Univerza *v Ljubljani*
Fakulteta *za gradbeništvo in geodezijo*

Visual Basic vaje za gradbenike

Vlado Stankovski

Ljubljana, september 2019

Univerza
v Ljubljani

Fakulteta
za gradbeništvo
in geodezijo

Vlado Stankovski

Visual Basic
vaje za gradbenike

1. elektronska izdaja

Ljubljana, september 2019

Vlado Stankovski

Visual Basic
vaje za gradbenike

Univerzitetni učbenik

1. izdaja

Recenzenta:
Boštjan Brank
Matija Marolt

Lektorica:
Polona Štefanič

Oblikovalec naslovnice:
Janez Brežnik

Ilustracije:
Jan Mak Bevcl

Izdala in založila

Fakulteta za gradbeništvo in geodezijo,
Jamova 2, Ljubljana

Naklada: 50 izvodov
Publikacija je brezplačna. Publikacija je dostopna tudi v elektronski obliki.

Ljubljana, 2019

CIP – Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana
xxx.x(xxx.x)
STANKOVSKI, Vlado
Visual Basic vaje za gradbenike : [univerzitetni učbenik] / Vlado Stankovski. – 1. izd. – Ljubljana: Fakulteta za gradbeništvo in geodezijo, 2019
ISBN xxx-xxx-xxxx-xx-x (pdf)
1. Stankovski, Vlado
COBISS.SI-ID=xxxxxxxxx
ISBN 978-961-6884-66-2 (epub)
1. Stankovski, Vlado
COBISS.SI-ID=301905408

Predgovor

Kratka zgodovina računalništva je zelo bogata, vendar lahko rečemo, da je bilo le nekaj prelomnih trenutkov. V 19. stoletju, v času industrijske revolucije, so se ukvarjali z razvojem najrazličnejših strojev. Charles Babbage je najprej leta 1922 izdelal Diferenčni stroj, nato pa okrog leta 1837 načrt Analitskega stroja. Analitski stroj je bil utemeljen na nekaj pomembnih konceptih, ki označujejo začetek računalništva: aritmetično-logične enote, nadzora pretoka v obliki pogojnih stavkov in zank ter integriranega spomina.

Hitro zatem je pomembnost Babbageovega odkritja prepoznala baronica Ada Lovelace, nečakinja znamenitega Lorda Gordona Byrona. Ugotovila je, da lahko tak stroj izvaja ukaze, zapisane v določenem zaporedju. Krajši premislek nam da vedeti, da je Charles Babbage duhovni oče računalnika, baronica Ada Lovelace pa prva programerka.

Naslednja prelomnica pripada Johnu von Neumannu, ki je uresničil ideje svojih predhodnikov in izdelal prvi računalnik. Od takrat dalje postajajo računalniki hitrejši in zmoglivejši, vendar so po konceptualni zasnovi enaki. Zgodovina računalništva nas torej uči, da dobro zasnovan idejen koncept lahko pripelje do revolucije, tokrat informacijske.

Druga pomembna trenutka v razvoju računalništva sta vsekakor razvoj Interneta ter Svetovni splet. Pri razvoju računalništva je prispevalo in še vedno prispeva nešteto inženirjev in znanstvenikov. Omenimo Alana Turinga, ki je zastavil teorijo izračunljivosti, Noama Chomskega, ki je razvil teorijo računalniških jezikov ter Sira Tima Bernersa Leeja, ki je izumil Svetovni Splet.

Dandanes smo priča vse večji povezljivosti podatkov in programskih storitev, dostopnih prek Interneta, programska oprema pa postaja vse bolj neodvisna od naprav, na katerih se izvaja. V bližnji prihodnosti bodo naši programi prek Interneta upravljali najbolj različne naprave in senzorje.

Dandanes obstaja več sto računalniških jezikov. Vsaj enega se moramo naučiti, da bi lahko napisali računalniku razumljiv program. Pred tabo je knjiga, ki je namenjena študentom gradbeništva in predstavlja osnovi tečaj programiranja v programskem jeziku Visual Basic. Ta jezik je bil izbran zato, ker je splošno namenski in kot pove tudi samo ime, dokaj osnoven. Lahko je dobro izhodišče za učenje drugih programskih jezikov.

Uspešno na poti prve programerke, Ade Lovelace!

Vlado Stankovski

Ljubljana, september 2019

Kazalo

1 Prvi program

Da se naučimo programirati in izdelovati lastne aplikacije ne potrebujemo veliko. Zadošča svinčnik in list papirja na katerega zapišemo program. Vsekakor pa je veliko bolj zabavno, če imamo pred sabo računalnik in ustrezno razvojno okolje. Razvojno okolje nam omogoča različne pomožne funkcije, kot so na primer: odkrivanje sintaktičnih napak pri pisanju programov, svetovanje pri izbiri različnih funkcij, možnosti za timski razvoj aplikacij, možnost uporabe grafične orodjarne pri razvoju okenskih programov in tako naprej. Za namene tega predmeta bomo uporabljali urejevalnik programov Visual Studio, ki ga lahko z zastonjsko študentsko licenco prenesemo na svoj računalnik. Visual Studio omogoča urejanje programskih jezikov, kot so: C#, F# in tudi jezik, ki ga bomo uporabljali mi - Visual Basic.

Preden začnemo izdelovati prve programe je pomembno, da vemo, kaj je to program in na kakšen način se ga izvede. Program je zaporedje ukazov, ki se izvedejo v predpisanem vrstnem redu. Ukazi lahko uporabljajo spremenljivke ter tako berejo in pišejo podatke v spomin računalnika.

Računalnik je sestavljen iz centralno-procesne enote, notranjega pomnilnika ter sistema vhodno-izhodnih naprav, kot so miška, tipkovnica, zaslon, tiskalnik in podobno. Sama centralno-procesna enota je sestavljena iz aritmetično-logične enote, krmilne enote in registrov. V aritmetično-logični enoti so fizično, z uporabo vezij, realizirani različni ukazi, kot je ukaz za seštevanje dveh števil. Krmilna enota skrbi za pretok podatkov in ukazov po vodilih iz pomnilnika v procesor, registri pa so pomnilniške celice, ki služijo za začasno hranjenje in obdelavo podatkov in njihovih naslovov v pomnilniku. Različni procesorji podpirajo različne nabore takšnih ukazov, ki jih imenujemo strojni ukazi, izdelane programe pa strojni programi. Programiranje strojnih programov je zelo zamudno delo, zato so v zadnjih desetletjih razvili višjenivojske programske jezike, ki so bližje naravnim jezikom. Med takšne jezike spada tudi programski jezik Visual Basic, katerega osnove se bomo naučili tekom predmeta Računalništvo in informatika.

Pri višjenivojskih programskih jezikih obstajata dva načina izvajanja: izvajanje z interpretacijo in izvajanje s predhodnim prevajanjem v izvršljive programe. V prvem načinu se program postopoma prevaja v strojno kodo in hkrati tudi izvaja. Za ta način je potreben poseben program, ki mu pravimo interpreter. Jezik z možnostjo interpretacije je jezik javascript. Program, ki omogoča njegovo interpretacijo pa se imenuje Java Virtual Machine, in je vključen v spletne brskalnike ter druga orodja. V drugem načinu je program oz. prevajalnik vključen v razvojno okolje. Takšen jezik je Visual Basic, ki zahteva predhodno prevajanje. To pomeni, da moramo, ko je program napisan, najprej pognati prevajalnik, ki generira izvršljivi program, ki ga lahko nato kadarkoli izvajamo.

Kot vidimo, je računalniški program postopek, ki je zapisan v programskem jeziku in je namenjen izvajanju na računalniku. Računalniški programi so v osnovi namenjeni pohitritvi daljših izračunov ali pohitritvi dostopa do informacij. Tako kot vsi jeziki, se tudi programski jeziki razlikujejo po besedišču, sintaksi in izraznih možnostih. V preteklosti so morali programerji pisati programe v

strojnem jeziku, ki je razumljiv računalniku, programer pa hitro naleti na problem razumevanja tega jezika. Z razvojem računalništva so razvili takoimenovane prevajalnike. To so uporabniška okolja, ki avtomatsko prevedejo program, ki je napisan v programskem jeziku, v strojni jezik, ki je razumljiv procesorju. Tak jezik je Visual Basic, ki je bil zasnovan na način, da bi bil razumljiv vsem, ki se lotijo programiranja.

