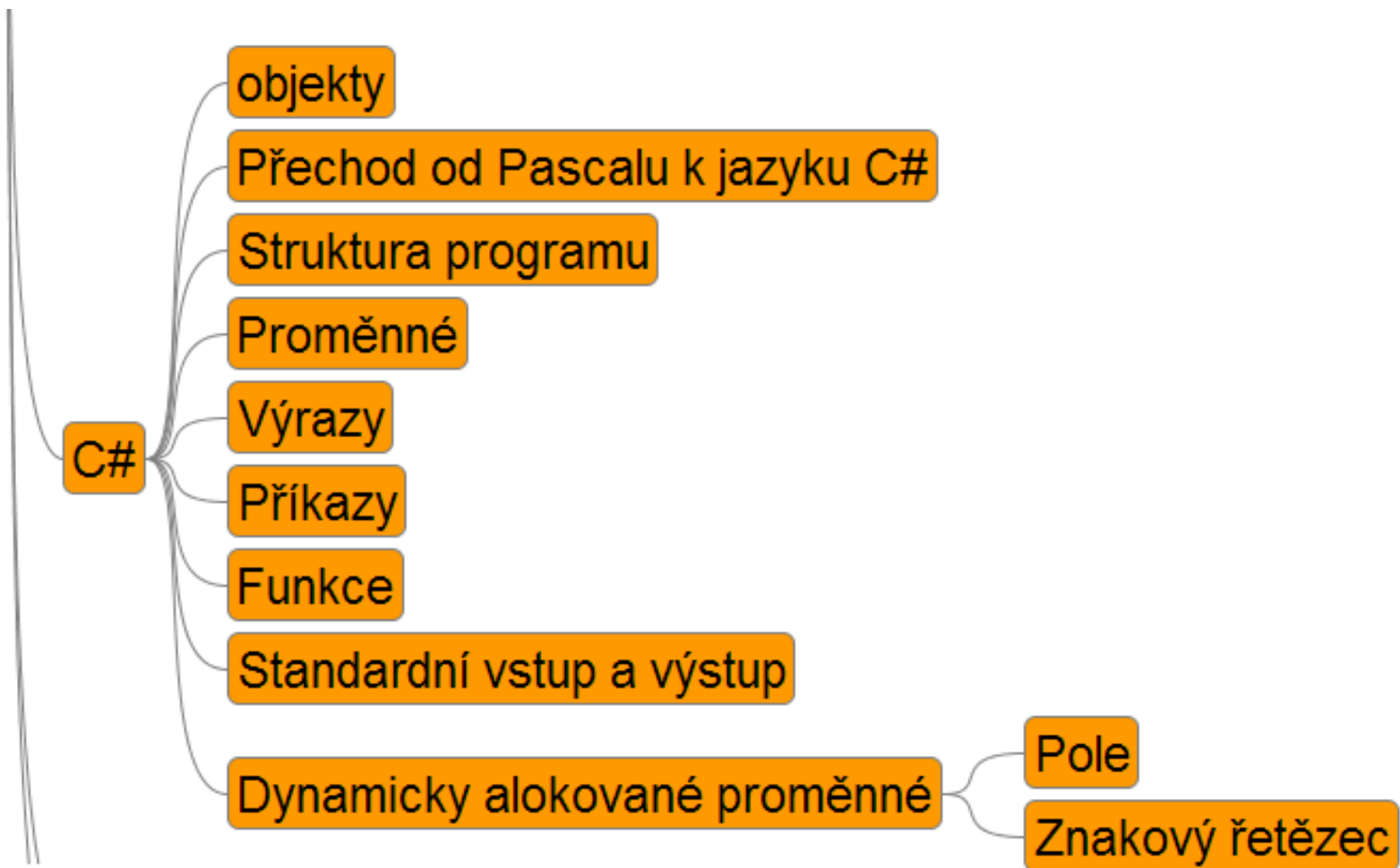




# Objekty

Svět se skládá z objektů!

