



ŠOLSKI CENTER POSTOJNA  
SREDNJA ŠOLA  
GIMNAZIJA ILIRSKA BISTRICA

**SEMINARSKA NALOGA**

# **PRIROČNIK ZA PROGRAMSKI JEZIK C#**

Mentor: Vinko Zbačnik

Projektna skupina: Boštjan Hren, Tilen Berlak, 4.R

Postojna, Maj 2016



# KAZALO

## KAZALO SLIK

Slika 1 Start Page.....	9
Slika 2 Zavihek File.....	10
Slika 3 Zavihek Edit.....	11
Slika 4 Zavihek Project.....	12
Slika 5 Zavihek Build.....	13
Slika 6 Zavihek Debug.....	13
Slika 7 Zavihek Team.....	14
Slika 8 Zavihek Tools.....	14
Slika 9 Zavihek Test.....	14
Slika 10 Zavihek Analyze.....	15
Slika 11 Zavihek Window.....	15
Slika 12 Zavihek Help.....	16
Slika 13 Orodna vrstica.....	16
Slika 14 Ustvarjanje novega projekta.....	17
Slika 15 Izbira konzolne aplikacije.....	18
Slika 16 Poimenovanje projekta.....	19
Slika 17 Prazna datoteka konzolne aplikacije.....	19
Slika 18 Zagon programa.....	20
Slika 19 Zaustavitev, premor in ponoven zagon programa.....	21

Slika 20 Dodajanje razreda.....	53
Slika 21 Poimenovanje razreda.....	53
Slika 22 Delovni zavihki.....	55
Slika 23 Okenska aplikacija.....	59
Slika 24 Ustvarjanje okenske aplikacije.....	60
Slika 25 Okno.....	60
Slika 26 Toolbox.....	61
Slika 27 Properties.....	61
Slika 28 GroupBox in gumbi.....	63
Slika 29 Trije GroupBox-i.....	64
Slika 30 Dodajanje predmetov v ComboBox.....	67
Slika 31 Izбира lastnosti lika.....	67
Slika 32 ComboBox.....	67
Slika 33 Rezultati.....	68
Slika 34 PictureBox.....	68
Slika 35 Dodajanje slik.....	69
Slika 36 Uvažanje slik.....	69
Slika 37 Izбира slik.....	70
Slika 38 Uvožene slike.....	70
Slika 39 Končni izgled okenske aplikacije.....	72
Slika 40 Argumenti z vrednostjo »true«.....	75
Slika 41 Izбира lika.....	76
Tabela 1 Aktivnosti (priprave).....	6
Tabela 2 Aktivnosti (pisanje).....	6
Tabela 3 Ponazoritev podatkovne strukture Array.....	42

V najini seminarski nalogi sva predstavila programski jezik C# in Microsoftovo orodje Visual Studio. Za pisanje priročnika sva se odločila zaradi pomankanja dobrih virov znanja glede programiranja. Še posebno težko je najti dobre vodiče v slovenščini.

Namen je bil v enem dokumentu strniti vse znanje, ki bo lahko kogarkoli pripravilo na pisanje programov in učenje bolj naprednih funkcij programskega jezika. Odločila sva se da bova poleg navodil in pojasnil v dokument vključila tudi slike, ki prikazujejo postopek in zagotovijo, da pri branju ne pride do nesporazuma.

## **POVZETEK**

Začela sva s opisom programskega jezika C#, ki se uporablja za veliko različnih namenov. V dokumentu sva predstavila ustvarjanje konzolskih in okenskih aplikacij v Visual Studiu 2015, ki podpira C# 6.0 ter .NET Framework 4.6.

.NET je programirni model, ki ga poleg J#, C, C++ uporablja tudi programski jezik C#. Omogoča nam uporabo knjižnice z že napisano kodo. Eden od razlogov za predstavitev jezika C# je bilo objektno programiranje, ki ga ta jezik omogoča. To pomeni da v kodi ustvarjamo objekte in razrede, ki omogočajo lažjo organizacijo programov in lažje pisanje večjih projektov. V Visual Studiu se nov razred ustvari samodejno za vsak grafični element, ki ga dodamo okenskim aplikacijam. Preden sva začela z s samim programskim jezikom pa sva preučila in predstavila vse elemente Visual Studia. Orodje je zastoj, razvilo pa ga je podjetje Microsoft. Ker je večnamensko nam omogoča ogromno število funkcij od katerih nam ne pridejo v poštev vse. Naštela sva vse funkcije in natančno opisala tiste, ki so pogosto uporabljene ter tiste, ki sva jih med delom potrebovala midva. Najbolj pomembna so različna okna, ki nam nudijo informacije ali funkcije. Microsoftovo orodje nam omogoča pisanje, pregledovanje, zaganjanje in popravljanje kode. Glede grafične podobe programov pa omogoča dodajanje in spreminjanje elementov, ki jih vidimo v vseh Microsoftovih okenskih programih.

Učenje programskega jezika sva začela s preprostim programom, ki v konzolo izpiše besedilo »pozdravljen svet«. To je standarden začetniški program, ki se za predstavitev jezika uporablja v večini priročnikov. Nadaljevala sva s različnimi tipi spremenljivk, stavkov in zank ter podatkovne strukture. Preučila sva še delovanje metod in organizacijskih enot. Nato sva se lotila oblikovanja najinega grafičnega programa. Ustvarila sva okensko aplikacijo, ki je z danimi podatki izračunala obseg in ploščino šest različnih likov.

Maturitetna naloga je bila uspešna. V dokument sva vključila vse kar sva načrtovala in ga oblikovala tako, da se je iz njega enostavno naučiti uporabe Visual Studia in pisanja programov v C#.

Ključne besede: programski jezik C#, orodje Visual Studio, priročnik, vodič, slike, konzolna aplikacija, okenska aplikacija.

## **PROJEKTNA ZAMISEL**

Programiranje ima temelje v logiki in matematiki. Učenje programiranja pa poleg razumevanja teh, zahteva tudi znanje posameznih programskih jezikov ter orodij, ki jih za pisanje programov uporabljamo. Vsak jezik in orodje deluje na podobni podlagi in hkrati vsebuje svoje posebnosti. Učenci programiranja se pri pridobivanju znanja

pogosto zanašamo na video-vodiče, forume in spletne priročnike. Problem pri spletnih vsebinah je, da nas pogosto ne morejo naučiti vsega in zato je potrebno pregledati številna spletna mesta za polno znanje o neki temi. Alternativa so fizične knjige ali priročniki, ki so navadno bolj organizirani in imajo več znanja na enem mestu. Kljub temu pa niso popolni saj lahko z novimi različicami jezikov in orodij postanejo zastareli. V slovenščini so spletni in tradicionalni priročniki zelo redki, zato sva se odločila sama napisati priročnik za programski jezik C# in orodje Visual Studio.

C# sva izbrala, ker spada v družino C jezikov. C jeziki se uporabljajo v številne namene in v različnih industrijah. C# se uporablja za objektno programiranje in v Visual Studiu z njim ustvarjamo okenske aplikacije. Namen najine naloge je prikazati uporabo jezika C# s konzolnimi aplikacijami in to znanje povezati s oblikovanjem okenske aplikacije. Znanje v dokumentu bo kogarkoli pripravilo na razvijanje aplikacij za Windows naprave in poda dobro podlago za nadaljno učenje programskih jezikov družine C in drugih.

Sestava projektne skupine: Boštjan Hren ( projektni sodelavec ), Tilen Berlak ( projektni sodelavec )

Projekt se je začel 27. decembra 2015, predviden zaključek pa je 1. maj 2016. Razen morebitnih okvar računalnikov ni posebnih stroškov. Plačati bova morala le za elektriko, ki jo bodo porabili računalniki in papir ter črnilo, ki bo uporabljeno za tiskanje dokumentov.

Maturitetno nalogo lahko ogrozijo okvare računalnikov, programov in nestabilne verzije C# ali Visual Studia.

Ob koncu dela bo ustvarjen dokument, ki bo predstavil delo s pogramskim jezikom C# in orodjem Visual Studio. Imel bo poglavja, ki po logičnem vrstnem redu predstavijo programski jezik, Visual Studio in ostalo znanje, ki je potrebno za razumevanje le teh. Za lažje sledenje navodilom in pojasnilom v dokumentu bova vključila tudi slike, ki bodo prikazale postopek med delom.

## **NAČRTOVANJE PROJEKTA**

Aktivnosti, ki naju bodo pripeljale do končnega izdelka se delijo na priprave, ki vključujejo učenje in eksperimentacijo s kodo ter pisanje dokumenta.

Tabela 1 Aktivnosti (priprave)

<b>AKTIVNOSTI (PRIPRAVE)</b>	<b>IZVAJALEC</b>	<b>DATUM</b>	<b>PORABLJEN ČAS</b>
Iskanje virov	Boštjan, Tilen	30.12.2015	7 ur
Učenje teorije	Boštjan, Tilen	10.1.2016	10 ur
Učenje C# in uporabe Visual Studia	Boštjan, Tilen	24.1.2016	25 ur

Tabela 2 Aktivnosti (pisanje)

<b>AKTIVNOSTI (PISANJE)</b>	<b>IZVAJALEC</b>	<b>DATUM</b>	<b>PORABLJEN ČAS</b>
Predstavitev jezika in teorije	Boštjan, Tilen	30.1.2016	8 ur
Uporaba Visual Studia	Boštjan	7.2.2016	6 ur
Pozdravljen svet	Tilen	13.2.2016	3 ure
Spremenljivke	Tilen	21.2.2016	20 ur
If in else stavek	Tilen	27.2.2016	8 ur
Switch stavek	Tilen	28.2.2016	6 ur
Logični operatorji	Tilen	30.2.2016	4 ure
Zanke	Boštjan	30.2.2016	10 ur
Array, list	Boštjan	15.3.2016	8 ur
Metode	Boštjan	26.3.2016	5 ur
Razredi	Boštjan	10.4.2016	10 ur
Namespace	Boštjan, Tilen	15.4.2016	9 ur
Okenska aplikacija_1 del	Boštjan, Tilen	25.4.2016	6 ur
Okenska aplikacija_2 del	Tilen	25.4.2016	15 ur
Pisanje maturitetne naloge	Boštjan	5.5.2016	10 ur

## IZVEDBA PROJEKTA



## **KAJ JE C# IN ZAKAJ GA UPORABLJAMO**

C# (C Sharp) je eden izmed .NET programskih jezikov. Namenjen je objektnemu programiranju ter izdelavi vseh vrst aplikacij za operacijske sisteme Windows.

Ima veliko podobnosti s programskima jezikoma Java in C++. Zmogljiv je kot C in C++ saj je evolucija iz družine programskih jezikov C. Glede na objektno programiranje pa ga primerjamo s jezikoma Java ter Pascal (Delphi).

Številne knjižnice ter ogrodja nam omogočajo uporabo C Sharp-a za vrsto namenov. Najbolj pogosto se uporablja za izdelavo namiznih aplikacij in programov v ukaznem pozivu. Z njim razvijamo tudi aplikacije namenjene mobilnim telefonom z operacijskim sistemom Windows Phone. Uporabljen je pri pisanju programskih skript za omrežne serverje ter dodajanje funkcij spletnim stranem.

To so najbolj pogoste uporabe jezika C#, najdemo pa ga tudi pri razvijanju video iger ter drugih tehnologijah.

## **ZGODOVINA JEZIKA**

Jezik je razvila Microsoft ekipa, katere vodja je bil Anders Hejlsberg. C# se je pojavil 26. junija 2000. Prva stabilna verzija C# 1.0 pa je izšla 13. februarja 2002. Uporabljalo se je ogrodje .NET Framework 1.0 skupaj s Visual Studio .NET 2002.

Skozi leta razvijanja so uvedli veliko novosti in sprememb kot so: statični razredi, anonimni tipi spremenljivk, razširitvene metode, interpolacija string spremenljivk ...

Ob času pisanja tega priročnika za delo s C# uporabljamo Visual Studio 2015, ki podpira C# 6.0 ter .NET Framework 4.6.

## **OGRODJE .NET FRAMEWORK**

Ogrodje .NET Framework je Microsoftov programirni model upravljanja programske kode. Uprablja se za razvoj aplikacij na Microsoftovih odjemalcih, strežnikih in mobilnih ter vgradnih napravah.

To ogrodje podpira sledeče programske jezike: C#, J#, C, C++.

Ob prevodu programa se izvorna koda teh jezikov prevede v vmesno kodo (IL - Intermediate Language). Ob zagonu programa pa prevajalnik JIT (Just in Time) vmesno kodo prevede v strojno kodo na podoben način kot pri programskem jeziku Java.

Skupna vmesna koda teh jezikov pa nam omogoča, da so posamezni deli istega programa napisani v različnih .NET jezikih.

Pod ogrodje .NET Framework spada izvajalno okolje **CLR (Common Language Runtime)** in zbirka razrednih knjižnic **NET Framework class library**, ki vsebujejo podatkovne baze (ADO.NET), spletne aplikacije (ASP.NET), Windows Presentation Foundation (WPF) in Windows forms (okna).

Okolje **CLR** lahko omeji dostop do sistemskih datotek in drugih pomembnih podatkov.

## **OBJEKTNO PROGRAMIRANJE**

Pri objektnem programiranju se ukvarjamo z objekti in pri nekaterih programskih jezikih tudi z razredi.

Pri programskem jeziku C# ustvarjamo razrede, katerim lahko dodelimo podatke in metode. Nato pa ustvarimo objekte, ki tem razredom pripadajo. Berejo in spreminjajo lahko vrednosti spremenljivk iz svojega razreda ter uporabljajo njihove metode.

Ob deklaraciji več objektov istemu razredu vsi objekti prejmejo enake podatke in metode. Po deklaraciji pa je vsak objekt posamezna entiteta, saj ob spreminjanju podatkov v enem objektu ne vplivamo na druge objekte istega razreda.

Poleg izdelave razredov in deklaracij objektov lahko uporabimo za natančnejšo in podrobnejšo objektno programiranje naslednje postopke: **dedovanje** (inheritence), **združevanje** (encapsulation), in **večličnost** (polymorphism).

Najpogosteje se uporablja dedovanje, saj nam omogoča, da objekt razreda dostopa do podatkov in metod drugega razreda. Kot pri navadnih razredih tudi pri dedovanju objekti nimajo vpliva na druge, saj objekti še vedno prejemajo spremenljivke in metode iz svojega in podedovanega razreda, brez vpliva že prej deklariranih objektov.

V programskem jeziku C# lahko pri dedovanju določimo, kateri podatki in metode iz razreda bodo dedovani drugemu razredu.

## **UPORABA VISUAL STUDIA**

Ker se bomo osredotočili na Windows aplikacije bomo za urejanje kode in oblikovanje oken uporabljali Microsoftov Visual Studio 2015.

Na spletni strani <https://www.visualstudio.com/> nam Microsoft ponuja več različic: Community, Professional in Enterprises. Prvi je popolnoma brezplačen in ponuja vse funkcije Visual Studia. Drugi dve pa ponujata še nekaj dodatnih funkcij, ki so v glavnem osredotočene na skupinsko delo ali delo v podjetju.

Po prenosu Visual Studio Community ga namestimo kot vsak drugi program, izberemo pa si tudi privzeto lokacijo naših projektov na računalniku.

Ko zaženemo program se nam prikaže start page. Na tej strani lahko vidimo nazadnje urejene projekte. Ustvarimo lahko nove ali odpremo shranjene projekte. Pomembno je

omeniti da Visual Studio v nekaterih primerih projekte kliče rešitev (solution). V tem programu sta imeni sopomenki.

Na tej strani se prikažejo tudi novice in razne ponudbe ter oglasi. To stran pa lahko zapremo saj ni nujna za uporabo Visual Studia. Omogoča nam še nastavitve za prikaz tega okna v prihodnje.

Start Page

## Visual Studio

Start

- New Project...
- Open Project...
- Open from Source Control...

Recent

- while
- WindowsFormsApplication7
- WindowsFormsApplication8
- WindowsFormsApplication6
- WindowsFormsApplication2
- WindowsFormsApplication3
- WindowsFormsApplication4
- WindowsFormsApplication5
- WindowsFormsApplication1
- WindowsFormsApplication2

### Discover Visual Studio Community 2015

New to Visual Studio? Check out coding tutorials and sample projects  
Get training on new frameworks, languages, and technologies  
Create a private code repo and backlog for your project  
See how easy it is to get started with cloud services  
Discover ways to extend and customize the IDE

Ready to Cloud-power your experience?  
[Connect to Azure](#)

### News

#### New feature to enable C# 6 / VB 14

The model to leverage updated versions of C# or VB in web apps has changed over the past few years. In the past to leverage a new version of C# or VB you would...

sreda, december 16, 2015

#### Clang with Microsoft CodeGen in VS 2015 Update 1

One of the challenges with developing and maintaining cross-platform C++ code is dealing with different C++ compilers for different platforms. You...

sreda, december 16, 2015

#### Windows IoT Core and Azure IoT Hub – Putting the 'I' in IoT

Earlier this year, we announced the availability of Windows IoT Core – a new edition of Windows designed for maker boards and small devices. We've...

sreda, december 16, 2015

#### The .NET Journey: Recapping the last year

Having just completed Connect() // 2015, we thought to take a moment to review everything that's happened with .NET over the last year, between last year's and t...

sreda, december 16, 2015

#### Improving your build times with IncrediBuild and Visual Studio 2015

IncrediBuild is a software acceleration technology that allows builds, tests, and other development processes to execute in parallel over a distributed network. It wo...

četrtak, december 10, 2015

#### Announcing Public Preview of Visual Studio Marketplace

We're excited to announce the Public Preview of the new Visual Studio Marketplace – the one place to discover and acquire extensions, integrations and sub...

četrtak, december 10, 2015

#### Improvements for C++ Edit and Continue in Visual Studio 2015 Update 1

In Visual Studio 2015 RTM we announced Edit and Continue (EnC) support for both X86 and X64 C++ in the default Debug Engine with the VC++ 2015 toolset...

četrtak, december 10, 2015

#### Developer update for Microsoft Band

It's been a year since we launched Microsoft Band and Microsoft Health. We've made a commitment to the developer community to make our ecosystem open a...

četrtak, december 10, 2015

#### What's New in Visual Studio Update 1 for .NET Managed Languages

Hold on to your hats, cowboys and cowgirls! A lot of exciting things are coming out of the .NET Managed Languages team for Visual Studio 2015 Update 1. Rea...

torek, december 8, 2015

#### Node.js Tools 1.1 for Visual Studio Released

Since our first stable release of Node.js Tools 1.0 for Visual Studio (NTVS) earlier this year, we have seen a lot of interest in this free extension. We are both humble...

torek, december 8, 2015

### New on Microsoft Platforms

- Windows
- Microsoft Azure
- ASP.NET and Web

### Featured Videos

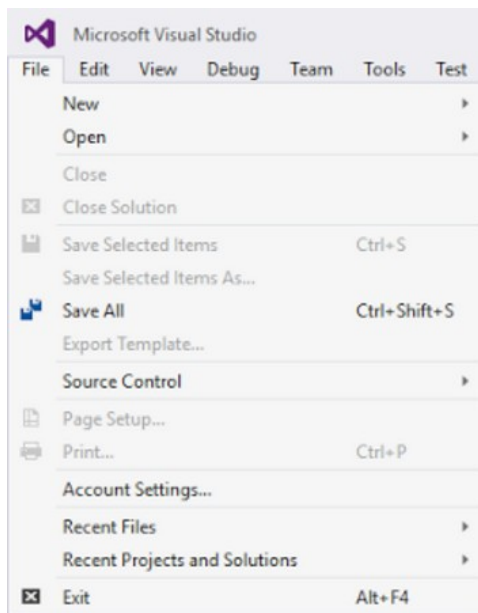
- AppStudio Publish to Windows 10 Store from Source Code 24:11
- Windows Presentation Foundation (WPF) Application Development 15:49
- What's new in C# 6.0 7:49
- What's New for .NET 2015 8:21
- Connecting to Services with Visual Studio 7:12

[More videos](#)

☐ Keep page open after project loads  
☐ Show page on startup

Slika 1 Start Page

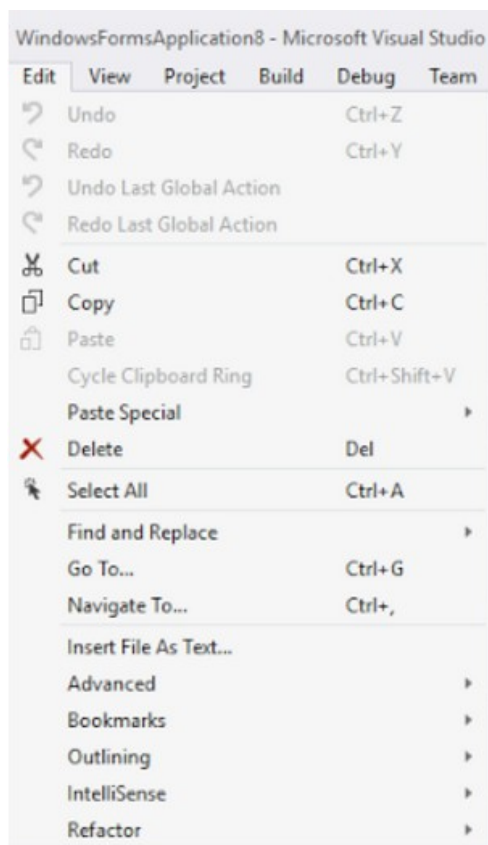
Najprej si bomo ogledali vse zavihke v orodni vrstici, nato pa še vsa okna, ki jih bomo potrebovali med našim delom.



