

3. POGLAVLJE

Programski jezici

U ovom poglavlju:

- Osnovno o programskim jezicima
- Jezici dostupni u .NET-u
- Vaš prvi program
- Vodič kroz sintaksu jezika VB.NET i C#

Opće je poznato da svaki programer ima preferirani programski jezik koji koristi, voli i u koji se kune. Iako svaki od jezika ima svojih pozitivnih i negativnih strana, omiljeni programski jezik bira na temelju slučaja i okolnosti u kojima počinjete programirati. Mi vam tu ne dajemo neki izbor – cijela knjiga, izuzevši ovo poglavlje, bavit će se isključivo C# programskim jezikom, što iz uredničko-tehničkih razloga (nisu nam dali pisati knjigu od tisuću stranica), što iz političko-religijskih (i mi imamo svoj omiljeni jezik). Naravno, postoji i objektivni razlog za favoriziranje – radi se o jeziku koji je stvoren upravo za .NET platformu, pokušavajući sintetizirati jednostavnost Visual Basic a i snagu C++-a.

Ipak, u ovom poglavlju ćemo se detaljno pozabaviti i sintaksom jezika Visual Basic .NET kako bismo onima kojima su bliže osnove toga jezika omogućili bezbolniji prelazak na novu, “ceoliku” sintaksu. Ne pokušavamo vas obratiti na novu “religiju” – samo želimo napomenuti da je novi VB.NET značajno promijenjen u odnosu na stari VB te da je učenje tih novotarija idealna prilika za prihvatanje još poneke sitnice i prelazak na C#.

I. DIO: .NET IZNUTRA

Imajte na umu da se .NET Framework prema svim programskim jezicima odnosi ravnopravno te nema tehničkih razloga zbog kojih ne biste programirali u nekom kronično nepopularnom jeziku kao što je, recimo, Haskell. Stoga ćemo prije seciranja sintakse dvaju favoriziranih spomenuti riječ-dvije o najpoznatijim “ostalim” jezicima podržanima u .NET-u.

Jezici u .NET-u

Visual Studio .NET dolazi s podrškom za četiri programska jezika. To su Visual Basic .NET, Visual C++ .NET, Visual C# .NET i Visual J# .NET. Izbor jezika strogo je “političke” naravi – svaki od nabrojanih jezika ima svoju zadaću u Microsoftovu marketinškom planu: VB kao mamac za početnike, C++ kao udica za iskusne programere, J# za brojne ljubitelje programiranja u Javi i C# – za sve ostale.

Prilikom instalacije Visual Studia možete odabrati koji od tih jezika želite koristiti. Naravno, izbor nije ograničen isključivo na nabrojane jezike. Postoji niz drugih jezika za koju su podršku razvili ostali proizvođači softvera, i koje možete instalirati kao dodatak Visual Studiju te koristiti u programiranju. Neki od tih dodataka imaju svoju cijenu, dok su neki besplatni, no u svakom slučaju o alternativnim jezicima razmišljajte tek ukoliko želite iskoristiti već postojeći kôd – za svako programiranje “od nule” preporučujemo prihvaćanje jednog od “originalnih” jezika.

Razlog tomu je jednostavan – zapnete li negdje, što će se sigurno dogoditi, potražiti ćete pomoć od prijatelja ili na Internetu. Kako većina ljudi programira u nekoliko najpoznatijih jezika, primjer koji rješava problem najlakše ćete pronaći u njima, dok ćete za ostale morati dobiveno rješenje “prevoditi”, ako uopće tako nešto bude moguće.

Osim tisuća primjera na grupama Useneta i raznim web-stranicama, najbolji argument za netom izrečenu konstataciju je originalna Microsoftova dokumentacija koja u većini slučajeva primjere navodi u dva jezika – VB.NET i C#.

Karakteristike jezika u .NET-u

Osnovna i najvažnija karakteristika svakog jezika koji želi raditi u okruženju .NET-a jest objektna orijentiranost. Više o tom pojmu i općenito objektno orijentiranom programiranju čitat ćete u šestom poglavlju, pa ovdje nećemo previše duljiti.

Kao što smo već natuknuli u prošlom poglavlju, bez obzira na jezik koji koristite, služiti ćete se bibliotekama klasa ugrađenim u sâm .NET Framework. Korištenje istih biblioteka jezike međusobno čini puno sličnijima, a paralelno programiranje u više njih jednostavnijim. Štoviše, ta će vam karakteristika omogućiti da “pročitajte” kôd pisan u jeziku čiju sintaksu možda niti ne poznajete.

Odgovor na vječno pitanje – koji jezik izabrati – u okruženju .NET-a nešto je jednostavnije dati. Stvari koje više ne mogu utjecati na odluku su brzina izvršavanja i proširivost. Bez obzira u kojem jeziku

U početku bijaše naredba...

Prvo “nešto” što bi se moglo nazvati pretečom programskih jezika koje danas poznajemo i volimo nastalo je davne 1946. u bavarskim Alpama, skrovištu njemačkog inženjera Konrada Zusea. Svoj je jezik nazvao Plankalkul, no on nije bio korišten u elektroničkim računalnim uređajima. Prvi takav stvoren je 1949. i nazvan je Short Code.

U godinama nakon toga rodilo se nekoliko novih programskih jezika, a jedan od njih, stvoren 1957. godine, bio je Fortran, najstariji i danas živući jezik, ujedno preteča većine današnjih jezika.

Riječ “Basic” zapravo je akronim izraza *Beginner's All-purpose Symbolic Instruction Code*. Napravljen je 1963. godine na koledžu Dartmouth, a tvorci su mu John G. Kemeny i Thomas E. Kurtz. Godinu dana kasnije je prvi put korišten na računalu IBM 704. Zahvaljujući jednostavnosti, jezik je vrlo brzo stekao veliku popularnost i implementiran je na brojne operativne sustave i strojeve, uključujući i svojevrmeno izuzetno popularne Commodore 64 i ZX Spectrum.

Basic je od početaka bio vrlo zanimljiv Microsoftu, te ga je koristio na nekoliko načina i oblika. Uz DOS je neko vrijeme dolazio QBasic, a Windowse je u stopu pratio pod imenom Visual Basic, u nekoliko inačica. Postoji i skriptna inačica jezika nazvana VBScript koja je podržana u Windows Scripting Hostu (WSH) za skripte u Windowsima, ASP-u za serversko te Internet Exploreru za klijentsko skriptiranje web-stranica.

Tvorac jezika C bio je Dennis Ritchie iz Bell Labsa, daleke 1972. godine. Nećete nam vjerovati,

no postojao je i jezik B koji je bio preteča C-a. Nemojte pogađati – njihov se prethodnik nije zvao A, već BCPL, a obiteljsko stablo seže sve do Algola i već spomenutog Fortrana.

Svoju popularnost jezik C duguje ponajprije Unixu zahvaljujući kome se nastanio u srca brojnih programera. Ipak, danas je broj ljudi koji programira baš u C-u manji – većina ih se koristi objektno orijentiranom inačicom C-a nazvanom C++ . (Čisto kao zanimljivost – postoji i programski jezik D, nasljednik dvaju spomenutih C-a, koji nije previše zainteresirao programe-re, no još ima šanse – razvijen je 2003.)

Na temeljima C-a razvijen je, među ostalim, i objektno programski jezik Java. Njega je razvio James Gosling iz kompanije Sun Microsystems, a osnovni adut – portabilnost među platformama – donio mu je veliku popularnost među programerima, posebno onima nevezanim uz Microsoft.

JavaScript, skriptni jezik razvijen od strane tvrtke Netscape, služi za klijentsko skriptiranje u web-preglednicima i razvijen je nevezano s Javom iako s njom dijeli slično ime i sintaksu. Postoji i Microsoftova varijacija na temu JavaScripta nazvana JScript.