Objekt = data + funkce (metody)

konkrétní x abstraktní  
hmatatelné x nehmatatelné  
(letadlo) x (chyba v programu)

Objekty URČITÝM ZPŮSOBEM PODOBNÉ můžeme  
považovat za instance jedné třídy (*pes*).

# Objekty

Další pokus oddělit

CO x JAK

VENKU x UVNITŘ

INTERFACE x IMPLEMENTACE

Strukturované programování

blok, funkce

Modulární programování

modul, unit

Objektové programování

objekt a třída

# Objekty v programu

Způsob jak izolovat část kódu  
(příkaz-blok-procedura-modul-objekt).

Způsob jak uvažovat o problému  
Objekt sdružuje DATA (datové složky, vlastnosti)  
i KÓD (funkce+procedury=METODY)  
= ČLENY (members)

OBJEKT = exemplář, instance TŘÍDY.

Zapouzdření  
ukrývání vnitřku, díky tomu konsistentní stav

**Příklad:**

Napište program, který čte ze vstupu slova  
a tiskne je na řádky dané délky.

# Jazyk C#

- \* C#

- Java 1995, bytecode
- C# 2002, .NET
  - Anders Hejlsberg

- \* Microsoft Visual Studio, „Community“ verze zdarma

- \* Projekt MONO (macOS, Linux, Windows)

# Přechod od C/C++ k jazyku C#

...raději ne.

## Přechod od Pascalu k jazyku C#

### Používání mezer a řádkování

- volné, stejně jako v Pascalu
- doporučená a prostředím podporovaná indentace

## Identifikátory

- case-sensitivní
- možnost používat diakritiku
- klíčová slova malými písmeny
- konvence (zvyklosti):
  - proměnné malými písmeny, konstanty velkými písmeny
  - jména prostorů\*), tříd, metod a vlastností, veřejné členy
    - > „PascalskáNotace“
- např. `Math`, `DivideByZeroException`, `Main`, `WriteLine`
- soukromé metody začínají malým písmenem
  - > „velbloudíNotace“

# Struktura programu

Celý program se skládá ze tříd,  
vše se deklaruje a používá uvnitř tříd  
(proměnné, konstanty, funkce, ...).

Položky deklarované ve třídě:

- datové složky třídy = **členské proměnné**
- metody = **členské funkce**

**Prozatím:** celý program je tvořen jedinou statickou\*)  
metodou (její obsah tedy odpovídá celému programu)

**Někdy příště:** jak jinak to může vypadat se třídami

# Proměnné

- zápis deklarace

  - **syntaxe:** `int alfa;`

- umístění deklarace:

**BUĎ** členská proměnná třídy (tzn. datová složka objektu)

**NEBO** lokální kdekoliv ve funkci, ale nesmí zakrýt jinou stejnojmennou deklaraci uvedenou v téže funkci

(pozor na kolize!)

- lokální platnost deklarace v bloku, kde je uvedena

- možnost inicializace v rámci deklarace: `int alfa = 15;`

- v programu nelze použít nedefinovanou hodnotu proměnné

  - (kontrola při překladu) 

- hodnotové a referenční typy

- všechno\*) je objekt (instance nějaké třídy)

# Konstanty

- syntaxe jako inicializované proměnné, specifikátor `const`:  
`const int ALFA = 15;`
- číselné konstanty podobné jako v Pascalu (různé typy)
- konstanty typu `char` v apostrofech: `'a'`,  
typu `string` v uvozovkách: `"aaa"` 

# Typy

## Hodnotové

### celé číslo

`int`                      `System.Int32`      32 bitů

další typy: `byte`, `sbyte`, `short`, `ushort`, `uint`, `long`, `ulong`

### desetinné číslo

`double`                  `System.Double`    64 bitů

další typy:      `float`, `decimal`

### logická hodnota

`bool`

### znak

`char`            `System.Char`    16 bitů Unicode

### výčtový typ

`enum`

### struktura

`struct`

## Referenční

### pole

[ ]      System.Array

### znakový řetězec

string      System.String

### třída

class

(standardní třídy, např. ArrayList,  
StringBuilder, List<>)

## Hlavní rozdíl:

Dosazuje se hodnota nebo reference.