Slika 2 Zavihek File

Zavihek **File** nam omogoča:

- Ustvarjanje novega projekta.
- Visual Studio omogoča tudi ustvarjanje drugih reči kot so spletne strani ali skupinski projekti, ukvarjali pa se bomo le z osnovami jezika C#.
- Odpiranje, zapiranje in shranjevanje projektov ter posameznih datotek.
- Uredimo lahko stran za ogled kode, lahko jo tudi natisnemo.
- Urejamo lahko nastavitve računa.



Slika 3 Zavihek Edit

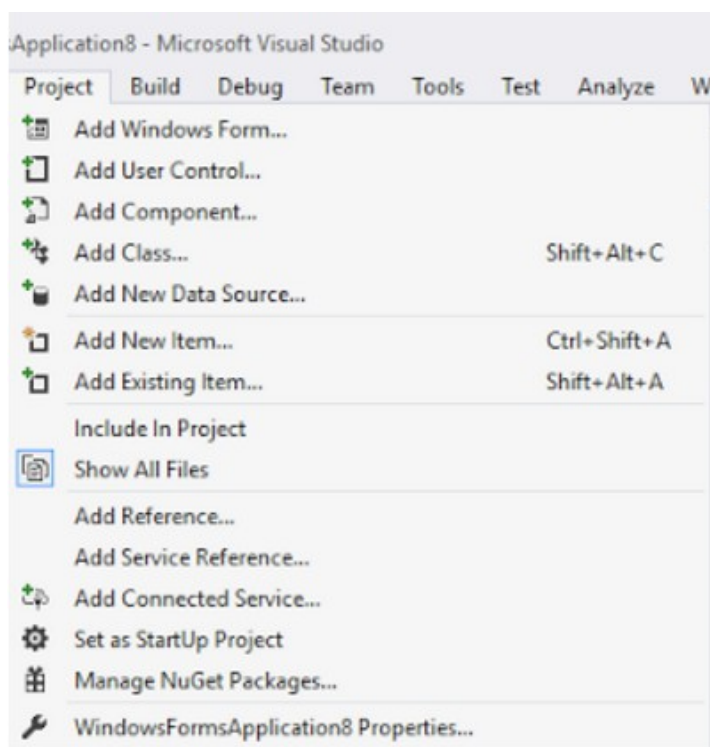
Zavihek **Edit** nam ponuja podobne možnosti kot ostali programi za urejanje kode ali besedila:

- Razveljavi in uveljavi, kopiraj, izreži, prilepi, izbriši.
- Izberi vse, najdi in zamenjaj.
- Z iskanjem po strani in datotekah nam pomagata "Go To" in "Navigate To".
- Imamo tudi nekaj naprednih možnosti za urejanje, kot so oznake besedila in sprememba med velikimi in malimi črkami.
- Zelo pomembni funkciji pa sta spreminjanje zamika kode, ki naredi kodo bolj pregledno ter IntelliSense, ki nam pomaga pri pisanju tako da dopolnjuje besede ali nam da seznam možnosti. IntelliSense nam da tudi informacije o kodi, ki jo uporabljamo in zato je to odlično orodje za začetnike in tudi bolj napredne uporabnike.

Zavihek **View** nam omogoča prikaz vseh možnih oken, ki jih pri uporabi Visual Studio uporabljamo. Našteli bomo vsa okna, ki jih bomo uporabljali pri delu. Nekaj pa jih bomo zanemarili, ker pri naših projektih nimajo pomena npr. Team Explorer ali SQL Server Object Explorer.

- Solution Explorer prikaže vse kar pripada našemu projektu (glavni program, razredi, oblikovana okna...)

- Call Hierarchy nam prikaže povezave ali klice med različnimi deli programa. Pri zapletenih projektih je to pomembno saj omogoča lažji pregled nad programom in morebitnimi možnostmi za izboljšavo. Pri našem delu pa tega okna ne bomo potrebovali saj naši programi ne bodo tako kompleksni.
- Class View je uporabljen za lažji pregled naših razredov. Pokaže nam posamezne razrede in vsako od njihovih komponent. Tudi tega okna ne bomo potrebovali saj naši razredi ne vsebujejo veliko komponent.
- Code Definition Window nam pove na kaj kaže izbrana referenca. Če želimo videti kaj vsebuje npr. funkcija, kliknemo na njeno referenco in v oknu se nam pokaže celota katero kličemo.
- Object Browser nam pokaže vse objekte v projektu, vsakemu doda še kratek opis in pove kje v projektu se nahaja.
- Error List je okno v katerem se prikažejo morebitne napake v kodi in je eno izmed pomembnejših oken saj nam olajša iskanje napak.
- Output nam v oknu prikaže razna sporočila o statusu našega programa.
- Start page odpre začetno stran.
- Task List nam omogoča zapis opravi v projektu. Poleg komentarja lahko določimo še datoteko in vrstico kode. To okno je priročno za večje projekte s veliko komentarji, saj jih tako lažje uredimo in beremo.
- Toolbox uporabljamo pri oblikovanju naših obrazcev ali oken. Omogoča nam dodajanje novih elementov kot so gumbi, števc, območje za besedilo ...
- Properties Windows nam prikaže vse lastnosti izbranega objekta npr. barva, pozicija, ime ...

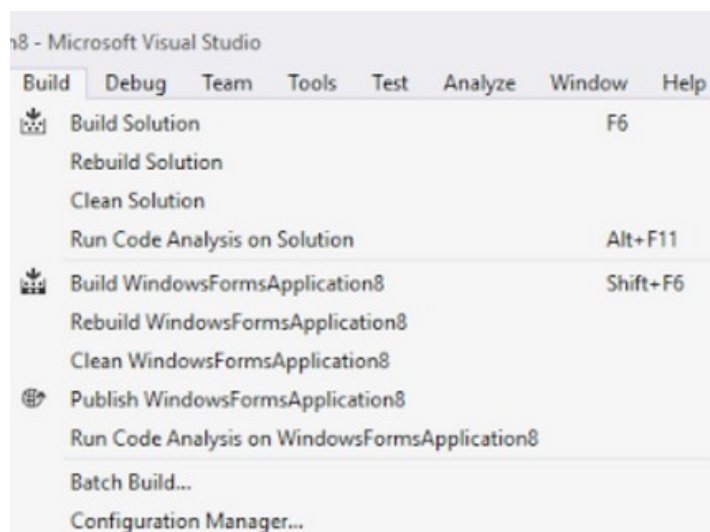


Slika 4 Zavihek Project

Zavihek **Project** omogoča dodajanje raznih objektov: Windows obrazci, uporabniški nadzor, komponente, razredi, podatkovni viri...

Dodamo lahko nove ali že obstoječe objekte ter reference na knjižnice.

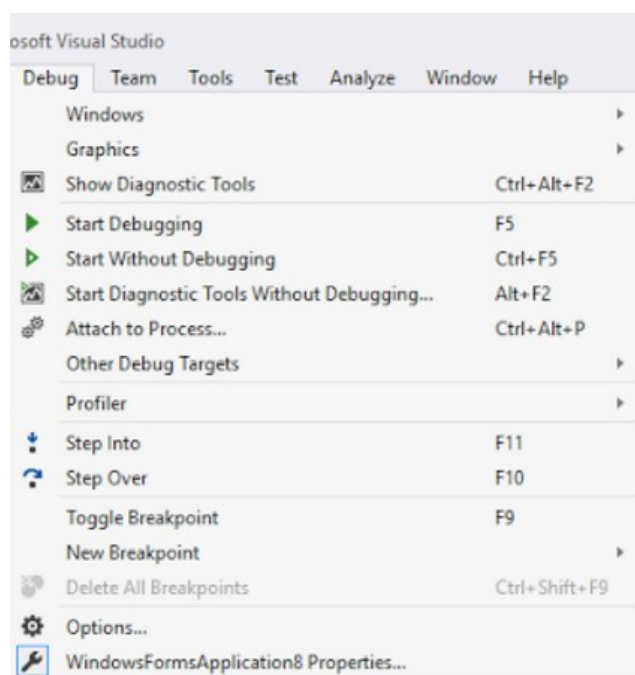
Nudi nam tudi druge bolj napredne možnosti ter spreminjanje nastavitev projekta.



Slika 5 Zavihek Build

Zavihek **Build** nam omogoča grajenje kode v projektu in splošno preverjanje delovanja našega projekta. Naš projekt lahko tudi objavimo lokalno ali na FTP strežnik.

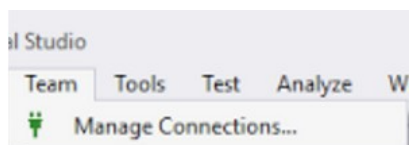
Ta zavihek se uporablja po končanem delu, ko hočemo program zagnati izven Visual Studia 2015.



Slika 6 Zavihek Debug

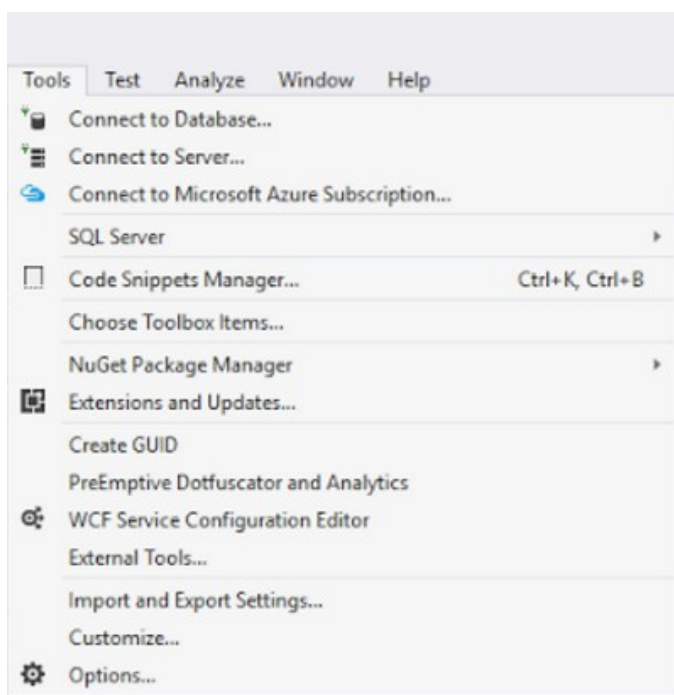


Zavihek **Debug** uporabljamo za iskanje morebitnih napak v naših programih.



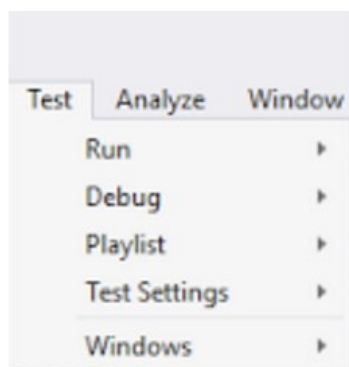
Slika 7 Zavihek Team

Zavihek **Team** uporabljamo pri ekipnem delu saj nam omogoča lažji pregled skupine, ki dela na istem projektu.



Slika 8 Zavihek Tools

Zavihek **Tools** nam omogoča različne nastavitve in napredne funkcije kot npr. povezava na podatkovno bazo ali server, povezava na Microsoft Azure, ustvarjanje SQL tabel, izberemo lahko kaj se prikaže v oknu "Toolbox", upravljamo s razširitvami in posodobitvami, uvažamo in izvažamo nastavitve Visual Studia...



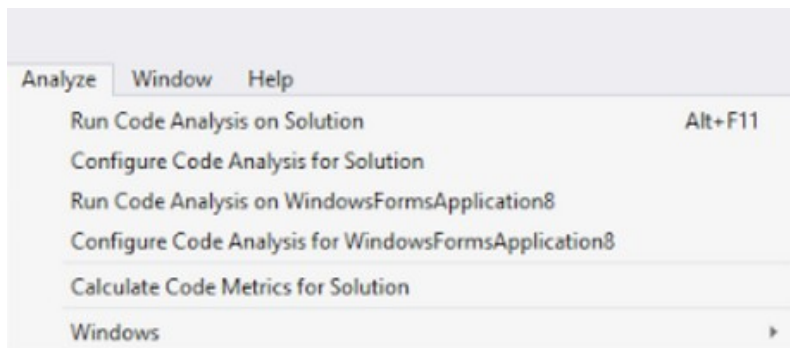
Slika 9 Zavihek Test



Zavihek **Test** se uporablja za testiranje delovanja projekta med delom.

Prednost tega zavihka je da lahko v primeru napak zaženemo teste na različnih delih programa. Test Explorer je okno ki nam bo pokazalo kateri testi so uspešni, neuspešni ali preskočeni.

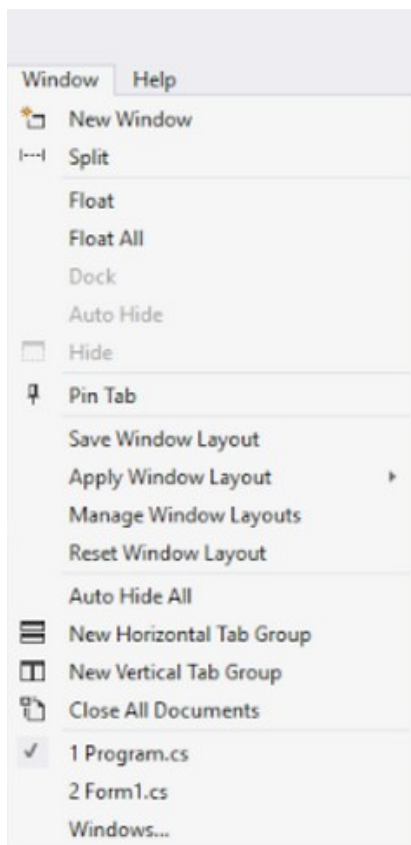
V Test Settings lahko spreminjamo arhitekturo procesorja med 32 in 64 bitno da se dodatno prepričamo o delovanju programa.



Slika 10 Zavihek Analyze

Zavihek **Analyze** nam podobno kot nekateri prejšnji zavihki pomaga z iskanjem težav v projektih. Razlika je v tem da lahko tukaj analiziramo projekt in ta postopek nam poda številne podatke o našem programu.

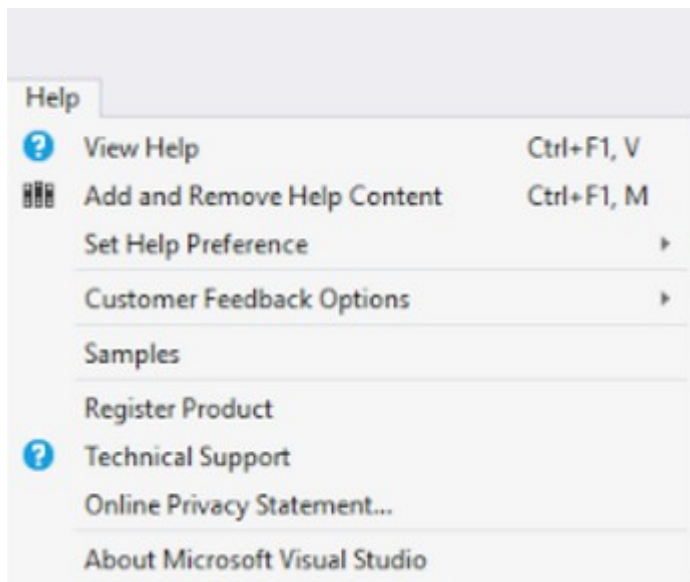
Analiziranje pride prav ob ukvarjanju s velikimi projekti



Slika 11 Zavihek Window

Zavihek **Window** nam omogoča odpiranje novih oken, razdeljevanje in ločevanje od glavnega okna. Shranjujemo in urejamo lahko pozicije ter ustvarjamo nove predele za okna.

Na dnu lahko vidimo katere strani so odprte in če kliknemo na "Windows..." lahko s odprtimi okni upravljamo.



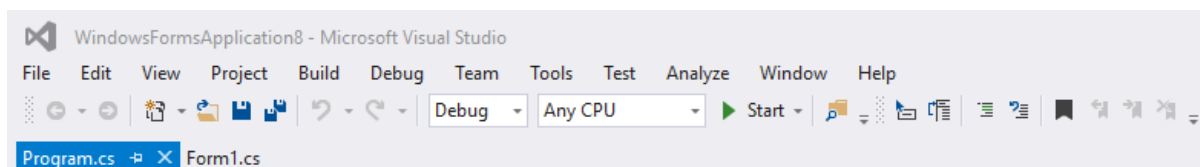
Slika 12 Zavihek Help

Zavihek **Help** nam omogoča dostop do pomoči za Visual Studio 2015, nastavitve uporabniške pomoči npr. namigi o orodjih.

Preberemo lahko o Visual Studiu ter kontaktiramo uporabniško podporo.

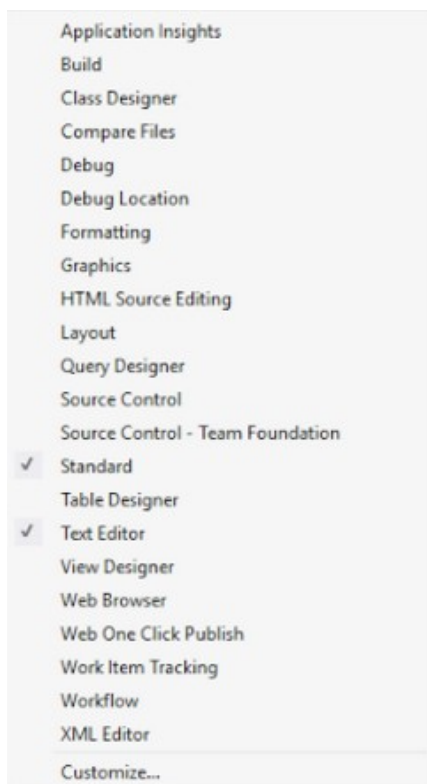
Če kliknemo na Samples pa nam program omogoči odpiranje raznih že narejenih projektov, ki nam lahko pomagajo pri učenju programskih jezikov podprtih s strani Visual Studia.

## Orodna vrstica ter izbira strani



Slika 13 Orodna vrstica

Orodna vrstica ima bližnjice do funkcij iz prej omenjenih zgornjih zavihkov. Z desnim klikom jo lahko prilagodimo, tako da vsebuje vse kar potrebujemo. Mi bomo orodno vrstico pustili nastavljeno kot je privzeto ob inštalaciji.



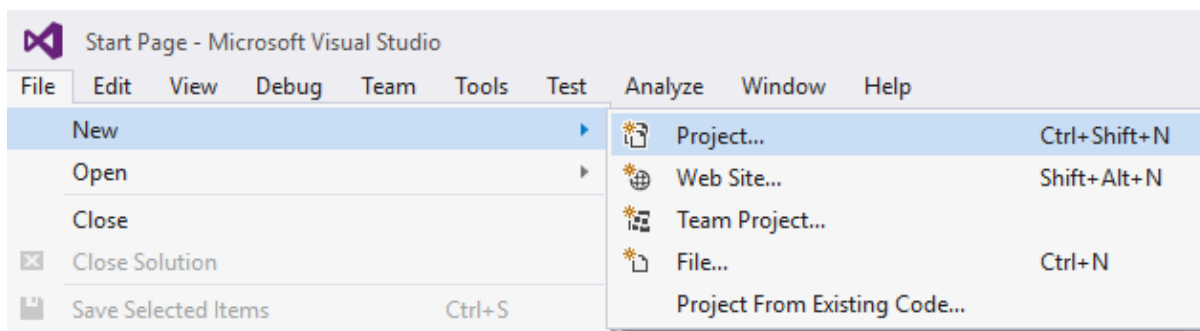
Pod orodji pa lahko izbiramo med odprtimi stranmi, ki jih odpremo tako da v oknih kjer imamo pregled nad projektom odpremo strani.