Postoji nekoliko tisuća programskih jezika, što mrtvih što živih, a vjerojatno najduži (iako ne i najvažniji) popis možete potražiti na adresi:

<http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm>

I. DIO: .NET IZNUTRA

Tablica 3-1:

Popis trenutno dostupnih alternativnih jezika u .NET-u koje su razvili nezavisni proizvođači

Jezik	Naziv	Adresa
Ada	Ada	http://www.usafa.af.mil/dfcs/bios/mcc_html/a_sharp.html
APL	Dyalog APL	http://www.dyadic.com/
AsmL	AsmL is the Abstract State Machine Language for .NET	http://research.microsoft.com/fse/asml/
CAML	Microsoft Research F#	http://research.microsoft.com/projects/ilx/fsharp.htm
Cobol	Fujitsu Cobol	http://www.adtools.com/info/whitepaper/net.html
Forth	Delta Forth	http://www.dataman.ro/dforth/
Eiffel	Interactive Software Engineering Eiffel for .NET	http://www.eiffel.com/products/envsn10/
Fortran	Lahey/Fujitsu Fortran for .NET	http://www.lahey.com/dotnet.htm
Fortran	Salford FTN95 for Microsoft .NET	http://www.salfordsoftware.co.uk/compilers/ftn95/dotnet.shtml
Haskell	Hugs98 for .NET	http://galois.com/~sof/hugs98.net/
Mercury	Melbourne University Mercury Project	http://www.cs.mu.oz.au/research/mercury/dotnet.html
ML	Standard ML from Microsoft Research	http://www.research.microsoft.com/Projects/SML.NET/index.htm
Mondrian	Mondrian for .NET	http://www.mondrian-script.org/
Oberon	ETH Active Oberon for .net	http://www.oberon.ethz.ch/lightning/
Pascal	Queensland University Component Pascal	http://www.citi.qut.edu.au/research/plas/projects/cp_files/ComponentPascal.html
Perl	ActiveState Perl Dev Kit with PerlNET	http://aspn.activestate.com/ASPN.NET/
Python	ActiveState Python for .NET Research	http://starship.python.net/crew/mhammond/dotnet
Scheme	Northwestern University Hotdog Scheme	http://rover.cs.nwu.edu/~scheme/
Smalltalk	SmallScript from SmallScript LLC	http://www.smallscript.net/

.NET-a programirali, on će na kraju biti preveden u MSIL koji je uvijek jednako brz, iz kojeg god jezika bio preveden. Također, zahvaljujući zajedničkim temeljima, u jednom je modulu moguće koristiti druge module pisane u bilo kojem jeziku.

Drugim riječima, izbor jezika ovisit će ponajprije o vašim vlastitim navikama, potrebama i željama. Ili o autorima knjige koju upravo čitate. Ipak, da vas ne bismo ostavili u neznanju, evo nekoliko crtica o jezicima *po defaultu* podržanim u Visual Studiju .NET.

Visual Basic .NET

Glavni adut jezika Basic (pa sukladno tome i Visual Basica .NET) je njegova čitljivost. Gdje god je moguće naredbe nisu kratice, specijalni znakovi ili akronimi, već obične engleske riječi. Zahvaljujući tome, pukim čitanjem naredbi možete shvatiti o čemu se radi i što bi koji redak mogao značiti. Čitljivost i jasnoća čine ga idealnim jezikom za učenje, što direktno utječe na njegovu popularnost i rasprostranjenost.

Projekcije su da polovica programera koji rade na platformi .NET za to koriste jezik Visual Basic .NET. Osim jednostavnosti, takav rezultat VB.NET može zahvaliti i velikom broju ljudi koji su dosad koristili njegove starije inačice, pa im se prilikom prelaska na novu platformu činilo najlogičnije ostati pri svom jeziku.

Takva odluka je očekivana, no tek djelomično utemeljena. Naime, razlika između “starog” Visual Basica i ovoga označenog nastavkom .NET je velika – mnoge su stvari dodane ili promijenjene kako bi mogao zadovoljiti stroga pravila i zahtjeve koje svim jezicima nalaže specifikacija .NET-a. Microsoft u svojoj dokumentaciji ide toliko daleko da napominje kako se radi o različitim, no srodnim jezicima.

Iako se programerima u “starom” Visual Basicu moglo spočitnuti mnogo toga što im je bilo nedostižno u odnosu na druge jezike, s Visual Basicom .NET ta se razlika osjetno smanjila, ako ne i potpuno nestala. Ostaju, naime, još neke sitnice na koje utječe sama sintaksa, no sjetite se da .NET u svojoj arhitekturi ne favorizira niti jedan od dostupnih jezika – u svima je moguće napraviti jednako dobre i brze aplikacije.

C# .NET

C# (čita se *C-sharp*) pozicionira se kao moderan i inovativan, u potpunosti objektno orijentiran jezik koji balansira između jednostavnosti Visual Basica i snage jezika C++. Ipak, kao što i samo ime govori, radi se o jeziku koji se bazira na sintaksi C-a, tako da će programeri s iskustvom u C++-u, Javi i srodnim jezicima vrlo brzo savladati ovaj potpuno nov jezik.

Iako ga je Microsoft izmislio, C# nije isključivo Microsoftov jezik. Dvije najveće standardizacijske agencije, europska ECMA i međunarodni ISO, proglasile su C# i neke druge tehnologije iz .NET-a standardom, pa već sada taj jezik možemo pronaći u razvojnim alatima drugih proizvođača.

I. DIO: .NET IZNUTRA

Ne treba ignorirati činjenicu da je C# stvoren upravo za .NET Framework. Iako Microsoft stalno naglašava da nema niti će biti favoriziranja C#-a u odnosu na ostale jezike, činjenica je da se upravo on koristi u internim projektima. Ipak, nemojte zbog toga misliti da će ostali jezici trenutno podržani u Visual Studiu .NET ikada biti dovedeni u neravnopravan položaj – jedan od najjačih aduta .NET platforme jest višezjezičnost i od njega Microsoft vjerojatno nikada neće odustati.

J# .NET

Kako bi privukao što veći broj programera u Javi, nastao je jezik nazvan J#. Iako bi se svaki programer u Javi vrlo brzo snašao i u C#-u, postojanje ovog jezika omogućava puno jednostavniju migraciju postojećih aplikacija i iskorištavanje postojećeg znanja. J# je Microsoftov adut i u akademskim krugovima koji su u velikom broju prihvatili Javu.

Treba imati na umu da, unatoč izuzetnoj sličnosti, J# nije Java, nema veze s platformom Java Virtual Machine i može se koristiti isključivo za razvoj aplikacija unutar okruženja .NET.



Iako za njega postoji tek djelomična podrška u Visual Studiju, postoji i jezik nazvan JScript.NET. Radi se o nadogradnji jezika JScript prilagođenoj okruženju .NET-a koja, iako podudarna u nekim stvarima, nije isto što i J#.

Visual C++ .NET

Probajte nekom programeru u C++ reći da njegov jezik može ravnopravno zamijeniti neki tamo C# – dobit ćete takvu bujicu rečenica da će vam biti žao što ste uopće išta govorili. Što je najgora – bit će u pravu!

Naime, u priči o C++ .NET morate zaboraviti na postulat o ravnopravnosti svih jezika jer on, osim mogućnosti dostupnih svakom jeziku, omogućava zaobilaznje .NET Frameworka (pisanje tzv. neupravljanog kôda) i rad na “stari način” – direktno s operativnim sustavom. Riječ “stari” u ovom slučaju nije negativna – ma koliko okruženje poput .NET Frameworka pružalo dodatnih mogućnosti i olakšavalo razvoj stvari koje je predvidio, teško da može zadovoljiti sve potrebe. To se posebno odnosi na pisanje specifičnih aplikacija poput *drivera* za hardverske uređaje, raznih sistemskih alata ili aplikacija kojima je krucijalna izuzetna brzina.

