IDT, Přednáška 1

Libor Váša

Katedra informatiky a výpočetní techniky, Západočeská univerzita v Plzni

12. 2. 2024

Úvod

O čem je kurz IDT?

- ujasnit, jak se pozná dobrý program od špatného (výpočetní složitost)

O čem je kurz IDT?

- ujasnit, jak se pozná dobrý program od špatného (výpočetní složitost)
- vyzbrojit studenty základní sadou vzorů, které by v úlohách měli vidět (abstraktní datové struktury)

O čem je kurz IDT?

- ujasnit, jak se pozná dobrý program od špatného (výpočetní složitost)
- vyzbrojit studenty základní sadou vzorů, které by v úlohách měli vidět (abstraktní datové struktury)
- dát studentům příležitost programovat, aby si znalosti osvojili (těžko na cvičišti, lehko na bojišti)

O čem je kurz IDT?

- ujasnit, jak se pozná dobrý program od špatného (výpočetní složitost)
- vyzbrojit studenty základní sadou vzorů, které by v úlohách měli vidět (abstraktní datové struktury)
- dát studentům příležitost programovat, aby si znalosti osvojili (těžko na cvičišti, lehko na bojišti)
- získat programátorskou zručnost (kouknu a vidím, vím co dělat, když to nefunguje)

O čem je kurz IDT?

- ujasnit, jak se pozná dobrý program od špatného (výpočetní složitost)
- vyzbrojit studenty základní sadou vzorů, které by v úlohách měli vidět (abstraktní datové struktury)
- dát studentům příležitost programovat, aby si znalosti osvojili (těžko na cvičišti, lehko na bojišti)
- získat programátorskou zručnost (kouknu a vidím, vím co dělat, když to nefunguje)
- naučit se alespoň jeden jazyk (C#) - ostatní jsou podobné

Motto předmětu

Programovat vás nenaučíme, musíte se to naučit sami.

O čem není kurz IDT?

- podrobně probrat vlastnosti jazyka C#

O čem není kurz IDT?

- podrobně probrat vlastnosti jazyka C#
- podrobně probrat vlastnosti prostředí .NET

O čem není kurz IDT?

- podrobně probrat vlastnosti jazyka C#
- podrobně probrat vlastnosti prostředí .NET
- podrobně probrat možnosti vývojového prostředí (VS Code, Visual Studio)

O čem není kurz IDT?

- podrobně probrat vlastnosti jazyka C#
- podrobně probrat vlastnosti prostředí .NET
- podrobně probrat možnosti vývojového prostředí (VS Code, Visual Studio)

na to je předmět KIV/PNET

Jazyk C#

Historie

Architekt: Anders Hejlsberg (předtím Turbo Pascal, Borland Delphi)
Leden 2002: C# 1.0
Listopad 2005: C# 2.0 (generické datové typy)
Listopad 2007: C# 3.0 (LINQ)
...
Listopad 2023: C# 12.0
Nyní využíván např. jako skriptovací jazyk v herním enginu Unity.

Vlastnosti

- objektově orientovaný

Vlastnosti

- objektově orientovaný
- silná typová kontrola

Vlastnosti

- objektově orientovaný
- silná typová kontrola
- managed memory

Vlastnosti

- objektově orientovaný
- silná typová kontrola
- managed memory
- důraz na přenositelnost mezi platformami

Středníky

na konci každého příkazu
na jedné řádce může být více příkazů (není to ale obvyklé)

Komentáře

překladač je ignoruje

```
// az do konce radky
/*
mezi symboly
*/
```

Proměnné

Musí se *deklarovat*

- před prvním použitím

Proměnné

Musí se *deklarovat*

- před prvním použitím
- kdekoli v programu

Proměnné

Musí se *deklarovat*

- před prvním použitím
- kdekoli v programu
- sděluje překladači datový typ proměnné

Proměnné

Musí se *deklarovat*

- před prvním použitím
- kdekoli v programu
- sděluje překladači datový typ proměnné
- datový typ je s proměnnou trvale svázán

Formát:

```
<typ> <nazev>;
<typ> <nazev1>, <nazev2>;
```

Příklad:

```
int x;
double y, z;
```

Názvy proměnných

- bez mezer

Názvy proměnných

- bez mezer
- záleží na velikosti písmen

Názvy proměnných

- bez mezer
- záleží na velikosti písmen
- nezmí začínat číslicí

Názvy proměnných

- bez mezer
- záleží na velikosti písmen
- nezmí začínat číslicí
- nesmí kolidovat s klíčovými slovy (if, for, while, ...)