## OSNOVE PROGRAMSKEGA JEZIKA

### Pozdravljen svet

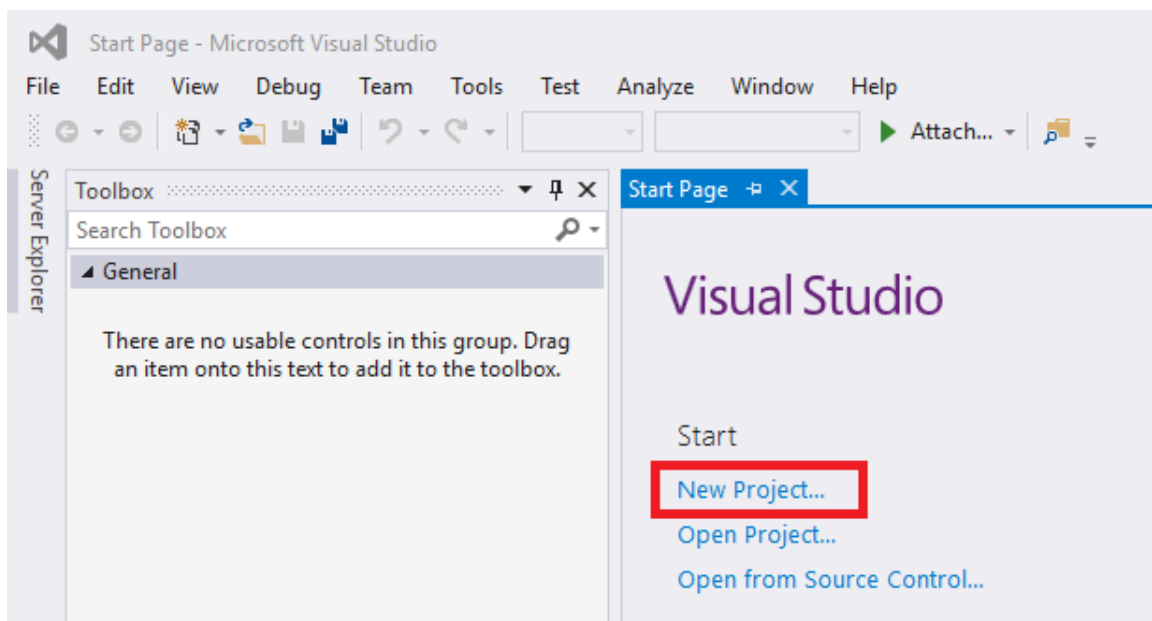
Sedaj, ko poznamo Visual Studio lahko začnemo z osnovami programskega jezika C# in izdelamo naš prvi projekt.

Za začetek bomo izdelali najpreprostejši program **Pozdravljen svet** (Hello world), ki v ukaznem pozivu izpiše vrstico "Pozdravljen svet!". Če želimo izdelati nov projekt, lahko to naredimo na dva načina, ki sta prikazana na spodnjih slikah.



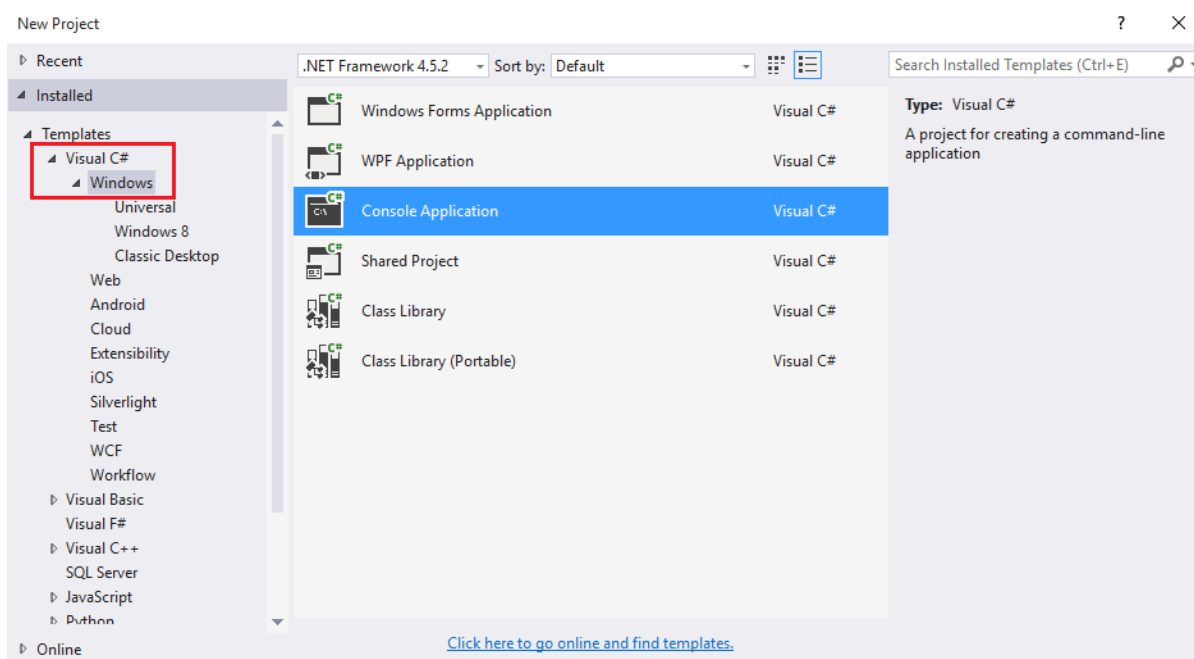
Slika 14 Ustvarjanje novega projekta

V orodni vrstici pod zavihkom **File** izberemo **New** in **Project** oziroma uporabimo bližnico **Ctrl+Shift+N**, ki jih lahko vidimo na desni strani vseh odprtih zavihkov.



Drugi način je še enostavnejši, saj lahko v našem delovnem zavihku (Start Page) kliknemo na **New Project**, kot je prikazano na sliki.

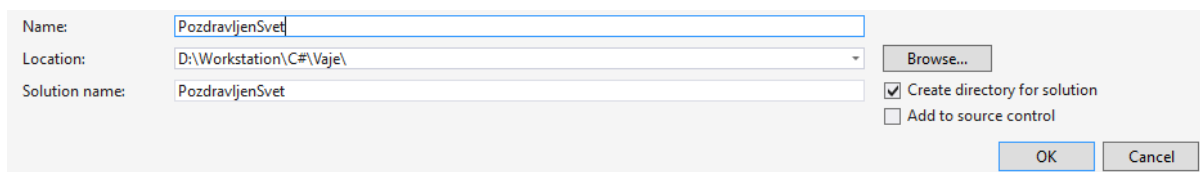
Pri ustvarjanju novega projekta se nam bo prikazalo okno, kjer lahko izberemo programski jezik, platformo, vrsto aplikacije in lokacijo na računalniku, kjer bomo shranjevali naš projekt.



**Slika 15** Izbira konzolne aplikacije

V našem primeru bomo uporabili programski jezik **Visual C#** in platformo **Windows** ter za vrsto aplikacije **Console Application**, saj želimo, da je naš program čim bolj preprost.

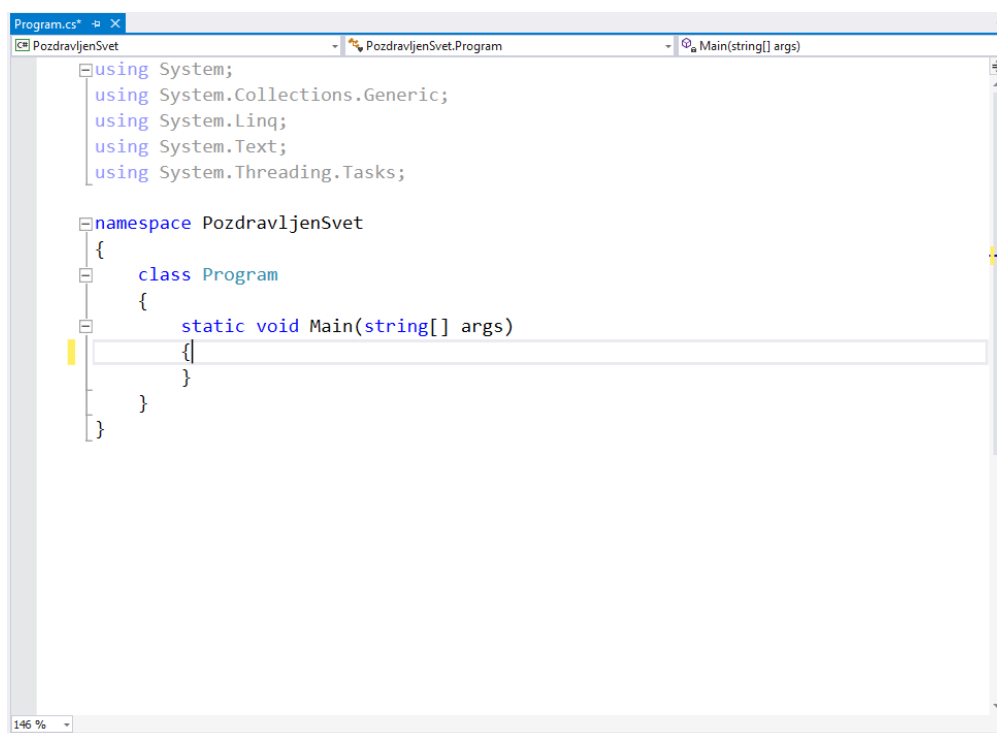
Zgoraj imamo še možnost za izbiro ogrodja .NET Framework, ampak priporočeno je, da to nastavitev pustimo kot je oziroma uporabimo različico **.NET Framework 4.5** ali novejšo.



Slika 16 Poimenovanje projekta

Na koncu še določimo ime in lokacijo našega projekta in pritisnemo na gumb OK.

Visual Studio bo ustvaril nov projekt in prikazalo se nam bo naslednje okno z nekaj kode.



Slika 17 Prazna datoteka konzolne aplikacije

Zaenkrat nas trenutna koda ne zanima, saj bomo za izdelavo naše aplikacije uporabili le prostor med zavitima oklepajema na sliki.

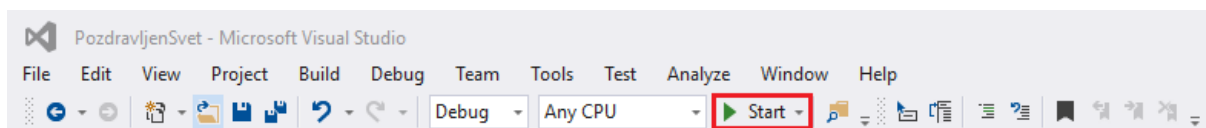
```
static void Main(string[] args)
{
    |
}
```

Za izpis v ukaznem pozivu bomo napisali funkcijo »**Console.Write(“Pozdravljen svet!”);**«

```
static void Main(string[] args)
{
    Console.WriteLine("Pozdravljen svet!");
}
```

Kot v večini programskih jezikov moramo na koncu ukaza napisati podpičje, saj ga s tem zaključimo.

Naš program je skoraj končan, zato ga bomo zagnali v orodni vrstici s pritiskom na ukaz **Start** oziroma uporabimo bližnjico **F5** (tipka na tipkovnici).



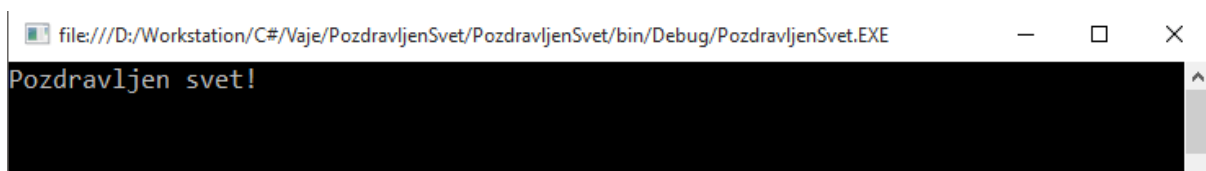
Slika 18 Zagon programa

Če smo dobro opazovali po pritisku na **Start**, smo videli ukazni poziv, ki se je odprl in takoj zatem zaprl. Problem je v tem, da se program konča takoj, ko izpiše besedilo v ukazni poziv.

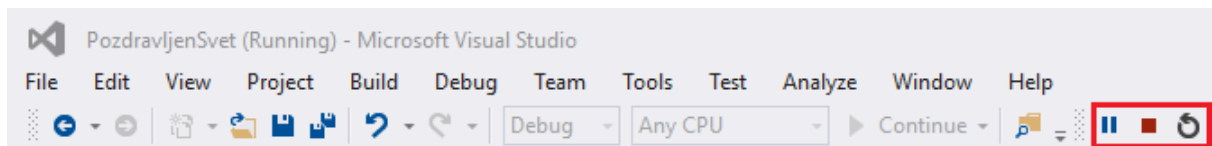
Problem lahko rešimo, tako da pod funkcijo »**Console.WriteLine("Pozdravljen svet!");**« napišemo še funkcijo »**Console.ReadKey(true);**«, ki bo zahtevala od uporabnika pritisk katerekoli tipke na tipkovnici.

```
static void Main(string[] args)
{
    Console.WriteLine("Pozdravljen svet!");
    Console.ReadKey(true);
}
```

Sedaj lahko ponovno zaženemo naš program s pritiskom na **Start** ali bližnjico **F5** in prikazal se nam bo ukazni poziv z izpisom "Pozdravljen svet!".



Ob pritisku katerekoli tipke na tipkovnici se bo program končal. Za zapiranje programa lahko uporabimo tudi ukaz **Stop Debugging** v orodni vrstici. Poleg njega sta še ukaz za premor in ponoven zagon programa.



Slika 19 Zaustavitev, premor in ponoven zagon programa

## Spremenljivke

V tem poglavju se bomo poglobili v pomembno in najosnovnejšo temo pri programiranju, spremenljivke.

Vsi programski jeziki uporabljajo spremenljivke, zato je zelo enostavno prehajanje med njimi.

Spremenljivka je nekaj kar lahko spreminja svojo vrednost, ali bolj natančno je prostor v pomnilniku, v katerem so lahko shranjeni različni podatki ob različnem času. V programskem jeziku C# se najpogosteje uporabljajo naslednji tipi spremenljivk: **int**, **float**, **double**, **byte**, **char**, **string**, **bool** in **object**.

### INT

Začeli bomo z vrsto spremenljivke **int**, zato izdelajmo nov projekt z ukaznim pozivom in se osredotočimo na prostor med zavitima oklepajema v glavni **Main** funkciji.

```
static void Main(string[] args)
{
    |
}
```

Spremenljivko deklariramo tako, da najprej napišemo vrsto (v našem primeru je to **int**) in potem poljubno ime, ki ne vsebuje **posebnih znakov**, **ločil**, **presledkov** in se **ne začne** s številko. Za ime bomo uporabili "stevilo". Na koncu deklaracije zapišemo še podpičje.

```
static void Main(string[] args)
{
    int stevilo;
}
```

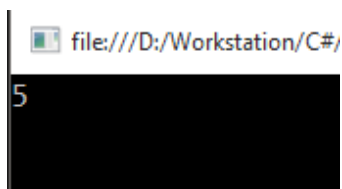
Naši spremenljivki lahko sedaj določimo vrednost, tako da napišemo ime spremenljivke in z uporabo enačaja na desni strani zapišemo še vrednost. Spremenljivki tipa **int** lahko dodelimo za vrednost le **cela števila**.

```
static void Main(string[] args)
{
    int stevilo;
    stevilo = 5;
}
```

Vrednost naše spremenljivke lahko zdaj izpišemo v ukaznem pozivu z uporabo funkcije »**Console.Write(stevilo.ToString());**«. Ker funkcija lahko izpiše le vrednosti tipa **string** in naša spremenljivka "stevilo" ima tip **int**, moramo dodati na koncu za normalen izpis še funkcijo »**ToString();**«

```
static void Main(string[] args)
{
    int stevilo;
    stevilo = 5;
    Console.Write(stevilo.ToString());
    Console.ReadKey(true);
}
```

Ko zaženemo program se bo v našem ukaznem pozivu izpisala vrednost spremenljivke "stevilo".



Vrednost spremenljivke lahko določimo tudi že ob deklaraciji, tako da uporabimo enačaj in na desni strani dodelimo vrednost.

```
static void Main(string[] args)
{
    int stevilo = 5;
    Console.Write(stevilo.ToString());
    Console.ReadKey(true);
}
```

Če za vrednost pri tipu **int** uporabimo kaj drugega kot celo število, nas bo Visual Studio obvestil, da smo storili napako oziroma spremenljivka ne bo delovala kot bi morala.

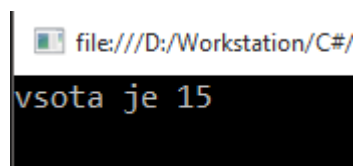
Spremenljivke tipa **int** so predvsem namenjene, da jih uporabljamo pri računanju, zato bomo deklarirali še eno spremenljivko tipa **int**, jo poimenovali "vsota", za vrednost ji bomo dodelili vsoto števila 10 in vrednost spremenljivke "stevilo", ki je v našem primeru



5. Vrednost spremenljivke "vsota" bi sedaj morala biti 15. Na koncu namesto spremenljivke "stevilo" izpišemo spremenljivko "vsota" in za lepšo preglednost bomo dodali funkciji za izpis še besedilo v narekovajih.

```
static void Main(string[] args)
{
    int stevilo = 5;
    int vsota = 10 + stevilo;
    Console.WriteLine("vsota je " + vsota.ToString());
    Console.ReadKey(true);
}
```

Program zaženemo in lahko vidimo, da se nam bo izpisala vrednost spremenljivke "vsota", ki je v našem primeru 15.



## FLOAT

Naslednji tip spremenljivke, ki ga bomo obdelali je **float**. Ta tip lahko shrani 32 bitna decimalna števila do **sedem mest natančno**.

Deklaracija je podobna, kot pri tipu **int** s tem, da namesto **int** pred imenom spremenljivke zapišemo **float**. Podamo poljubno ime z uporabo pravil in na desni strani enačaja decimalno vrednost ter malo črko "f" za zadnjim decimalnim mestom.

Za začetek deklarirajmo spremenljivko **float** z imenom "pi" in vrednostjo 3.14 ter jo izpišimo v ukazni poziv.

```
static void Main(string[] args)
{
    float pi = 3.14f;
    Console.WriteLine(pi.ToString());
    Console.ReadKey(true);
}
```

Program zaženemo in prikazala se nam bo vrednost pi.

file:///D:/Workstation/C#/

3,14

Če ima vrednost več kot sedem mest, bo zadnje decimalno mesto zaokroženo glede na naslednja števila.

```
float pi = 3.14159265359f;
```

file:///D:/Workstation/C#/

3,141593

Spremenljivka **float** lahko shrani tudi cela števila, ker pa porabi več prostora na pomnilniku to ni priporočeno.

Lahko jo uporabimo tudi za računanje z drugimi vrstami spremenljivke, zato bomo deklarirali še spremenljivko **int** z imenom "x" ter vrednostjo 2 in "y" z tipom **float** ter vrednostjo 0. Spremenljivki "pi" in "x" bomo med seboj množili, produkt shranili v "y" in ga izpisali v ukazni poziv.

Za deklariranje več spremenljivk enakega tipa istočasno lahko uporabimo naslednji zapis:

```
float pi = 3.14f, y = 0;
```

Če računamo z decimalnimi števili moramo rezultat vedno shraniti v spremenljivko z vrsto, ki lahko shranjuje decimalne vrednosti.

```
static void Main(string[] args)
{
    float pi = 3.14f, y = 0;
    int x = 2;
    y = pi * x;
    Console.WriteLine(y.ToString());
    Console.ReadKey(true);
}
```

Zaženemo program in izpisal se nam bo produkt spremenljivk "pi" in "x".

file:///D:/Workstation/C#/

6,28

## DOUBLE

Spremenljivke s tipom **double** so zelo podobne tipu **float**. Razlikujejo se v tem, da lahko shranjujejo **64 bitna** števila, do **16** mest natančno.

Deklariramo jo tako, da pred imenom spremenljivke zapišemo tip **double**, nato podamo poljubno ime spremenljivke z uporabo pravil, enačaj in na desni strani vrednost ter na koncu za zadnjim številom črki "d" ali "D". Deklaracijo zaključimo s podpičjem.

```
static void Main(string[] args)
{
    double x = 4.1234567891234567;
    Console.WriteLine(x.ToString());
    Console.ReadKey(true);
}
```

file:///D:/Workstation/C#/

4,12345678912346

Pri računanju se tip **double** obnaša enako kot **float**.

## BYTE

Tip **byte** je podoben tipu **int** s tem, da lahko shranjuje le 8 bitna pozitivna cela števila. To so števila od 0 do 255. Če shrani število, ki je manjše od 0 ali večje od 255 nas bo Visual Studio obvestil, da smo storili napako.

Spremenljivko s tipom **byte** deklariramo na enak način, kot druge spremenljivke. Pred imenom zapišemo tip **byte**, nato podamo poljubno ime z uporabo pravil, enačaj in na desni strani vrednost, ki je na intervalu od 0 do 255.

```
static void Main(string[] args)
{
    byte x = 147;
    Console.Write(x.ToString());
    Console.ReadKey(true);
}
```

file:///D:/W

147

## CHAR

Tip **char** se nekoliko razlikuje od ostalih spremenljivk, saj namesto števil shranjuje znake, kot so črke, posebni znaki, ločila in številke (Unicode 16-bit znaki). Shranimo lahko le en znak.