Štoviše, čak i klasične Windows-aplikacije zasada nisu preporučljive za pisanje u okruženju .NET-a. Naime, da bi ih korisnik mogao pokrenuti, mora imati instaliran .NET Framework, što je zahtjev koji ne ispunjava velik broj osobnih računala. Situacija će biti drugačija tek kada .NET Framework postane standardni dio svakog Windows operativnog sustava.

3. POGLAVLJE: PROGRAMSKI JEZICI

Pogledamo li stvari iz drugog kuta, Visual C++ .NET zapravo je ostao isti onaj Visual C++ koji smo poznavali još prije ere .NET-a. Dolazak .NET-a proširio mu je mogućnosti u obliku C++ Managed Extensions, biblioteke koja omogućava korištenje svega što .NET Framework nudi. Sukladno tome, C++ .NET može poslužiti za postepenu migraciju postojećih aplikacija na platformu .NET.

Karakteristike, prednosti i mogućnosti Visual C++ .NET-a (i jezika C++ općenito) izlaze iz okvira ove knjige, pa se njima nećemo baviti.

Odlučite li koristiti C++ isključivo za pisanje upravljanih aplikacija, radit ćete u podskupu ovog jezika koji se naziva Managed C++. On je po mogućnostima jednak i karakteristikama usporediv sa svakim drugim upravljanim jezikom u .NET-u. Tom programeru, onome koji radi isključivo u Managed C++, možete slobodno reći da mu jezik nije ništa bolji od C#.

Vaš prvi program

Kako se proučavanje sintakse što slijedi ne bi svelo na teoretsko razglabiranje, prije ćemo naučiti jednostavan način kako možete sami isprobati svaki primjer i eksperimentirati mijenjajući ga.

Za taj posao poslužiti će nam kompajleri – alati koji će prevoditi kôd koji napišemo u izvršnu datoteku s nastavkom “.exe” koju ćemo moći pokrenuti i na ekranu vidjeti rezultat. Nemojte očekivati čuda – u ovom poglavlju zadržat ćemo se na tzv. konzolskim aplikacijama dok prave poslastice čuvamo za kasnija poglavlja.

Za početak, napišimo jednostavan program...

Zdravo, svijete!

Na početku učenja bilo kojeg jezika tradicionalno se poseže za komadom kôda popularno nazvanu “Hello, world”. Radi se o programčiću kojem je jedini zadatak ispisati na ekranu izraz “Zdravo, svijete!”, no već iz njega možete vizualno doživjeti jezik i primijetiti osnovnu strukturu programa.

VB.NET

```
Imports System
Public Module ZdravoSvijete
    Sub Main()
        Console.WriteLine ("Zdravo, svijete!")
    End Sub
End Module
```

I. DIO: .NET IZNUTRA

C#

```
using System;
class ZdravoSvijete
{
    static void Main()
    {
        Console.WriteLine ("Zdravo, svijete!");
    }
}
```



U primjerima što slijede nećemo ponavljati kostur kôda već samo dio koji obrađujemo. Vama na dušu stavljamo da, ukoliko kôd želite isprobati u praksi, ne zaboravite ga umetnuti u blok ovog primjera na kojem se nalazi linija koja počinje s `Console.WriteLine`.

Kompajliranje

Prepišite jedan od primjera u datoteku, snimite je pod imenom koje vam se sviđa u jednu od mapa na računalu. Preporuka je da kôd pisan u C#-u snimate s nastavkom ".cs" (primjerice, "zdravovsvijete.cs"), a datotekama s kôdom u VB.NET-u dajete nastavak ".vb".

Pronađite na disku mapu u kojoj se nalaze datoteke "csc.exe" i "vbc.exe". Radi se o mapi koja se nalazi unutar systemske mape "Windows" (odnosno "WinNT", ovisno o verziji operativnog sustava), zatim "Microsoft.NET", pa "Framework" te na kraju mapa istog imena kao i inačica .NET Frameworka koji imate instaliran. U našem slučaju je to verzija 1.1.4322, pa cijela putanja glasi:

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322
```

Zatim pokrenite komandnu liniju (engl. *Command Prompt*) i upišite sljedeću naredbu (ne zaboravite promijeniti putanju do mape, ukoliko se ona na vašem računalu razlikuje od naše:

```
path "C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322",%PATH%
```

Nakon toga trebate pomoću komandne linije doći do mape u koju ste smjestili datoteku s kôdom. Pretpostavljamo da osnovne naredbe DOS-a za snalaženje po mapama znate, no ipak evo nekoliko natuknica za čitatelje mlađe generacije.

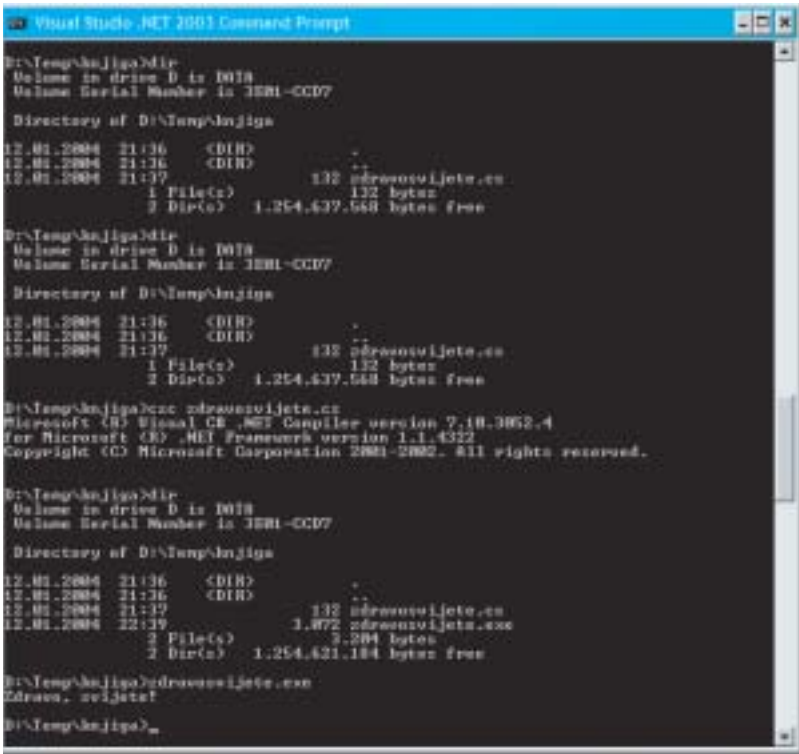
3. POGLAVLJE: PROGRAMSKI JEZICI

Ukoliko imate instaliran Visual Studio .NET, ne morate se mučiti s traženjem mape s kompajlerima i njezine registracije naredbom “path”. Dovoljno je u izborniku “Start” potražiti stavku “Visual Studio .NET 2003 Command Prompt”.



Prvo treba doći na disk na kojem se nalazi mapa, a zatim upisati i samu putanju do mape. Primjerice, ukoliko je putanja mape “D:\My Documents\Primjeri”, do nje ćete doći na sljedeći način:

```
d:
cd "\My Documents\Primjeri"
```



Slika 3-1:
Postupak kompajliranja datoteke “zdravosvijete.cs” u “zdravosvijete.exe” i pokretanje te male aplikacije

Nakon ovih srednjovjekovnih muka, na red dolazi uživancija – kompajliranje. Da biste kompajlirali datoteku “zdravosvijete.cs” u kojoj je kôd pisan u C#-u, upišite sljedeće:

```
csc zdravosvijete.cs
```

I. DIO: .NET IZNUTRA

Slična je procedura i ako ste se odlučili za kôd u VB.NET-u:

 vbc zdravosvijete.vb

U oba slučaja, nakon nekoliko trenutaka, u mapi će se pojaviti datoteka “zdravosvijete.exe”. Ona je rezultat kompajliranja datoteke s kôdom; da biste je izvršili, dovoljno ju je pokrenuti. Rezultat će biti linija s natpisom “Zdravo, svijete!”. Vaš prvi program radi!

Kad smo savladali kompajliranje aplikacija možemo se baciti na proučavanje sintakse odabranih jezika. Zapamtite da svaki od primjera što slijede možete isprobati na gore opisani način: snimite kôd u datoteku, kompajlirajte je i pokrenite dobivenu izvršnu datoteku. I, ne zaboravite na najzabavniji dio – nakon što se uvjerite da primjer radi (odnosno da ste ga dobro prepisali), probajte ga malo izmijeniti i proučavajte kako se onda ponaša.

Sintaksa

Kao što ljudski jezik ima svoj pravopis i gramatiku, tako i računalni jezici imaju pravila pisanja koje možemo ujediniti pod pojmom sintakse. Sintaksa je, jednostavno rečeno, način na koji morate pisati naredbe u nekom programskom jeziku da bi ih kompajler mogao razumjeti i pravilno protumačiti.

Svi programski jezici imaju slične mogućnosti, no različit način na koji se te mogućnosti koriste. To je posebno naglašeno u okruženju .NET-a gdje je isti kôd pisan u različitim jezicima sličniji nego ikada (zbog istih biblioteka klasa koje koriste), pa se razlike među njima svode na razlike u sintaksi.



Iako to sintaksa niti jednog jezika ne zahtijeva, preporučujemo vam da kôd pišete uredno i dosljedno – uvlačite redove, ne ostavljajte nepotrebne razmake i pišite komentare kako biste se kasnije u njemu lakše snalazili.

Kako bi poznavatelji C-olike sintakse lakše razumjeli VB-ovu i obrnuto, kroz primjere pisane u Visual Basicu .NET i C#-u prolaziti ćemo paralelno. Naravno, slobodni ste jedan od jezika u potpunosti ignorirati i usvajati saznanja samo o onome koji vam se više dopada, no voljni smo se kladiti da ćete se kad-tad vratiti i pročitati preskočeno.

Velika i mala slova

Jedna od osnovnih razlika između jezika VB.NET i C# jest tretiranje malih i velikih slova. Tradicionalno, VB.NET ne pravi razliku, pa je sasvim svejedno na koji ćete od sljedećih načina pisati kôd:

VB.NET

```
Console.WriteLine ("Zdravo, svijete!")
Console.WRITELINE ("Zdravo, svijete!")
console.writeline ("Zdravo, svijete!")
CoNsOlE.WrItElIne ("Zdravo, svijete!")
```

Ipak, radi urednosti i preglednosti, preporučujemo vam da kôd pokušavate pisati što dosljednije i urednije, po mogućnosti prema pravilima koja vrijede u drugim jezicima jer će vam to olakšati čitanje i prevođenje.

C#

U C# jeziku od krucijalne je važnosti koristite li mala ili velika slova. Za razliku od VB-a .NET, kompajler će prepoznati samo prvi redak, dok će za sve ostale prijaviti grešku:

```
Console.WriteLine ("Zdravo, svijete!");
Console.WRITELINE ("Zdravo, svijete!");
console.writeline ("Zdravo, svijete!");
CoNsOlE.WrItElIne ("Zdravo, svijete!");
```

Kraj naredbe

Prošla dva primjera razlikuju se samo u znaku točka-zarez. Kao što vidimo, svaka naredba u jeziku C# mora završavati tim znakom, dok je u VB-u .NET dovoljno prijeći u novi red. Mogli bismo reći da C# ignorira prelazak u novi red i da on služi isključivo ljudima kako bi kôd bio pregledniji.

Evo primjera kako napisati tri naredbe u dva reda. Kod primjera pisanog u VB-u .NET uočite da, ukoliko u istom redu želimo staviti dvije naredbe, moramo koristiti znak dvotočke.

VB.NET

```
Console.WriteLine ("Prva linija!")
Console.WriteLine ("Druga linija!") : Console.WriteLine ("Treća linija!")
```

C#

```
Console.WriteLine ("Prva linija!");
Console.WriteLine ("Druga linija!"); Console.WriteLine ("Treća linija!");
```

I. DIO: .NET IZNUTRA

Komentari

Iako bismo ih teško mogli nazvati najvažnijima, zgodno je da odmah na početku naučimo kako se u kôd upisuju komentari jer ćemo ih u primjerima što slijede i sami koristiti. Radi se o izrazima koje kompajler ignorira, a služe isključivo programerima kako bi se lakše snalazili u svom i tuđem kodu.

VB.NET

U VB-u .NET komentare označavamo znakom apostrofa ili naredbom “rem” (skraćeno od *remark*). Jednom korištena oznaka “komentira” sve napisano do kraja reda. Ova dva retka su sinonimi:

```
' Ovo je komentar koji kompajler ignorira  
Rem Ovo je komentar koji kompajler ignorira
```

Oznaka komentara ne mora biti korištena na početku retka. Dozvoljeno je da u prvom dijelu bude neka naredba, a nakon nje komentar:

```
Console.WriteLine ("Testiranje komentara!") ' Komentar
```

Iako je logično, mnogi se ne sjete da su komentari zapravo zgodan način da neke naredbe privremeno, iz bilo kojeg razloga, ne budu izvršene. Primjerice:

```
Console.WriteLine ("Testiranje komentara!")  
' Console.WriteLine ("Ova linija neće biti izvršena!")
```

C#

Za C# vrijede sve varijacije kao i kod VB-a .NET. Razlika je u tome što se umjesto znaka apostrofa koriste dvije kose crte:

```
// Ovo je komentar koji kompajler ignorira
```

Osim toga, C# pruža mogućnost komentiranja više linija. U VB-u .NET svaka bi linija trebala na početku imati poseban znak apostrofa, dok u C#-u to izgleda ovako:

```
/*  
Ovo je komentar  
koji se proteže  
kroz nekoliko linija  
*/
```

Varijable i njihovo deklariranje

Varijable su svojevrsni spremnici u koje spremate određene vrijednosti. Kada vam ta vrijednost kasnije u kodu zatreba, jednostavno upišete ime varijable i dobit ćete jednak efekt kao da ste upisali njenu vrijednost. Evo jednostavnog primjera u kojem prvi i drugi red imaju isti efekt:

VB.NET

```
Console.WriteLine (5)  
broj = 5 : Console.WriteLine (broj)
```

C#

```
Console.WriteLine (5);  
broj = 5; Console.WriteLine (broj);
```

Međutim, da biste neku varijablu mogli koristiti, morate je deklarirati. Postupak deklaracije nije ništa drugo nego način da kažete kompajleru da ćete koristiti neku varijablu te da joj odredite koji tip vrijednosti će moći poprimiti. O samim varijablama i tipovima više ćete saznati nešto kasnije, u petom poglavlju, a ovdje nam je zadatak naučiti sintaksu za deklariranje varijabli. Stoga ćemo u primjerima koristiti najpoznatije tipove – *string* (znakovni niz), *integer* (cijeli broj) i *object* (o tome nešto kasnije – spominjemo ga radi specifične sintakse).

Varijable možete imenovati gotovo proizvoljno. Kažemo “gotovo” jer postoje određena pravila kojih se treba držati. Najvažnije je da ime varijable mora počinjati slovom, ne smije sadržavati razmake i razne druge specijalne znakove (među koje ubrajamo i “naša slova”) te ne smije biti istog imena kao ključne riječi, naredbe i slično. Treba imati na umu da C# pazi i na način na koji ste napisali ime varijable, tako da varijable “broj” i “Broj” neće biti iste. VB.NET ne pravi razliku među različito napisanim imenima varijabli.

Varijable možete imenovati i samo jednim slovom (kao što mi najčešće radimo u primjerima), no u većim programima to neće biti praktično (ako ništa drugo, neće biti dovoljno slova). Zato vam preporučujemo da varijable pišete u obliku riječi, tako da vas njihovo ime automatski asocira na vrijednost koju sadrži. Evo nekoliko zgodnih primjera:

```
ImeDatoteke  
Ime_i_Prezime  
UkupnaVrijednostNarudzbe  
DatumZaposlenja
```

I. DIO: .NET IZNUTRA

VB.NET

Glavna razlika između naša dva jezika na području varijabli jest što VB.NET podržava tzv. implicitno deklariranje varijabli odnosno, razumljivijim rječnikom, moguće je isključiti obavezno deklariranje varijabli prije korištenja (što se, kad se već razbacujemo pojmovima, zove eksplicitno deklariranje). Iako vam to nikako ne bismo preporučili, to možete učiniti tako da na početku datoteke s kôdom napišete:

```
Option Explicit Off
```

Nadajući se da to ipak nećete učiniti (i tako si na prvi pogled olakšati, no dugoročno zakomplicirati programerski život), evo kako se deklariraju varijable u VB-u .NET:

```
Dim n As Integer
Dim s As String
Dim s1, s2 As String
Dim objekt1      ' ukoliko se ne navede, podrazumijeva se tip Object
Dim objekt2 As New Object()
```

C#

U C#-u deklariranje istih tih varijabli izgledalo bi ovako:

```
int n;
String s;
String s1, s2;
Object objekt1;
Object objekt2 = new Object();
```

Prilikom deklariranja moguće je varijablama odmah dodijeliti neke početne vrijednosti. Naravno, ništa vas ne sprečava da te početne vrijednosti dodijelite nekoliko linija kasnije, no na ovaj način svoj kôd činite preglednijim, a dobivate i na brzini. Dakako, ponekad vam istovremeno deklariranje i pridruživanje vrijednosti neće biti pogodno, no ako vam zatreba, evo kako ga postići:

VB.NET

```
Dim s As String = "Zdravo, svijete!"
Dim n As Integer = 1
```

C#

```
string s = "Zdravo, svijete!";
int n = 1;
```

Odluke

Uzmimo za primjer da želimo napisati programčić koji će, ovisno o vrijednosti neke varijable, ispisati na ekranu neki tekst. Za to nam treba provjera vrijednosti te varijable, što radimo naredbom *if*. Ta je naredba prisutna u svim programskim jezicima, pa ako imate imalo programerskog iskustva znate čemu ona služi. (Ako slučajno nemate, shvatit ćete iz primjera.)

U primjerima što slijede prvo ćemo deklarirati varijablu, dodijeliti joj vrijednost te nakon toga, ako je vrijednost veća od 100, ispisati adekvatan tekst. (Uočite da je uvjet u C# **uvijek** u zagradama, dok uvjet u VB-u .NET može, ali ne mora biti.)

VB.NET

```
Dim n As Integer  
n = 123  
If n > 100 Then Console.WriteLine("Broj je veći od sto!")
```

C#

```
int n;  
n = 123;  
if (n > 100) Console.WriteLine("Broj je veći od sto!");
```

U praksi su rijetki slučajevi u kojima postoji samo jedna naredba koju želite izvršiti u slučaju zadovoljenja uvjeta. Najčešće je potrebno napraviti više stvari, poput ispisivanja dvaju redaka teksta kao u našem sljedećem primjeru:

VB.NET

```
' izostavljena je deklaracija varijabli  
If n > 100 Then  
    Console.WriteLine("Broj je veći od sto!")  
    Console.WriteLine("Hvala što ste koristili program.")  
End If
```

C#

```
// izostavljena je deklaracija varijabli  
if (n > 100)
```

I. DIO: .NET IZNUTRA

```
{  
    Console.WriteLine("Broj je veći od sto!");  
    Console.WriteLine("Hvala što ste koristili program.");  
}
```

U gornjim će primjerima ako uvjet nije ispunjen program ignorirati naredbe u *if*-bloku. Međutim, često postoji potreba da i u tom slučaju izvršite neku naredbu; primjerice, ispišete da je broj manji od sto. Evo kako:

VB.NET

```
If n > 100 Then  
    Console.WriteLine("Broj je veći od sto!")  
Else  
    Console.WriteLine("Broj je manji od sto!")  
End If
```

C#

```
if (n > 100)  
{  
    Console.WriteLine("Broj je veći od sto!");  
}  
else  
{  
    Console.WriteLine("Broj je manji od sto!");  
}
```

Naravno, niti ovo ne mora zadovoljiti sve vaše potrebe. U sljedećim primjerima imat ćemo tri različita natpisa: jedan za brojeve do sto, drugi za one između sto i dvjesto, te treći za sve ostale.

VB.NET

```
If n > 200 Then  
    Console.WriteLine("Broj je veći od dvjesto!")  
ElseIf n > 100 Then  
    Console.WriteLine("Broj je između sto i dvjesto!")  
Else  
    Console.WriteLine("Broj je manji od sto!")  
End If
```


C#

```

if (n > 200)
{
    Console.WriteLine("Broj je veći od dvjesto!");
}
else if (n > 100)
{
    Console.WriteLine("Broj je između sto i dvjesto!");
}
else
{
    Console.WriteLine("Broj je manji od sto!");
}

```

Primjećujete i sami – kada bi postojao velik broj *e/se*-blokova, kôd bi postao nezgrapan i nepregledan. Zato u pomoć pozivamo naredbu *case*. Slijedi kôd iste funkcionalnosti kao i prošli primjer:

VB.NET

```

Select Case n
    Case Is > 200
        Console.WriteLine("Broj je veći od dvjesto!")
    Case 100 To 200
        Console.WriteLine("Broj je između sto i dvjesto!")
    Case Else
        Console.WriteLine("Broj je manji od sto!")
End Select

```

C#

Očekujete primjer u C#? Nažalost, ovakav tip uvjeta u tom je jeziku nemoguć. Moguće je samo navođenje konkretnih vrijednosti varijable, no ne i raspona. Dakle, pomoću naredbe *case* u C#-u možemo napisati samo provjeru je li varijabla jednaka određenoj vrijednosti, dok ćemo za rasponne morati koristiti naredbu *if*.

```

switch (n) {
    case 200 :
        Console.WriteLine("Broj je jednak dvjesto!");
        break;
}

```

I. DIO: .NET IZNUTRA

```

case 100 :
    Console.WriteLine("Broj je jednak sto!");
    break;
case 0 :
    Console.WriteLine("Broj je jednak nuli!");
    break;
default :
    // ukoliko niti jedan uvjet nije zadovoljen, izvršit će se blok "default"
    Console.WriteLine("Radi se o nekom drugom broju!");
    break;
}

```

Unatoč tom prilično velikom ograničenju, C# nudi neke mogućnosti koje VB.NET ne poznaje. Pogadate i sami – ključ je u naredbi *break*, koju može zamijeniti izrazima poput *goto case 0*. Na pišemo li nešto poput ovoga, izvršit će se i zadovoljeni blok i blok na koji skaćemo.