# Aritmetické výrazy

- obvyklé symboly operací i priority stejné jako v Pascalu  
+ - \* /
- **POZOR:** symbol / představuje reálné i celočíselné dělení (zvolí se podle typu argumentů) = zdroj chyb! 
- znak % pro modulo (zbytek po celočíselném dělení)
- klíčová slova checked, unchecked - určení, zda se má kontrolovat aritmetické přetečení v celočíselné aritmetice
- použití jako checked(výraz) nebo checked{blok}
- standardní matematické funkce  
= statické\*) metody třídy Math

# Středník

- ukončuje každý příkaz  
(musí být i za posledním příkazem bloku!)
- nesmí být za blokem ani za hlavičkou funkce
- odděluje sekce v hlavičce for-cyklu

# Čárka

- odděluje deklarace více proměnných téhož typu
- odděluje parametry v deklaraci funkce i při volání funkce
- odděluje indexy u vícerozměrného pole

## Komentáře

- jednořádkové `// xxx` do konce řádku
- víceřádkové `/* xxx */`
- dokumentační `///`

## Blok (složený příkaz)

- závorky `{ }` místo pascalského `begin - end`

## Dosazovací příkaz

- syntaxe: `proměnná = výraz` např. `i = 2*i + 10;`

# Příkaz modifikace hodnoty

```
i++; ++i;  
i--; --i;
```

```
i += 10;  
i -= 10;  
i *= 10;  
i /= 10;  
i %= 10;
```

# Podmíněný příkaz

- podmínka = výraz typu bool v závorkách

```
if (a == 5)    b = 17;
```

```
if (a == 5)    b = 17;
```

```
    else      b = 18;
```

- relační operátory: ==    !=    <    >    <=    >=

- logické spojky

- && and (zkrácené vyhodnocování)

- || or (zkrácené vyhodnocování)

- & and (úplné vyhodnocování)

- | or (úplné vyhodnocování)

- ! not

- ^ xor

# For-cyklus

- **syntaxe:**

for (inicializace; podmínka pokračování; příkaz iterace)  
příkaz těla

```
for (int i=0; i<N; i++)    a[i] = 3*i+1;
```

- některá sekce může být prázdná (třeba i všechny)

(pokud víc příkazů, oddělují se čárkou)

# Cykly while a do-while

- cyklus while stejný jako while-cyklus v Pascalu (podmínka je opět celá v závorce a nepíše se „do“)

```
while (podmínka) příkaz;
```

- cyklus do-while má podmínku na konci jako cyklus repeat-until v Pascalu, ale význam podmínky je proti Pascalu obrácený,  
tzn. dokud podmínka platí, cyklus se provádí

```
do příkaz while (podmínka) ;
```

- více příkazů v těle cyklu musí být uzavřeno v bloku { }

## Ukončení cyklu

### - příkazy

`break;`

`continue;`

### - stejný význam jako v Pascalu

# Příkaz switch

- analogie pascalského příkazu case
- varianta se může rozhodovat podle výrazu celočíselného, podle znaku nebo také stringu
- sekce **case**, za každým case jediná konstanta, ale pro více case může být společný blok příkazů
- poslední sekce může být default:
- je povinnost ukončit každou sekci **case** (i sekci default, neboť ta nemusí být uvedena poslední) příkazem

`break, příp. return nebo goto`

```
int j, i = ...;  
switch (i)  
{  
    case 1:  
        i++; break;  
    case 2:  
    case 3:  
        i--; break;  
    default:  
        i=20; j=7; break;  
}
```

(pozor: v C, C++, Java, PHP... se může propadat mezi  
sekcemi = zdroj chyb, v C# opravený)



# Funkce

- pascalské procedury a funkce
  - > metody (členské funkce) nějaké třídy
- procedura
  - > funkce typu void
- v deklaraci i při volání vždy píšeme ( ),  
i když nemá žádné parametry
- ve funkci nelze lokálně definovat\*) jinou funkci,  
strukturu nebo třídu,  
Ize tam ale deklarovat lokální proměnné  
(ve třídě lze deklarovat jinou třídu  
ta může mít své metody)



\*) V C#7 už lze...