Deklaracija poteka tako, da najprej napišemo tip **char**, ime spremenljivke, enačaj in na desni strani uporabimo **dva enojna zgornja narekovaja** in med njima znak, ki ga želimo shraniti.

Spremenljivko bomo za poiskus še izpisali v ukazni poziv z uporabo funkcije »**Console.Write()**«. Tokrat nam ni treba uporabiti funkcije »**ToString()**«, ker tip **char** deluje na podoben način kot tip **string**.

```
static void Main(string[] args)
{
    char c = '!';
    Console.Write(c);
    Console.ReadKey(true);
}
```

file:///D:/Workstation/C#

!


## STRING

Spremenljivke **string** so precej podobne tipu **char** s tem, da lahko shranjujejo besede in povedi z vsemi Unicode znaki. Deklariramo jih tako, da najprej zapišemo tip **string**, nato ime spremenljivke, enačaj in na desni strani vrednost obdano z **dvema dvojnima zgornjima narekovajema**. Deklaracijo kot vedno zaključimo s podpičjem.

```
static void Main(string[] args)
{
    string poved = "Programirna je zabavno!";
    Console.Write(poved);
    Console.ReadKey(true);
}
```

Pri izpisu nam ni treba uporabiti funkcije »**ToString()**«, saj ima naša spremenljivka že tip **string**.

Program zaženemo in izpisala se nam bo poved.


 file:///D:/Workstation/C#/Vaje/Pozdra

Programirna je zabavno!

Več **string** spremenljivk lahko s seštevanjem med seboj združimo na podoben način, kot seštevamo cela števila. Za poizkus deklarirajmo še dve spremenljivki tipa **string**. V našem primeru jo bomo poimenovali "drugaPoved" in ji dodali poljubno vrednost, drugo pa "zdruzenaPoved" ter ji prišteli spremenljivki "poved" in "drugaPoved".

```
static void Main(string[] args)
{
    string poved = "Programirna je zabavno!";
    string drugaPoved = " Ali se strinjate?";
    string zdruzenaPoved = poved + drugaPoved;
    Console.Write(zdruzenaPoved);
    Console.ReadKey(true);
}
```

Na koncu s funkcijo »**Console.Write()**« izpišemo spremenljivko "zdruzenaPoved" in program zaženemo.

 file:///D:/Workstation/C#/Vaje/PozdravljenSvet/PozdravljenSvet/bin/Debug

Programirna je zabavno! Ali se strinjate?

## BOOL


Preostala sta nam še dva pomembna tipa spremenljivk, ki se nekoliko razlikujeta od drugih. Eden izmed teh je **bool**, ki lahko shrani le dve različni vrednosti. Ti dve vrednosti sta **true** ali **false**. Vrednost **true** pomeni, da nekaj **drži**, vrednost **false** pa, da nekaj **ne drži**. Spremenljivke tipa **bool** najpogosteje uporabljamo pri pogojih za logične operacije

**if**, **else**, za pogoje v **zankah** ter za **argumente v funkcijah**. Naštete stvari bomo obdelali v naslednjih poglavjih.

Pri deklaraciji, kot vedno najprej napišemo tip spremenljivke (v našem primeru **bool**), nato ime spremenljivke, enačaj in na desni strani enačaja vrednost **true** ali **false**.

```
static void Main(string[] args)
{
    bool soncenDan = true;
    Console.Write(soncenDan);
    Console.ReadKey(true);
}
```

Pri izpisu nam ni treba uporabiti funkcije »**ToString()**« ker vrednosti **true** in **false** nista števili.

 file:///D:/Workstation,

True

## OBJECT

Prišli smo do zadnjega podpoglavja pri spremenljivkah in tokrat bomo obdelali najbolj zanimiv tip spremenljivke **object**. Te spremenljivke lahko shranijo vrednosti katerega koli tipa. Uporabimo jih takrat, ko ne vemo kakšnega tipa bo vrednost, ki jo želimo shraniti. Tudi če **object** spremenljivke lahko shranijo vse vrednosti, je zaradi boljšega delovanja programa še vedno priporočeno uporabiti ostale tipe spremenljivk.

Deklaracija še vedno poteka enako, kot pri ostalih tipih spremenljivk. Najprej napišemo tip spremenljivke **object**, nato ime, zatem enačaj in na desni strani enačaja še katerokoli vrednost. V našem primeru bomo ustvarili dve **object** spremenljivki enega z vrednostjo tipa **string** in drugega z vrednostjo tipa **int**.

```
static void Main(string[] args)
{
    object ime = "Janez Novak ";
    object starost = 35;
    Console.Write(ime + starost.ToString());
    Console.ReadKey(true);
}
```

Pri izpisu za spremenljivko "starost" še vedno upoštevamo, da imamo vrednost **int**, zato moramo klicati funkcijo »**ToString()**«

Prišli smo do konca poglavja s spremenljivkami in v nadaljnjih poglavjih bomo spoznali veliko možnih načinov uporabe spremenljivk z drugimi pomembnimi znanji v programskem jeziku C#.

## If stavek

V tem poglavju se bomo osredotočili na **if stavek** (ang. if statement), ki je v programskem jeziku C# precej uporabna logična operacija. Z **if stavkom** preverimo, če določen pogoj drži (je true) se bo izvršila koda med zavitima oklepajema. Osnovna struktura **if stavka** je prikazana na spodnji sliki.

```
if (POGOJ)
{
    KODA, KI SE IZVRŠI V PRIMERU,
    DA JE POGOJ PRAVILEN.
}
```

Za začetek bomo ustvarili preprost **if stavek**, ki bo preveril ali je število **a** manjše od števila **b**. V primeru, da pogoj drži bomo v ukaznem pozivu izpisali odgovor.

Najprej deklarirajmo spremenljivki **a** in **b** s tipom **int**, ter vrednostmi 1 in 2. Nato ustvarimo **if stavek** s pogojem **a < b**. **If stavek** deluje na naslednji način: če je **a** manjši od **b**, se izvrši koda med zavitima oklepajema. V našem primeru bo ta koda izpis v ukazni poziv.

```
static void Main(string[] args)
{
    int a = 1;
    int b = 2;
    if (a < b)
    {
        Console.WriteLine("a je manjsi od b.");
    }
    Console.ReadKey(true);
}
```

Če program zaženemo, vidimo, da se nam je izpisal odgovor, saj pogoj **a < b** (1 je manjše od 2) res drži.

file:///D:/Workstation/C#

a je manjši od b.

Pri pogojih lahko za primerjavo vrednosti uporabimo naslednje znake: < (manjše), > (večje),

<= (manjše ali enako), >= (večje ali enako), == (je enako). Pri preverjanju, če sta dve vrednosti enaki vedno uporabimo dva enačaja, saj z uporabo enega dodelimo novo vrednost spremenljivkam.

Za vajo bomo z uporabo dvojnega enačaja primerjali spremenljivki a in b, če sta enaki.

```
static void Main(string[] args)
{
    int a = 1;
    int b = 2;
    if (a == b)
    {
        Console.WriteLine("a je enak b.");
    }
    Console.ReadKey(true);
}
```

Če program zaženemo, se v ukaznem pozivu ne bo izpisalo nič, saj a in b nista enaka.

## Else stavek

Stavek **else** (else statement), lahko uporabimo le skupaj z **if stavkom**, kadar mu želimo dodati kodo, ki se bo izvršila, če pogoj ne drži (**false**). **Stavek else** vedno napišemo pod **if stavkom**. Osnovna struktura **else stavka** je prikazana na spodnji sliki.


```
if (POGOJ)
{
    KODA, KI SE IZVRŠI V PRIMERU,
    DA POGOJ DRŽI.
}
else
{
    KODA, KI SE IZVRŠI V PRIMERU,
    DA POGOJ NE DRŽI.
}
```



Za primer bomo uporabili **if stavek** iz prejšnje teme, kjer smo primerjali enakost spremenljivk **a** in **b**. Na koncu **if stavka** bomo dodali **else stavek**, kjer se bo izpisal odgovor v ukazni poziv.

```
static void Main(string[] args)
{
    int a = 1;
    int b = 2;
    if (a == b)
    {
        Console.WriteLine("a je enak b.");
    }
    else
    {
        Console.WriteLine("a ni enak b.");
    }
    Console.ReadKey(true);
}
```

Ker **a** ni enak **b**, se nam bo izvršila koda iz **else stavka**.

 file:///D:/Workstation/

a ni enak b.

**Stavku if** lahko nakopičimo še dodatne pogoje s tem, da mu pod zavitimi oklepaji pripišemo **else if stavek** in nato še pogoj v oklepajih. S tem lahko nakopičimo nešteto pogojev, kjer se bo izvršil le prvi, ki je pravilen (**true**). Vsi ostali pogoji po prvem pravilnem bodo preskočeni.


Za primer bomo spremenili nazadnje napisano kodo, tako da bomo spremenljivko **a** enačili z 4 in **b** z 5. Pod **if stavkom** bomo dodali dva **else if stavka**, s pogojema **a < b** in **a == b-1**. Na koncu bomo dodali še **else stavek**, kjer bomo izpisali odgovor še za preostalo možnost za pogoj.

```

static void Main(string[] args)
{
    int a = 4;
    int b = 5;
    if (a == b)
    {
        Console.WriteLine("a je enak b.");
    }
    else if (a < b)
    {
        Console.WriteLine("a je manjši od b.");
    }
    else if (a == b-1)
    {
        Console.WriteLine("a je za 1 manjši od b.");
    }
    else
    {
        Console.WriteLine("a je zagotovo večji od b.");
    }
    Console.ReadKey(true);
}


```

Po zagonu programa se nam bo v ukaznem pozivu izpisal odgovor prvega pravega pogoja. V našem primeru je to prvi **else if stavek**.

 file:///D:/Workstation/C#

a je manjši od b.

Če **else if stavka** med seboj zamenjamo in program znova zaženemo, se nam bo izvršila koda znotraj zavrtih oklepajev **else if stavka** s pogojem **a == b-1** (prvi pravilen pogoj).

 file:///D:/Workstation/C#/Vaje/Pri

a je za 1 manjši od b.

## Uporaba logičnih operatorjev v if/else stavku

Pri kreiranju **if** in **else if stavkov** je mogoče preveriti več pogojev hkrati. To lahko storimo z uporabo logičnih operatorjev: **&&**, **||** in **!** (z uporabo klicaja pogoj zanikamo).

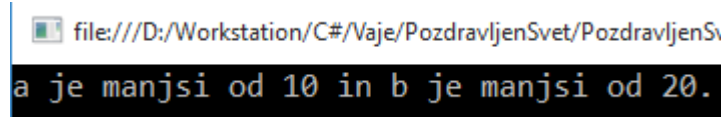
Logični operator **&&** (ang. **and** in slo. **in**) nam omogoči preverjanje dveh pogojev hkrati (več operatorjev kot imamo, več pogojev lahko preverimo). Operator preveri, če sta oba pogoja pravilna, vrne **true** in v našem primeru se izvrši **if stavek**.

Na sliki je prikazan preprost primer uporabe.

```
static void Main(string[] args)
{
    int a = 4;
    int b = 15;

    if(a < 10 && b < 20)
    {
        Console.WriteLine("a je manjši od 10 in b je manjši od 20.");
    }
    Console.ReadKey(true);
}
```

Pogoj lahko opišemo z naslednjimi besedami: "če je a manjši od 10 in b manjši od 20, se izvrši koda med zavitima oklepajema."



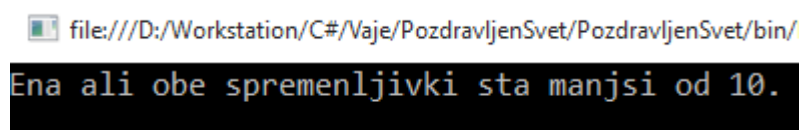
file:///D:/Workstation/C#/Vaje/PozdravljenSvet/PozdravljenSv  
a je manjši od 10 in b je manjši od 20.

Logični operator **||** (ang. **or** in slo. **ali**) nam prav tako omogoči preverjanje več pogojev hkrati, z izjemo, da je dovolj, če drži le en od pogojev.

```
static void Main(string[] args)
{
    int a = 4;
    int b = 15;

    if(a < 10 || b < 10)
    {
        Console.WriteLine("Ena ali obe spremenljivki sta manjši od 10.");
    }
    Console.ReadKey(true);
}
```

Pogoj lahko preprosto opišemo z naslednjimi besedami: "če je a manjši od 10 ali b manjši od 10, se izvrši koda med zavitima oklepajema."



file:///D:/Workstation/C#/Vaje/PozdravljenSvet/PozdravljenSvet/bin/  
Ena ali obe spremenljivki sta manjši od 10.

Logični operator **!** omogoča zanikanje **boolean** vrednosti. Za primer bomo deklarirali **bool** spremenljivki **x = true** in **y = false**.

Ustvarili bomo še **if stavek** s pogojem "**(x && !y)**". Z enostavnimi besedami pogoj pomeni: "če je **x** enak **true** in **y** ni enak **false**, vrne **true** in **if stavek** se izvrši. "

```
static void Main(string[] args)
{
    bool x = true;
    bool y = false;
    if(x && !y)
    {
        Console.WriteLine("x = " + x);
        Console.WriteLine("!x = " + !x);
        Console.WriteLine("y = " + y);
        Console.WriteLine("!y = " + !y);
    }
    Console.ReadKey(true);
}
```

Spremenljivka **y** ima v našem primeru vrednost **false**, ampak z uporabo logičnega operatorja **!** spremenljivki **y** določimo vrednost, ki ni enaka **false**. Ker ima tip **boolean** lahko shranjeni le dve različni vrednosti, bo v primeru **!y** vedno **true**.

Za boljše razumevanje logičnega operatorja **!** bomo v telesu **if stavka** z uporabo funkcije **Console.WriteLine**, ki po izpisu naredi novo vrstico, prikazali njegovo delovanje.

 file:///D:/Workst

```
x = True
!x = False
y = False
!y = True
```

## Switch stavek

Včasih se bomo soočili s situacijo, kjer bo treba preveriti veliko pogojev. S gnezdenjem **if** in **else stavkov**, se bo delovanje programa upočasnilo in koda bo postala slabše pregledna. V takem primeru je priporočeno uporabiti **stavek switch**. Osnovna struktura **stavka switch** je prikazana na spodnji sliki.

```

switch(SPREMENLJIVKA)
{
    case VREDNOST:
        //Koda ki se izvrši, ko je SPREMENLJIVKA == VREDNOST
        break; //Prekinitev stavka switch
    case VREDNOST2:
        //Koda ki se izvrši, ko je SPREMENLJIVKA == VREDNOST2
        break; //Prekinitev stavka switch
    default:
        /* Koda ki se izvrši, ko SPREMENLJIVKA ni enaka
           nobeni zgoraj navedeni vrednosti. */
        break; //Prekinitev stavka switch
}

```

---

**Stavek switch** lahko uporabimo z vsemi tipi spremenljivk. V telesu imamo lahko neomejeno vrednosti, ki bodo preverjene s spremenljivko.

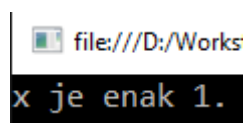
Za preprost primer bomo deklarirali spremenljivko s celoštevilsko vrednostjo in jo uporabili s **stavkom switch**.

```

static void Main(string[] args)
{
    int x = 1;
    switch(x)
    {
        case 1:
            Console.WriteLine("x je enak 1.");
            break;
        case 2:
            Console.WriteLine("x je enak 2.");
            break;
        default:
            Console.WriteLine("x ni enak 1 ali 2.");
            break;
    }
    Console.ReadKey(true);
}

```

Spremenljivki x smo dodelili vrednost 1 in jo vnesli v **stavek switch**. V našem primeru se bo izvršila koda med "case 1:" in "break;".




file:///D:/Works  
x je enak 1.

Če spremenljivki x zamenjamo vrednost v 2 se bo izvršila koda v "case 2:" predelu. V drugih primerih, ko x ni enak 1 ali 2 se bo izvršila koda v predelu "default".

Za boljše razumevanje **stavka switch** bomo napisali še en primer, kjer bomo uporabili spremenljivko tipa **string**.

```
static void Main(string[] args)
{
    string ime = "Janez";
    switch(ime)
    {
        case "Franci":
            Console.WriteLine("Pozdravljen Franci!");
            break;
        case "Peter":
            Console.WriteLine("Dobrodosel Peter!");
            break;
        default:
            Console.WriteLine("Lepo, da sva se spoznala " + ime + ".");
            break;
    }
    Console.ReadKey(true);
}
```

 file:///D:/Workstation/C#/Vaje/PozdravljenSvet

Lepo, da sva se spoznala Janez.

## ZANKE

Naša naslednja tema bodo zanke. To so deli programov, ki ponavljajo ukaze dokler ni izpolnjen njihov pogoj.

Začeli bomo z najpreprostejšo zanko **while**. Ustvarili bomo konzolno aplikacijo, ki bo izpisala števila od 20 do 100 in za to uporabila omenjeno zanko.

Najprej ustvarimo konzolno aplikacijo in deklariramo novo spremenljivko **a**, ki bo celo število 20 in bo uporabljeno za spodnjo mejo.

```
static void Main(string[] args)
{
    int a = 20;
}
```

Sedaj bomo ustvarili osnovno strukturo **while** zanke in v pogoj vpisali »**a<=100**«. To pomeni, da se bo vsebina zanke izvajala dokler je spremenljivka **a** manjša ali enaka 100 oziroma dokler pogoj velja.

```
int a = 20;
while(a <= 100)
{

}
```

V zanko (med zavirami oklepaji { }) bomo vpisali vrstico »**a=a+1**«;. To bo povzročilo da bo vsakič ko se vsebina zanke izvede a povečal za 1. Lahko pa uporabimo tudi krajši zapis »**a++**«, ki opravi enako nalogo. Ta vrstica je zelo pomembna saj se brez nje spremenljivka a ne spremeni in zato bi program tekel neskončno. Temu pravimo neskončni krog (infinite loop).

```
int a = 20;
while(a <= 100)
{
    //a++;
    a = a + 1;
}
```

Zelen tekst pred novo vrstico kode je komentar ki ga označujeta dve poševnici. Komentar je del kode, ki pri zagonu programa ni zagnan in je pogosto uporabljen za lažji pregled in oznako kode. Namesto // pa lahko uporabimo tudi znake **/\* \*/**, ti so uporabljeni za označitev več vrstic kot komentar namesto //, ki označi le eno vrstico.

```
/*
a++;
To je komentar.
*/

//a++;
```

Naša while zanka deluje pravilno. Sedaj moramo poskrbeti še za izpis.

V zanko vnesemo vrstico »**Console.WriteLine(a)**«, ki vsak obhod zanke izpiše takratno vrednost spremenljivke a v novo vrstico. Nato pa vpišemo še »**Console.ReadKey(true)**«, ki omogoči da izpis opazujemo. Ob vsakem pritisku na tipkovnico se izvede naslednji korak zanke in izpiše novo število.

```

static void Main(string[] args)
{
    int a = 20;
    while(a <= 100)
    {
        /*
        a++;
        To je komentar.
        */

        //a++;
        Console.WriteLine(a);
        Console.ReadKey(true);
        a = a + 1;
    }
}

```

Program deluje pravilno in lahko bi ga zagnali, a najprej bomo vrstico »**Console.ReadKey(true);**« izbrisali iz zanke in jo napisali na konec. Tako bo program ukazni poziv ustavil komaj po izvedbi celotne zanke in zato nam ne bo treba po vsakem koraku pritisniti tipke na tipkovnici.



```

static void Main(string[] args)
{
    int a = 20;
    while(a <= 100)
    {
        /*
        a++;
        To je komentar.
        */

        //a++;
        Console.WriteLine(a);
        a = a + 1;
    }
    Console.ReadKey(true);
}

```

Program lahko zaženemo.

20	40	60	80
21	41	61	81
22	42	62	82
23	43	63	83
24	44	64	84
25	45	65	85
26	46	66	86
27	47	67	87
28	48	68	88
29	49	69	89
30	50	70	90
31	51	71	91
32	52	72	92
33	53	73	93
34	54	74	94
35	55	75	95
36	56	76	96
37	57	77	97
38	58	78	98
39	59	79	100

Sedaj bomo program spremenili tako da bo za isti cilj izkoristil **for** zanko namesto **while**.

Posebnost te zanke je v tem, da ji ne podamo le pogoja temveč tudi spremenljivko, ki nam služi kot spodnja meja ter dejanje, ki se zgodi v vsakem krogu zanke. Ti elementi so ločeni s podpičjem.

```
for(int a = 20; a <= 100; a++)
```

To kodo napišemo na mesto kjer je bila prej postavljena while zanka. Opazili boste da smo for zanki podali kodo, ki je bila prej izven zanke saj pri while zanki to ni možnost.