```

switch (n) {
    case 200 :
        Console.WriteLine("Broj je jednak dvjesto!");
        goto case 0; // skok u blok "case 0"
    case 100 :
        Console.WriteLine("Broj je jednak sto!");
        goto default; // skok u blok "default"
    case 0 :
        Console.WriteLine("Broj je jednak nuli!");
        break;
    default :
        Console.WriteLine("Radi se o nekom drugom broju!");
        break;
}

```

Uvjeti

U priči o odlukama koristili smo uvjete na temelju kojih bi naredbe *if* i *case* donosile odluku o izvršavanju jednog od blokova kôda. Mi smo u primjerima koristili najjednostavnije (“veće od”, “manje od”) koje možemo naći i u udžbenicima iz matematike u osnovnoj školi.

Vrijeme je da upoznamo i naprednije oblike uvjeta. Svima će biti zajedničko da daju logičku vrijednost “istina” ili “laž”. Pogledajmo sljedeći uvjet:

```
n > 100
```

3. POGLAVLJE: PROGRAMSKI JEZICI

Ukoliko je vrijednost varijable *n* veća od 100, rezultat će biti istina (engl. *true*), a ako je vrijednost varijable manja (ili jednaka) broju 100, dobit ćemo logičku vrijednost neistine (engl. *false*). Usporedni prikaz osnovnih operatora uspoređivanja možete vidjeti u tablici 3-1.

VB.NET	C#	Opis
=	==	jednako
<	<	manje od
<=	<=	manje od ili jednako
>=	>=	veće ili jednako
>	>	veće
<>	!=	nije jednako (različito)

Tablica 3-1:
Usporedba osnovnih operatora uspoređivanja

Složene uvjete možemo slagati i kombiniranjem više jednostavnih. Pokušajte odgonetnuti što znače sljedeći uvjeti. (Imajte na umu da, za razliku od klasične matematike, uvijek koristimo zaobljene zagrade, u koliko god nivoa postojale).

VB.NET

```
(n > 100) AND (n < 200)
(n < 100) OR (n > 200)
(n > 100) OR ((n > 0) AND (n < 50))
NOT (n > 100)
```

C#

```
((n > 100) && (n < 200))
((n < 100) || (n > 200))
((n > 100) || ((n > 0) && (n < 50)))
(!(n > 100))
```

(Mala digresija: sjetite se kada smo pričali o VB-u .NET i njegovoj glavnoj prednosti – čitljivosti napisanog kôda. Upravo u gornjim primjerima ona dolazi do izražaja!)

Nakon što smo odgonetnuli ove jednostavnije primjere, možemo se upoznati i s nekim naprednijim logičkim operatorima (vidi tablicu 3-2).

Tablica 3-2:
Usporedba osnovnih logičkih operatora

VB.NET	C#	Rezultat je istinit ako...
AND	&&	...su obje vrijednosti istinite
OR		...je barem jedna vrijednost istinita
XOR	^	...je samo jedna vrijednost istinita
NOT	!	...negira vrijednost izraza koji slijedi

Kao što vidimo u primjerima, uvjeti u C#-u uvijek moraju biti omeđeni zagradama, dok u VB-u .NET to nije slučaj. Zagradama određujemo, baš kao u pučkoškolskoj matematici, i redoslijed računanja vrijednosti. Uočite kako sljedeća dva retka ne daju identične rezultate (a usput i kako drugi redak zapravo nema smisla, no ostavimo ga kako bismo naglasili ulogu zagrada):

VB.NET

```
(n > 100) OR ((n > 0) AND (n < 50))  
((n > 100) OR (n > 0)) AND (n < 50)
```

C#

```
((n > 100) || ((n > 0) && (n < 50)))  
(((n > 100) || (n > 0)) && (n < 50))
```

Spomenimo još i da rezultat, logičku vrijednost nekog uvjeta možete spremiti u varijablu. Taj tip varijable naziva se *boolean*. Evo kako to izgleda u praksi:

VB.NET

```
Dim b1, b2 As Boolean  
b1 = True  
b2 = n > 100
```

C#

```
bool b1, b2;  
b1 = true;  
b2 = (n > 100);
```

Sve napisano u ovom odlomku tek je vrh ledene sante. Uvjeti su mnogo kompleksnijih od ovih nekoliko operacija, no ograničen prostor ne dozvoljava nam dublji ulazak u problematiku. Neke druge poloviti ćete u hodu kroz primjere koji slijede, a utješno je što ćete u većini slučajeva koristiti upravo ove, najjednostavnije oblike.

Računske operacije

Iako je to računalu trivijalan zadatak, možete ga zadužiti za najjednostavnije računske operacije kao što su zbrajanje, oduzimanje, množenje i dijeljenje. U tablici 3-3 možete vidjeti kako pojedine računske operacije izgledaju u pojedinom jeziku

VB.NET	C#	Opis
10 + 50	10 + 50	zbrajanje, rezultat je 40
200 – 100	200 – 100	oduzimanje, rezultat je 100
20 * 10	20 * 10	množenje, rezultat je 200
432 / 10	432 / 10	dijeljenje, rezultat je 43,2
432 \ 10	(nije direktno dostupno)	dijeljenje s cjelobrojnim rezultatom, rezultat je 43
432 mod 10	432 % 10	ostatak cjelobrojnog dijeljenja, rezultat je 2

Tablica 3-3:
Usporedba osnovnih računskih operacija

S dobivenim rezultatima možete raditi svašta, uključujući i dodjeljivanje varijablama. Primjerice, kod pridruživanja vrijednosti nekoj varijabli, tu vrijednost ne morate eksplicitno napisati; ona može biti dana u obliku neke računske operacije. Od jednostavnih operacija slobodno možete slagati složene, sa zagradama ili bez njih. Osim brojeva, u izrazima možete koristiti i varijable. Evo nekoliko primjera:

VB.NET

```
n = 10 + 50      ' n = 40
m = 200 – 100     ' m = 100
n = m / 10        ' n = 100 / 10 = 10
n = n + 10        ' n = 10 + 10 = 20
m = m / (n - 10)  ' m = 100 / (20 - 10) = 100 / 10 = 10
```

I. DIO: .NET IZNUTRA

C#

```
n = 10 + 50;      // n = 40
m = 200 - 100;    // m = 100
n = m / 10;       // n = 100 / 10 = 10
n = n + 10;       // n = 10 + 10 = 20
m = m / (n - 10); // m = 100 / (20 - 10) = 100 / 10 = 10
```



U računskoj operaciji dijeljenja vrlo je važno kojeg je tipa varijabla kojoj pridružujemo vrijednost. Naime, dok cjelobrojnoj varijabli nećete niti moći eksplicitno dodijeliti decimalni broj, ako se dogodi da rezultat vašeg dijeljenja bude decimalni broj, a varijabla kojoj ga želite pridružiti cjelobrojnog tipa, rezultat će automatski biti pretvoren u cijeli broj.

Neke od računskih operacija često se koriste. Najčešći slučaj je povećanje vrijednosti varijable za jednu jedinicu (recimo: `n = n + 1`). Kako bi programeri bili produktivniji, sintakse jezika dozvoljavaju skraćeno pisanje nekih češće korištenih izraza (tablica 3-4).

Tablica 3-4:
Skraćeni oblici računskih operacija; umjesto broja 5 može stajati bilo koji broj ili varijabla

VB.NET	C#	Dugi oblik
<code>n += 5</code>	<code>n += 5</code>	<code>n = n + 5</code>
<code>n -= 5</code>	<code>n -= 5</code>	<code>n = n - 5</code>
<code>n *= 5</code>	<code>n *= 5</code>	<code>n = n * 5</code>
<code>n /= 5</code>	<code>n /= 5</code>	<code>n = n / 5</code>
(nema posebnog izraza)	<code>n ++</code>	<code>n = n + 1</code>
(nema posebnog izraza)	<code>n --</code>	<code>n = n - 1</code>

C#

Jezik C# časti nas još jednom vrlo praktičnom poslasticom zvanom uvjetno pridruživanje. Da ne kompliciramo teorijom, objasniti ćemo vam primjerom. Sljedeća dva retka daju identičan rezultat (varijabla `n` cjelobrojnog je tipa, dok je varijabla `s` *string*):

3. POGLAVLJE: PROGRAMSKI JEZICI

