \rangle$	64bitové číslo bez znaménka
char	U+0000 až U+ffff	Unicode znak
float	$\langle -3.4 \cdot 10^{38}, 3.4 \cdot 10^{38} \rangle$	32bitové číslo s plovoucí řádovou čár
double	–	64bitové číslo s plovoucí řádovou čár
decimal	–	128bitové číslo se znaménkem
string	dle paměti systému	Řetězec znaků Unicode

Deklarace a inicializace proměnných

- Deklarace: `datový typ` + `název proměnné`;

```
1 int cislo;
2 string retezec;
```

- Deklarace s inicializací: `datový typ` + `název proměnné` = `hodnota`;

```
1 int cislo = -34;
2 string retezec = "Nejaky text";
```

- Více proměnných:

```
1 int cislo = 2, dalsiCislo = 42, jesteJedno = 33;
```

- Platnost proměnných pouze v bloku, kde vznikly

Automatická inference typu, konstanty

- Automatická inference typu: `var π = 3.14;` proměnná π bude typu `Double`
  - Proměnná musí být inicializována (kompilátor by nepoznal typ)
  - Nesmí se přiřazovat `null`
- Sice .NET podporuje Unicode, ale asi není dobrý nápad používat π jako název proměnné
- Konstanty: `const int x = 15;` – nejde změnit hodnota,
  - Musí být inicializovány při deklaraci
  - Přiřazovaná hodnota musí být dostupná v době kompilace

Přetypování – převod mezi datovými typy

- Občas potřebujeme převést jeden datový typ na druhý
- Implicitní(automatický) převod mezi (číselnými) datovými typy
  - Dochází u převodu „menšího typu“ do „většího“
  - Nedojde ke ztrátě dat
- Explicitní převod
  - Kompilátor nás nenechá vložit „větší typ“ do „menšího“
  - Musíme výslovně říci, že souhlasíme se ztrátou dat
  - Pozor na možné nestandardní chování

```
1 int a = -32;
2 double b = a; // implicitne pretypovani
3 uint c = (uint) a; // nutne explicitni
4 uint d = 2147483650;
5 a = (int)d;
```

- Některé typy neleze mezi sebou převést ⇒ chyba kompilátoru

Komentáře

- Jako ve většině programovacích jazycích můžeme (různě) komentovat kód
- Řetězce sloužící jako informace/poznámky pro programátory $\Rightarrow$ čitelnost kódu
- Před kompilací odstraněny preprocesorem, neovlivňují výsledný program
- Jednořádkový komentář pomocí `//` vše za lomítky je bráno jako komentář
- Blokový komentář uvozen `/*` a ukončen `*/`

```
1 // Komentar od zacatku radku
2 int x = fib(36); // Komentar za kodem
3
4 /* Tady probiha inicializace dulezitych promennych
5 a ... delka strany
6 v ... vyska trojuhelniku
7 */
```

Operátory

- Konstrukty jazyka poskytující základní operace s datovými typy
- Pevně daná arita (unární, binární, ternární)
- Daná priorita operátorů (přirozená)
  - `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/#operator-precedence`
- Jdou uživatelsky definovat (tzv. překrýt, přetížit – někdy příště)

Aritmetické operátory

- Unární: `++`, `--`, `+`,`-`
  - Inkrementace, dekrementace, hodnota operandu, opačné číslo
  - Pozor na pořadí inkrementace a dekrementace

```
1 int a = 5;
2 Console.WriteLine(a++);
3 Console.WriteLine(++a);
```

- Binární: `*`, `/`, `%`, `+`, `-`
  - Násobení, dělení, zbytek po dělení, sčítání, odčítání
  - Pozor na datové typy při dělení (ukázka)
- „Zrychlené“ přiřazení výsledku po operaci

```
1 x+=y; // Pricte y k x a vysledek ulozi do x
2 x*=y; // Vynasobi x a x a vysledek ulozi do x
```

Operátory

- Porovnání – výsledkem je `true` nebo `false`
  - `==`, `!=`, `<`, `>`, `<=`, `>=`
- Logické operátory – manipulace s datovým typem `bool`
  - Unární: `!` – negace
  - Binární líné – `&&`, `||`
  - Binární „pilné“ – `&`, `|`, `^`
  - Priorita (zleva): `!`, `&`, `|`, `^`, `&&`, `||`