Za pravilno delovanje programa pa moramo izbrisati dve vrstici kode, ki nista več v uporabi. Če tega ne naredimo bo prišlo do napake pri delovanju.

```
static void Main(string[] args)
{
    for(int a = 20; a <= 100; a++)
    {
        /*
        a++;
        To je komentar.
        */

        //a++;
        Console.WriteLine(a);
    }
    Console.ReadKey(true);
}
```

Program lahko zaženete in vidite, da je izpis enak prejšnjemu.

Ogledali si bomo še **do while** zanko. Do sedaj so vse zanke in logični operatorji delali na podlagi izpolnitve pogoja. To pomeni da se nek dogodek ni zgodil če pogoj ni bil izpolnjen. Tukaj je do while poseben saj ne glede na pogoj izvede dogodke v zanki vsaj enkrat.

Najprej bomo odprli novo konzolno aplikacijo ter deklarirali spremenljivko a, ki ima vrednost 5. Napisali bomo tudi osnovno strukturo zanke **do while** ki se začne s **do** nato postopkom, ki se izvrši vsak krog ter **while** in pogojem za izvrševanje zanke.

```
static void Main(string[] args)
{
    int a = 5;

    do
    {

    } while ();
}
```

Dodali bomo tudi pogoj za izvršitev ter dogodek, ki se zgodi če je pogoj izpolnjen. Kot pogoj bomo dodali `a<0` in pod dogodke bomo dodali vrstico `a=a+5`. To bomo naredili za pridobitev dokaza o izvršitvi zanke kljub neustreznem pogoju.

```
do
{
    a = a + 5;
} while (a<0);
```

Nastavili bomo še izpis vrednosti spremenljivke `a` po vsakem koraku. Dodali bomo tudi zahtevo po pritisku na tipkovnico za konec programa.

Program lahko sedaj zaženemo.



Kot lahko vidimo je zanka potekla le enkrat saj pogoj ni bil izpolnjen nikoli a zaradi uporabe zanke **do while** in ne **while** se je vrednost spremenljivke `a` vseeno povečala enkrat.

## ARRAY

Array je podatkovna struktura, ki vsebuje več podatkov istega tipa. Predstavljamo si jo lahko kot tabelo saj imamo lahko tudi več dimenzionalne a mi se bomo posvetili le enodimnezijskim.

Do podatkov v array-u dostopamo s indexom, to je unikatna številka vsakega

elementa. Oznake se začnejo s številom 0. To pomeni da bo imel prvi element oznako 0, drugi 1 itd. kot je prikazano spodaj.

Tabela 3 Ponazoritev podatkovne strukture Array

Podatek (string)	Ime1	Ime 2	Ime3
Oznaka	0	1	2

Array lahko shranjuje vse tipe spremenljivk dokler so enake tipu array-a. Shranjuje lahko celo druge array-e.

Ustvarili bomo tabelo stevila, ki lahko shrani pet celih števil.  
Najprej vpišemo tip naše tabele, oglati oklepaj ter ime.

```
static void Main(string[] args)
{
    int[] stevila;
}
```

V novi vrstici določimo število mest v tabeli. V komentarju pa imamo prikazano deklaracijo in določitev mest v eni vrstici.

```
int[] stevila;
stevila = new int[5];
//int[] stevila = new int[5];
```

Vrednosti dodelimo tako da zapišemo ime tabele ter oznako mesta v oglati oklepaj. Nato zapišemo enačaj ter željeno vrednost.

```
int[] stevila = new int[5];
stevila[0] = 10;
```

Vrednost prvega mesta bomo sedaj izpisali. Spomniti se moramo dodati »**Console.ReadKey(true);**« da se ukazni poziv ne zapre avtomatsko.

```
int[] stevila = new int[5];
stevila[0] = 10;
Console.Write(stevila[0]);
Console.ReadKey(true);
```

Program lahko sedaj zaženete in preverite rezultat.

A screenshot of a console window with a black background. The number '10' is displayed in white text at the top left of the window.

Če ne veste koliko mest bo imela vaša tabela jo lahko deklarirate na drugi način.

```
static void Main(string[] args)
{
    int[] stevila={5,13,20,32,34};
}
```

Tabela ima sedaj toliko mest kolikor vrednosti smo vpisali. Seveda lahko vrednosti še dodajamo ali pa ne vpišemo nobene in to storimo naknadno kot v prejšnjem primeru.

## ARRAY IN ZANKE

Sedaj znamo spreminjati in dodeljevati vrednosti le posameznim mestom. Za lažje in hitrejše delo s tabelami lahko uporabimo zanke, ki smo se jih že naučili.

Ustvarili bomo zanko s spremenljivko, ki se začne z 0 in se veča do 10. Uporabili jo bomo kot oznako mest v tabeli namesto števil. Tako bomo v tabelo imena vpisali 10 imen in jih nato v drugi zanki izpisali.

Začnemo s deklaracijo tabele imena, ki bo vsebovala spremenljivke tipa string, nato pa ustvarimo for zanko. Spremenljivko v zanki bomo poimenovali kazalec.

```
string[] imena = new string [10];
for(int kazalec=0;kazalec<10;kazalec++)
{

}
```

Za dodajanje vrednosti v tabelo bomo uporabili uporabniški vnos oz. `Console.ReadLine` funkcijo. Vsak krog naše zanke bo kazalcu, ki kaže na mesta naše tabele povečal vrednost za 1. Vsako mesto tabele bomo tako enačili s funkcijo, ki bo prebrala uporabnikov vnos dokler ne pritisne na tipko enter.

Ko uporabnik izpolni 10 vnosov se bo zanka ustavila.

```
string[] imena = new string [10];
for(int kazalec=0;kazalec<11;kazalec++)
{
    imena[kazalec]=Console.ReadLine();
}
```

Za izpis teh imen bomo najprej vnesli prazno funkcijo `write line`. Ta služi za lažji pregled teksta v ukaznem pozivu. Nato ustvarimo for zanko, ki je identična prejšnji, le da uporabimo `kazalec2` namesto `kazalec`. To naredimo ker ne moremo deklarirati dveh spremenljivk z enakima imenoma. Spremenljivke `kazalec` pa ne moremo znova uporabiti saj lahko spremenljivko uporabljamo le v zanki kjer je deklarirana.

V novi zanki pa uporabimo funkcijo `Console.Write` za izpis vseh elementov tabele. Za lažjo preglednost izpisa smo v funkcijo dodali presledek(med navedkoma) ter plus znak kar pred izpis vsak krog zanke doda presledek med imena.

```
string[] imena = new string [10];
for(int kazalec=0;kazalec<10;kazalec++)
{
    imena[kazalec] = Console.ReadLine();
}

Console.WriteLine();

for(int kazalec2 = 0;kazalec2<10;kazalec2++)
{
    Console.Write(" "+imena[kazalec2]);
}
Console.ReadKey(true);
```

Sedaj lahko kodo zaženemo.

```
Jaka
Miha
Luka
Leon
Nika
Maja
Kaja
Mateja
Matej
Vitomir

Jaka Miha Luka Leon Nika Maja Kaja Mateja Matej Vitomir
```

Naša koda deluje pravilno, naredili pa bomo še eno izboljšavo. Namesto števila 10 v pogoju bomo vpisali `imena.Length`, kar pomeni dolžina array-a `imena`.

To spremembo naredimo zaradi manjše možnosti napake. Program je deloval pravilno s številom 10 a če bomo vrednosti spreminjali imamo tako manjše možnosti za napake v kodi.

```
string[] imena = new string [10];
for(int kazalec=0;kazalec<imena.Length;kazalec++)
{
    imena[kazalec] = Console.ReadLine();
}
```

```
Console.WriteLine();
```

```
for(int kazalec2 = 0;kazalec2<imena.Length;kazalec2++)
{
    Console.Write(" "+imena[kazalec2]);
}
Console.ReadKey(true);
```

## ***LIST***

List si prav tako kot array lahko predstavljamo kot tabelo. Razlika med dvema je to, da list lahko širimo in krčimo glede na porabo, medtem, ko array-u ne moremo spreminjati velikosti.

Prednost pri array-ih pa je večdimenzionalnost. Kljub temu pa večina programerjev polaga preferenco na list-e.

Spodaj je prikazana deklaracija in dodajanje treh elementov v list imenovan seznam.

```
static void Main(string[] args)
{
    List<int> seznam = new List<int>();
    seznam.Add(12);
    seznam.Add(20);
    seznam.Add(7);
}
```

Naslednji program pa prikazuje zapisovanje in izpisovanje iz list-a s zanko. Kot lahko vidite se pri preprostih programih kot so naši razlik razen v strukturi kode ne pozna. Pri naprednejših programih pa so razlike lahko zelo pomembne za optimizacijo delovanja.

```
static void Main(string[] args)
{
    List<string> seznam_imen = new List<string>();

    int spremenljivka;
    for(int oznaka=0;oznaka<5;oznaka++)
    {
        seznam_imen.Add(Console.ReadLine());
    }

    Console.WriteLine();

    for(int oznaka1=0;oznaka1<5;oznaka1++)
    {
        Console.WriteLine(seznam_imen[oznaka1]);
    }

    Console.ReadKey(true);
}
```

Kot vidite smo v list vpisali pet imen ter jih nato izpisali v stolpcu.

Array-e in list-e torej uporabljamo pri večjih količinah podatkov istega tipa. Omogočajo nam boljšo preglednost ter delo s kodo.



## METODE

Metode so bloki kode, ki omogočajo lažje delo pri programiranju. V nekaterih drugih programskih jezikih se imenujejo funkcije. Namen metod je ponavljanje vrstic kode z drugimi parametri. Z metodami skrajšamo programe, sebi pa zmanjšamo količino dela.

Vsi naši programi do sedaj so bili napisani v metodi Main, ki je temelj vsakega programa. Klicali pa smo tudi druge metode. Ko metodo kličemo se izvede koda, ki je v njej napisana, za delovanje ji podamo parametre, nekatere metode pa teh ne zahtevajo.

Na spodnji sliki je prikazan naš prvi program. V Main metodi smo poklicali metodo Console.Write();, ki je v ukazni poziv izpisala podani parameter.

```
static void Main(string[] args)
{
    Console.Write("Pozdravljen svet!");
}
```

Pred pregledom nekaterih bolj naprednih lastnosti metod bomo napisali dva programa. Enega brez uporabe, drugega pa z uporabo metod. Oba programa bosta iz tipkovnice prejela ime in letnico rojstva dijaka ter izpisal prejeto ime ter starost dijaka leta 2016. Vnesli bomo podatke za tri dijake.

```
static void Main(string[] args)
{
    string Ime1;
    string Ime2;
    string Ime3;
    int D_Rojstva1;
    int D_Rojstva2;
    int D_Rojstva3;

    Console.Write("Vpisi ime: ");
    Ime1 = Console.ReadLine();
    Console.Write("Vpisi datum rojstva: ");
    D_Rojstva1 = int.Parse(Console.ReadLine());
    Console.WriteLine();

    Console.WriteLine(Ime1 + ", Starost: " + (2016 - D_Rojstva1)+"\n");

    Console.Write("Vpisi ime: ");
    Ime2 = Console.ReadLine();
    Console.Write("Vpisi datum rojstva: ");
    D_Rojstva2 = int.Parse(Console.ReadLine());
    Console.WriteLine();

    Console.WriteLine(Ime2 + ", Starost: " + (2016 - D_Rojstva2)+"\n");

    Console.Write("Vpisi ime: ");
    Ime3 = Console.ReadLine();
    Console.Write("Vpisi datum rojstva: ");
    D_Rojstva3 = int.Parse(Console.ReadLine());
    Console.WriteLine();

    Console.WriteLine(Ime3 + ", Starost: " + (2016 - D_Rojstva3)+"\n");

    Console.ReadKey(true);
}
```

```
Vpisi ime: Jaka
Vpisi datum rojstva: 1997
Jaka, Starost: 19
Vpisi ime: Janez
Vpisi datum rojstva: 1990
Janez, Starost: 26
Vpisi ime: Lara
Vpisi datum rojstva: 2000
Lara, Starost: 16
```

```
static int metoda(int datum)
{
    return 2016 - datum;
}

static void Main(string[] args)
{
    string Ime;
    int D_Rojstva;
    int starost;

    for (int i = 0; i < 3; i++)
    {
        Console.Write("Vpisi ime: ");
        Ime = Console.ReadLine();
        Console.Write("Vpisi datum rojstva: ");
        D_Rojstva = int.Parse(Console.ReadLine());
        starost = metoda(D_Rojstva);
        Console.WriteLine();

        Console.WriteLine(Ime + ", Starost: " + (2016 - D_Rojstva) + "\n");
    }
    Console.ReadKey(true);
}
```

Oba programa v ukaznem pozivu dosežeta isti izpis. V drugem lahko vidite primer uporabe metode z imenom metoda v kombinaciji s for zanko.

```
static int metoda(int datum)
{
    return 2016 - datum;
}
```

Tukaj je zapis za statično int metodo z imenom metoda, ki ko jo kličemo prejme celo število in ga deklarira kot **int datum**. Vse metode razen void metod pri klicu vrnejo neko vrednost. Naša metoda vrne številko datum, odšteto od 2016.

V glavni metodi pa to metodo kličemo, kar pomeni da se dogodki v njej izvedejo. Če metoda zahteva vnos spremenljivk (v našem primeru celo število), jih pri klicu podamo.

```
starost = metoda(D_Rojstva);
```

V našem primeru smo morali spremenljivki starost dodeliti vrednost. Zato smo jo enačili s klicem metode, ki vrne primerno vrednost. Ob klicu, metoda deklarira spremenljivko datum, zato smo v oklepaj postavili D\_Rojstva katere vrednost bo prenesena na spremenljivko v klicani metodi. V oklepaj lahko vstavimo tudi vrednost in ne le ime spremenljivke. Bistveno je le da ima enak podatkovni tip kot spremenljivka v novi metodi.

Ob deklaraciji metode določamo podatkovni tip, ki določi kakšne spremenljivke lahko metoda vrne. Ko za metodo uporabimo podatkovni tip void v njej ne zapišemo vrstice, ki vrednost vrne (**return**). Ker metoda ne vrača spremenljivk je pogosto uporabljena za izpis podatkov.

## **RAZREDI**

Vsak program, ki smo ga napisali do tega poglavja je vseboval po en razred.

```
class Program
{
    static void Main(string[] args)
    {
    }
}
```

Razred Program vsebuje nam zelo znano **Main()** funkcijo.

Omenili smo jih že na začetku. Govorili smo o objektnih programskih jeziki in razredi so osnove objektnega programiranja. Opišemo jih lahko kot najmočnejši podatkovni tip saj vsebuje podatke, metode, omogoča pa tudi dedovanje.

Sedaj bomo deklarirali nov razred po imenu **MojRazred**. Ob deklaraciji bomo določili tudi **public** dostopni nivo. Dostopne nivoje lahko uporabljamo pri deklaracijah vseh podatkovnih tipov. Pomenijo pa kateri deli programa lahko te tipe uporabljajo. Vse spremenljivke imajo privzeto vrednost public in to vrednost bomo uporabljali še vnaprej.

Dostopne nivoje pa je vredno omeniti saj se uporabljajo v pisanju bolj zapletenih programov.

```
namespace ConsoleClass
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }

    public class MojRazred
    {

    }
}
```

Ustvarili smo prazen razred, ki še nima nobenih spremenljivk ali funkcij. V njem bomo ustvarili void funkcijo pozdrav ki bo izpisala zdravo.

```
public class MojRazred
{
    void pozdrav()
    {
        Console.WriteLine("Zdravo");
        Console.ReadKey(true);
    }
}
```

Ta funkcija pa se ne izvede saj je nismo klicali iz naše glavne **Main** funkcije.

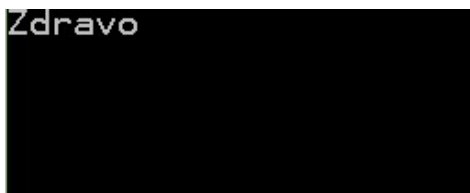
Naši funkciji moramo dodati **public** dostopni nivo saj jo brez tega zapisa lahko uporabljamo le znotraj razreda. V glavni funkciji pa bomo naredili objekt novega razreda, poimenovali ga bomo prvi. Objekti so nekakšne povezave do razredov in uporabili ga bomo za klicanje funkcije pozdrav.

```

class Program
{
    static void Main(string[] args)
    {
        MojRazred prvi = new MojRazred();
        prvi.pozdrav();
    }
}

public class MojRazred
{
    public void pozdrav()
    {
        Console.WriteLine("Zdravo");
        Console.ReadKey(true);
    }
}

```



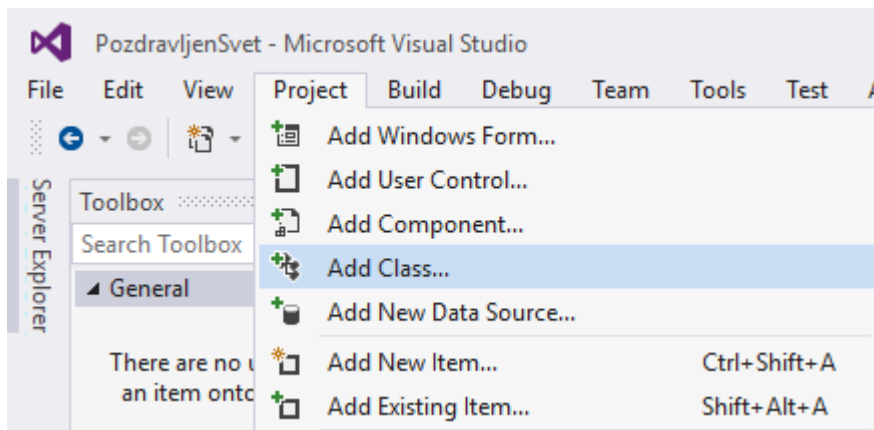
Kot vidite, program deluje pravilno.

## ***NAMESPACE***

### **DELO Z RAZREDI V ENEM NAMESPACE-U**

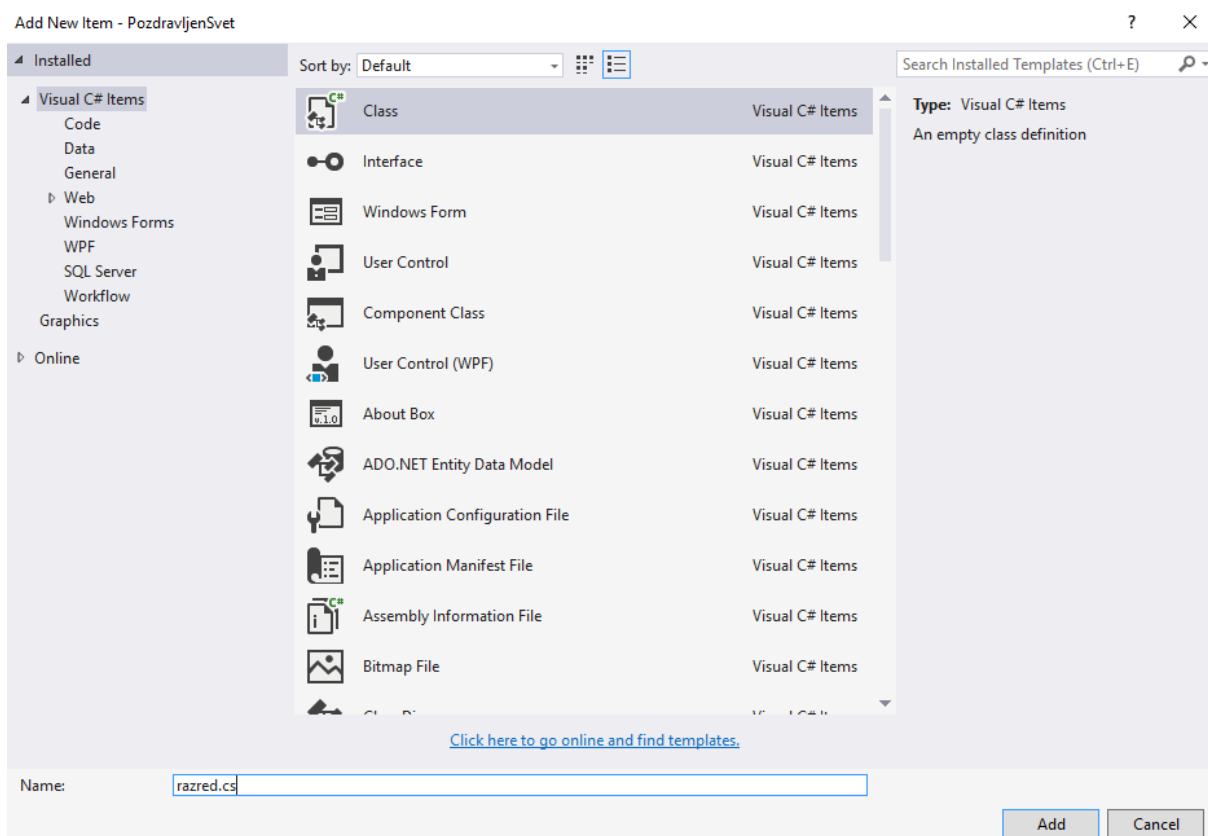
Pri izdelavi večjih programov bomo imeli tudi do več tisoč vrstic. Z uporabo razredov se lahko temu izognemo, ampak obstaja še boljša rešitev. Za boljšo preglednost programa uporabljamo datoteke in namespace.

