Nasveti za programiranje vaj

Tečaj C#

Igor Grešovnik

1 KORISTNI NAMIGI IN NASVETI

1.1 Gradiva in viri

Vsa gradiva za tečaj najdete na naslovu
http://dl.dropbox.com/u/12702901/csharp_course/index.html

1.2 Datotečna struktura projektov

Programe in projekte v C# je priporočljivo umestiti v dogovorjeno datotečno strukturo. Osnova naj bo direktorij *csharp*, ki je dostopen preko spletne strani tečaja. Ta direktorij naj si vsak skopira z FTP računa na lokalni disk in si ga posodablja, kadar so v njem kakšne. Znotraj tega direktorija naredi vsak svoj **osebni direktorij**, katerega ime je sestavljeno iz priimka in imena, ki sta ločena s piko. Zntraj osebnega direktorija naj bodo ločeni direktoriji, ki vsebujejo projekte z narejenimi nalogami. V osebnem direktoriju naj bo ena sama rešitev (”solution”, končnica ”.sln”), ki naj vsebuje vse projekte z domačimi. Tudi ime rešitve naj bo sestavljeno iz priimka in imena avtorja, ki sta ločena s piko (seveda mora imeti še ustrezno končnico, ”.sln”).

Projektov iz direktorijev csharp/external in csharp/gresovnik naj se ne kopira v svoj osebni direktorij, ampak naj se v svoji rešitvi te projekte doda iz originalnih lokacij.

Priporočljivo je, da se pri pošiljanju in prenašanju projektov zbriše iz osebnega direktorija vse datoteke s končnicami *”.pdb”, ”.exe”* in *”.dll”*. Tako zavzamejo projektni direktoriji precej manj prostora. Zaradi varnosti je priporočljivo, da se brisanje omenjenih datotek izvede na kopiji, ki je pripravljena za kopiranje ali pošiljanje, in ne na originalu.

1.3 Namig: Nastavljanje projektov za naloge

Pričujoči namig govori o tem, kako si na enostaven način urediš projekte in rešitev v C# za izdelavo domačih in končne naloge tako, da so projektu urejeni v skladu z navodili.

Pri domačih nalogah bo večkrat treba uporabiti projekte, ki so v direktoriju csharp/. V csharp/external/ so zunanje knjižnice, ki jih bomo včasih uporabljali, v direktoriju csharp/gresovnik/ pa je nekaj primerov programov in knjižnic za učenje in pomoč pri reševanju problemov. V direktoriju csharp/gresovnik/ je več poddirektorijev, ki vsebujejo vsak svoj projekt, projekti pa so povezani v rešitev *test-solution.sln*. V C# je projek osnovna enota prevajanja, rešitev pa je formalna struktura za povezovanje več sorodnih projektov, ki omogoča pregledno grupiranje programske kode in enostavno referenciranje med projekti v razvojnem projektu.

Svoj direktorij z domačimi nalogami narediš v direktoriju csharp/. Tu je tudi direktorij *gresovnik.igor/* , ki vsebuje rešitev *gresovnik.igor.sln*, ki je podobno organizirana kot naj bi bili projekti z nalogami. Zato je najlažje, da v direktoriju csharp/ narediš kopijo direktorija *gresovnik.igor/* in skopiran direktorij preimenuješ s svojim priimkom in imenom. Podobno preimenuješ še rešitev *gresovnik.igor.sln* in

to rešitev uporabiš kot rešitev za svoje naloge. Rešitev odpreš v programu Visual Studio in v tej rešitvi dodajaš projekte z domačimi nalogami. V rešitvi sta že vključena dva projekta, ki predstavljata knjižnici – 00library in Iridium. Ker boš ta projekta v prihodnje potreboval, ju pusti vključena. Vključen je še projekt *domaca_naloga_03_gresovnik*, ki vsebuje primer rešitve domače naloge, lahko ga pustiš v rešitvi ali odstraniš.

Za vsako nalogo narediš v rešitvi nov projekt. V oknu Solution Explorer-ja klikneš z desnim gumbom na re rešitev (čisto na vrhu), izbereš »*Add*« in nato »*New Project*«. Nato izbereš »*Console Application*« in spodaj daš še ime projektu (npr. domaca_naloga_01a_novak, če je tvoj priimek Novak). S tem je narejen projekt, v katerem lahko začneš programirati rešitev domače naloge. V projekt je že vključena datoteka Program.cs, ki vsebuje razred z metodo Main, kjer se začne izvajanje programa. To datoteko lahko tudi preimenuješ, ne smeš pa preimenoveti metode Main, ker je to dogovorjeno ime za metodo, kjer se začne izvajanje programa.

Od tu naprej je izdelava projekta zelo enostavna. Nove metode lahko dodajaš v razred, kjer je metoda Main(), lahko tudi definiraš nove razrede v isti datoteki. Koda pa bo bolj pregledna, če jo organiziraš po datotekah, v katerih dodajaš razrede. Novo datoteko z razredi narediš tako, da z desnim gumbom klikneš na projekt, izbereš »*Add*«, »*New Item*« in nato »*Class*«. Poskrbi za to, da dodajaš projekte in datoteke v pravem jeziku, torej v C# (v okolju Visual Studio je včasih možno dodajati projekte v različnih jezikih).

