

Distribuirani procesi Vježbe 01 – Uvod i threadovi

Vinko Petričević



Gradivo i način polaganja

- gradivo: osnove jezika C# potrebne za razumijevanje distribuiranog programiranja
- način vježbi – svaki 3. tjedan 3 sata u komadu
- Na praktikumu 3 će se detaljnije raditi C#



Sintaksa

- Vrlo slična C++-u, case sensitive
- .net framework identičan za sve .net alate
- Gotovo svi tipovi podataka su GC pokazivači (class – GC pokazivač, zauzima prostor na heapu, nemora se uništiti odmah na završetku programskog bloka; struct – kao C++ klasa, zauzima prostor na stogu)
- klasa se može naslijediti samo iz jedne klase, ali može implementirati više sučelja
- struktura ne može naslijediti klasu, niti se iz nje može nasljeđivati, ali može implementirati sučelje



Osnove jezika

- Programi su organizirani kao projekti
- Svaki projekt sadrži neke obavezne datoteke, ali možemo dodati proizvoljan broj datoteka i poddirektorija (dobro je svaki klasu pisati u zasebnoj datoteci)
- Kompajliranjem iz svih datoteka projekta nastaje jedan assembly (.dll ili .exe), koji je osnovna jedinica za ponovnu uporabu na .net platformi



Osnove jezika

- C# je dio .net platforme, može koristiti Framework Class Library (FCL)
- Programi napisani u C# se izvršavaju u Common Language Runtime-u (CLR), što omogućuje korištenje kasa implementiranih u bilo kojem .net programskom jeziku



Jednostavan program

```
class Hello
{
    static void Main()
    {
        // ispis teksta na ekran
        System.Console.WriteLine("Hello World");
    }
}
```

- Možemo koristiti bilo koji tekst-editor i csc.exe ili VS: File -> New -> Project...
 - Project Types: Visual C# -> Windows
 - Templates: Console Application
 - Odabrati ime (npr: Hello)



Zadatak 1

- Koristeći Vusal Studio, napišite program koji ispisuje pozdravnu poruku
- Uočite razlike u programu kojeg je definirao wizzard

Osnove jezika

- C# je strogo tipizirani jezik
- Imamo ugrađene i korisnički definirane tipove
- Prema načinu spremanja u memoriji se dijele na vrijednosne (stog) i referentne (heap, GC)

Vrijednosni tipovi

Tip	Veličina (u bajtovima)	.NET tip	Opis
bool	1	Boolean	true ili false
byte	1	Byte	od 0 do 255 (od 0 do 2 ⁸ -1)
char	2	Char	pojedinačni Unicode znak ('A' ili '\u0041' ili '\u0000' ili '\u0000')
sbyte	1	SByte	od -128 do 127 (od -2 ⁷ do 2 ⁷ -1)
short	2	Int16	od -32 768 do 32 767 (od -2 ¹⁵ do 2 ¹⁵ -1)
ushort	2	UInt16	od 0 do 65 535 (od 0 do 2 ¹⁶ -1)
int	4	Int32	od -2 147 483 648 do 2 147 483 647 (od -2 ³¹ do 2 ³¹ -1)
uint	4	UInt32	od 0 do 4 294 967 295 (od 0 do 2 ³² -1)
float	4	Single	7 decimala, od ±1,5 x 10 ⁻⁴⁵ do ±3,4 x 10 ³⁸
double	8	Double	15-16 decimala, od ±5,0 x 10 ⁻³²⁴ do ±3,4 x 10 ³⁰⁸
decimal	16	Decimal	28-29 decimala, od ±1,0 x 10 ⁻²⁸ do ±3,4 x 10 ²⁸
long	8	Int64	od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807 (od -2 ⁶³ do 2 ⁶³ -1)
ulong	8	UInt64	od 0 do 18 446 744 073 709 551 615 (od 0 do 2 ⁶⁴ -1)

Enum i struct

Osnove jezika

- Implicitne pretvorbe rade, ako se ne gube podaci

```
int x=5;
double y = x;
```

- Za obrnuto moramo navesti da 'znamo šta radimo'

```
double x=5;
int y = (int)x;
```

Jednostavan program

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            string s = Console.ReadLine();
            int x = Convert.ToInt32(s);
            for(int i=0; i<x; ++i)
                Console.WriteLine(i);
        }
    }
}
```

Jednostavan program

```
int x;
try
{
    x = Convert.ToInt32(s);
}
catch
{
    x = 10;
}
```

Varijable

- Prije korištenja, svaku varijablu moramo inicijalizirati

```
int x=5, y;
y = 10;
WriteLine("x={0}, y={1}", x, y);
```

- Možemo im mijenjati vrijednost
- Konstante moramo odmah inicijalizirati, i kasnije im se vrijednost ne može mijenjati

```
const int z=15;
```

string

- string je niz znakova. Razmaci se ne zanemaruju

```
string s = "abc def";
s = "abc\ndef";
s = @"abc\ndef";
s = Console.ReadLine();
```

Operatori i tijek programa

- Operatori su isti kao u C++-u
- Kontrola tijeka vrlo slična:
 - foreach – prolazi kroz sve elemente nekog spremnika
 - switch – ne mora biti cijelobrojni tip i možemo koristiti goto case slučaj