Novo datoteko, ki bo vsebovala tudi nov razred lahko ustvarimo z naslednjim ukazom v orodni vrstici **Project>Add Class...**



Slika 20 Dodajanje razreda

Po tem ukazu, bomo izbrali ime razreda.



Slika 21 Poimenovanje razreda

Naša nova datoteka vsebuje enak namespace kot izvirna datoteka. Imenuje se **PozdravljenSvet**. Zaradi tega, se bo naša nova koda obnašala kot, da bi bila napisana v izvorni datoteki.

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

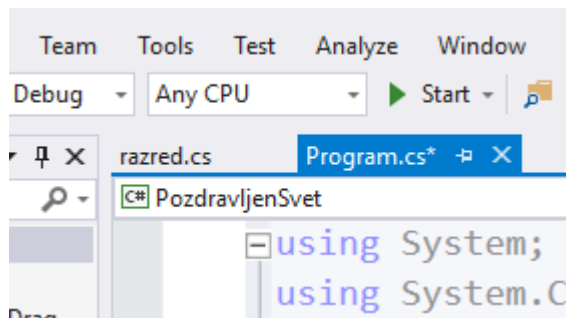
namespace PozdravljenSvet
{
    class razred
    {
    }
}
```

Ko imamo nov razred bomo za začetek napisali preprosto public funkcijo, ki za parameter prejeme spremenljivko tipa int in nato izpiše njen produkt v ukazni poziv.

```
namespace PozdravljenSvet
{
    class razred
    {
        public void funkcija(int x)
        {
            int y = x * x;
            Console.WriteLine(x + " * " + x + " = " + y);
            Console.ReadKey();
        }
    }
}
```

Naš razred je končan, zato se lahko vrnemo nazaj k naši **Main()** funkciji v prvi datoteki. Med datotekami se lahko premikamo s klikom na zavihke pod orodno vrstico.





Slika 22 Delovni zavinki

V Main funkciji bomo izdelali objekt razreda razred in klicali našo funkcijo.

```
namespace PozdravljenSvet
{
    class Program
    {
        static void Main(string[] args)
        {
            razred objekt = new razred();
            objekt.funkcija(4);
        }
    }
}
```

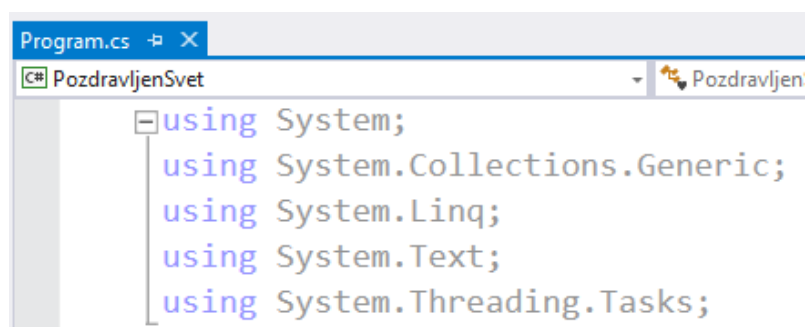
Main funkcija uporablja kodo iz novega razreda. Ker sta obe funkciji v istem namespace-u, se deli programa povežejo pravilno in izpišejo željen rezultat.

file:///D:/Wo

4 \* 4 = 16

## DELO Z RAZREDI V RAZLIČNIH NAMESPACE-IH

Verjetno ste pri delu s C# že opazili vrstice kode, ki vsebujejo using. To so klici na namespace-e, katere so ustvarili pri podjetju Microsoft in jih vključili v namestitev Visual Studia.



Do zdaj smo pogosto uporabili funkcijo **Console.Write** in **Console.WriteLine**. Ti funkciji sta del System namespace-a.

Za boljše razumevanje bomo izbrisali vrstico **using System;** in pred funkcijo za izpis pripisali **»System.«**. Program ni več povezan s tem namespace-om in zato moramo pred funkcijami pripisati namespace iz katerega izvirajo.

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

namespace DeloVRazličnihNamespaceih
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.Write("Pozdravljen svet!");
            System.Console.WriteLine("Primer!");
        }
    }
}
```

Za naslednji primer bomo ustvarili nov namespace in ga povezali z izvirnim preko stavka using.

Kot prej, bomo uporabili ukaz **Project>Add Class...** Razlika je v tem, da bomo namespace preimenovali, saj bi radi za boljšo preglednost uporabili dva namespace-a.

```
Razred.cs* x Program.cs
DeloVRazličnihNamepaceih DeloVRazličnihNamepac

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DeloVRazličnihNamepaceih
{
    class Razred
    {
    }
}
```

Iz **DeloVRazličnihNamepaceih**, smo ga preimenovali v **NovNamespace**.

```
namespace NovNamespace
{
    class Razred
    {
    }
}
```

V Razred bomo dodali funkcijo, ki izpiše število 10.

```
class Razred
{
    public void izpis()
    {
        Console.WriteLine(10.ToString());
    }
}
```

Da lahko to funkcijo uporabljamo v prvi datoteki, moramo dodati vrstico **using NovNamespace**;

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NovNamespace;

namespace DeloVRazličnihNamepaceih
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }
}

```

Namespace-a sta zdaj povezana in v prvotni datoteki lahko uporabimo funkcijo, ki pripada novemu Namespacu.

```

static void Main(string[] args)
{
    Razred primer = new Razred();
    primer.izpis();
    Console.ReadKey(true);
}

```

Program lahko sedaj preizkusimo.