Omeniti velja bistveno razliko med prevajanjem in interpretacijo. Če smo izdelani program prevedli na svojem osebnem računalniku, ki teče recimo na platformi z operacijskim sistemom Windows, to sploh ne pomeni, da bo izvršljivi program deloval tudi na računalniku, na katerega je nameščen kateri drug operacijski sistem. Da se lahko tak program izvaja na različnih platformah, mora biti ustrezno preveden za točno določeno platformo. Programe, ki jih izvajamo z interpretacijo so bistveno bolj prenosljivi. Program se namreč lahko izvede na katerikoli platformi, v kolikor zanjo obstaja ustrezen interpreter.

Pri predmetu Računalništvo in informatika se bomo tako naučili nekaj najbolj osnovnih ukazov. Zanima nas, kako izpišemo poljubno besedilo na zaslonu, kako shranimo podatke, ki jih poda uporabnik, ter kako izvajamo različne računske operacije ter vrnemo rezultate uporabniku na vpogled. Za izvajanje takšnih operacij moramo spoznati nekaj osnovnih ukazov in sintaktičnih pravil jezika Visual Basic. Pa poglejmo po vrsti.

1.1 Hello World!

Na področju računalništva se je uveljavila tradicija, ko začnemo razvijati aplikacije v novem programskem jeziku ali v novem razvojnem okolju, najprej napišemo aplikacijo "Hello World!". Namen zapisa te aplikacije je dobiti občutek, kako deluje razvojno okolje in koncept samega programskega jezika. Razvoj te preproste aplikacije poteka po naslednjih korakih.

Najprej moramo zagnati razvojno okolje, ki je v našem primeru Visual Studio 2010, nato pa izberemo programski jezik Visual Basic. Razvojno okolje omogoča uporabo nekaj predlog za razvoj preprostih konzolnih aplikacij, naprednejše okenske aplikacije, aplikacije za delo s podatkovnimi bazami in tako dalje. Izberemo podlogo za konzolni program Console Application. Razvojno okolje nato od nas zahteva, da določimo pot, kam se program shrani. Programsko okolje na izbranem mestu ustvari celo vrsto direktorijev in datotek, ki so generirani v točno določenih mapah. Vrhnja mapa ima ime našega programa oziroma projekta. Torej, če se kdaj pozneje odločimo program kopirati in ga prenesti na druge medije, ne zadošča samo datoteka v kateri je besedilo programa, temveč moramo kopirati vse ostale spremljajoče datoteke. Sedaj lahko začnemo z zapisom programske kode v urejevalniku.

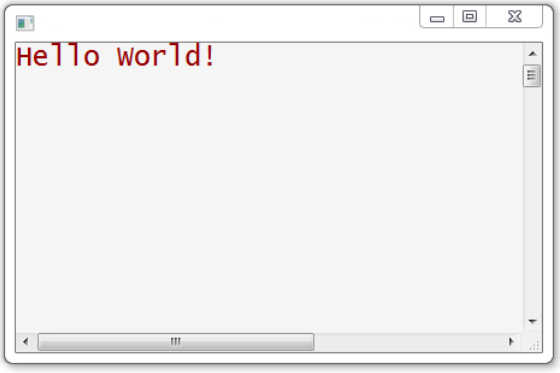
Hello World!

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World!")
        Console.ReadKey()
    End Sub

End Module
```

Potrebno je vedeti, da ima vsak programski stavek določen učinek in da se programski stavki izvajajo v določenem vrstnem redu. Prvi programski stavek `Console.WriteLine("Hello World!)`, nalaga konzolnemu oknu, da izpiše besedilo "Hello World!". Drug programski stavek `ReadKey()` ima funkcijo, da začasno ustavi izvajanje programa v konzolnem oknu in počaka, da uporabnik pritisne katerokoli tipko na tipkovnici. Ko se to zgodi, program nadaljuje z izvajanjem. V primeru, da ni nadaljnih ukazov, se program zaključi.

Da se izdelani program izvede, mora programsko orodje Visual Studio najprej prevesti programske stavke iz programskega jezika Visual Basic v zaporedja strojnih ukazov, ki so razumljivi procesorju, saj lahko le tako izvede zahtevano nalogo. S klikom na gumb `Debug|Start Debugging` se zažene prevajalnik, ki je vgrajen v razvojno okolje. V tem postopku računalnik ustvari še nekaj dodatnih datotek, ki kasneje omogočajo izvajanje programa. Če pogledamo v mapo projekta, najdemo tudi mapo bin, kjer Visual Studio shrani tudi izvršljivo kodo programa v datoteko. Ta datoteka ima enako ime kot začetna mapa programa, na primer: `Program.exe`. Torej, glavna razlika med programom v programskem jeziku Visual Basic in izvršljivim programom je v tem, da je prvi razumljiv programerju, drug pa je namenjen izključno računalniškemu procesorju.

Tako smo izdelali naš prvi program in v primeru, da nismo naredili kakšne napake bo program v konzolnem oknu izpisal pozdrav "Hello World!".



Slika 1.1: Prvi program Hello World!

Na začetku si oglejmo natančneje podlogo za konzolne programe.

```
Module Module1

    Sub Main()
        'Tukaj napišemo program
    End Sub

End Module
```

Vidimo, da vsebuje zaenkrat nerazumljive besede, na primer `Module` in `Sub`, ki sta rezervirani besedi iz programskega jezika Visual Basic. Beseda `Module` označuje začetek programskega modula poimenovanega `Module1`. Sintaktično pravilo programskega jezika Visual Basic določa, da se vsak programski modul mora zaključiti z ukazom `End Module`. Če pri pisanju programa ne bi upoštevali vseh sintaktičnih pravil, na primer, če bi izpustili ime podprograma, bo prevajalnik med prevajanjem programa v strojno kodo javil napako. Torej, vsak programski jezik ima točno določeno besedišče, ki ga moramo upoštevati. Če uporabimo napačno besedo, bo prevajalnik javil napako med prevajanjem programa.

Programski modul je lahko sestavljen iz več podprogramov (procedur ali funkcij), ki se med izvajanjem programa lahko medsebojno kličejo. Vsak podprogram mora imeti unikatno ime znotraj modula. V našem primeru imamo le eden podprogram, ki je označen z rezervirano besedo `Sub`. Imena procedur in funkcij lahko izbiramo sami, vendar je le ena lahko glavna in se začne izvajati prva. Če si ogledamo program "Hello World!", vidimo, da ima samo eno proceduro in to je glavna procedura `Main()`. V podprograme se bomo poglobili kasneje.

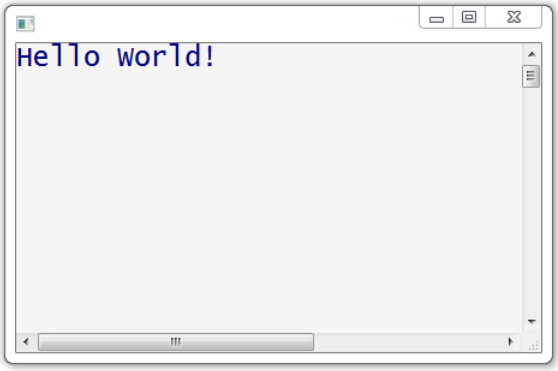
1.2 Pozdrav v drugi barvi

Poskušajmo dodelati izdelani program tako, da spremenimo barvo konzolnega izpisa. To izvedemo tako, da dodamo nov programski stavek, ki se glasi `Console.ForegroundColor = ConsoleColor.DarkBlue`. Najbolje, da ga dodamo kar na začetek programa.

Pozdrav v drugi barvi