# Funkce...

- mohou vracet i složitější\*) typy
- `return <hodnota>;`
  - definování návratové hodnoty a ukončení funkce
- v případě funkcí typu `void` pouze `return;`
- předávání parametrů:
  - standardně hodnotou
  - odkazem - specifikátor `ref` v hlavičce i při volání \*) 
  - výstupní parametr
    - specifikátor `out` v hlavičce i při volání 
    - (`out` je také odkazem, nemá ale vstupní hodnotu)

\*) V C#7 už lze vracet "tuple"

# Výchozí metoda Main()

- plní funkci hlavního programu  
(určuje začátek a konec výpočtu)
- je to statická\*) metoda nějaké třídy  
(nic „mimo třídy“ neexistuje),  
často se pro ni vytváří samostatná třída
- obvykle jediná v aplikaci  
→ je tak jednoznačně\*) určeno, kde má začít výpočet
- \*)může jich být i více, pak se ale při kompilaci musí  
přepínačem specifikovat, ze které třídy se má použít  
Main() při spuštění programu
- **syntaxe:** `static void Main(string[] args)`

# Standardní vstup a výstup

`Console.Read();`

vrací `int` = jeden znak ze vstupu (jeho kód)

`Console.ReadLine();`

vrací `string` = jeden řádek ze vstupu

`Console.Write(výraz);`

vypíše hodnotu zadaného výrazu

`Console.WriteLine(výraz);`

vypíše hodnotu zadaného výrazu a odřádkuje

# Formátovaný výstup

`Console.WriteLine(string);`

do stringu se dosadí hodnoty výrazů po řadě na místa vyznačená pomocí {0}, {1}, {2}, atd., případně i s požadovaným formátováním {0:N}

```
Console.WriteLine(  
    "x0={0} x1={1} x2={2}", x0, x1, x2  
);
```

od C#6: „string interpolation“

```
Console.WriteLine(  
    $"x0={x0} x1={x1} x2={x2}" );
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
        }
    }
}
```

## Příklad: Eukleidův algoritmus

```
static void Main(string[] args)
{
    Console.WriteLine(
        "Zadej dvě kladná celá čísla:");
    int a = int.Parse(Console.ReadLine());
    int b = int.Parse(Console.ReadLine());
    while (a != b)
    {
        if (a > b) a -= b;
        else b -= a;
    }
    Console.WriteLine(
        "Nejv. spol. dělitel: {0}", a);
    Console.ReadLine();
}
```

Příklady:

Prvočíselný rozklad

Hornerovo schéma – vstup čísla po znacích

```
static void Main(string[] args)
{
    int v;
    int c;
    c = Console.Read();
    // preskocit ne-cislice:
    while ((c < '0') || (c > '9'))
    {
        c = Console.Read();
    }
    // nacitat cislice:
    v = 0;
    while ((c >= '0') && (c <= '9'))
    {
        v = 10 * v + (c - '0');
        c = Console.Read();
    }
    Console.WriteLine(v);
    Console.ReadLine();
}
```

# Dynamicky alokované proměnné

- vytvářejí se pomocí zápisu  
new + konstruktor\*) vytvářeného objektu
- new je funkce, vrací vytvořenou instanci  
(ve skutečnosti ukazatel na ni)
- v odkazech se nepíšou ^
- string, pole, třídy - referenční typy
- konstanta null (jako NIL v Pascalu)
- automatická správa paměti  
nedostupné objekty jsou automaticky uvolněny z paměti  
(ne nutně úplně okamžitě, až to bude potřeba)