```

10

```

Kot vidite je izpis pravilen.

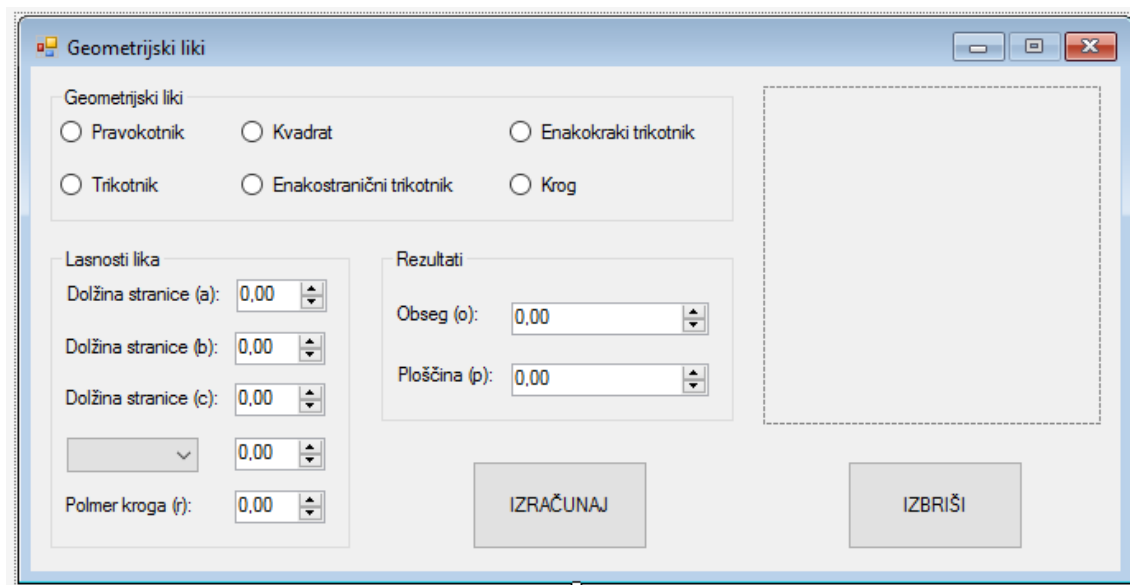
## GRAFIČNI PROGRAM

Naučili smo se osnov programskega jezika C#, ki nam omogočajo pisanje veliko različnih programov. Sedaj bomo svoje znanje uporabili v kombinaciji s grafičnimi knjižnicami Visual Studio(.NET). Te nam omogočajo ustvarjanje okenskih aplikacij kot jih poznamo v Windowsih. Večina oken, ki jih odpiramo in uporabljamo v Windows operacijskih sistemih so bila narejena s temi knjižnicami.

Uporabe grafičnih knjižnic se bomo naučili ob ustvarjanju programa za računanje lastnosti geometrijskih likov. Program bo vseboval različne oblike vnosa in prikaza podatkov, prikazali bomo skice likov ter izračunali obsege ter ploščine likov.

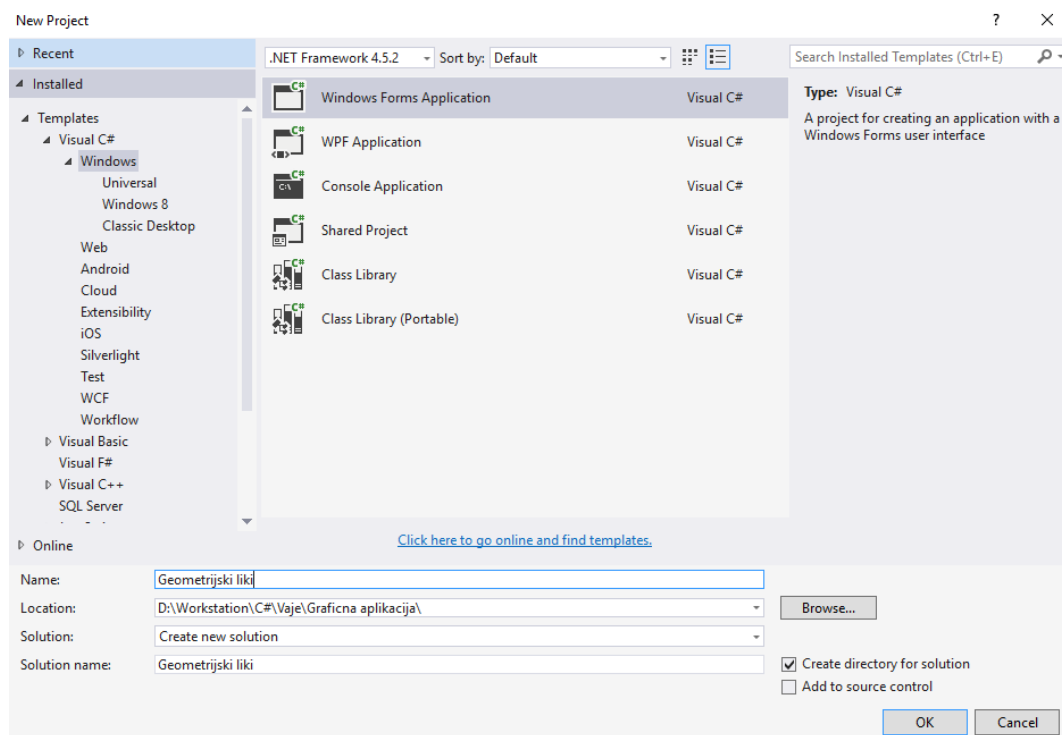
Vstavili bomo tudi gumb, ki prične izračun in gumb, ki zbriše podatke.

Začeli bomo z odprtjem novega projekta. Med izbirami bomo tokrat namesto Console Application izbrali Windows Forms Application. Poimenovali smo jo Geometrijski liki.



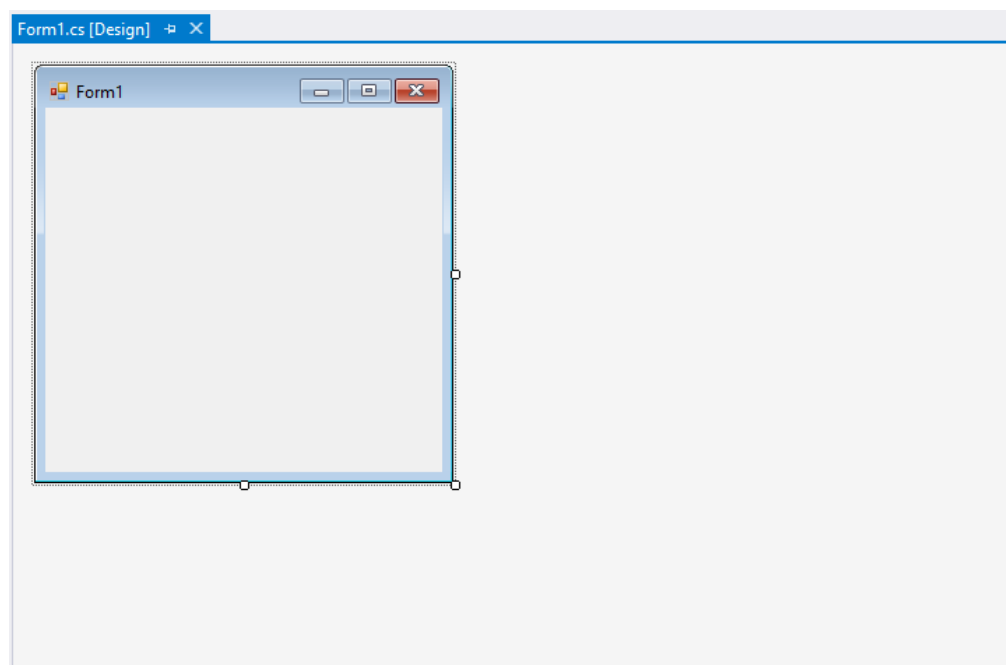
Slika 23 Okenska aplikacija

Na sliki je prikazan videz končnega izdelka.



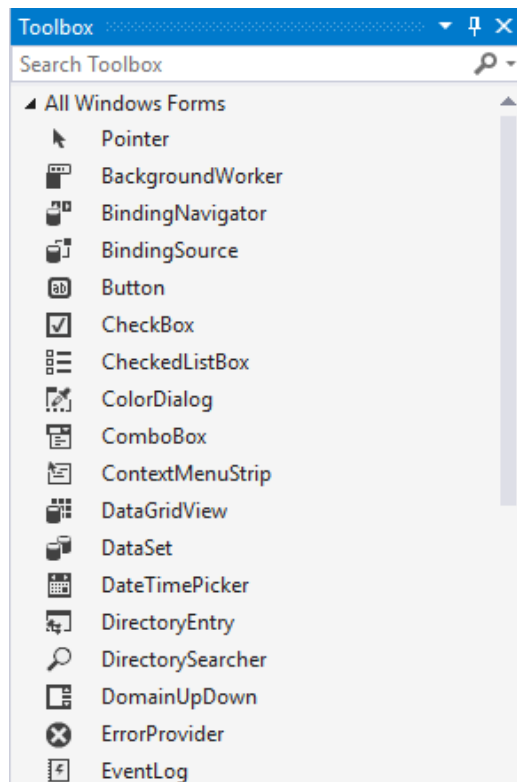
**Slika 24** Ustvarjanje okenske aplikacije

Spodaj lahko vidimo osnovno okno, ki še ne vsebuje nikakršnih elementov.

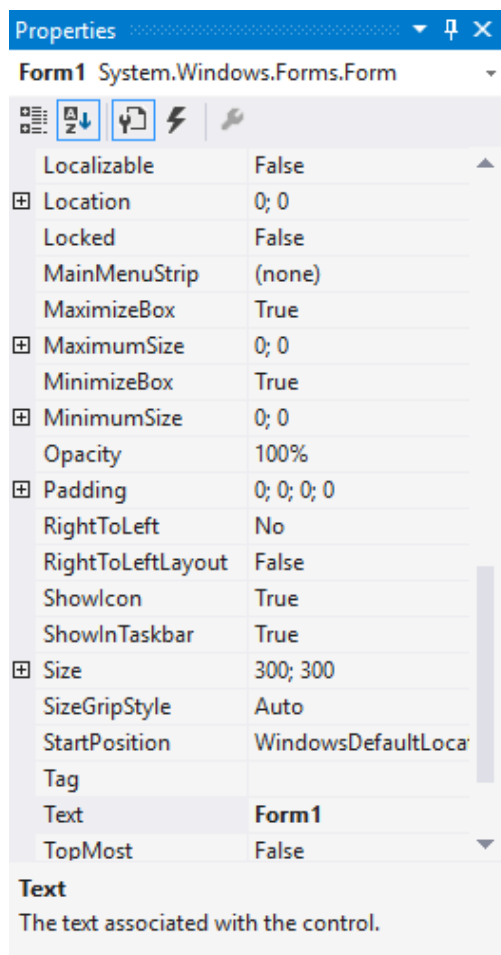


**Slika 25** Okno

Ob strani lahko vidite že odprto okno Toolbox. Uporabljali ga bomo za dodajanje grafičnih elementov programu.



Slika 26 Toolbox

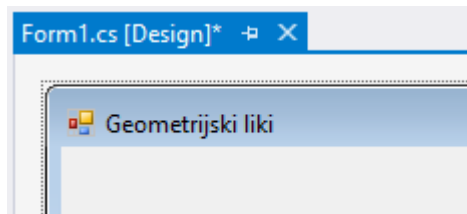


Slika 27 Propreties

Če kliknemo na properties pa vidimo lastnosti izbranih elementov. Tu lahko spreminjamo razne funkcije in videz elementov, ki jih lahko spreminjamo v tem grafičnem vmesniku ali preko kode.

Tag	
Text	Geometrijski liki
TopMost	False

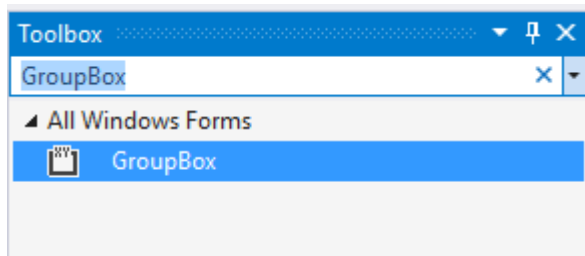
Okno bomo preimenovali preko lastnosti Text.



Da vidite spremembo pritisnite Enter ali kliknite izven vnosnega okenca.

MaximizeBox	False
MaximumSize	678; 345
MinimizeBox	True
MinimumSize	678; 345
Opacity	100%
Padding	0; 0; 0; 0
RightToLeft	No
RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
Size	678; 345

Velikost okna in drugih elementov je lahko poljubna. Te vrednosti smo vpisali zaradi željenega videza. Poleg spremembe velikosti okna smo onemogočili tudi MaximizeBox nastavev saj bi ta uničila izgled programa, ko bi uporabnik pritisnil na ta gumb.

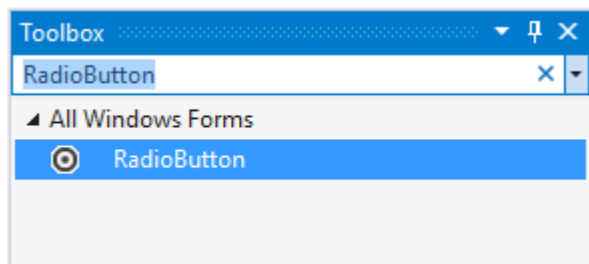


V Toolbox najdete GroupBox in ga potegnite na glavno okno programa. To je okvir za skupino elementov, ki jih bomo dodali naknadno.

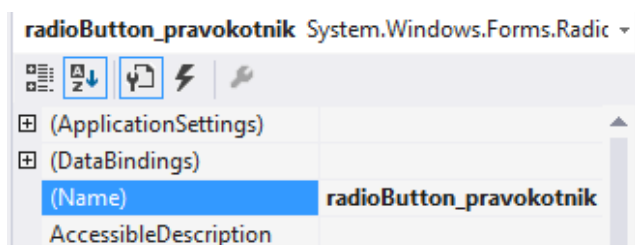
Size	415; 83
TabIndex	1
Tag	
Text	Geometrijski liki



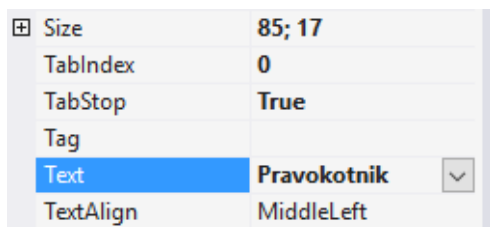
Na novo ustvajan GroupBox spremenimo v Properties.



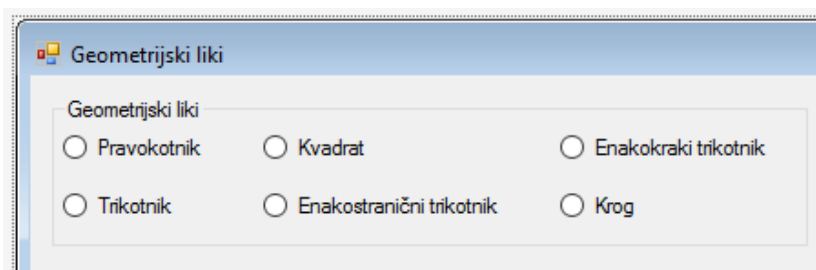
Po istem postopku bomo v GroupBox dodali RadioButton, ki je gumb z binarno vrednostjo (označen ali neoznačen).



Gumb preimenujemo tako, da bo njegovo ime prepoznavno kasneje, ko ga bomo uporabljali v kodi. Pri programiranju je pomembna organizacija in zato je pomembno, da spremenljivke imenujemo sistematsko.

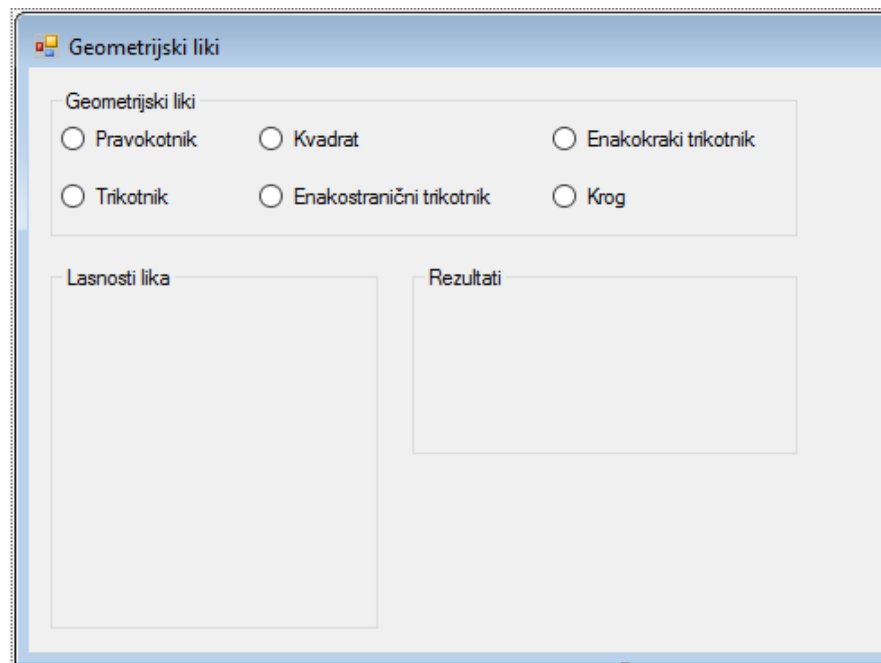


Ob gumbu se prikaže Text. Naši gumbi bodo poimenovani po likih, saj nam bodo omogočali izbiro lika katerega lastnosti si želimo izračunati.



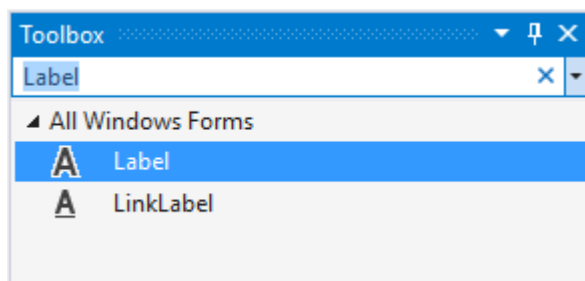
Slika 28 GroupBox in gumbi

Tako ustvarimo 6 gumbov za izbiro med liki.

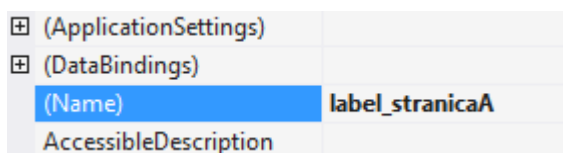


Slika 29 Trije GroupBox-i

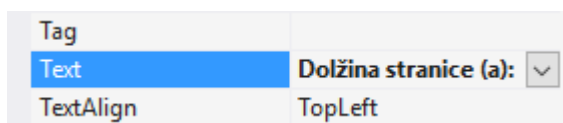
Dodali smo še 2 GroupBox-a. Njihove lokacije in oblike niso pomembne.



Iz Toolbox-a vzamemo element Label. Ta se vsebuje za prikaz besedila v programih. Služil nam bo kot oznaka številskih polj, saj želimo, da uporabnik ve v katero polje mora vnašati vrednosti.



Dodelimo mu prepoznavno ime.



Ob številskih poljih bomo napisali tudi oznake za vrednosti, ki se navadno uporabljajo pri matematiki npr. (a)

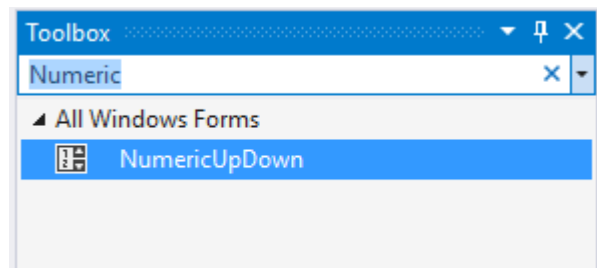
Lasnosti lika

Dolžina stranice (a):

Dolžina stranice (b):

Dolžina stranice (c):

Tu so oznake za naše 3 stranice.



Povedali smo da označujemo številska polja, ki jih bomo dodali sedaj. Ustvarite NumericUpDown element. Ta nam omogoča vnos spremenljivk ter spreminjanje vrednosti s kliki na gumba za večanje in manjšanje.

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	numericUpDown_stranicaA
AccessibleDescription	

DecimalPlaces	2
Dock	None
Enabled	False
⊞ Font	Microsoft Sans Serif; 8,25pt
ForeColor	■ WindowText
GenerateMember	True
Hexadecimal	False
ImeMode	NoControl
Increment	0,01
InterceptArrowKeys	True
⊞ Location	113; 19
Locked	False
⊞ Margin	3; 3; 3; 3
Maximum	1000

Elementu spremenimo lastnosti. Takšne nastavitve niso nujne in lahko jih spreminjamo kakor želimo.

Lasnosti lika

Dolžina stranice (a): 0,00

Dolžina stranice (b): 0,00

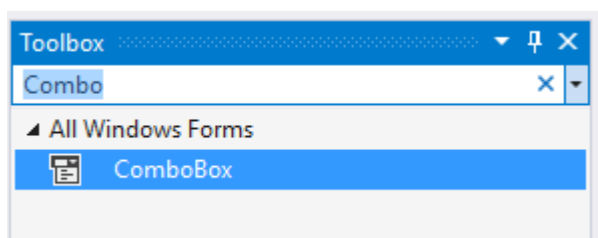
Dolžina stranice (c): 0,00

Dolžina stranice (d): 0,00

0,00

Polmer kroga (r): 0,00

Dodali smo še Polmer kroga in dva NumericUpDown elementa.



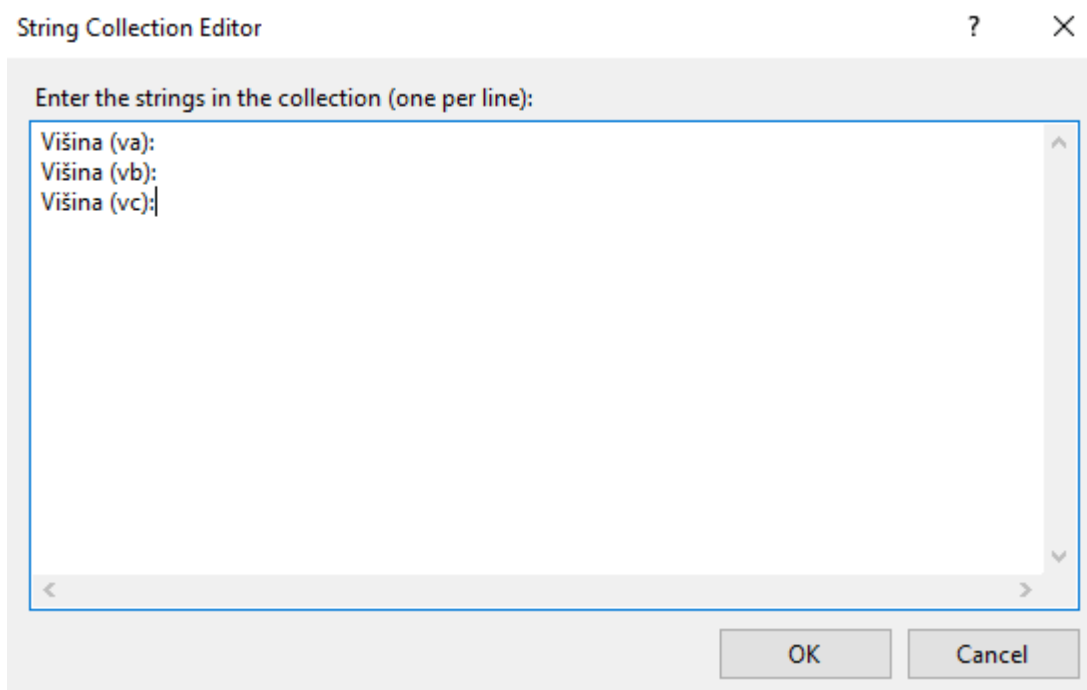
Kjer manjka Label vstavite ComboBox. To je okence, ki nam omogoča izbiro več različnih možnosti.

⊕ (ApplicationSettings)	
⊕ (DataBindings)	
(Name)	comboBox_visina
AccessibleDescription	

DrawMode	Normal
DropDownHeight	106
DropDownStyle	DropDownList
DropDownWidth	80

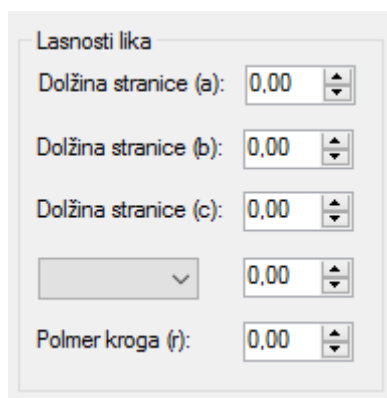
IntegralHeight	True
ItemHeight	13
Items	(Collection) ...

Spremenimo mu lastnosti in kliknemo na gumb označen s tremi pikami.



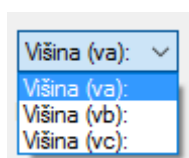
Slika 30 Dodajanje predmetov v ComboBox

Po prikazu zgornjega okna vnesemo tri vrednosti med katerimi bo lahko uporabnik izbiral. Ko preidemo v novo vrstico se naš zapis upošteva kot naslednja izbira.



Slika 31 Izbira lastnosti lika

Grafična podoba tega kosa okna naj bi izgledala tako.

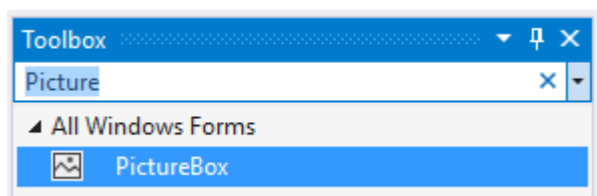


Slika 32 ComboBox

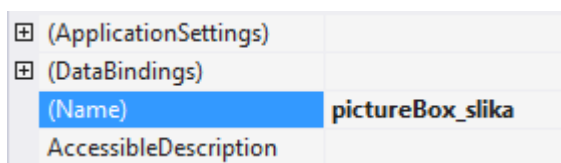
Ko program zaženemo bi mogel naš ComboBox imeti takšno funkcionalnost.

Slika 33 Rezultati

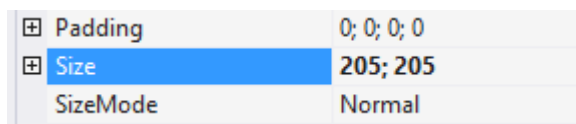
Enak postopek ponovimo za Obseg in Ploščino v Skupini Rezultati.



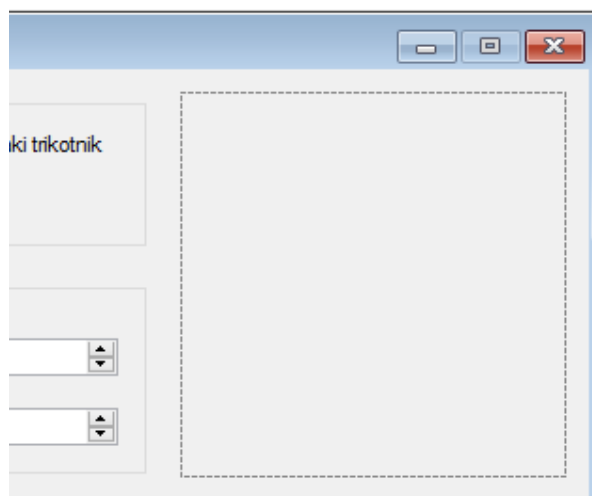
Zdaj bomo ustvarili okvir za slike. V okvir bomo vstavili skice likov.



Spremenimo ime okvirja.

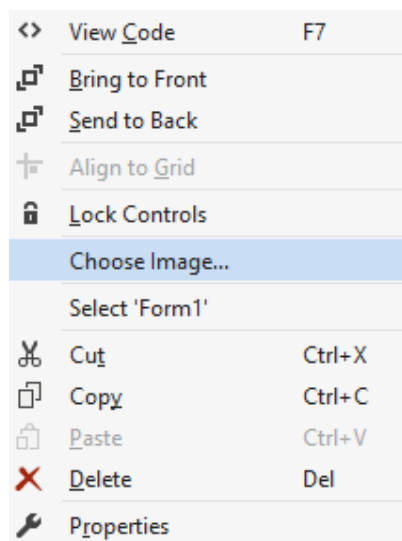


Določimo še velikost okvirja.



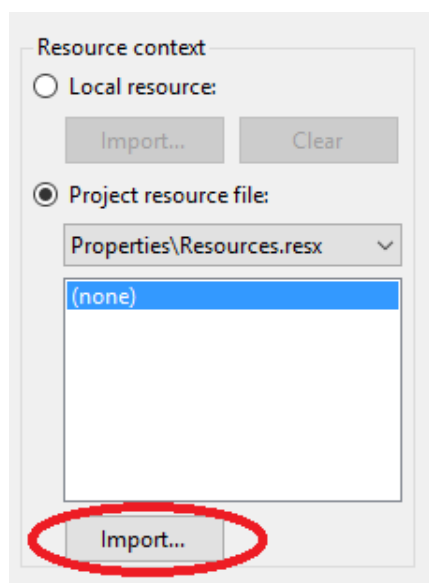
Slika 34 PictureBox

Zgoraj vidimo obliko našega okvirja.



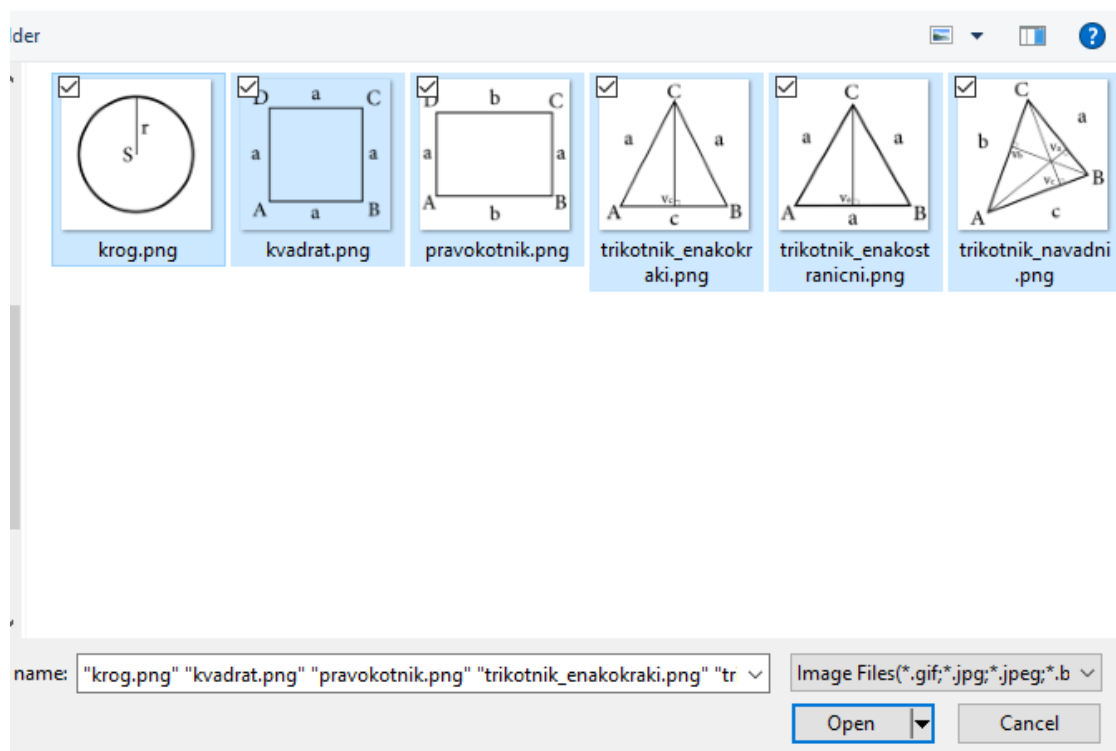
Slika 35 Dodajanje slik

Ob desnem kliku na okvir lahko izberemo slike.



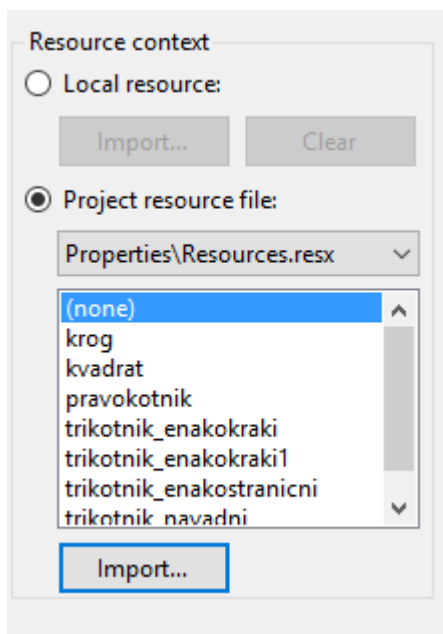
Slika 36 Uvažanje slik

Po prikazu okna kliknemo na Import, ki omogoča uvoz slik v program.



Slika 37 Izbira slik

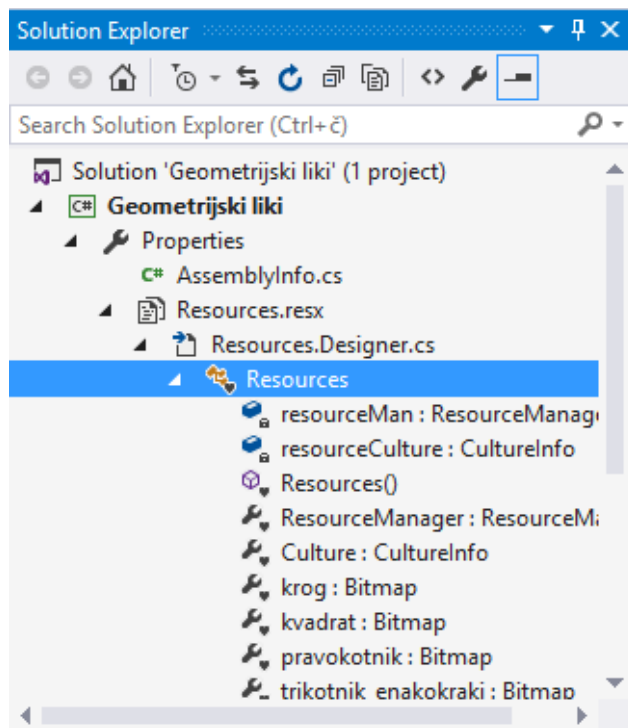
Izberemo lahko vse slike skupaj.



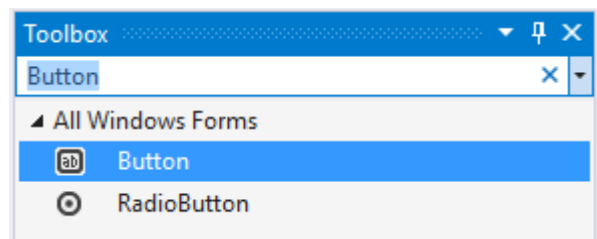
Slika 38 Uvožene slike

Po uvozu naše okno zgleda tako.

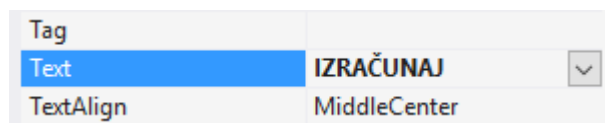




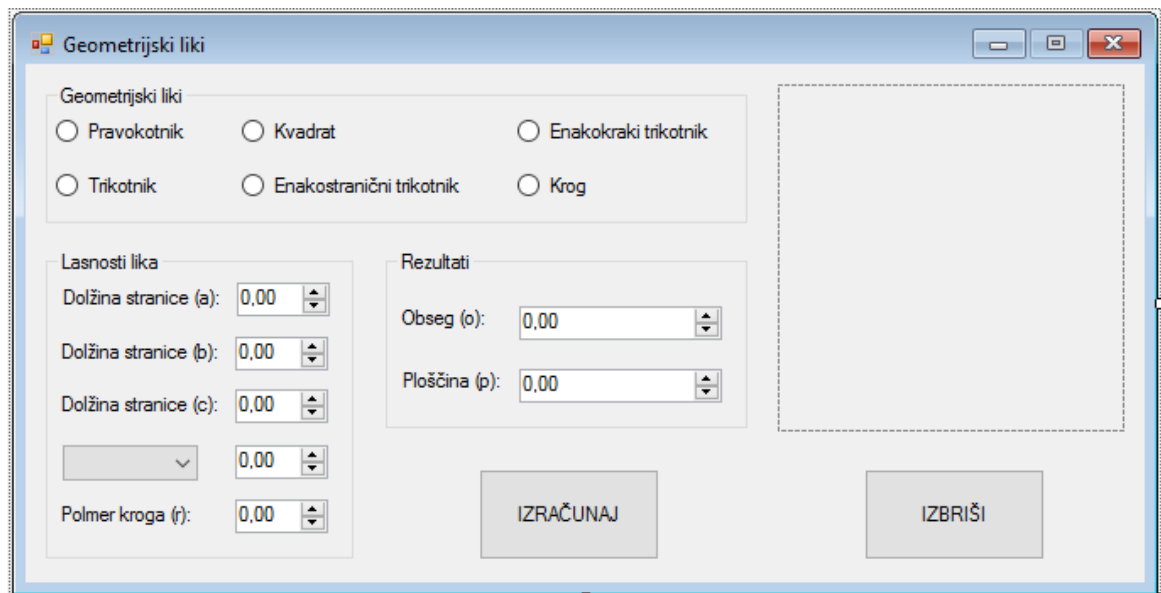
Uvožene slike lahko najdemo pod zavihkom Resources.



V Toolbox-u najdemo in ustvarimo Button.



Gumbu spremenimo nastavitve.



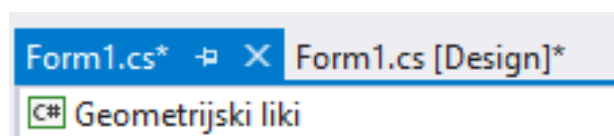
Slika 39 Končni izgled okenske aplikacije

Izgled končne razporeditve.

Grafični izgled našega programa je končan. Čas je, da začnemo s pisanjem kode, ki bo poskrbela za delovanje programa.

Začeli bomo s funkcijami, ki bodo ob izbiri geometrijskega lika prikazale ustrezno sliko v okvirju. Funkcija bo prepoznala pritisk na gumb za izbiro pravokotnika "radioButton\_pravokotnik" in prikazala skico pravokotnika v okvirju za prikazovanje slik na desni strani. Pri pisanju funkcij je pomembno, da imamo prepoznavna imena grafičnih elementov.

Funkcije lahko ustvarimo na dva načina. Osnovno funkcijo, ki bo preverila ali smo pritisnili na nekaj ali spremenili kakšno vrednost, lahko ustvarimo z dvojnim klikom na element v grafičnem pogledu programa.