```
Module Module1

    Sub Main()
        Console.ForegroundColor = ConsoleColor.DarkBlue

        Console.WriteLine("Hello World!")
        Console.ReadKey()
    End Sub

End Module
```

Vsebina izpisa je enaka kot v prejšnji nalogi, spremenjena je edino barva znakov.



Slika 1.2: Pozdrav v drugi barvi

`Console` je primer okna, prek katerega aplikacija komunicira z uporabnikom - izpisuje besedilo ali omogoča vnos podatkov za računanje. Okno `Console`, ima tako kot drugi predmeti, dve pomembni značilnosti:

- lastnosti - podatki, ki jih uporablja program pri svojem delovanju, kot so barva in velikost konzolnega okna;
- operacije - bodisi procedure ali funkcije, ki jih omogoča predmet. To je lahko procedura `WriteLine()`, ki omogoča izpis besedila na zaslonu, funkcija `ReadKey()`, ki počaka, da uporabnik pritisne katerokoli tipko za nadaljevanje izvajanja programa.

Do vseh lastnosti in operacij, ki jih podpira konzolno okno dostopamo tako, da napišemo piko za imenom predmeta. Razvojno okolje nam našteje vse možne lastnosti in operacije, ki jih omogoča predmet z imenom `Console`.

Lastnost `ConsoleColor.DarkBlue` predstavlja podatek o barvi pisave, ki se izpiše v konzolnem oknu. Ta podatek je že definiran z začetno nastavitvijo, vendar ga lahko sami spremenimo tako, da mu

dodelimo novo vrednost. Procedura poimenovana poimenovana `WriteLine()` nam omogoča izpis poljubnega besedila v konzolni aplikaciji. Uporablja se tako, da za argument ukaza `WriteLine()`, v navednicah napišemo besedilo, ki naj bo izpisano na zaslonu. Celoten ukaz za izpis besedila je tako `Console.WriteLine("Besedilo")`. Pika za imenom razreda `Console` pomeni, da dostopamo do njegovih podprogramov.

Procedura `ReadKey()` nima nobenih argumentov. Njen namen je, da začasno ustavi izvajanje programa in počaka na to, da uporabnik pritisne katerokoli tipko na tipkovnici. Ko se to zgodi, program nadaljuje z izvajanjem in ker ni drugih ukazov, se zaključi.

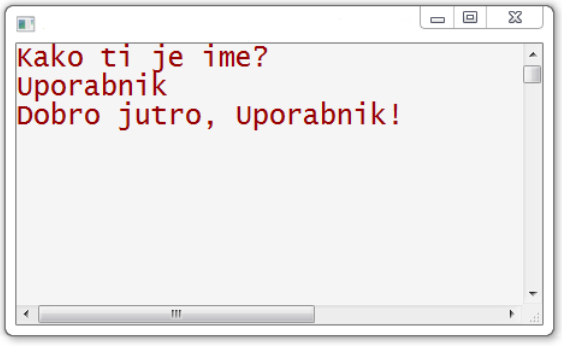
1.3 Dobro jutro, Uporabnik!

V naslednji različici programa "Hello World!"vključimo še nekaj novosti, da lahko program komunicira z uporabnikom. Najprej program vpraša po imenu uporabnika, nato pa ga osebno pozdravi.

Dobro jutro, Uporabnik!

```
Module Module1

    Sub Main()
        Dim ime As String ' Ime osebe
        Console.WriteLine("Kako ti je ime?")
        ime = Console.ReadLine
        Console.WriteLine("Dobro jutro, " & ime & "!")
        Console.ReadKey()
    End Sub

End Module
```

Ob zagonu programa odgovorimo na vprašanje, nato nas program pozdravi, kot je prikazano na sliki 1.3



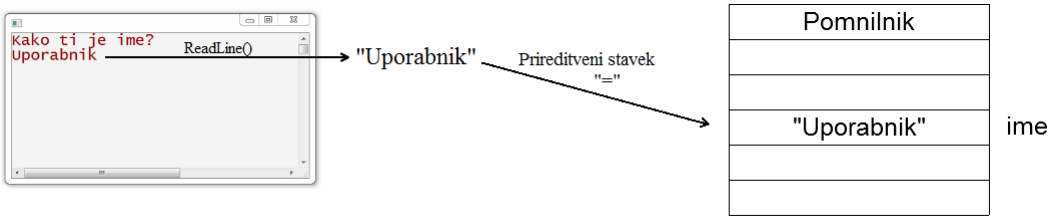
Slika 1.3: Dobro jutro, uporabnik!

Naš novi program uporablja funkcijo `ReadLine()`, ki jo omogoča predmet `Console`. Funkcija `ReadLine()` bere črke, posebne znake ali števila, ki jih uporabnik tipka na tipkovnici in jih sproti shranjuje. Za razliko od operacije `WriteLine()`, ki je procedura, je operacija `ReadLine()` funkcija, kar pomeni, da nam v trenutku, ko uporabnik pritisne na tipko Enter, funkcija vrne vnešeno besedilo. Če želimo to besedilo nadalje uporabljati ga moramo shraniti v neko spremenljivko.

S programerskega stališča uporaba funkcij poteka v dveh korakih. V prvem koraku moramo deklarirati ustrezno spremenljivko za shranjevanje rezultata funkcije, v drugem koraku pa določimo njeno vrednost. Naslednji odsek programske kode prikazuje postopek deklariranja ter prireditveni stavek, ko pridobi spremenljivka *ime* vrednost, ki jo vrne funkcija `ReadLine()`.

```
Module Module1

   Sub Main()
      Dim ime as String ' Deklaracija spremenljivke
      ime = Console.ReadLine ' Prireditveni stavek
      ...
   End Sub

End Module
```

Kaj se dejansko zgodi v spominu računalnika pa prikazuje naslednja slika.



Slika 1.4: Spremenljivka ime dobi vrednost, ki jo vrne funkcija.

Deklaracija spremenljivk

Spremenljivka je simbol, ki ima dodeljeno neko vrednost. V računalništvu ima spremenljivka bolj konkreten pomen, saj označuje prostor v spominu računalnika, kjer je določena vrednost shranjena. Preden začnemo uporabljati spremenljivko, moramo v programu izrecno povedati, koliko spomina potrebujemo za hranjenje njene vrednosti, saj je spomin v računalniku omejen. Zato rabimo pred shranjevanjem podatkov v spomin vedeti, kakšno količino podatkov bomo shranili in kakšnega tipa bodo sami podatki. Če želimo hraniti števila od ena do sto, nam zadoščata dva bajta (osem ničel ali enic) spomina za vsako posamezno celo število. Za hranjenje števila PI do petnajst mest natančno, bomo potrebovali

osem bajtov pomnilniškega prostora. To dosežemo s postopkom imenovanim deklariranje. Deklariracija je programski stavek, ki se začne z rezervirano besedo `Dim`. Nato sledi ime spremenljivke ter njen podatkovni tip. Spremenljivko ime, ki hrani poljubno kombinacijo znakov deklariramo na naslednji način.

```
Dim ime As String
```

Beseda `String` tako določa podatkovni tip spremenljivke, ki nam omogoča shranjevanje besed.

V naslednji tabeli so prikazani podatki o količini spomina, ki ga mora rezervirati računalnik za neko spremenljivko z določenim podatkovnim tipom.

Tabela 1.1: Zbirka osnovnih deklaracij

Deklaracija	*Količina spomina*	*Obseg*
Boolean	2 bajta	Da / Ne
Double	8 bajtov	od $-1.79769 \cdot 10^{308}$ do $-4.94066 \cdot 10^{-324}$ za negativne vrednosti in od $4.94066 \cdot 10^{-324}$ do $1.79769 \cdot 10^{308}$ za pozitivne vrednosti
Integer	4 bajti	od $-2\ 147\ 483\ 648$ do $2\ 147\ 483\ 647$
Single	4 bajti	od $-3.40282 \cdot 10^{38}$ do $-1.40129 \cdot 10^{-45}$ za negativne vrednosti in od $1.40129 \cdot 10^{-45}$ do $3.40282 \cdot 10^{38}$ za pozitivne vrednosti
String	2 bajta$\cdot 2^{31}$	od 0 do približno 2 bilijona Unicode znakov

Pri izpisu v konzolno aplikacijo moramo združiti dve vrsti zapisa. Prvo je pozdravno besedilo, ki ga vnesemo v sam program, drugo pa je vsebina spremenljivke. Znotraj operacije `WriteLine()` lahko vnesemo oba tipa tako, da ju povežemo z znakom &. Tako se s tem ukazom izpiše zapisano besedilo ter tudi vrednost spremenljivke, ki se prebere iz spomina. Glede na samo pozicijo v ukazu `Writeline()` dobi tudi točno določeno mesto, kamor se spremenljivka izpiše.

Konkatenacija

Konkatenacija je združevanje dveh besedil v eno samo. V ta namen pri programiranju uporabljamo znak &. Njegov pomen je tak kot je znak za operacijo seštevanja v matematiki, pri čem se drugo besedilo postavi na konec prvega. Ko v našem drugem programu uporabimo simbol ime, računalnik dejansko uporabi vrednost, ki je shranjena na pomnilniški lokaciji, ki jo označuje ta simbol.

1.4 Vsota dveh celih števil

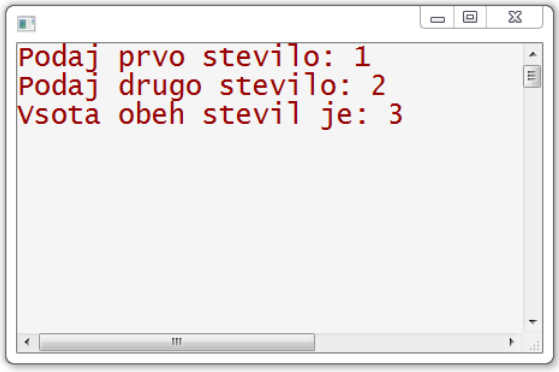
Nadaljevali bomo s programom, ki izračuna vsoto dveh celih števil. Za program bomo potrebovali dve spremenljivki a in b podatkovnega tipa Integer, ki omogoča shranjevanje celih števil, ki jih bo

vpisal uporabnik. Potrebujemo še tretjo spremenljivko c, ki jo bomo uporabili za hranjenje vsote števil a in b. Uporabnika najprej po vrsti povabimo, da vpiše števili a in b, nato števili seštejemo in rezultat priredimo spremenljivki c. Na koncu izpišemo rezultat v konzolnem oknu.

Vsota dveh celih števil