Primitivní datové typy

- celá čísla

Primitivní datové typy

- celá čísla
- desetinná čísla (float)

Primitivní datové typy

- celá čísla
- desetinná čísla (float)
- znak

Primitivní datové typy

- celá čísla
- desetinná čísla (float)
- znak
- pravdivostní hodnota (boolean)

Primitivní datové typy

- celá čísla
- desetinná čísla (float)
- znak
- pravdivostní hodnota (boolean)
- reference

Celočíselné typy

znaménkové/neznaménkové

- 8 bit: sbyte/byte
- 16 bit: short/ushort
- 32 bit: int/uint
- 64 bit: long/ulong

Typy s plovoucí desetinnou čárkou

bity na mantisu/exponent

- float: 23/8, cca 7 platných desetinných míst
- double: 52/11, cca 16 platných desetinných míst

Typy s plovoucí desetinnou čárkou

bity na mantisu/exponent

- float: 23/8, cca 7 platných desetinných míst
- double: 52/11, cca 16 platných desetinných míst

Speciální hodnoty:

- NaN
- infinity, negative infinity

konstanty s desetinnou tečkou: 1.0 (double) 1.0f (float)

Pravdivostní hodnota

bool

- true
- false (default)

zabírá celý byte v paměti

Znak

char

- konstanty v jednoduchých uvozovkách (apostrofech)

```
'a'
' '
'\n' odradkovani
'\'' apostrof
'\\' backslash
```

Řetězec

string

- konstanty v horních uvozovkách

```
""
"␣"
"ahoj"
"dobry\nden"
"c:\\data\\idt"
```

Přiřazení

Syntaxe:

```
<promenna>=<vyraz>;
```

Přiřazení

Syntaxe:

```
<promenna>=<vyraz>;
```

Vyhodnocení:

1. zjistí se hodnota (výsledek) výrazu
2. hodnota proměnné se nahradí hodnotou výrazu

Přiřazení

Syntaxe:

```
<promenna>=<vyraz>;
```

Vyhodnocení:

1. zjistí se hodnota (výsledek) výrazu
2. hodnota proměnné se nahradí hodnotou výrazu

Datový typ výsledku výrazu musí odpovídat typu proměnné

- kontroluje překladač, pokud neodpovídá, nepřeloží se

Přiřazení

Syntaxe:

```
<promenna>=<vyraz>;
```

Vyhodnocení:

1. zjistí se hodnota (výsledek) výrazu
2. hodnota proměnné se nahradí hodnotou výrazu

Datový typ výsledku výrazu musí odpovídat typu proměnné

- kontroluje překladač, pokud neodpovídá, nepřeloží se

Nealokuje se (trvale) žádná paměť

- pokud nezpůsobí alokaci samotný výraz

Přiřazení - rozdíl oproti Pyhtonu

Vyhodnocení v Pythonu:

1. zjistí se hodnota výrazu
2. alokuje se paměť (na haldě) pro výsledek
3. do proměnné se přiřadí odkaz na výsledek

Alokuje se vždy nová paměť

Přiřazení - rozdíl oproti Pyhtonu

Vyhodnocení v Pythonu:

1. zjistí se hodnota výrazu
2. alokuje se paměť (na haldě) pro výsledek
3. do proměnné se přiřadí odkaz na výsledek

Alokuje se vždy nová paměť

- i když se pracuje s primitivními typy
- i když datový typ výsledku výrazu odpovídá současnému datovému typu na který odkazuje proměnná

Důvod: při překladu není známý datový typ výsledku výrazu, může se tedy lišit (velikostí) od datového typu na který odkazuje proměnná

Operace s čísly

Celá čísla:

```
+
-
*
/
(unarni) -
% (zbytek po deleni)
```

neřešte precedenci, závorkujte
výsledkem dělení je celé číslo (!)