```
switch(i) {
    case 1: Console.Write(1); goto case 3;
    case 2: Console.Write(2); break;
    case 3: Console.Write(3); break;
    default: Console.Write("nesto drugo"); break;
}
```

Zadatak 2

- Napišite program koji za uneseni broj provjerava je li prost
- Napišite program koji uneseni broj rastavlja na proste faktore

Slanje parametara funkciji

- Svaka funkcija može imati proizvoljan broj parametara proizvoljnog tipa
- Svaki parametar se može slati po vrijednosti, ili po referenci (ref), a može biti i čisto izlazni parametar (out). Tada se prije korištenja u funkciji treba inicijalizirati.

Slanje parametara funkciji

```
static void f1(int x) { ++x; }
static void f2(ref int x) { ++x; }
static void f3(out int x) {
    // ++x; ovo bi bila greska
    x = 0; ++x;
}
static void Main(string[] args)
{
    int x = 10;
    f1(x); Console.WriteLine(x); // 10
    f2(ref x); Console.WriteLine(x); // 11
    f3(out x); Console.WriteLine(x); // 1
}
```

Zadatak 3

- Napišite funkciju koja vraća najmanji prosti broj veći od danog broja
- Napišite naredbu koja ispisuje sve proste brojeve manje od danog broja
- Napišite naredbu koja od dobivenog stringa napravi palindrom

Enumeracije

```
enum imeEnumeracije [:osnovniTip = int]
{
    const1 = vrijednost1, // ako ne stavimo podrazumijeva se 0
    const2 = vrijednost2, // ako ne stavimo, bit ce prethodna+1
    ...
}
```

```
enum Temperature {
    ledisteVode = 0,
    vrelisteVode = 100,
}
static void Main(string[] args) {
    Console.WriteLine("Ledište vode: {0}",
        (int)Temperature.ledisteVode);
    Console.WriteLine("Vreliste vode: {0}",
        (int)Temperature.vrelisteVode);
}
```

Klase

```
class automobil
{
    public int gorivo = 100;
    public automobil()
    {
        Console.WriteLine(gorivo);
    }
    ~automobil()
    {
        Console.WriteLine(gorivo);
    }
}
class Program
{
    static void Main()
    {
        automobil a = new automobil();
        ++a.gorivo;
    }
}
```

Nasljeđivanje

```
class kupe : automobil
{
    public kupe()
    {
        Console.WriteLine("Kupel");
    }
    ~kupe()
    {
        Console.WriteLine("@Kupel");
    }
}
class Program
{
    static void Main()
    {
        automobil a = new kupe();
        ++a.gorivo;
        Console.WriteLine("kraj\n");
    }
}
```

Threadovi

- programi koje smo do sada susretali izvršavali su se uglavnom u jednom threadu (na jednoj jezgri procesora)
- ukoliko bi željeli upotrijebiti više, ili zbog drugih stvari, potrebno je kreirati više threadova
- svaki thread ima svoj stog (lokalne varijable, parametri) i svoju početnu rutinu

Threadovi

- Postoji više načina kreiranja threadova:
- void pocetak() { ... }
...
Thread t = new Thread(new ThreadStart(pocetak));
t.Start();
- void radi(object o) { ... }

ThreadPool.QueueUserWorkItem(new
WaitCallback(radi), parametar);

Kreiranje threada

using System.Threading;

```
int threadNo = 0;
Program(int no) { threadNo = no; }
void drugiThread()
{
    for (int i = 0; i < 10; ++i)
    {
        Thread.Sleep(100);
        Console.WriteLine("Thread: {0:D}, i={1:D}", threadNo, i);
    }
}
static void Main()
{
    Program p1 = new Program(1);
    Thread t = new Thread(new ThreadStart(p1.drugiThread)); t.Start();
    // Program p = new Program(0); p.drugiThread();
}
```

Thread objekt

- Sleep(int miliSec) – čeka, procesor nije opterećen
- SpinWait(int vrijeme) – vrti petlju određeno vrijeme. Ovisi o procesoru koliko, procesor je opterećen
- VolatileRead(obj) – čita podatak iz memorije
- VolatileWrite(obj) – piše podatak u memorije

Thread klasa

- Start() – pokreće thread
- Abort() – prekida izvršavanje threada
- Suspend() – pauzira thread
- Resume() – pokreće pauzirani thread
- Join(int miliSec) – čeka završetak threada
- Interrupt() – prekida čekanje threada.

Primjeri

- Primjere startamo tako da u VS-u kreiramo novi projekt, odaberemo C# i Console Application, te kopiramo kôd primjera u glavnu .cs datoteku

Lock – primjer 1.1

- globalne i statičke varijable (a i drugi resursi mogu biti) zajednički threadovima, pa je ponekad potrebno osigurati da samo jedan thread pristupa varijabli
- lock(obj) – zaključava kritični dio kôda. Ako je objekt zaključan, drugi thread nemože ući u taj dio kôda.
 - obj je bilo koji objekt izveden iz Object (dakle svaka klasa)

AutoResetEvent – primjer 1.2, 1.3

- Wait(int miliSec) – čeka sve dok ne istekne vrijeme ili objekt ne bude signaliziran naredbom Set. Samo jedan thread može nastaviti s radom, te objekt odmah prelazi u nesignalizirano stanje
- Set() – signalizira da jedan thread može nastaviti sa radom (ako čeka)
- Reset() – ako je objekt u signaliziranom stanju, prelazi u nesignalizirano
- ManualResetEvent – nakon Set, svi čekajući threadovi mogu nastaviti s radom, sve dok se ne pozove Reset.

Ostale sinhronizacijske rutine



- Postoje i brojni drugi načini za kreiranje threada, te za asinhrono pozivanje funkcija (neki su navedeni u primjerima)
- System.Threading sadrži još mnoštvo sinhronizacijskih mehanizama, poput Semafora, Mutexa, za detalje pogledati help od System.Threading