```
Module Module1

    Sub Main()
        Dim a, b, c As Integer
        Console.Write("Podaj prvo stevilo: ")
        a = Console.ReadLine()
        Console.Write("Podaj drugo stevilo: ")
        b = Console.ReadLine()
        c = a + b
        Console.WriteLine("Vsota obeh stevil je: " & c)
        Console.ReadKey()
    End Sub

End Module
```

Primer izvajanja programa je prikazan na sliki spodaj.



Slika 1.5: Vsota dveh celih števil

Za realizacijo programa nam zadoščajo tri spremenljivke podatkovnega tipa `Integer`, ki jih lahko deklariramo z enim samim programskim stavkom `Dim a, b, c As Integer`.

Pri zapisu besedila v konzolno okno smo uporabili dve proceduri `Write()` in `WriteLine()`, ki jih omogoča predmet `Console()`. Razlika med tema procedurama je, da procedura `Write()` izpiše zahtevano besedilo, vendar ne doda znaka za novo vrstico. Procedura `WriteLine()` pa izpiše besedilo ter na koncu doda (nevidni) znak za novo vrstico tako, da se izpis nadaljuje v naslednji vrstici konzole.

Prireditveni stavek

Prireditveni stavek je programski stavek, v katerem se pojavi znak $=$. V računalništvu ima znak za enačaj drugačen pomen, kot ga ima v matematiki. V matematiki enačaj pove, da sta leva in desna stran enaki. V računalništvu pa se najprej izvede funkcija na desni strani enačaja, rezultat izvajanja pa se dodeli spremenljivki na levi strani enačaja.

V praksi to pomeni, da če hočemo shraniti število 30 v spremenljivko a, lahko to storimo le na en način:

```
a = 30 ' pravilen zapis
30 = a ' povsem nepravilen zapis
```

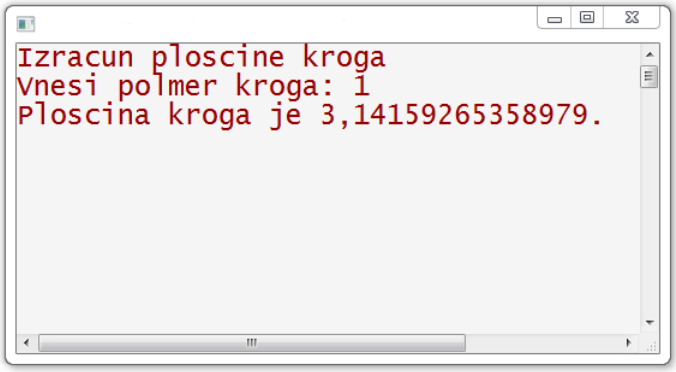
1.5 Program za izračun ploščine kroga

Poglejmo program, ki uporabniku omogoča izračun ploščine kroga. Program najprej vpraša uporabnika, kolikšen je polmer kroga. Glede na podano vrednost nato izračuna in izpiše vrednost ploščine.

Ploščina kroga

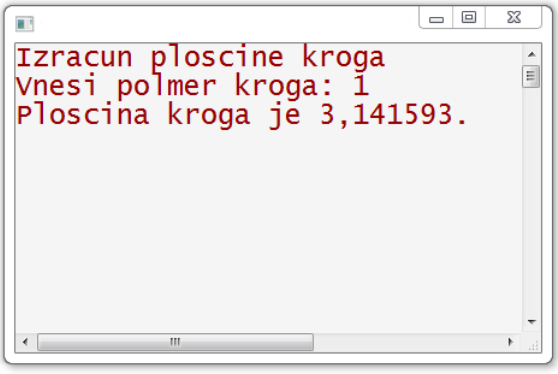
```
Module Module1

    Sub Main()

        Dim r, p As Double 'deklaracija potrebnih spremenljivk

        Console.WriteLine("Izracun ploscine kroga") 'začetno obvestilo in branje
        polmera
        Console.Write("Vnesi polmer kroga: ")
        r = Console.ReadLine()

        p = 3.14159265358979 * r ^ 2 'izračun ploščine

        Console.WriteLine("Ploscina kroga je " & p & ".") 'izpis rezultata
        Console.ReadKey()
    End Sub

End Module
```

Izdelani program deluje kot prikazuje naslednja slika.



Slika 1.6: Ploščina kroga

Pri izdelavi programa smo uporabili dve spremenljivki r in p podatkovnega tipa `Double`, ki se uporablja za shranjevanje realnih števil z dvojno natančnostjo. V praksi to pomeni, da uporabimo ta podatkovni tip, ko operiramo z realnimi števili, ki imajo veliko števk. Alternativno bi lahko uporabili podatkovni tip `Single`, ki omogoča hranjenje realnih števil z enojno natančnostjo. Takšna realna števila bi imela manjše število števk, kar pomeni, da bodo posledično vsi izračuni manj natančni. Preizkusimo program tako, da zamenjamo deklaracijo spremenljivk `As Double` z deklaracijo `As Single`. Z istim vnosom program sedaj deluje tako.



Slika 1.7: Ploščina kroga

V prvi različici programa je izpis ploščine narejen z natančnostjo petnajstih števk, štirinajst števk je za decimalno piko. V drugi različici programa pa je izpis ploščine s sedmimi števkami, kar posledično predstavlja manj natančen izračun.

Zgornji program lahko še optimiziramo tako, da sami določimo, na koliko decimalnih mest natančno se bo ploščina izpisala. To najlažje storimo z uporabo vgrajene funkcije `ToString()`, ki je implementirana za spremenljivke tipa `Double`. Če želimo, da bo izpis narejen na 3 decimalna mesta natančeno potem v program zapišemo `Console.WriteLine("Ploscina kroga je "& p.ToString("F3")& ".")`. Oznaka $F3$ označuje število decimalnih mest in je vhodni parameter za funkcijo `ToString()`.

Število PI smo zapisali na štirinajst decimalk natančno. Visual Basic pa ima vgrajeno knjižnico matematičnih funkcij ter konstant, ki jih lahko direktno uporabljamo pri programiranju. Števila PI tako ne rabimo ročno vnašat, ampak ga preprosto pokličemo iz knjižnice tako: `p = Math.PI * r ^ 2`. Če je spremenljivka a deklarirana kot `Double`, bo tako pridobila vrednost števila PI na 16 decimalnih mest natančno.

Knjižnica Math

V uporabniškem vmesniku Visual Studio je vgrajenih veliko knjižnic, med katere spada tudi knjižnica `Math`. Če v programu ne bomo uporabljali veliko matematičnih konstant in funkcij, nam ni potrebno uvoziti celotne knjižnice, ampak kličemo posamezne ukaze s sklicevanjem na to knjižnico - `Math.PI`.

V primeru, da bomo v program vključili veliko matematičnih elementov, je smiselno, da na začetku programa uvozimo celotno knjižnico ukazov, iz katere nato kličemo posamezne ukaze. To pomeni, da pred vso programsko kodo progama dodamo ukazno vrstico z vsebino `Imports System.Math`. S tem zmanjšamo količino programske kode, saj število PI priličemo samo z ukazom `PI`.

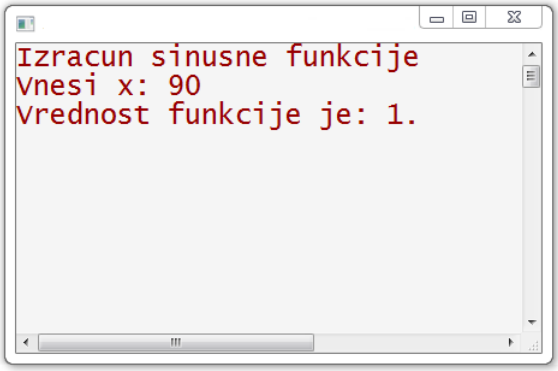
1.6 Program za izračun vrednosti sinusne funkcije

Izdelajmo še program, ki za podano vrednost kota, na primer, $x = 60°$ izračuna vrednost sinusne funkcije. Program izgleda takole:

Vrednost sinusne funkcije