```
if (n > 0) s = "n je pozitivan"; else s = "n je negativan";
s = n > 0 ? "n je pozitivan" : "n je negativan";
```

Ukoliko je uvjet (u našem slučaju $n > 0$) zadovoljen, varijabli *s* bit će dodijeljena vrijednost “n je pozitivan”. Ako taj uvjet nije zadovoljen, *s* će biti jednak izrazu “n je negativan”.

Računsku operaciju zbrajanja, osim nad brojevima, možete vršiti i nad tekстом (*stringovima*). Radi se o najobičnijem spajanju dvaju (ili više) *stringova* u jedan. To se “zbrajanje *stringova*” zove konkatencija, a u praksi izgleda ovako:

```
str = "Mali " + "zeko";    // str = "Mali zeko"
str += " i prijatelji";    // str = " Mali zeko i prijatelji"
```

Isto je moguće i VB.NET-u – samo izbacite točka-zarez i zamijenite oznaku za komentar. U tom se jeziku, osim operatora “+” u ovu namjenu koristi i operator “&”.



Petlje

Petlje služe za ponavljanje nekog dijela kôda određeni broj puta ili dok se neki uvjet ne ispuni. U svakodnevnom životu primjera za to ima koliko hoćete. Ako u dućanu znate točan broj komada nekog proizvoda koji želite kupiti, istu ćete radnju uzimanja i spremanja u košaricu ponoviti točno određen broj puta. Kada, pak, dođete na blagajnu, iz novčanika ćete vaditi novac sve dok ne izvadite iznos veći ili jednak potrošenom.

Za početak ćemo se zabaviti prvim slučajem – ponavljanjem neke radnje određen broj puta. Za to će nam poslužiti petlja *for*. U sljedećem primjeru na ekran ćemo ispisati brojeve od 1 do 100:

VB.NET

Sintaksa u VB-u .NET je jednostavna – nakon naredbe *for* odredite varijablu koju želite “vrtiti” u petlji te nakon znaka jednakosti odredite raspon kojim će se ona kretati. Prilikom svakog izvršavanja bloka naredbi između riječi *for* i *next*, odabrana varijabla će imati drugu vrijednost. Prvi put jedan, zatim dva, pa tri... i tako sve do sto.

```
Dim n As Integer ' ne zaboravite prije deklarirati varijablu!
For n = 1 To 100
    Console.WriteLine(n)
Next
```

I. DIO: .NET IZNUTRA

C#

Ova sintaksa na prvi je pogled znatno složenija, no zato pomoću nje, kao što ćemo vidjeti kasnije, možemo postići puno više varijacija. Primijetite da se nakon naredbe *for* u zagradi navode tri parametra odijeljena točkom-zarezom. U prvom određujemo koja će se varijabla koristiti kao brojač i njenu početnu vrijednost. U drugom se parametru stavlja uvjet koji treba biti ispunjen da bi se blok naredbi izvršavao. Treći parametar služi najčešće za uvećavanje brojača (što se u VB-u .NET ne mora definirati, već je *defaultna* vrijednost). U našem ga slučaju uvećavamo za jedan, pa pišemo izraz tome namijenjen (vidi dio teksta o računskim operacijama).

```
for (int n = 1; n <= 100; n++) // varijablu možete deklarirati i ovdje
{
    Console.WriteLine(n);
}
```

For-petlja nije ograničena isključivo na povećavanje vrijednosti za jedan. U primjerima što slijede pokazat ćemo kako napisati petlju kojoj će brojač povećavati vrijednost za dva, u kojoj će vrijednost brojača padati i, konačno, gdje će se brojač povećavati eksponencionalno.

VB.NET

```
For n = 1 To 100 Step 2 ' vrijednosti 1, 3, 5, 7, 9 .. 97, 99
    ...
Next

For n = 100 To 1 Step -1 ' vrijednosti 100, 99, 98 .. 3, 2, 1
    ...
Next

' zahvaljujući dosjetkama u bloku, vrijednosti će biti 1, 2, 4, 8, 16, 32, 64
For n = 1 To 100
    ...
    n = n * 2 ' ili n *= 2
    n = n - 1 ' ili n -= 1
Next
```

C#

```
for (int n = 1; n <= 100; n+=2) // vrijednosti 1, 3, 5, 7, 9 .. 97, 99
{ ... }
```


3. POGLAVLJE: PROGRAMSKI JEZICI

```
for (int n = 100; n >= 1; n--) // vrijednosti 100, 99, 98 .. 3, 2, 1
{ ... }

for (int n = 1; n <= 100; n*=2) // vrijednosti 1, 2, 4, 8, 16, 32, 64
{ ... }
```

Osim *for*-petlje, postoji i oblik petlje nazvan *while*. Osnovna razlika među ovim petljama jest što potonja nema brojač – promjenu uvjeta koji odlučuje o ponavljanju petlje morat ćete sami kontrolirati u bloku unutar petlje. Ovaj tip petlje najčešće se koristi kada unaprijed ne znamo koliko puta će se neka petlja trebati izvršiti.

Ipak, i petlji *while* možemo dodati brojač te tako dobiti isti efekt kao i s petljom *for*. Evo primjera:

VB.NET

```
Dim n As Integer
n = 1
Do While n <= 100
    Console.WriteLine(n)
    n += 1
Loop
```

C#

```
int n = 1;
while (n <= 100) {
    Console.WriteLine(n);
    n += 1;
}
```

Primjećujete i sami da je petlja *for* u ovakvim slučajevima jednostavnija za korištenje. Prava vrijednost petlje *while* očitovat će se u sljedećem primjeru. U njega uvodimo jednu novu naredbu – *Console.ReadLine*. Dok kod naredbe *Console.WriteLine* određeni izraz ispisujemo na ekran, nova će naredba tražiti od korisnika programa da sam nešto upiše. Sljedeća petlja izvršavat će se sve dok korisnik ne upiše riječ “izlaz”.

VB.NET

```
Dim unos As String = ""
Do While unos <> "izlaz"
```

I. DIO: .NET IZNUTRA

```
Console.WriteLine("Unesite tajnu riječ:")
unos = Console.ReadLine()
Loop
```

C#

```
string unos = "";
while (unos != "izlaz")
{
    Console.WriteLine("Unesite tajnu riječ:");
    unos = Console.ReadLine();
}
```

Metode ili funkcije

Uхватite li se prilikom programiranja da neki komad kôda često ponavljate, mogli biste si uštedjeti puno vremena kreiranjem funkcije. Nakon što određeni komad kôda smjestite u funkciju, bit će dovoljno pozvati njeno ime i taj će kôd biti izvršen. U sljedećem primjeru funkcija se brine o ispisivanju izraza “Pozdrav svima!”.

VB.NET

```
Imports System
Public Module ZdravoSvijete

    Sub IspisiPozdrav()
        Console.WriteLine("Pozdrav svima!")
    End Sub

    Sub Main()
        IspisiPozdrav()
        IspisiPozdrav()
        IspisiPozdrav()
    End Sub