Najbolj pregledno je, da je v metodi Main (ki je začetna oz. vstopna točka izvajanja programa) samo kličeš druge metode, ki so organizirane v razredih, ali po potrebi narediš kak objekt. Koda naj bo organizirana v razredih po vsebini, razredom in metodam pa dajaj imena, iz katerih je možno sklepati, čemu so namenjeni.

Bodi pozoren/a na imenske prostore, v katerih so definirani razredi in v njih metode. Če nekje v kodi naslavljaš razred, ki je v definiran drugem *imenskem prostoru* (blok *namespace*, v katerm je razred vključen), potem pri naslavljanju ali navedeš polno ime (fully classified name) sestavljeno iz imenskega prostora in imena razreda, ali pa vključiš imenski prostor z direktivo *using* na vrhu datoteke, npr.:

```
using domaca_naloga_01a_novak;
```

Za vse, kar je znotraj danega projekta, ni potrebno vključevati referenc v projekt. Če pa v projektu uporabljaš kodo, ki je definirana izven projekta (enako velja za prevedene knjižnice), moraš dodati ustrezne reference. Najbolj enostavno je dodati referenco na projekt, ki je vključen v rešitvi. Zato v rešitev vedno vključiš vse projekte, katerih dele potrebuješ pri delu, tudi, če so projekti definirani v direktoriju izven direktorija, ki vsebuje rešitev. V rešitev, ki služi kot šablona za domače naloge, je že vključen projekt *00library*, ki je definiran v direktoriju csharp/gresovnik. Dodaten projekt vključiš tako, da z desnim gumbom miške klikneš na rešitev v Solution Explorer-ju, izbereš »Add« in nato »Exsisting Project«. V okencu, ki se odpre, navigiraš do direktorija, ki vsebuje željeni projekt, in projektno datoteko (končnica .csproj) vključiš v rešitev.

Če kjerkolu v projektu za domačo nalogo uporabljaš kakšno stvar (recimo tam definiran razredrazred) iz drugega projekta vključenega v rešitev, moraš v projektu z nalogo dodati referenco na tisti projekt. Na vrhu datoteke, v kateri naslavljaš stvari iz drugega projekta, pa zapišeš direktivo

```
using<imenski_prostor>;
```

Tu imenski_prostor nadomestiš z dejanskim imenom imenskega prostora, v katerem je definirana stvar, ki jo želiš uporabiti.

Referenco na projekt, ki je vključen v rešitev, dodaš v svojem projektu tako, da klikneš z desnim gumbom miške na svoj projekt in izbereš »*Add Reference...*« (po tem navadno nekaj časa traja, preden se odpre okence za izbiranje referenc), v odprtem okencu izbereš zavihek »*Projects*« in izbereš projekt, ki ga želiš nasloviti. Še enkrat je treba opozoriti, da sama referenca na projekt ni dovolj, v datoteki, kjer uporabljaš stvari iz tistega projekta, moraš doddati tudi ustrezne direktive *using*, ki vključijo imenski prostor, ki sledi direktivi. Tako ti razredov ne bo potrebno naslavljati s polnimi imeni (fully qualified names).

Kadar je več nalog v isti rešitvi, si kodo organizirajte tako, da v metodi *Main* le kličete nekaj metod, ki izvedejo vse rešitve nalog.

Določen **projekt**, ki naredi program, **poženeš** tako, da ga najprej narediš za aktivnega (v Solution explorer-ju klikneš z desnim gumbom na dotični projekt in izbereš "*Set as Startup Project*"), potem pa **pritisneš tipko F5**.

1.4 Namig: Osnovne aritmetične operacije v C#

Kot večina sodobnih programskih jezikov ima tudi C# preddefinitanih nekaj osnovnih matematičnih funkcij in konstant, kot so kotne funkcije, potence z realno osnovo in eksponentom, eksponentna fukcija in logaritem, število π ter osnova naravnega logaritma e. Te stvari so definirane v razredu **Math** v imenskem prostoru **System**.

Izčrpne podatke o tem zlahka najdete na straneh MSDN. Če v polju za iskanje vtipkate iskalni niz "Math class", dobte med zadetki glavno stran z opisom razreda Math:

http://msdn.microsoft.com/en-us/library/system.math.aspx

Na strani je zgoraj kratek opis razreda s podatkom o imenskem prostrom, kjer je razred definiran. Sledijo primeri in podrobnejše razlage, na koncu strani pa je v poglavju "See also" tudi povezava z naslovom "Math members" do naslednje strani:

http://msdn.microsoft.com/en-us/library/system.math_members.aspx

Na tej strani so podatki o vseh metodah in poljih, ki so definirani v razredu Math. Tu zlahka najdete ime metode, ki jo potrebujete. Za kakšno stvar je dobro vedeti angleški izraz; angleški izraz za potenco je "power", z iskanjem tega izraza na zgornji strain hitro ugotovimo, da je ime metode za izračun potence "Pow". Tako na primer izraz

```
double x = Math.Pow(Math.PI, 1.5)
```

izračuna $\pi^{3/2}$. Da se zgornja vrstica pravilno prevede, morate seveda nekje na vrhu datoteke vključiti stavek

```
using System;
```