```
Module Module1

    Sub Main()

        Dim x, n As Single 'deklaracija potrebnih spremenljivk

        Console.WriteLine("Izracun sinusne funkcije") 'začetno obvestilo in branje
        vnosa
        Console.Write("Vnesi x: ")
        x = Console.ReadLine()

        n = Math.Sin(x * Math.PI / 180) 'izračun vrednosti funkcije

        Console.WriteLine("Vrednost funkcije je: " & n & ".") 'izpis rezultata
        Console.ReadKey()
    End Sub

End Module
```

Še slika programa:



Slika 1.8: Ploščina kroga

Pri izdelavi programa smo uporabili funkcijo `Sin()` iz knjižnice `Math`. Pri klicanju vseh funkcij in procedur moramo biti pozorni na število in tip zahtevanih argumentov. Funkcija `Sin()` zahteva le en argument, realno število, ki predstavlja vrednost kota v radianih. Ker bo uporabnik podal vrednost kota v stopinjah, moramo v programu poskrbeti še za pretvorbo kota iz stopinj v radiane.

Programi, ki jih napišemo, so lahko namenjeni avtomatizaciji določenih postopkov. Zna se zgoditi, da se postopki spremenijo, zato je potrebno programe popraviti ali celo spisati na novo. Če se to zgodi po daljšem časovnem obdobju, navadno pozabimo, s kakšnim namenom smo spisali določeno vrstico programske kode. Prav zaradi tega je skoraj nujno komentirati programske stavke. Komentiramo tako, da dodamo na poljubnem mestu v vrstici znak ' in nadaljujemo z našim komentarjem. Pomembno je tudi, da pri programiranju uporabljamo pomenljiva imena spremenljivk, procedur in funkcij.

Dobra praksa komentiranja

Komentar je prosto spisano besedilo zraven programske kode, ki ga prevajalnik pri prevajanju ne upošteva. Namenjen je boljšemu razumevanju programske kode. Poglejmo si primer komentarja, ki razloži bistvo uporabljene formule.

```
n = Math.Sin(x * Math.PI / 180) 'izračun vrednosti funkcije
```

Poglejmo si še tabelo rezerviranih besed jezika Visual Basic, ki jih bomo uporabljali pogosto v naših programih.

Tabela 1.2: Zbirka rezerviranih besed

Beseda	*Pomen*
1	2
3	4
5	6

Uporaba razvojnega okolja, poznavanje rezerviranih besed iz programskega jezika, podatkovni tipi, postopki deklariranja, prirejanja in komentiranja spadajo med najbolj osnovne stvari, ki jih mora obvladati sleherni programer. V zadnjem poglavju so naloge za utrjevanje temeljnih spoznanj. Veselo na delo!

2 Algoritmi

Naši prvi programi nam omogočajo izvajati preproste računske naloge, ki jih enostavno lahko opravimo tudi s kalkulatorjem. Izračun ploščine kroga računalniku ne predstavlja posebnega izziva. Reševanje sistema večjega števila enačb in drugih matematičnih problemov ter urejanje ogromnih količin podatkov pa je zelo zamudno delo, zato ga raje prepustimo računalniku. Reševanje kompleksnejših nalog namreč zahteva izdelavo natančnih računskih postopkov oz. algoritmov. Algoritem je postopek, ki vsebuje podatke, je natančno določen, izvedljiv in se zaključi v končnem številu korakov oz. ukazov ter običajno vrne rezultat.

Algoritme lahko zapišemo na več različnih načinov: v naravnem jeziku, grafično s pomočjo diagrama poteka, s t.i. psevdokodo ali kot program zapisan v nekem programskem jeziku. Od namena uporabe algoritma je odvisno na kakšen način in kako podrobno bo zapisan. V nadaljevanju si poglejmo primer preprostega algoritma, ki je zapisan na vse štiri omenjene načine.

Preprosti algoritem v naravnem jeziku:

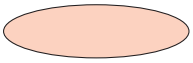
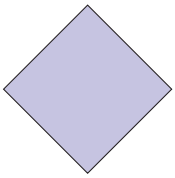
> Uporabnik vpiše števili a in b. Preverimo ali je a<b. Če je res, vrnemo odgovor, da je število b večje, sicer vrnemo odgovor, da je število b manjše ali enako številu a.

Ta isti algoritem lahko zapišemo v psevdokodi na nekoliko bolj strukturiran način:

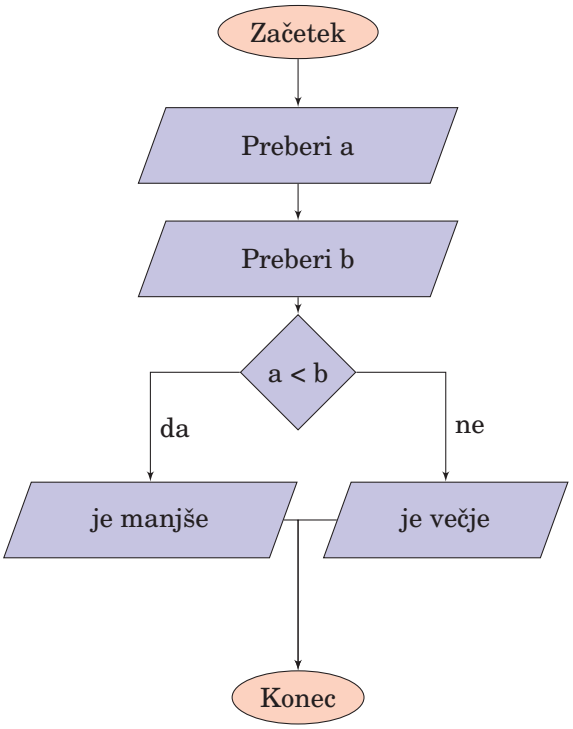
```
Začetek
Vpiši a
Vpiši b
Če je a < b
izpiši “b je večji od a."
sicer
izpiši “b je manjši ali enak a.”
Do sem
Konec
```

Algoritem v obliki blok diagrama nam omogoča natančen vpogled v sosledje izvajanja programskih stavkov. Osnovni elementi blok diagramov so prikazani v tabeli 2.1. Na podlagi teh osnovnih elementov lahko zapišemo tudi zelo kompleksne algoritme.

Tabela 2.1: Tabela elementov blok diagramov

Simboli	*Razlaga*
	Začetni ali končni blok
	Blok vhodno-izhodnih operacij
	Odločitveni blok
	Prireditveni blok

Blok diagram za naš algoritem je predstavljen na sliki 2.1.



Slika 2.1: Blok diagram

Algoritem lahko zapišemo zelo natančno, če uporabimo katerega od programskih jezikov, na primer, Visual Basic.

Algoritem v obliki programa

```
Module Module1

    Sub Main()
        Dim a, b As Integer
        Console.Write("Vpisi a: ") : a = Console.ReadLine
        Console.Write("Vpisi b: ") : b = Console.ReadLine
        If a < b Then
            Console.WriteLine("a je manjse od b.")
        Else
            Console.WriteLine("a je vecje ali enako b.")
        End If
        Console.ReadKey()
    End Sub