Operace s čísly

`++`

- za proměnnou: použije současnou hodnotu a pak zvýší o jedna
- před proměnnou: zvýší o jedna a pak použije zvýšenou hodnotu

```
int a = 5;
int b = a++;
int c = ++a;
b++;
++c;
```

Operace s čísly

```
--
```

- za proměnnou: použije současnou hodnotu a pak sníží o jedna
- před proměnnou: sníží o jedna a pak použije sníženou hodnotu

```
int a = 5;
int b = a--;
int c = --a;
b--;
--c;
```

Operace s čísly

Čísla s plovoucí desetinnou čárkou:

```
+
-
*
/
(unarni) -
```

výsledkem dělení je číslo s plovoucí desetinnou čárkou

Operace s čísly

```
int a = 3;
int b = 2;
int c = a/b;
float d = a/b;
```

Operace s čísly

```
int a = 3;
int b = 8;
bool q1 = (a == b);
bool q2 = (a < b);
bool q3 = (a >= b);
bool q4 = (a != b);
```

(pseudo)náhodná čísla

```
Random r = new Random(0);
int x = r.Next(10); // 0-9
double y = r.NextDouble(); // 0-1
```

Logické operace

```
bool a = true;
bool b = false;
bool q1 = (a && b); AND
bool q2 = (a || b); OR
bool q3 = !q2; NOT
```

Konzolový výstup

Statické metody třídy Console

```
string s = "Hello world";
Console.WriteLine("Ahoj svete");
Console.WriteLine(s);
Console.WriteLine("s");
int x = 42;
Console.WriteLine(x);
float y = 42.0;
Console.WriteLine(y);
```

Konzolový vstup

Statické metody třídy Console

```
string s = Console.ReadLine();
Console.ReadKey();
```

Bloky

- tam, kde se očekává jen jeden příkaz, ale chceme jich napsat víc
- typicky: větvení, cykly

```
{
  Console.WriteLine("Ahoj␣svete");
  Console.WriteLine("Jak␣se␣mas?");
}
```

Bloky

- tam, kde se očekává jen jeden příkaz, ale chceme jich napsat víc
- typicky: větvení, cykly

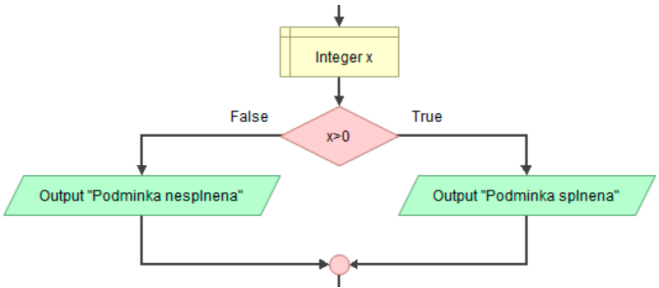
```
{
  Console.WriteLine("Ahoj␣svete");
  Console.WriteLine("Jak␣se␣mas?");
}
```

- proměnné „žijí“ jen ve svém bloku a ve vnořených blocích

```
{
  int y = 5;
}
{
  float y = 8.2f;
}
```

- odsazení: *zvyk, bez syntaktického významu!*

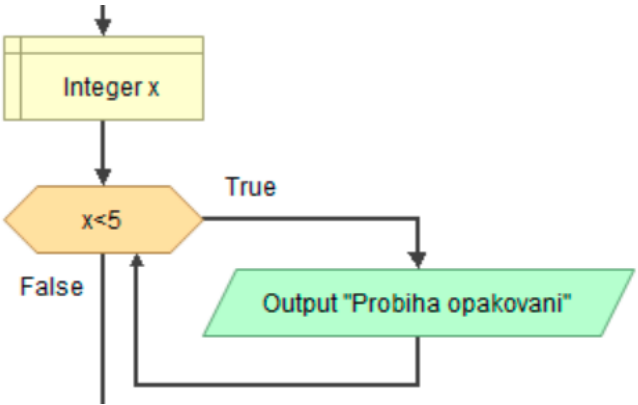
Větvení




```
int x;

if (x > 0)
{
    Console.WriteLine("Podminka splnena");
}
else
{
    Console.WriteLine("Podminka nesplnena");
}
```

Cyklus while




```
int x;

while (x < 5)
{
    Console.WriteLine("Probiha opakovani");
}
```

Cyklus for

Ve skutečnosti jen zkrácený while

```
for(<a>;<b>;<c>)<d>

...

<a>
while(<b>)
{
  <d>;
  <c>;
}
```

Cyklus for

Typické použití

```
for(int i = 0;i<10;i++)
  Console.WriteLine(i);

...

int i = 0;
while(i<10)
{
  Console.WriteLine(i);
  i++;
}
```

Cyklus for - častý problém

Co vypíše následující program:

```
for(int i = 0;i<10;i++);
  Console.WriteLine(i);
```

Switch

```
switch (x)
{
  case 0:
    Console.WriteLine("nula");
    break;
  case 1:
    Console.WriteLine("jedna");
    break;
  case 2:
    Console.WriteLine("dva");
    break;
  default:
    Console.WriteLine("Neco␣jineho");
    break;
}
```

