C++ vs C#

C++	C# [sí šarp]
Od 80. let jako objektové rozšíření C	Od 00. let jako .NET jazyk vycházející z Javy
Především low-lovel aplikace orientované na HW	Především high-level aplikace frameworku
Správu paměti řídí kód	Správu paměti řídí OS
GUI: winAPI, wxWidgets	GUI: WinForms, WPF, GTK
Framework Qt	Framework .NET
IDE Code Blocks, Dev-C++	IDE Visual Studio, SharpDevelop

Hello world

C++

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello world";
}
```

C#

```
using System;
namespace HelloWorld {
  class Program {
    public static void Main() {
      Console.WriteLine("Hello world");
    }
  }
}
```

C# nemá preprocesor, příkaz `using` zde má jiný význam, načítá knihovny. Uzavírat kód do jmenného prostoru není povinné, ale známá vývojová prostředí tak činí automaticky. V C# nemůže existovat samostatná funkce (není to *first-class citizen*), `Main` (pozor, velké M, jmenná konvence) musí být definována jako statická metoda a měla by být jediná v celém projektu. Díky tomu není nutné používat hlavičkové soubory, kompilátor si je snadno vygeneruje z tříd, neboť téměř všechen kód musí být ve třídě, nejsou žádné globální proměnné. Modifikátory přístupu (zde `public`) je nutno uvádět u každého členu zvlášť.

Vývojové prostředí

Nejrozšířenější jsou následující IDE pro vývoj aplikací v C#:

- **Microsoft Visual Studio** vyskytující se v následujících verzích: 2005, 2008 (velmi malé a rychlé), 2011 (podpora frameworků .NET 4+), 2013, 2015 (velmi pomalé, mnoho GB), 2017 (modulární, ale přesto pomalé, schůdné na strojích s i7 a lepším procesorem), 2017 Community Edition (pouze editor, rychlé a univerzální, lze použít i na Arduino, automatické instalace doplňků, ale obtížné řešení čehokoliv neautomatického)

- **MonoDevelop** (nově Xamarin Studio) relativně rychlé, multiplatformní, pro Windows nutno zkompilovat ze zdrojáků s několika závislostmi. Podporuje GUI GTK.
- **SharpDevelop** méně než 100 MB, portabilní, velmi podobné Visual Studiu, velmi rychlé a jednoduché (chybějící rozšířené možnosti v tomto kurzu nevyužijeme). Tohoto prostředí se přidržíme ve škole.

Funkce IDE

IDE usnadňuje práci zejména v

- automatické doplňování kódu
- odhalování syntaktických chyb při psaní (tj. před kompilací)
- automatická kompilace (tlačítko play zajistí sestavení celého projektu a referencí)
- ladění se zarážkami a krokováním
- vizuální programování GUI

Správa paměti

Klíčové slovo `struct` neoznačuje třídu, ale hodnotový datový typ[1], který má stejně jako třída výchozí modifikátor přístupu privátní. Není typ ukazatele, objekty všech tříd se přenáší odkazem. Nevolá se `delete`, program je řízený (*managed*[2]): operační systém udržuje vnitřní počítadlo referencí na objekt a ve vhodnou chvíli (typicky když dochází paměť) zavolá nástroj *garbage collector*, který z paměti uvolní všechny objekty s vynulovaným počítadlem odkazů.

```
class Point { public int X, Y; }

void test() {
  Point P = new Point { X=1, Y=2 };
  // není únik paměti, garbage collector časem paměť uvolní
}
```

V C# jsou ukazatele automatické (není žádný typ hvězdička), objekty struktur se přenáší hodnotou, viz příklad:

```
struct PointS { public int x, y; }
class  PointC { public int x, y; }

PointS s1 = new PointS { x=1, y=2 };
PointC c1 = new PointC { x=1, y=2 };
PointS s2 = s1; // kopírování hodnotou
PointC c2 = c1; // kopírování odkazem
s2.x = 42; // nezmění s1
c2.x = 42; // změní c1
Console.WriteLine("s1.x: {0}, c1.x: {1}", s1.x, c1.x); // 1, 42
```

Pole se kopíruje odkazem jako objekt třídy (ověřte si). Pozor na syntaktickou odlišnost s C++:

```
int numbers[] = {1,2,3}; // C++
int[] numbers = {1,2,3}; // C# (z Javy)
```

[1] stále může mít konstruktor a metody, ale je zamýšlen především jako kontejner malých datových položek
[2] opak řízené aplikace je *nativní*: dynamická správa paměti je řešena kódem

Pole v C# dědí ze třídy `Array`, která obsahuje zejména vlastnost `Length` (opět si všimněte velkého L). Žádná proměnná není pouze ukazatel na místo v paměti (každá proměnná kopírovaná odkazem dědí minimálně ze třídy `Object`), v C# tedy nejsou Cčkové řetězce typu `const char*`, pouze typu `String`. Nepotřebujeme tedy předávat délku pole jako parametr a signatura metody Main s parametry je

```
public static void Main(string[] args)
```

kde `string` je alias k `System.String`.

Verze .NET

Při kompilaci je nutné zvolit správnou verzi .NET frameworku (aplikace spoléhá, že .NET je na cílovém stroji nainstalován).

- Ve Windows XP ve výchozím nastavení není .NET a lze nainstalovat max verzi 3.5 (vyšší verze pouze částečně formou hacku).
- Ve Windows 7 je jako výchozí verze 3.5, ve Windows 10 verze 4.6
- Na systémech založených na Linuxu lze obvykle nainstalovat libovolnou verzi .NET, některé funkce ale nejsou portabilní (jsou svázány s Windows) a nelze je použít.

Dokumentace

Dokumentace k jednotlivým třídám .NET je na stránkách Microsoftu. Prozkoumejte třídu System.Console[3] a vyzkoušejte si konstrukce `if`, `for` a `while`, které jsou stejné jako v C++:

- napište program, který vypíše mezerou oddělené čísla 0 - 99
- upravte program tak, aby vypsal pouze lichá čísla
- napište program, který přečte od uživatele znak z konzole a převede ho na velký
- upravte program tak, aby převáděl tak dlouho, dokud nestiskneme `Esc`

[3] vpravo nahoře je možné zvolit si mezi kostrbatým automatickým překladem do češtiny a mezi napůl přeloženým textem z angličtiny