End Module
```

Opazimo, da je zapis programa v psevdokodi nekakšen predhodnik zapisa v programskem jeziku. Na primer, del programskega stavka: če je a < b potem, se v jeziku Visual Basic zapiše kot: If a<b Then.

Vsi programski stavki znotraj tako zapisanega programa se izvajajo po vrstnem redu v katerem so zapisani. V eni vrstici lahko zapišemo tudi dva ali več programskih stavkov tako, da dodamo znak ":". Vrstica `Console.Write("Vpisi a: "): a = Console.ReadLine()` tako vsebuje dva programska stavka in sicer `Console.Write("Vpisi a: ")` in `a = Console.ReadLine()`. To storimo bodisi z namenom varčevanja s prostorom, bodisi zaradi logičnega sosledja posameznih programskih stavkov. V primeru zapisa več programskih stavkov v isti vrstici, se bodo stavki izvajali v vrstnem redu zapisa iz leve proti desni.

Zasnova in ustreznost algoritma sta zelo pomembni pri izdelavi programa. Oblika samega zapisa pri temu ne igra večje vloge in jo lahko izberemo glede na trenutne zahteve. Razvoj algoritma je ponavadi zapleten in dolgotrajen postopek, v katerem pridemo do končne rešitve po določenem številu korakov, pri čemer predstavlja vsak korak nek njen približek. Med vsakim korakom namreč razvijemo program do večje podrobnosti. Glavni namen tega je, da se čim bolj natančno približamo končni rešitvi, ni pa potrebno, da je oblika izpisana do zadnje potankosti. Določeni deli algoritma so zapisani z diagramom poteka, matematičnimi formulami in z uporabo naravnega jezika. Potrebno se je torej naučiti, kako izdelati zasnovo algoritma, saj si tako močno olajšamo samo programiranje.

2.1 Izrazi in izjave

Končna oblika algoritma naj bi bila zapisana v obliki programa z vsemi sintaktični pravili izbranega programskega jezika, v našem primeru v jezuku Visual Basic. Če zapis programa ni pravilen, se

kasneje sam program ne bo mogel izvajati. Med najvažnejše sintaktične enote programskega jezika spadajo izrazi in stavki.

Izraz je formula oziroma računski predpis, ki določa neko vrednost ali rezultat. Izraz vsebuje operante ter operatorje. Operanti so bodisi konstante (števila), spremenljivke ali vrednosti, ki jih vrnejo funkcije. Če izraz vsebuje več operatorjev, ki združujejo operante brez uporabe oklepajev, programski jezik določa tudi pravilo v katerem vrstnem redu se bodo izvajale operacije. V programskem jeziku Visual Basic se izrazi vrednotijo iz leve proti desni.

Rezultat izvajanja izrazov je lahko karkoli: celo ali realno število, določen digitalni predmet, ki ga vrne funkcija in tako dalje. Nekaj primerov izrazov si poglejmo v tabeli 2.2.

Tabela 2.2: Pogojni in iterativni stavki

Izraz	*se izvede kot*
$x + y - z$	$(x + y) - z$
$x/y/z$	$(x/y)/z$
$x - y * z + w$	$(x - (y * z)) + w$
$Math.Sin(x) + n * Math.Cos(x/2)$	$(Math.Sin(x))+(n*Math.Cos(x/2))$
$a\ and\ b\ or\ c > 23$	$(a\ and\ b)\ or\ c > 23$

V programih pogosto nastopajo tudi izjave. Izjava je logični izraz, ki je lahko ovrednoten kot resničen (True) ali neresničen (False). Logične izraze običajno tvorimo z uporabo relacijskih operatorjev, ki so predstavljeni v tabeli 2.3. Na primer, izjava 5>3 je resnična (True), izjava 2>=7 pa neresnična (False).

Tabela 2.3: Relacijski operatorji

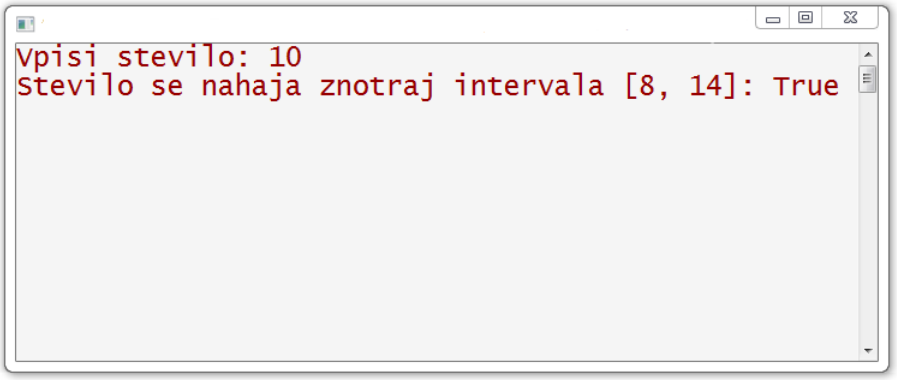
Operator	*Relacija*
=	je enako
<	je manjše
>	je večje
<>	ni enako
<=	je manjše ali enako
>=	je večje ali enako

Poleg teh imamo na voljo tudi logične operacije, kot so negacija (Not), konjunkcija oz. logični in (And) in disjunkcija oz. logični ali (Or). Za ilustracijo uporabe logičnih izrazov si oglejmo naslednji program, ki preveri ali podano število leži na intervalu [8, 14].

```
Module Module1

    Sub Main()
        Dim a, b, rezultat As Boolean
```

```
        Dim podatek As Integer
        Console.Write("Vpisi stevilo: ") : podatek = Console.ReadLine
        a = podatek >= 14
        b = podatek <= 8
        rezultat = Not (a Or b)
        Console.WriteLine("Stevilo se nahaja znotraj intervala [8, 14]: " & rezultat)
        Console.ReadKey()
    End Sub

End Module
```

Program deluje takole:



Slika 2.2: Ploščina kroga

Spremenljivki a in b sta tokrat deklarirani kot spremenljivki podatkovnega tipa Boolean. Spremenljivke tega tipa zavzameta lahko le dve možni vrednosti: True ali False.

Opiši po vrsti delovanje programa stavek za stavkom.

2.2 Programski stavki

Pri programskih stavkih je bistveno, da imajo določen učinek. Delimo jih na nekaj osnovnih tipov: prireditveni stavek, pogojni stavek, iterativni stavek oziroma zanka, izbirni stavek in klice procedur.

V nadaljevanju bomo skozi primere spoznali različne vrste programskih stavkov.

Prireditveni stavek ima enostavno obliko S = I, pri čem je S spremenljivka, I pa izraz. Prireditvene stavke smo uporabili že pri prvih izdelanih programih, na primer pri programu za izračun ploščine kroga. Učinek prireditvenega stavke je, da se izračunana vrednost izraza na desni strani enačaja dodeli spremenljivki na levi strani enačaja. Tip izraza se mora ujemati s tipom spremenljivke.

Pogojne stavke smo spoznali na začetku tega poglavja, med temi spadata stavka If-Then in If-Then-Else in se jih hitro naučimo uporabljati.

Več pozornosti bomo v nadaljevanju namenili *iterativnim stavkom* oziroma zankam. To so stavki:

Tabela 2.4: Pogojni in iterativni stavki

VB.net	*Psevdokoda*

`If <izjava> Then`
`  A`
`End If`
	`Ce je <izjava> potem`
	`  naredi A`
	`Do  sem`

`If <izjava> Then`
`  A`
`Else`
`  B`
`End If`
	`Ce je <izjava> potem`
	`  naredi A`
	`Sicer`
	`  naredi B`
	`Do  sem`

`Do`
`  A`
`Loop While <izjava>`
	`Ponavljaj`
	`  naredi A`
	`Dokler velja <izjava>`

`Do While <izjava>`
`  A`
`Loop`
	`Dokler velja <izjava> ponavljaj`
	`  naredi A`
	`Do sem`

`Do Until <izjava>`
`  A`
`Loop`
	`Dokler je <izjava> ponavljaj`
	`  naredi A`
	`Do sem`

`Do`
`  A`
`Loop Until <izjava>`
	`Ponavljaj`
	`  naredi A`
	`Dokler je <izjava>`

`For i=1 To n`
`  A`
`Next i`
	`Za i=1 do n ponavljaj`
	`  naredi A`
	`Do  sem`

Izbirni stavek je prikazan v tabeli 2.5.

Tabela 2.5: Izbirni stavek

VB.net	*Psevdokoda*

`Select N`
`  Case 1`
`  A`
`  Case 2`
`  B`
`End Select`
	`Glede na vrednost N`
	`  Primer 1`
	`  naredi A`
	`  Primer 2`
	`  naredi B`
	`Do  sem`

V nadaljevanju bomo za ilustracijo posameznih programskih stavkov izdelali več različnih programov.

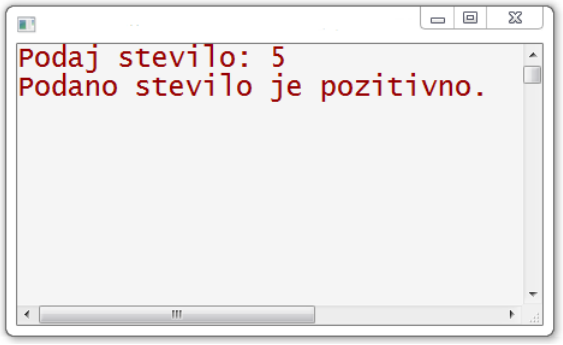
2.3 Ali je število pozitivno?

Najprej bomo izdelali program, ki komunicira z uporabnikom in ugotovi, ali je vpisano število pozitivno. Program izgleda takole:

Pozitivno število