Pole

Datový typ

```
<typ>[]
```

Příklad:

```
int[] array1;
double[] array2;
```

Obsah: reference (default null)
Nutno alokovat paměť pro konkrétní počet prvků: operátor `new`.
Příklad:

```
new int[5];
```

1. alokuje paměť pro 5 intů (pole)
2. vrátí referenci na alokované pole

```
array1 = new int[8];
double[] array3 = new double[1024];
```

Přístup k prvku

Indexování vždy od nuly!

```
int[] data = new int[5];
data[0] = 1;
int x = data[1];
data[4] = 8;
data[5] = 9; // chyba
```

Přístup k prvku

Indexování vždy od nuly!

```
int[] data = new int[5];
data[0] = 1;
int x = data[1];
data[4] = 8;
data[5] = 9; // chyba
```

Počet prvků pole

```
int[] data = new int[5];
int c = data.Length; // 5
```

Dvourozměrné pole

Datový typ

```
<typ>[,]
```

Příklad:

```
int[,] array1;
double[,] array2;
```

Obsah: reference (default null)
Nutno alokovat paměť pro konkrétní počet prvků: operátor `new`.
Příklad:

```
new int[5,8];
```

1. alokuje paměť pro 5x8 = 40 intů (pole)
2. vrátí referenci na alokované pole

```
array1 = new int[5,8];
double[,] array3 = new double[3,3];
```

Vícerozměrné pole

```
int[,,,,] cosi = new int[3,3,3,3,3];
cosi[0,1,2,0,1] = 42;
```

Pole je reference

```
int[] a = new int[3];
int[] b = a;
b[0] = 42;
Console.WriteLine(a[0]);
```

Pole polí

Použiju pole jako datový typ v definici pole

```
int[][] a;
```

Obsah proměnné: reference na pole referencí (teď zrovna null)

```
a = new int[5][];
```

Obsah proměnné: reference na pole referencí (teď zrovna pět referencí, všechny null)

```
a[0] = new int[6];
a[1] = new int[6];
```

Pole polí

Použiju pole jako datový typ v definici pole

```
int[][] a;
```

Obsah proměnné: reference na pole referencí (teď zrovna null)

```
a = new int[5][];
```

Obsah proměnné: reference na pole referencí (teď zrovna pět referencí, všechny null)

```
a[0] = new int[6];
a[1] = new int[6];
```

```
for (int i = 0;i<5;i++)
{
  a[i] = new int[6];
}
```

Pole polí

Použiju pole jako datový typ v definici pole

```
int[][] a;
```

Obsah proměnné: reference na pole referencí (teď zrovna null)

```
a = new int[5][];
```

Obsah proměnné: reference na pole referencí (teď zrovna pět referencí, všechny null)

```
a[0] = new int[6];
a[1] = new int[6];
```

```
for (int i = 0;i<5;i++)
{
  a[i] = new int[6];
}
```

```
a = new int[5][6]; // chyba!
```

Konverze datových typů

Nehrozí-li ztráta přesnosti, pak není třeba dělat nic

```
int x = 5;
long y = x;
float a = 3.1415f;
double b = a;
```

(přesto ke konverzi dochází a je dobré o ní vědět)

Konverze datových typů

Hrozí-li ztráta přesnosti, pak je třeba provést přetypování

```
long x = 5;
int y = (int)x;
double a = 3.1415;
float b = (float)a;
```

(výpočetní náročnost konverze stejná jako v předchozím případě)

Dělení celých čísel

```
int a = 3;
int b = 5;
double d = (double)a/b;
```

Převod na řetězec

```
int a = 5;
double b = 13.2;
string s = String.Format("{0}, {1}", a, b);
Console.WriteLine(s);
```

Převod z řerězce

```
string s1 = Console.ReadLine();
int x = Integer.Parse(s1);
string s2 = Console.ReadLine();
double y = Double.Parse(s2);
```

Když se nepovede (např. řetězec není číslo), tak program havaruje.

Souborový výstup

```
FileStream fs = new FileStream("file.txt", FileMode.Create);
StreamWriter sw = new StreamWriter(fs);
sw.WriteLine("jedna␣radka␣textu");
sw.WriteLine("druha␣radka␣textu");
sw.Close();
```

Souborový vstup

```
FileStream fs = new FileStream("file.txt", FileMode.Open);
StreamReader sr = new StreamReader(fs);
string line = sr.ReadLine();
while (line!=null) {
    Console.WriteLine(line);
    line = sr.ReadLine();
}
sr.Close();
```