End Module
```

C#

```
using System;
class ZdravoSvijete
```

3. POGLAVLJE: PROGRAMSKI JEZICI

```
{
    static void IspisiPozdrav()
    {
        Console.WriteLine("Pozdrav svima!");
    }

    static void Main()
    {
        IspisiPozdrav();
        IspisiPozdrav();
        IspisiPozdrav();
    }
}
```

Sigurno ste primijetili da prvi put u nekom primjeru ponovo koristimo kostur iz primjera “Zdravo, svijete!”. Kao što smo tada napomenuli, sav kôd treba ići u blok Main(), za koji sada znamo da je zapravo funkcija, osnovna funkcija koja se izvršava po pokretanju programa.

Kostur pokazujemo zato što deklariranje funkcija ne može biti unutar neke druge funkcije, već isključivo izvan. Ono što možemo (i moramo) raditi unutar neke druge funkcije jest pozivanje, što mi u primjeru radimo unutar funkcije Main(). U primjerima što slijede ponovno ćemo, radi uštede prostora, izostavljati kostur kôda, a vi imajte na umu da se deklariranje funkcija radi izvan, a pozivanje funkcija unutar glavne funkcije.

Pozivanje funkcija moguće je vršiti i iz drugih funkcija, a ne samo osnovne. Takvih ćemo primjera imati u kasnijim poglavljima. Također, pojedinu je funkciju moguće pozvati i iz nje same. Te se pak funkcije nazivaju rekurzivnima i u radu s njima treba biti vrlo oprezan. I njih ćemo susresti nešto kasnije u knjizi.



Gornji primjeri u praksi nemaju nekog smisla – više bismo profitirali da smo triput zaredom napisali naredbu Console.WriteLine s pripadajućim parametrom. Međutim, zamislite da pozivi te funkcije nisu jedan ispod drugoga, već razbacani po programu, a vama dođe zadatak da tekst koji biva ispisan izmijenite u “Dobar dan!”. Puno je jednostavnije izmijeniti stvar na jednom mjestu, nego tražiti gdje se sve spomenuta fraza koristi i mijenjati je na svim tim mjestima.

Još bolji primjer korisnosti funkcija je slučaj u kojem kôd unutar funkcije nije, kao u našem primjeru, samo jedna linija, nego neki složeniji skup naredbi. U takvim ćete slučajevima uštedjeti i na samom tipkanju, a i kôd koji producirate bit će puno pregledniji.

I. DIO: .NET IZNUTRA

Funkcije možemo pozivati i s parametrima. U sljedećem primjeru ćemo funkciji kao parametre poslati dva broja, a ona će imati zadatak oduzeti veći broj od manjeg i rezultat ispisati na ekranu.

VB.NET

```
Sub Oduzmi(broj1 As Integer, broj2 As Integer)
    If broj1 > broj2 Then
        Console.WriteLine(broj1 - broj2)
    Else
        Console.WriteLine(broj2 - broj1)
    End If
End Sub
```

C#

```
static void Oduzmi(int broj1, int broj2)
{
    if (broj1 > broj2)
    {
        Console.WriteLine(broj1 - broj2);
    }
    else
    {
        Console.WriteLine(broj2 - broj1);
    }
}
```

Pozivanje ove funkcije izgledat će ovako:

VB.NET

```
Oduzmi(10, 20) ' rezultat 10
Oduzmi(5, 11) ' rezultat 6
```

C#

```
Oduzmi(123, 321); // rezultat 198
Oduzmi(100, 100); // rezultat 0
```

3. POGLAVLJE: PROGRAMSKI JEZICI

Funkcije možemo nazivati i njihovim drugim imenom – metode – a funkcije koje smo dosad upoznali možemo zvati i procedurama. Za procedure je specifično da ne vraćaju nikakvu povratnu vrijednost, dok one koje ne pripadaju pod značenje tog pojma, vraćaju vrijednosti. Te vraćene vrijednosti možemo spremiti u varijablu, uvrstiti u računski izraz ili pak iskoristiti kao parametar za neku drugu metodu.

U sljedećem primjeru modificirat ćemo prethodnu funkciju tako da rezultat oduzimanja ne ispisuje na ekran, nego vraća u obliku svoje povratne vrijednosti.

VB.NET

```
Function Oduzmi(ByVal broj1 As Integer, ByVal broj2 As Integer) As Integer
    If broj1 > broj2 Then
        Return broj1 - broj2
    Else
        Return broj2 - broj1
    End If
End Function
```

C#

```
static int oduzmi(int broj1, int broj2)
{
    if (broj1 > broj2)
    {
        return broj1 - broj2;
    }
    else
    {
        return broj2 - broj1;
    }
}
```

Funkcija koja vraća vrijednost ima dva obilježja koja ne nalazimo u ostalim funkcijama. U redu deklariranja funkcije dodan je parametar koji označava tip vrijednosti koji će funkcija vraćati. Sintaksa za to vrlo je slična deklaraciji varijabli – u VB-u .NET koristi se riječ “As”, nakon koje slijedi tip vrijednosti (u našem slučaju “Integer”), dok se u C#-u tip vrijednosti stavlja na mjesto riječi “void” (hrv. praznina; u našem slučaju “int”).

Drugo obilježje je definiranje povratne vrijednosti. Za taj posao koristimo riječ “return”, nakon koje navodimo vrijednost koju će funkcija vratiti pozivatelju (u našem slučaju rezultat izraza “broj1 – broj2” odnosno “broj2 – broj1”).

I. DIO: .NET IZNUTRA

Uz ove dvije, Visual Basic pravi još jednu razliku među funkcijama koje vraćaju i ne vraćaju vrijednosti. One koje ne vraćaju, procedure, započinju naredbom “Sub”, a završavaju s “End Sub”. Ove druge pak na početku imaju “Function”, a na kraju “End Function”.

I ovakvu funkciju možete pozvati na isti način kao i prošle, samo što će onda njezina povratna vrijednost biti izgubljena. Kako to u većini slučajeva nema smisla, u sljedećem ćemo primjeru pokazati neke od načina na koje možete tu vrijednost “uhvatiti” i iskoristiti.

VB.NET

```
Console.WriteLine(Oduzmi(90, 50)) ' vraćena vrijednost je ispisana
Dim n As Integer = Oduzmi(400, 500) ' vraćena vrijednost je pridružena varijabli n
n = Oduzmi(Oduzmi(100, 200), Oduzmi(500, 600))
```

C#

```
Console.WriteLine(Oduzmi(90, 50)); ' vraćena vrijednost je ispisana
int n = Oduzmi(400, 500); ' vraćena vrijednost je pridružena varijabli n
n = Oduzmi(Oduzmi(100, 200), Oduzmi(500, 600));
```

Klase

Sada kada razumijemo pojam, možemo reći da su naredbe koje smo koristili u primjerima (WriteLine i ReadLine) zapravo metode. Naravno, njih nismo morali sami definirati jer postoje u osnovnoj biblioteci klase, i to u klasi Console. Zato prilikom pozivanja navodimo *namespace* klase u kojoj se metoda nalazi, kako bi kompajler znao gdje da je nađe.

Čak štoviše, da mu eksplicitno ne kažemo, kompajler ne bi znao niti gdje da traži određenu klasu. Zato na početku kôda pišemo “Imports System” odnosno “using System”, ovisno o jeziku. To je direktiva kompajleru da sve vanjske klase na koje se pozivamo potraži u baznoj klasi koja ima *namespace* “System”.

I sami primjeri koje smo koristili sami po sebi zapravo predstavljaju klasu. Naravno, da bi oni funkcionirali na način na koji to rade bazne klase, potrebne su određene adaptacije, posebno u primjeru pisanom u VB-u .NET, koji poznaje drugačiju sintaksu za nešto zvano standardni modul koji smo u primjerima koristili. No o svemu tome više u poglavlju posvećenom objektno-orijentiranom programiranju...

Zbogom, vizualni bejziče...

Naše druženje s Visual Basicom .NET ovdje prestaje iako se nesumnjivo o Visual Basicu .NET još štošta može reći. Međutim, kao što smo već naglašavali, fokus knjige bit će na jeziku C#, pa će odsada svi primjeri biti isključivo u tom jeziku.

Ipak, vjerujemo da ćete se i dalje vraćati na ovo poglavlje. Ono je, naime, osim jednostavnijeg

svladavanja novog jezika osobama s programerskim iskustvom u Basicu, zamišljeno i kao vodič za razumijevanje primjera u VB-u .NET na koje možete naići. Dogodit će se, naime, da nećete moći naći rješenje za problem koji vas muči pisano u C#-u. Zahvaljujući usporednim primjerima u ovom poglavlju, moći ćete lakše odgonetnuti o čemu se radi i prevesti primjer kako bi se uklopio u vašu aplikaciju.