```
Module Module1

    Sub Main()
        Dim a As Single

        Console.Write("Podaj stevilo: ") : a = Console.ReadLine()

        ' If-Then pogojni stavek
        If a > 0 Then
            Console.Writeline("Podano stevilo je pozitivno.")
        End If
        ' Konec kontrolnega stavka

        Console.ReadKey()

    End Sub
End Module
```

Program deluje takole:



Slika 2.3: Pozitivno število

Spremenljivka a je deklarirana kot `Integer` in bo hranila število, ki ga vpiše uporabnik. Potem, ko uporabnik vpiše število, s pogojnim stavkom If-Then preverimo, če je izpolnjena izjava $a > 0$. Če je to res, program izpiše "Podano število je pozitivno."

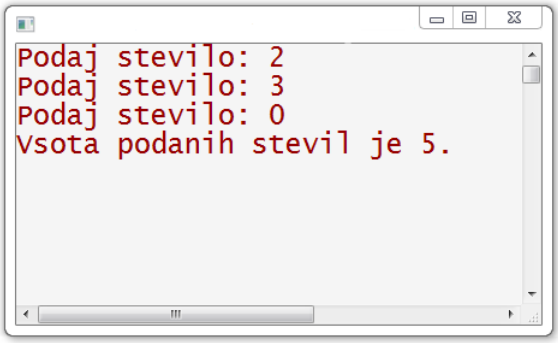
2.4 Vsota podanih števil

V prvem poglavju smo izdelali preprost program, ki nam omogoča seštevanje dveh števil. V tem poglavju ga nadgradimo z Do-Loop-While zanko, z namenom, da se seštevanje ponavlja toliko časa, dokler ne vpišemo števila nič. Nov program izgleda takole.

Vsota podanih števil

```
Module Module1

    Sub Main()
        Dim stevilo, vsota As Single

        vsota = 0 ' inicializacija spremenljivke
        ' Do-Loop-While zanka
        Do
            Console.Write("Podaj stevilo: ") : stevilo = Console.ReadLine()
            vsota = vsota + stevilo
        Loop Until stevilo = 0
        ' konec zanke

        Console.WriteLine("Vsota podanih stevil je " & vsota & ".")

        Console.ReadKey()

    End Sub
End Module
```

Program deluje tako kot prikazuje slika 2.4



Slika 2.4: Vsota števil

Z namenom izdelave programa smo deklarirali dve spremenljivki *stevilo* in *vsota* podatkovnega tipa `Single`. Spremenljivka *stevilo* se uporablja za sprotno shranjevanje vpisanih števil, spremenljivka *vsota* pa za sprotno shranjevanje vsote. Pri uporabi zank, kot je Do-Loop-While, je zelo pomembno, da pravilno zastavimo ustavitveni pogoj, v našem primeru $stevilo = 0$. Torej, zanka se bo prenehala izvajati v trenutku, ko bo uporabnik vpisal 0, v vseh ostalih primerih pa se bodo izvajali programski stavki znotraj zanke.

Sedaj si oglejmo nekoliko bolj natančno, kaj se v programu dogaja s spremenljivko *vsota*. Na levi strani prireditvenega stavka `vsota = vsota + stevilo` je spremenljivka vsota, na desni strani pa matematični izraz vsota + stevilo. Vsakič, ko se bo ta programski stavek izvajal, se bo najprej ovrednotil izraz na desni strani enačaja. Računalnik bo segel v pomnilnik in tam prebral vrednosti obeh spremenljivk, izračunal bo njuno vsoto in jo potem zapisal na pomnilniško lokacijo rezervirano za spremenljivko vosta, kot je to določeno na levi strani izraza. Potek izvajanja tega stavka je nazorno prikazan na naslednji sliki:

2.5 Poštevanka

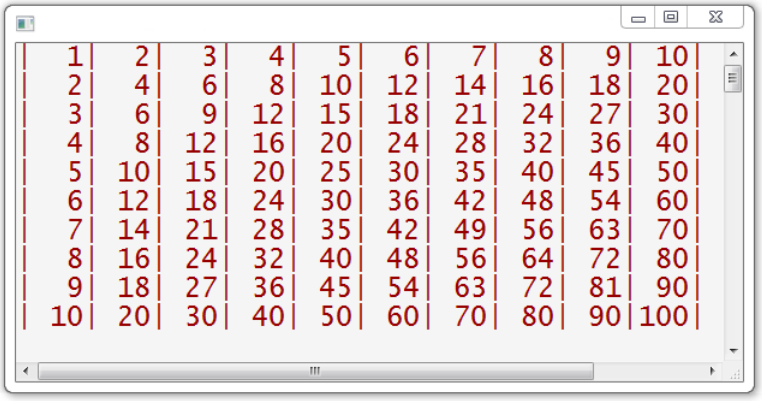
Že v osnovni šoli smo se seznanili s poštevanko števil od 0 do 10, sedaj pa izdelajmo program, ki avtomatično izpiše poštevanko števil.

Poštevanka

```
Module Module1

    Sub Main()
        Dim produkt As String
        'For zanka
        For i = 1 To 10 'vrstice
            For j = 1 To 10 'stolpci
```

```
                produkt = i * j
                ' izpis z dodajanjem presledkov
                If produkt.Length = 1 Then
                    Console.Write("|  " & produkt)
                ElseIf produkt.Length = 2 Then
                    Console.Write("| " & produkt)
                Else
                    Console.Write("|" & produkt)
                End If
            Next
            Console.WriteLine("|")
        Next
        Console.ReadKey()

    End Sub
End Module
```

Program deluje tako, kot prikazuje naslednja slika:



Slika 2.5: Poštevanka

Za izdelavo poštevanke smo uporabili dvojno For-Next zanko, prvo za izpis posamezne vrstice, drugo pa za zapis stolpcev v posamezno vrstico. Tako, se programski stavek If-Then bo izvajal natanko $10x10 = 100$ krat in bo za vsako kombinacijo števil $i * j$ izpisal njihov produkt.

Spremenljivko *produkt* smo definirali kot `String` z namenom, da uredimo ustrezne zamike pri izpisu števil. Produkt dveh števil je včasih lahko enomestno število, včasih dvomestno in v primeru $i = 10$ in $j = 10$, je produkt tromestne število. Da bomo vedeli kolikšno je število cifer v produktu bomo izkoristili lastnost `.Length`, ki jo imajo vse spremenljivke podatkovnega tipa `String`.

Prireditveni stavek `produkt = i * j` najprej izračuna vrednost produkta dveh števil na desni strani izraza (na primer, 6 * 8). Rezultat izvajanja te operacije je celo število. Nato se to celo število vpiše v spremenljivko podatkovnega tipa `String`. Na tem mestu velja pripomniti, da je takšen

vpis možen zato ker podatkovni tip `String` omogoča zadosti prostora za vpis celih števil. Pomen enačaja v prireditvenem stavku torej ni samo prirejanje vrednosti spremenljivki produkt, temveč tudi pretvorba celega števila 48 v besedilo "48". V nadaljevanju lahko spremenljivko `produkt.Length` uporabimo za dodajanje ustreznega števila presledkov, zato, da bo prikaz bolj urejen. Pri dolžini 1 se dodata dva presledka, pri dolžini 2 se doda 1 presledek. Če ima spremenljivka dolžino tri pa ni treba dodajati dodatnih presledkov.

2.6 Program z menijem

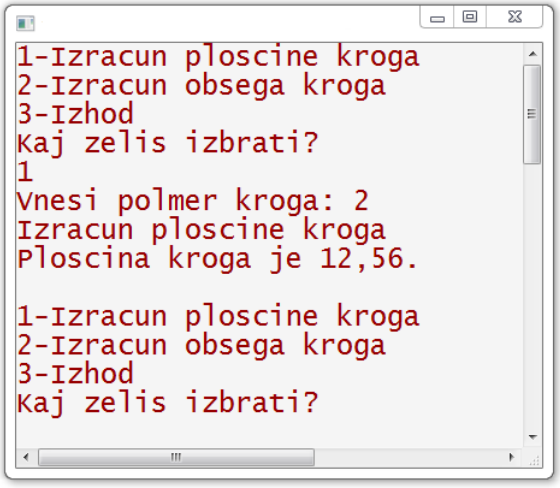
V naslednjem programu želimo uporabniku omogočiti izbiro bodisi izračuna ploščine kroga bodisi izračuna obsega kroga na podlagi menija.

Program z menijem

