Základní programové konstrukce

Základní programové konstrukce, proměnné, datové typy, reference a hodnota, programy a podprogramy, operátory, priorita operátorů

Joseph Soukup – P4

Proměnná

- Místo v paměti, do kterého ukládáme data
- Nutné deklarovat
- Obsahuje datový typ, název a hodnotu
-
```
int value = 0;

string word;
word = "opice";
```

Datový typ

- Dělení:
  - Hodnotové (přímo jí obsahují)
  - Referenční (odkazují na místo v paměti)
- **Hodnotové**
  - Dělí se na:
    - čísla (byte, int, long, double, float…)
    - logické (bool)
- **Referenční**
  - Class [object, string, class], interface, array…
- **Struktura**
  - Skupina proměnných
  -
```
struct kousky
{
    string Zichovec;
    string Albrecht;
    List<int> Farmář;
}
```
  - Rychlejší jak class
  - Nemá funkce jako class (dědičnost, …)
  - Ve struct nejde přímo inicializovat data (string pepa = „pepa“)
  - Používá se u jednodušších konstrukcí (souřadnice)

Reference a hodnota

- **Reference**
  - Označuje odkaz na proměnnou, nebo instanci objektu
  -
```
public int DivideByTwo(int param)
{
    return param / 2;
}
```
  - Dělíme na 4 druhy:
    - V signatuře metody a ve volání metody předejte argument metodě odkazem.
    - V signatuře metody pro vrácení hodnoty volajícímu odkazem

- V těle člena, aby označoval, že návratová hodnota odkazu je uložená místně jako odkaz, který volající cche změnit, nebo obecně místní proměnná přistupuje k jiné hodnotě odkazem
- Ve struct deklaraci pro deklaraci ref struct nebo readonly ref struct

- **Předání argumentu odkazem**
  - Předán odkazem, nikoli hodnotou

```C#
void Method(ref int refArgument)
{
    refArgument = refArgument + 44;
}

int number = 1;
Method(ref number);
Console.WriteLine(number);
// Output: 45
```

```
class Product
{
    public Product(string name, int newID)
    {
        ItemName = name;
        ItemID = newID;
    }

    public string ItemName { get; set; }
    public int ItemID { get; set; }
}

private static void ChangeByReference(ref Product itemRef)
{
    // Change the address that is stored in the itemRef parameter.
    itemRef = new Product("Stapler", 99999);

    // You can change the value of one of the properties of
    // itemRef. The change happens to item in Main as well.
    itemRef.ItemID = 12345;
}

private static void ModifyProductsByReference()
{
    // Declare an instance of Product and display its initial values.
    Product item = new Product("Fasteners", 54321);
    System.Console.WriteLine("Original values in Main.  Name: {0}, ID: {1}",
        item.ItemName, item.ItemID);

    // Pass the product instance to ChangeByReference.
    ChangeByReference(ref item);
    System.Console.WriteLine("Back in Main.  Name: {0}, ID: {1}\n",
        item.ItemName, item.ItemID);
}

// This method displays the following output:
// Original values in Main.  Name: Fasteners, ID: 54321
// Back in Main.  Name: Stapler, ID: 12345
```

- **Referenční návratové hodnoty**
  - Návratové hodnoty odkazu jsou hodnoty, které metoda vrátí odkazem na volajícího
  - => Volající může změnit hodnotu vrácenou metodou a tato změna se projeví ve stavu objektu v metodě volání
  - **V signatuře metody**
    - Označuje, že **GetCurrentPrice** metoda vrací Decimal hodnotu odkazem
    -

```
public ref decimal GetCurrentPrice()
```