```
private void radioButton_pravokotnik_CheckedChanged(object sender, EventArgs e)
{
}
}
```

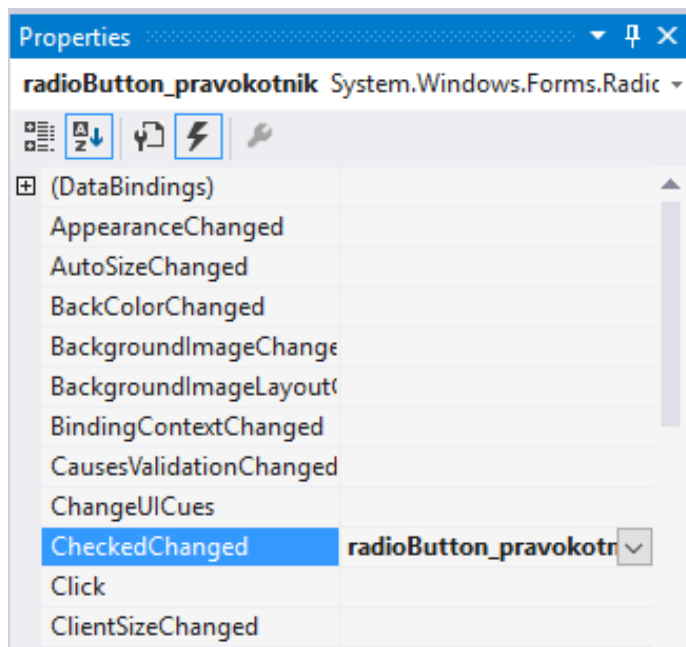
Odprl se nam bo nov zavihek s kodo grafične aplikacije in samodejno se nam bo ustvarila funkcija, ki bo preverila ali je element "radioButton\_pravokotnik" označen.

Kot smo že zgoraj omenili bo funkcija prikazala sliko pravokotnika, zato bomo funkciji dodali kodo prikazano na spodnji sliki.

```
private void radioButton_pravokotnik_CheckedChanged(object sender, EventArgs e)
{
    pictureBox_slika.Image = Geometrijski_lik.Properties.Resources.pravokotnik;
}
```

Sliko pravokotnika, ki smo jo uvozili določimo elementu za prikazovanje slik "pictureBox\_slika", da jo prikaže ob klicu funkcije.

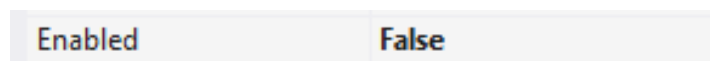
Če želimo funkcijo izbrisati, se moramo vrniti v zavihek z grafično aplikacijo in označiti element, kateremu funkcija, ki jo želimo izbrisati, pripada. V oknu "Properties" (lastnosti), kliknemo na gumb "Events", kjer je narisana strela.



Prikazale se nam bodo vse funkcije, ki jih označen element lahko kliče. Izberimo jo, tako da z desnim gumbom miške kliknemo nanjo in pritisnemo "Reset". Po teh korakih se bo funkcija samodejno izbrisala iz kode in ostalih datotek, zato lahko nanjo pozabimo.

V enakem oknu, kjer smo izbrisali funkcijo lahko ustvarimo tudi novo. To je drugi način ustvarjanja funkcije in nam omogoča kreiranje različnih funkcij.

Pri ustvarjanju elementov za vpis podatkov posameznega lika, smo v lastnostih nastavili, možnost "Enabled" na "False", kar pomeni, da po zagonu programa elementi ne bodo uporabni. To smo storili, ker bomo omogočili le tiste elemente, ki jih bo označen lik potreboval.



V zavihku s kodo bomo ustvarili dve novi funkciji, ki bosta poskrbeli, kateri elementi bodo omogočeni. Tokrat bomo funkcijo ustvarili na roke, ker ni povezana z grafičnimi elementi.

```
private void vklopiLastnostiLabel(bool a = false, bool b = false,
    bool c = false, bool v = false, bool r = false)
{
    label_stranicaA.Enabled = a;
    label_stranicaB.Enabled = b;
    label_stranicaC.Enabled = c;
    comboBox_visina.Enabled = v;
    label_polmer.Enabled = r;
}
```

Naša funkcija bo spreminjala lastnost "Enabled" elementov "Label", za prikaz preprostega besedila. V našem primeru ti elementi prikazujejo imena lastnosti (npr. "Dolžina stranice (a):").

Za parametre bomo deklarirali pet spremenljivk tipa **bool**, ki bodo imele brez dodajanja argumentov vrednost **false**. Pri klicanju funkcije bomo za argument uporabili vrednost **true** ali **false** in s tem povedali funkciji ali naj vklopi ali izklopi posamezen element.

```
vklopiLastnostiLabel(true);
```

Za boljše razumevanje bomo klicali funkcijo in jih podali prvi argument, ki bo imel vrednost **true**.

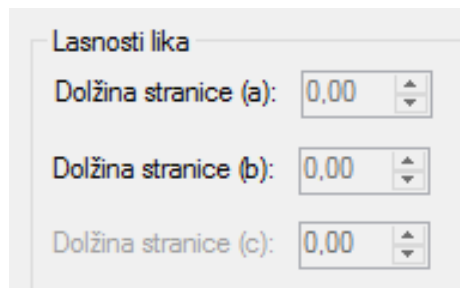
```
label_stranicaA.Enabled = a; // a = true
label_stranicaB.Enabled = b; // b = false
label_stranicaC.Enabled = c; // c = false
comboBox_visina.Enabled = v; // v = false
label_polmer.Enabled = r; // r = false
```

Tako bo izgledalo telo funkcije, če smo za prvi argument podali vrednost **true**. V tem primeru, bo omogočen le en element "label\_stranicaA".

Naša funkcija je končana, zato jo lahko kličemo v funkciji, ki preveri ali je element "radioButton\_pravokotnik" označen.

```
private void radioButton_pravokotnik_CheckedChanged(object sender, EventArgs e)
{
    vklopiLastnostiLabel(true, true);
    pictureBox_slika.Image = Geometrijski_liki.Properties.Resources.pravokotnik;
}
```

Pravokotnik ima stranici a in b, zato bomo podali vrednost **true** prvemu in drugemu argumentu funkcije.



Slika 40 Argumenti z vrednostjo »true«

Vklopiti moramo še "numericUpDown" elemente, zato bomo napisali še eno funkcijo, ki bo delovala na enak način, kot ta, ki smo jo nazadnje naredili.

```
private void vklopiLastnostiNumeric(bool A = false, bool B = false,
    bool C = false, bool V = false, bool R = false)
{
    numericUpDown_stranicaA.Enabled = A;
    numericUpDown_stranicaB.Enabled = B;
    numericUpDown_stranicaC.Enabled = C;
    numericUpDown_visina.Enabled = V;
    numericUpDown_polmer.Enabled = R;
}
```

Vsa struktura in spremenljivke so enake, kot pri prejšni funkciji s tem, da bomo pri tej spremenlili lastnosti "Enabled" elementov "numericUpDown".

```
private void radioButton_pravokotnik_CheckedChanged(object sender, EventArgs e)
{
    vklopiLastnostiLabel(true, true);
    vklopiLastnostiNumeric(true, true);
    pictureBox_slika.Image = Geometrijski_lik.Properties.Resources.pravokotnik;
}
```

Program poženemo, označimo pravokotnik in odkleniti bi se nam morale vse lastnosti izbranega lika.

Slika 41 Izbira lika

Funkcija, ki preverja stanje elementa `radioButton_pravokotnik` je končana, zato ponovimo postopek, za ostale like. Tokrat bomo naredili funkcijo, ki bo preverila ali je označen krog.

Dvakrat kliknemo na element, kjer označimo krog in ustvarila se nam bo funkcija. V njej kličemo funkcije, za vklop in izklop elementov za vpis in funkcijo, ki spremeni sliko v okvirju.

```
private void radioButton_krog_CheckedChanged(object sender, EventArgs e)
{
    vklopiLastnostiLabel(false, false, false, false, true);
    vklopiLastnostiNumeric(false, false, false, false, true);
    pictureBox_slika.Image = Geometrijski_lik.Properties.Resources.krog;
}
```

Enak postopek ponovimo še za vse ostale like in s tem bomo pridobili boljšo preglednost pri računanju z liki.

Program bo ob pritisku na gumb izračunal obseg in ploščina izbranega lika. Z dvojnim klikom na gumb za izračun, se nam bo kreirala funkcija, ki bo preverila ali je bil gumb pritisnjen. Izračunati morami šest likov, zato lahko uporabimo **if** in **else if stavke**. V primeru, da bi imeli več likov, bi morali uporabiti **stavek switch**, saj prekomerno gnezdenje **if** in **else stavkov** ni priporočeno zaradi počasnejšega delovanja programa. Za shranjevanje vrednosti elementov "numericUpDown" potrebujemo spremenljivke tipa **decimal**.

```

private void button1_Click(object sender, EventArgs e)
{

    if(radioButton_kvadrat.Checked) // Računanje kvadrata
    {
        decimal a = numericUpDown_stranicaA.Value;

        decimal obseg = 4 * a;
        decimal ploscina = a * a;

        numericUpDown_obseg.Value = obseg;
        numericUpDown_ploscina.Value = ploscina;
    }

```

Na zgornji sliki, v jedru **if stavka**, je prikazana koda za izračun kvadrata. V spremenljivko "a" smo shranili vrednost stranice a. Pri računanju kvadrata je to edini podatek, ki ga potrebujemo. Ustvarili smo še spremenljivki "obseg" in "ploscina", kjer smo pri računanju uporabili stranico a. Po izračunu smo spremenili vrednosti elementov "numericUpDown", ki izpišeta obseg in ploščino lika.

Enako je potrebno storiti za ostale like. Za vsak lik je potrebno shraniti podatke, ki jih potrebujemo za izračun in nato pravilno uporabiti v matematični formuli.

```

else if(radioButton_pravokotnik.Checked) // Računanje pravokotnika
{
    decimal a = numericUpDown_stranicaA.Value;
    decimal b = numericUpDown_stranicaB.Value;

    if(a == b)
    {
        pictureBox_slika.Image = Geometrijski_lik.Properties.Resources.kvadrat;
    }
    else
    {
        pictureBox_slika.Image = Geometrijski_lik.Properties.Resources.pravokotnik;
    }

    decimal obseg = (2 * a) + (2 * b);
    decimal ploscina = a * b;
    numericUpDown_obseg.Value = obseg;
    numericUpDown_ploscina.Value = ploscina;
}

```

Pri računanju pravokotnika, smo dodali še poseben pogoj, ki preveri ali sta stranici a in b enaki. V primeru, da sta spremenimo sliko pravokotnika v kvadrat, saj ima kvadrat vse stranice enake. Enako, kot pri kvadratu izračunamo ploščino in obseg ter na koncu izračune enačimo z vrednostjo "numericUpDown" elementi, ki izpišejo rezultate.

```

else if(radioButton_trikotnik.Checked) // Računanje navadnega trikotnika
{
    decimal a = numericUpDown_stranicaA.Value;
    decimal b = numericUpDown_stranicaB.Value;
    decimal c = numericUpDown_stranicaC.Value;
    decimal v = numericUpDown_visina.Value;

    decimal obseg = a + b + c;
    decimal ploscina = 0;

    switch(comboBox_visina.Text) // Preveri katera visina je izbrana
    {
        case "Višina (va)":
            ploscina = (a * v)/2;
            break;
        case "Višina (vb)":
            ploscina = (b * v)/2;
            break;
        case "Višina (vc)":
            ploscina = (c * v)/2;
            break;
    }

    numericUpDown_obseg.Value = obseg;
    numericUpDown_ploscina.Value = ploscina;
}

```

Pri računanju navadnega trikotnika je potrebno napisati nekaj več kode. Tokrat bomo potrebovali štiri podatke. Obseg lahko izračunamo že na začetku, ploščino pa pod pogoji. Zaradi boljše preglednosti smo za izračun ploščine uporabili **stavek switch**, s tremi preverjanji.



```

else if (radioButton_enakokraki.Checked) // Računanje enakokrakega trikotnika
{
    decimal a = numericUpDown_stranicaA.Value;
    decimal c = numericUpDown_stranicaC.Value;
    decimal v = numericUpDown_visina.Value;

    decimal obseg = (2 * a) + c;
    decimal ploscina = 0;

    if (comboBox_visina.Text == "Višina (vc):")
    {
        if (a > 0 && c > 0)
        {
            decimal c2 = c / 2;
            ploscina = 2 * (a * c2 / 2);
        }
        else if (a > 0 && v > 0 && c == 0 && v < a)
        {
            decimal c2 = (decimal)Math.Sqrt((double)(a * a) - (double)(v * v));
            ploscina = 2 * (c2 * a / 2);
        }

        else if (a > 0 && v > 0 && c == 0 && v > a)
        {
            MessageBox.Show("Enakokraki trikotnik: višina ne more biti večja od kraka.");
        }
        else if (c > 0 && v > 0 && a == 0)
        {
            decimal a2 = (c / 2) * (c / 2) + v * v;
            ploscina = 2 * (a2 * (c/2) / 2);
        }

        numericUpDown_ploscina.Value = ploscina;
        numericUpDown_obseg.Value = obseg;
    }
}

```

Ploščino enakokrakega trikotnika smo izračunali na tri različne načine. Če uporabnik vnese ustrezne podatke je mogoče uporabiti različne formule.

```

else if (radioButton_krog.Checked)
{
    decimal r = numericUpDown_polmer.Value;

    decimal obseg = 2 * (decimal)Math.PI * r;
    decimal ploscina = (decimal)Math.PI * (r * r);

    numericUpDown_obseg.Value = obseg;
    numericUpDown_ploscina.Value = ploscina;

}

```

Za računanje kroga smo uporabili matematično funkcijo **Math.PI**, ki predstavlja vrednost pi (3.14159). Za normalen izpis v element "numericUpDown" smo morali pi pretvoriti v tip **decimal**.

Napisati moramo še zadnjo funkcijo za gumb "IZBRIŠI". Dvakrat kliknemo na element, da se nam ustvari funkcija in nato v notranjosti spremenimo vse vrednosti elementov za izpis "numericUpDown" na 0.

```

private void button2_Click(object sender, EventArgs e)
{
    numericUpDown_stranicaA.Value = 0;
    numericUpDown_stranicaB.Value = 0;
    numericUpDown_stranicaC.Value = 0;
    numericUpDown_visina.Value = 0;
    numericUpDown_polmer.Value = 0;
    numericUpDown_obseg.Value = 0;
    numericUpDown_ploscina.Value = 0;
}

```

S tem korakom je naša okenska aplikacija končana.

## **POROČILO OB ZAKLJUČKU PROJEKTA**

Projekt je končan uspešno. Izpolnila sva vse načrte v predvidenem času in obliki. K uspehu projekta je pripomoglo najino že obstoječe znanje programiranja, ki sva ga dobila v šoli in tudi sama. Prednosti skupine ležijo predvsem v zanimanju za programiranje in delavnosti. Pomankljivost pa lahko opazimo pri razporeditvi časa. Kljub temu, da nisva vseh ciljev izpolnila ob času sva pravočasno oddala končni izdelek.

Dobila sva izkušnje z organizacijo projektov, ki zahtevajo dolgo časa za izvršitev in se naučila vsega kar sva napisala v dokument ter še kakšno neomenjeno podrobnost več. Oba sva dobila veliko izkušenj in znanja ob ustvarjanju končnega izdelka. Razdelila sva si delo glede na prednosti in slabosti posameznika. Delo v projektni skupini je potekalo dobro. Problemi so bili le pri delih kjer je bilo potrebno skupno delo, saj je bilo težko najti čas v katerem sva bila oba prosta za delo. Kljub temu sva dobro sodelovala predvsem zaradi želje po znanju in čim boljšem končnem izdelku. Priročnik je odlično učno sredstvo in ob pisanju sva se tudi sama pridobila ogromno znanja na področju programiranja.

# LITERATURA IN VIR

Joe Mayo, C# Succinctly. Dostopno na  
<http://www.csharp-station.com/> 5.5.2016

Davor Bonačič, Kratek priročnik jezika C# in razlike z jezikom C++. Dostopno na  
[http://osebje.informatika.uni-mb.si/davor/predmeti/orodja\\_za\\_razvoj\\_aplikacij\\_in\\_vs3/CSharp/CSharp\\_zbrano\\_gradivo.pdf](http://osebje.informatika.uni-mb.si/davor/predmeti/orodja_za_razvoj_aplikacij_in_vs3/CSharp/CSharp_zbrano_gradivo.pdf) 5.5.2016

FERI Jure, EPF Nina, E-računalništvo - Objektno programiranje. Dostopno na  
<http://gradiva.txt.si/index.php/objektno-programiranje-2/> 5.5.2016

Microsoft, Overview of the .NET Framework. Dostopno na  
<https://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx> 5.5.2016

Matija Lokar, Objektno programiranje. Dostopno na  
<http://www.nauk.si/materials/1327/out/#state=1> 5.5.2016

Microsoft, Anders Hejlsberg 2010 Career Achievement. Dostopno na  
<https://www.microsoft.com/about/technicalrecognition/anders-hejlsberg.aspx> 5.5.2016

Microsoft, C# Operators. Dostopno na  
<https://msdn.microsoft.com/en-us/library/6a71f45d.aspx> 5.5.2016

Microsoft, C# Keywords. Dostopno na  
<https://msdn.microsoft.com/en-us/library/x53a06bb.aspx> 5.5.2016

Microsoft, C#. Dostopno na  
<https://msdn.microsoft.com/en-us/library/kx37x362.aspx> 5.5.2016