Intermezzo – zápis čísel

- Hodnoty čísel můžeme zapsat několika způsoby (vhodnost)
  - Dekadický zápis (klasický)

```
1 uint cislo = 9872;
```

  - Šestnáctkový (hexadecimální) – někdy vhodnější – masky, barvy

```
1 byte cislo = 0xFF;
```

  - Binární – masky, binární manipulace s čísly

```
1 int cislo = 0b00111000001010100001000;
```

- Pro přehlednost možno dodat podtržítka (libovolně!)

```
int cislo = 0b1111_1100_0101;
int dalsiCislo = 123_456_78_910;
```

Operátory

- Bitové operátory
  - Unární: `~` – bitový doplněk
  - Binární: `&`, `|`, `^` (logické „po složkách“)
  - Bitový posun doleva: `<<`, doprava `>>`
  - (ukázka)
- Ternární operátor `? :`

```
vyraz ? Prvni_vetev : Druha_vetev
```

- Vyhodnotí `výraz` a je-li pravdivý vyhodnotí `První_větev` jinak `Druháý_větev`

```
1 int a = c>=b ? 33 : b-c;
```

Řízení toku programu (1/2)

- Podmíněný příkaz `if` – Stejný jako v C

```
if ( výraz_na_bool ) {
blok, který se vykoná, když je splněno
} else if (další_výraz_na_bool) {
} else {
jinak
}
```

- Větve `else if` a `else` nejsou povinné

Řízení toku programu (2/2)

- `switch / case` – lze nahradit serií `if`

```
1  switch (title) {
2    case "Bc":
3    case "Mgr":
4      message = "Hello bro";
5      break;
6    case "PhD":
7    case "doc":
8      message = "Good morning";
9      break;
10   case "prof":
11     message = "Good morning, my lord";
12     break;
13   default:
14     message = "";
15     break;
16 }
```

- Za `case` musí být konstanta
- `goto` v C# máme, ale nebudeme používat. (proč?)

Cykly

- `for` cyklus: `for (inicializatory; podminka; interator) { výrazy }` – každá z částí může být prázdná
  - (Většinové) použití: Když dopředu známe počet iterací
- `while` cyklus: `while (podminka) { výrazy }`
- `do...while` cyklus `do { výrazy } while (podmínka)`
  - Rozdíl v době testování podmínky
  - (Většinové) použití: Když dopředu neznáme počet iterací
- `break` a `continue`

Řetězce

- Máme jako základní datový typ, nemusíme „hlídat konec“
- Základní metody: `Length`, `ToLower`, `ToUpper`, `IndexOf`, `StartsWith`, ...
- Každý objekt má metodu `ToString()` pro textovou reprezentaci objektu
- Sčítání řetězců: `a+= "Toto priddam na konec retezce"`
  - Pozor! Vznikne v paměti nový řetězec (a starý se může zahodit)
  - Řetězíme-li něco v cyklu může být krajně neefektivní
  - `StringBuilder` (ukázka)
- Vepsání hodnot proměnných do řetězce:

```
1 string text = String.Format("Hodnota a={0}, b={1} a c={2}", a,
      b, c);
2
3 string zkracene = $"Hodnota a={a}, b={b}, a c={c}";
```

- Přístup k jednotlivým znakům:

```
1 char c = text[5];
```

Úkol (1/2)

Naprogramujte metody:

1. `void PrintFactorial(uint n)`
   - Postupně vypíše čísla 1!, 2!, 3!, ... $n!$ `n`

     ```
     n=5:
     1,  2,  6,  24, 120,
     ```
2. `void Square(short n)`
   - Pro sudá `n` vykreslí do konzole pomocí `*` čtverec dané velikosti
   - Pro lichá `n` vykreslí do konzole pomocí `*` křížek dané velikosti
   - Například n=4 a n=5

     ```
     ****
     *  *
     *  *
     ****
     ```

     ```
       *
       *
     *****
       *
       *
     ```

Úkol (2/2)

3. `uint NumberOfOccurrences(string str, char c)`
   - Vrátí počet výskytů znaku `c` v řetězci `str`

     `NumberOfOccurrences("Retezec", 'e') => 3`

4. `string StringTimes(string str, uint n)`
   - Vrátí řetězec, který obsahuje n-krát `str` oddělený čárkou a mezerou

`StringTimes("Retezec", 5) => "Retezec, Retezec, Retezec, Retezec, Retezec"`