- **Mezi return tokenem a proměnnou vrácenou v return příkazu v metodě.**
  - ```
    return ref DecimalArray[0];
    ```
  - Aby volající mohl změnit stav objektu, návratová hodnota odkazu musí být uložena do proměnné, která je explicitně definovaná jako **místní referenční číslo**
  - ```
    public static ref int Find(int[,] matrix, Func<int, bool> predicate)
    {
        for (int i = 0; i < matrix.GetLength(0); i++)
            for (int j = 0; j < matrix.GetLength(1); j++)
                if (predicate(matrix[i, j]))
                    return ref matrix[i, j];
        throw new InvalidOperationException("Not found");
    }
    ```
    Find returning by reference
- **Lokální hodnoty REF**
  - Lokální proměnná ref slouží k odkazování na hodnoty vracené pomocí return ref
  - ```
    ref decimal estValue = ref Building.GetEstimatedValue();
    ```
  - K hodnotě můžete přistupovat stejným způsobem pomocí odkazu
  - ```
    ref VeryLargeStruct reflocal = ref veryLargeStruct;
    ```
  - Ref musí být použito na obou místech
- **Místní referenční hodnoty jen pro čtení**
  - Místní odkaz jen pro čtení se používá k odkazování na hodnoty vracené metodou nebo vlastnosti, která má ref readonly a jeho signaturu a používá return ref
- **Příklad odkazů vrátí a místní hodnoty REF**
  - ```
    public class Book
    {
        public string Author;
        public string Title;
    }

    public class BookCollection
    {
        private Book[] books = { new Book { Title = "Call of the Wild, The", Author = "Jack London" },
                                 new Book { Title = "Tale of Two Cities, A", Author = "Charles Dickens" }
                               };
        private Book nobook = null;

        public ref Book GetBookByTitle(string title)
        {
            for (int ctr = 0; ctr < books.Length; ctr++)
            {
                if (title == books[ctr].Title)
                    return ref books[ctr];
            }
            return ref nobook;
        }

        public void ListBooks()
        {
            foreach (var book in books)
            {
                Console.WriteLine($"{book.Title}, by {book.Author}");
            }
            Console.WriteLine();
        }
    }
    ```
    ```

```
var bc = new BookCollection();
bc.ListBooks();

ref var book = ref bc.GetBookByTitle("Call of the Wild, The");
if (book != null)
    book = new Book { Title = "Republic, The", Author = "Plato" };
bc.ListBooks();
// The example displays the following output:
//       Call of the Wild, The, by Jack London
//       Tale of Two Cities, A, by Charles Dickens
//
//       Republic, The, by Plato
//       Tale of Two Cities, A, by Charles Dickens
```

- **Hodnota**
  - Entita programu, se kterou může program pracovat
  - ```
    int value = 0;
    ```

Programy a podprogramy

- **Program**
  - Proces projektu, který řídí fungování aplikace
  - ```
    Počet odkazů: 0
    class Program
    {
        Počet odkazů: 0
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
    ```
- **Podprogram**
  - Část programu, která se dá opakovaně používat v různých místech programu
  - ```
    public void DivideString(string s)
    {
        s.ToCharArray();
    }
    ```
  - Logický celek programu
  - Část programu definovaná před startem programu
  - Po startu hlavního programu je podprogram z hlavního programu spouštěn
  - **Využití:**
    - Zkrácení, zpřehlednění a zjednodušení hlavního programu
    - Rozdělení algoritmu na více jednodušších částí
    - Jednoduchá kontrola a odstranění chyb

Základní programové konstrukce

- **Podmínky**
  - **Složený příkaz:**
    - Příkazy se provádí přímo za sebou
  - **Úplný podmíněný (dvě větve)**
    - Pokud platí podmínka, proveď příkaz, v jiném případě proveď příkaz č.2
    ```

```
if (c > 0)
{
    c++;
} else
{
    c--;
}
```

- **Neúplný podmíněný (jedna větev)**
  - Pokud platí podmínka, proveď příkaz

```
if (c > 0)
{
    c++;
}
```

- **Vícenásobné větvení**
  - Pokud se hodnota rovná 1, proveď příkaz 1, pokud se rovná 2, proveď příkaz 2, …

```
switch (c)
{
    case 1:
        c++;
        break;

    case 0:
        c++;
        break;

    default:
        break;
}
```

```
if (condition1)
{
  // block of code to be executed if condition1 is True
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
  // block of code to be executed if the condition1 is false and condition2 is False
}
```

- **Cykly**
  - **Podmínka na začátku (příkaz nemusí proběhnout)**
    - Dokud platí podmínka, konej příkaz

```
int c = 0;
while (c == 9)
{
    //neproběhne
    c--;
}
```

  - **Podmínka na konci**
    - Vykonej příkaz, opakuj, zda platí podmínka (do while)

```
do
{
    c++;
} while (c < 20);
```

  - **Známý počet průchodů**
    - Konej příkaz tolikrát, kolikrát je v podmínce zadáno (zvyšování proměnné)

-
```
for (int i = 0; i < 20; i++)
{
    c = c + 5;
}
```

Operátory

- **Aritmetické:** +, -, *, /, %
- **Logické:** &&(AND), ||(OR), !(NOT)
- **Bitové:** &(bitové AND), |(bitový OR), ~(bitový NOT)
- **Relační:** <, >, ==, <=, !=
- **Přístupové:** [] (indexer), . (přístup ke členu), () (invokace)
- **Ostatní:** = (operátory přihlášení), => (lambda operátor)