Brez tega stavka morate pri sklicevanju na razred Math navesti tudi imenski prostor, torej

```
double x = System.Math.Pow(System.Math.PI, 1.5)
```

1.5 Namig za kopiranje iz konzole

Kopiranje v konzoli je možno le, če to nastaviš pri lastnostih konzole. Klikneš na kono konzole v levem zgornjem kotu okna, izbereš Properties/Options/ in pod Edit Options vse odkljukaš.

Kljub temu je kopiranje lahko nerodno, če je teksta za več vrstic ali so v konzoli lomljene vrstice. Problem lahko rešiš tudi tako, da poženeš program, ki izpisuje v konzolo, in mu preusmeriš standardni izhod (ki je prednastavljen na konzolo) v izbrano datoteko. To narediš tako, da program poženeš na naslednji način:

program.exe argumenti... > out.txt

Tu je predpostavljeno ime programa *program.exe*. Ko prevedeš projekt v C#, katerega rezultat je program, lahko ta program najdeš nekje znotraj projektnega direktorija, tipično v direktoriju /bin/debug relativno glede na projektni direktorij.

Z opisanim načinom so lahko problemi, kadar program bere podatke, ki jih vstavi uporabnik preko konzole. V takšnem primeru je najbolje v programu izključiti vsa interaktivna branja in potem pognati

program s preusmerjenim standardnim izhodom. Droga možnost je, da v programu sprogramiraš tudi izpise v datoteko, pri čemer si lahko pomagaš z razredom StreamWriter.

1.6 Namig - zapisovanje rezultatov v datoteko

V datoteko se zapisujejo rezultati podobno kot v konzolo, za to lahko uporabiš razred StreamWriter. Primer:

```
static void FileTest()
{
    StreamWriter sw = new StreamWriter("c:\\temp\\test.txt", false  /* append */);
    sw.WriteLine("Zapis v testno datoteko... ");
    sw.WriteLine("  Cas: " + DateTime.Now.ToString());
    sw.WriteLine("  Pi ~ " + Math.PI.ToString());
    sw.WriteLine("Konec zapisa. ");
    sw.Close();
}
```

1.7 Namig - merjenje časov računanja

Čase računanja lahko merimo z razredom *Timer*, ki je definiran v
csharp/gresovnik/00library/time.cs.

Za uporabo teha rezreda moramo v rešitev vključiti projekt 00library in v projekt dodati referenco na ta projekt.

Z razredom merimo čas po uri in CPU čas. CPU čas je čas, ki ga je našemu procesu namenil procesor računalnika in je pri merjenju učinkovitosti algoritmov bolj pomemben podatek. To je zaradi tega, ker znotraj časa po uri procesor uporabljajo tudi drugi procesi, ki tečejo na računalniku, zaradi česar traja določebna operacija dlje časa, kadar se na računalniku izvajajo še drugi procesi, ki porabljajo veliko procesorskega časa. Vseh teh procesov v večopravilnem okolju ne moremo v celoti kontrolirati, zato podajamo tudi CPU čas, ki je dejanski čas, ko je centralno procesno enoto računalnika obremenjeval naš proces.

Primer uporabe:

```
// Naredimo štoparico, s katero merimo čas.
// V konstruktorju z argumentom podamo ime, po katerem
// lahko identificiramo narejeno štoparico.
Timer t = new Timer("Štoparica št. 1");
...
    t.Start();  // sprožimo merjenja časa
...  // Izvajanje računskih operacij
    t.Stop();  // ustavimo štoparico štoparico
// Izpišemo izmerjen čas
    Console.WriteLine("Čas potreben za računsko operacijo:");
    Console.WriteLine("    t = " + t.Time + " s (CPU: " + t.CpuTime + " s)");
    t.Start();  // ponovno sprožimo merjenja časa
...  // Izvajanje naslednjega bloka računskih operacij
    t.Stop();  // ustavimo štoparico štoparico
// Izpišemo izmerjen čas
    Console.WriteLine("Čas potreben za računsko operacijo:");
    Console.WriteLine("    t = " + t.Time + " s (CPU: " + t.CpuTime + " s)");
// Izpišemo lahko tudi skupen čas, ki je seštevek vseh izmerjenih časov med
// t.Start() in t.Stop():
    Console.WriteLine("Čas potreben za vse merjene operacije skupaj:");
```

```
            Console.WriteLine("    t = " + t.TotalTime + " s (CPU: " + t.TotalCpuTime
+ " s)");
        // Stanje štoparice (skupni čas in čas zadnjega merjenja) lahko izpišemo
        // tudi na bolj enostaven način z uporabo metode ToString():
            Console.WriteLine(t.ToString());
```