## Příklad

```
prvni = null;  
    // to je korektní zrušení celého spojového seznamu
```

```
class Uzel  
{  
    public int info;  
    public Uzel dalsi;  
}  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        Uzel prvni = new Uzel();  
        prvni.info = 123;  
        prvni.dalsi = null;  
        // ...  
    }  
}
```

# Pole

- deklarace: `int[] aaa;`
- referenční typ, je nutné vytvořit pomocí `new`:  
`int[] aaa = new int[10];`
- každé pole je instancí třídy odvozené\*) z abstraktní statické třídy `System.Array`
- indexování vždy od 0
- možnost inicializace:  
`int[] aaa = new int[3] { 2, 6, 8 };`  
`int[] aaa = { 2, 6, 8 };`
- počet prvků: `aaa.Length`
- vždy se provádějí kontroly přetečení mezí   
při indexování `aaa[i]`

# POZOR !

```
static void Main(string[] args)
{
    int[] aaa = { 2, 6, 8 };
    int[] bbb;
    bbb = aaa;
    aaa[0] = 27;

    Console.WriteLine(bbb[0]);
}
```

**dosazuje se ukazatel !!**

# Pole...

- připravené metody, např. `CopyTo`, `Sort`, `Reverse`, `BinarySearch`, `Array.Reverse(aaa)` ;
- vícerozměrné pole  
obdélníkové `[,]` a nepravidelné `[] []`

**Nepravidelné dvourozměrné pole** je ve skutečnosti pole polí

(tzn. pole ukazatelů na řádky,  
což jsou pole jednorozměrná),

- každý řádek je třeba zvlášť vytvořit pomocí `new`
- řádky mohou mít různou délku

```
int[] [] aaa = new int[3] [] ;  
aaa[0] = new int[4] ;  
aaa[1] = new int[6] ;  
aaa[2] = new int[2] ;
```

## Příklad: Třídění čísel v poli - přímý výběr

```
static void Main(string[] args)
{
    Console.Write("Počet čísel: ");
    int pocet = int.Parse(Console.ReadLine());
    int[] a;
    a = new int[pocet];
    int i = 0;
    while (i < a.Length)
        a[i++] = int.Parse(Console.ReadLine());
    i = 0;

    while (i < a.Length)
    {
        int k = i;
        int j = i+1;
        while (j < a.Length)
        {
```

```

        if (a[j] < a[k]) k = j;
        j++;
    }
    if (k != i)
    {
        int x = a[i];
        a[i] = a[k];
        a[k] = x;
    }
    i++;
}

i = 0;
while (i < a.Length)
    Console.Write(" {0}", a[i++]);
Console.WriteLine();
}

```

# Znakový řetězec

- deklarace: `string sss;`
- `string` - referenční typ, alias pro třídu `System.String`
- vytvoření instance: `string sss = "abcdefg";`
- nulou ukončené řetězce, nemají omezenou délku
- indexování znaků od 0
- délka = `sss.Length`
- obsah nelze měnit (na to je třída `StringBuilder`)
- všechny objekty mají konverzní metodu `ToString()`,  
pro struktury a objekty je vhodné předefinovat ji  
(jinak se vypisuje jenom jejich jméno)

# Struktura - struct

- „zjednodušená třída“
- má podobný význam a použití jako pascalský záznam (record)
- navíc může mít metody (jako třída)
- může mít i konstruktor  
(vlastní konstruktor musí inicializovat všechny datové složky, jinak má i implicitní bezparametrický konstruktor)
- je to **hodnotový** typ  
(na rozdíl od instance třídy se nemusí alokovat)
- některá omezení oproti třídám (např. nemůže dědit)

```
struct Bod
{
    public int x, y;
    public Bod(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