Priorita operátorů

Priorita	Operátor(y)
1	`()  []  ->  .`
2	`!  ~  ++ --  +  - (typ)  *  &  sizeof`
3	`*  /  %`
4	`+  -`
5	`<<  >>`
6	`<  <=  >  =>`
7	`==  !=`
8	`&` and bitový
9	`^` xor bitový
10	`|` or bitový
11	`&&`
12	`||`
13	`?: ternární operátor`
14	`= += -= *= /= %= >>= <<= &= |= ^=`
15	`,  operátor čárka`

-

Význam jednotlivých operátorů je:

1) primární operátory — vyhodnocování zleva doprava
   - `()` volání funkce
   - `[]` index do pole
   - `->` přístup k prvku struktury pomocí pointeru na strukturu
   - `.` přístup k prvku struktury pomocí jména struktury
2) unární operátory — vyhodnocování zprava doleva
   - `!` negace logického výrazu
   - `~` negace bit po bitu (jedničkový doplněk)
   - `++` inkrementace
   - `--` dekrementace
   - `+` unární plus (např. `i = +5;`)
   - `-` unární mínus (např. `i = -5;`)
   - `(typ)` přetypování
   - `*` pointer (dereference)
   - `&` adresový (referenční) operátor
   - `sizeof` velikost typu
3) multiplikativní operátory — vyhodnocování zleva doprava
   - `*` aritmetický součin
   - `/` celočíselné nebo reálné dělení
   - `%` dělení modulo
4) aditivní operátory — vyhodnocování zleva doprava
   - `+` aritmetický součet
   - `-` aritmetický rozdíl
5) operátory posunů — vyhodnocování zleva doprava
   - `<<` posun bitů doleva
   - `>>` posun bitů doprava
6) relační operátory — vyhodnocování zleva doprava
   - `<` menší než
   - `<=` menší nebo rovno než
   - `>` větší než
   - `>=` větší nebo rovno než
7) operátory rovnosti — vyhodnocování zleva doprava
   - `==` rovnost
   - `!=` nerovnost
8) `&` operátor bitového součinu (AND) — vyhodnocování zleva doprava
9) `^` operátor exkluzivního bitového součtu (XOR) — vyhodnocování zleva doprava
10) `|` operátor bitového součtu (OR) — vyhodnocování zleva doprava
11) `&&` operátor logického součinu (AND) — vyhodnocování zleva doprava
12) `||` operátor logického součtu (OR) — vyhodnocování zleva doprava
13) `?:` ternární podmínkový operátor — vyhodnocování zprava doleva
14) přiřazovací operátory — vyhodnocování zprava doleva
   - `=` přiřazení
   - `+=` součet a přiřazení
   - `-=` rozdíl a přiřazení
   - `*=` součin a přiřazení
   - `/=` dělení a přiřazení
   - `%=` dělení modulo a přiřazení
   - `>>=` bitový posun doprava a přiřazení
   - `<<=` bitový posun doleva a přiřazení
   - `&=` bitový součin a přiřazení
   - `|=` bitový součet a přiřazení
   - `^=` exkluzivnáce a přiřazení
15) `,` operátor čárky — vyhodnocování zleva doprava

Dají se použít následující přiřazení:

l-hodnota `+=` *výraz*	*l-hodnota* `=` *l-hodnota* `+` *výraz*
l-hodnota `-=` *výraz*	*l-hodnota* `=` *l-hodnota* `-` *výraz*
l-hodnota `*=` *výraz*	*l-hodnota* `=` *l-hodnota* `*` *výraz*
l-hodnota `/=` *výraz*	*l-hodnota* `=` *l-hodnota* `/` *výraz*
l-hodnota `%=` *výraz*	*l-hodnota* `=` *l-hodnota* `%` *výraz*
l-hodnota `>>=` *výraz*	*l-hodnota* `=` *l-hodnota* `>>` *výraz*
l-hodnota `<<=` *výraz*	*l-hodnota* `=` *l-hodnota* `<<` *výraz*
l-hodnota `&=` *výraz*	*l-hodnota* `=` *l-hodnota* `&` *výraz*
l-hodnota `^=` *výraz*	*l-hodnota* `=` *l-hodnota* `^` *výraz*
l-hodnota `|=` *výraz*	*l-hodnota* `=` *l-hodnota* `|` *výraz*

-