```
Module Module1

    Sub Main()
        Dim n As Integer ' izbira v meniju
        Dim r, p, o As Single ' radij, ploščina in obseg kroga

        Do
            Console.WriteLine("1-Izracun ploscine kroga")
            Console.WriteLine("2-Izracun obsega kroga")
            Console.WriteLine("3-Izhod")
            Console.WriteLine("Kaj zelis izbrati?") : n = Console.ReadLine() '
            uporabnik vpiše izbiro

            Select Case n
                Case 1
                    Console.Write("Vnesi polmer kroga: ")
                    r = Console.ReadLine()
                    Console.WriteLine("Izracun ploscine kroga")
                    p = 3.14 * r ^ 2
                    Console.WriteLine("Ploscina kroga je " & p & ".")
                    Console.WriteLine("")
                Case 2
                    Console.WriteLine("Vnesi polmer kroga:")
                    r = Console.ReadLine()
                    Console.WriteLine("Izracun obsega kroga")
                    o = 3.1415926 * r * 2
                    Console.WriteLine("Obseg kroga je " & o & ".")
                    Console.WriteLine("")
                Case 3
                    Console.WriteLine("Zahvaljujem se za zaupanje!")
                    Console.WriteLine("")
                Case Else
```

```
                    Console.WriteLine("Napacna izbira!")
                End Select
            Loop While n <> 3

        End Sub
    End Module
```

Program deluje tako, kot je prikazano na naslednji sliki.



Slika 2.6: Program z menijem

Poleg običajnih deklaracij, smo v program vgradili izbirni stavek `Select Case`, ki glede na vpisano število n zažene ustrezen del programske kode. Če uporabnik izbere napačno število program avtomatsko izpiše opozorilo "Napacna izbira!". Če uporabnik izbere tretjo opcijo v meniju pa program izstopi iz zanke Do-Loop-While in nato zaključi z delovanjem.

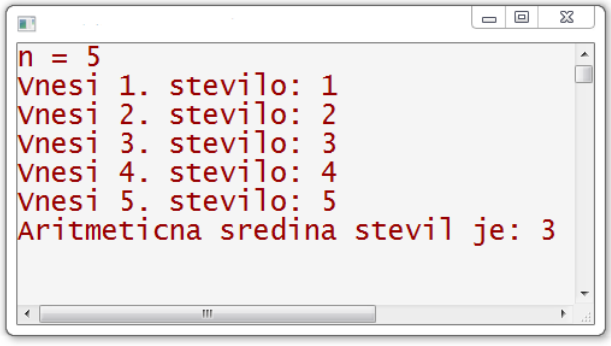
2.7 Aritmetična sredina

Naslednji program nam omogoča izračun aritmetične sredine števil. Uporabnik najprej vpiše koliko je takih števil, nato program zahteva, da uporabnik vpiše vsa števila. Na koncu izpiše aritmetično sredino podanih števil. Program je predstavljen v nadaljevanju.

Aritmetična sredina

```
Module Module1

    Sub Main()
        Dim n As Integer = 1
        Dim vsota As Single = 0
        Dim sredina As Single = 0

        Console.Write("n = ") : n = Console.ReadLine()

        For i = 1 To n
            Console.Write("Vnesi " & i & ". stevilo: ")
            vsota = vsota + Console.ReadLine()
        Next
        sredina = vsota / n

        Console.WriteLine("Aritmeticna sredina stevil je: " & sredina)
        Console.ReadKey()

    End Sub
End Module
```

Program deluje tako kot je prikazano na sliki 2.7.



Slika 2.7: Aritmetična sredina

Na začetku smo deklarirali nekaj spremenljivk, ki jih bomo potrebovali za shranjevanje števila podanih števil (n), sprotne vsote (vsota) ter aritmetične sredine (sredina). Ob deklaraciji smo vse tri spremenljivke smo jim dodelili začetne vrednosti oziroma smo jih inicializirali. Uporabnik najprej poda število n. Potem se zanka For-Next izvaja natančno n krat. Ob vsakem prehodu skozi zanko uporabnik vpiše novo število. Po poteku zanke program izračuna in izpiše aritmetično število.

Naš program, kot tudi vsi dosedanji, deluje brez težav. Ampak, kaj se zgodi z našim programom,

če na začetku uporabnik namesto števila poda neko naključno besedilo, na primer "abba". V tem primeru, se nam program sesuje. V resnici, program med izvajanjem poskuša dodeliti vrednost $abba$ spremenljivki n, ki je deklarirana kot spremenljivka podatkovnega tipa Integer. Vendar tip spremenljivke n (na levi strani prireditvenega stavka) ne ustreza predmetu, ki ga vrne funkcija `ReadLine()` (na desni strani stavka). Računalnik poskuša pretvoriti tekstovni podatek $abba$ v celo število, vendar mu to ne uspe in zato takoj preneha z izvajanjem. Če želimo reševati tovrstne probleme potem moramo v naš program vgraditi tudi določene kontrolne mehanizme, na primer programske stavke, ki preverijo ali je uporabnik vpisal celo ali realno število, ki uporabnika opozarjajo na nepravilnost vnosa.

Zamikanje

Zamikanje pri programiranju nam omogoča, da hitreje razberemo, kje se začne in kje se konča posamezen programski stavek. V naslednjem primeru, je programski stavek Console.WriteLine("Dober dan!") vrinjen znotraj zanke For-Next, kar pomeni, da se bo izvajal natančno 5-krat.

```
For i = 1 To 5
    Console.WriteLine("Dober dan!")
Next
```

2.8 Ugani število

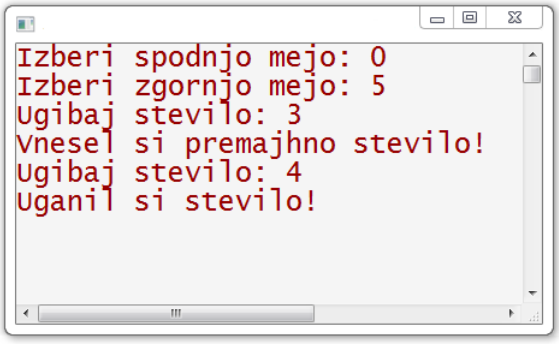
Ideja naslednjega programa je, da si program v spominu naključno generira celo število na podanem intervalu $[a, b]$. Uporabnik nato ugiba, za katero število gre, računalnik pa mu sporoča ali je število premajno ali preveliko.

Ugani število

```
Module Module1

    Sub Main()

        Dim a, b As Integer ' spodnja in zgornja meja
        Dim x As Single
        Dim stevilo, izbira As Integer ' naključno število in izbira uporabnika

        Console.Write("Izberi spodnjo mejo: ") : a = Console.ReadLine
        Console.Write("Izberi zgornjo mejo: ") : b = Console.ReadLine

        Randomize()
        x = Rnd()
        stevilo = Int((b - a) * x + a)
```

```
        Do
            Console.Write("Ugibaj stevilo: ")
            izbira = Console.ReadLine()
            If izbira < stevilo Then
                Console.WriteLine("Vnesel si premajhno stevilo!")
            ElseIf izbira > stevilo Then
                Console.WriteLine("Vnesel si preveliko stevilo!")
            End If

        Loop While izbira <> stevilo
        Console.WriteLine("Uganil si stevilo!")
        Console.ReadKey()

    End Sub
End Module
```

Slika 2.8 prikazuje delovanje tega programa



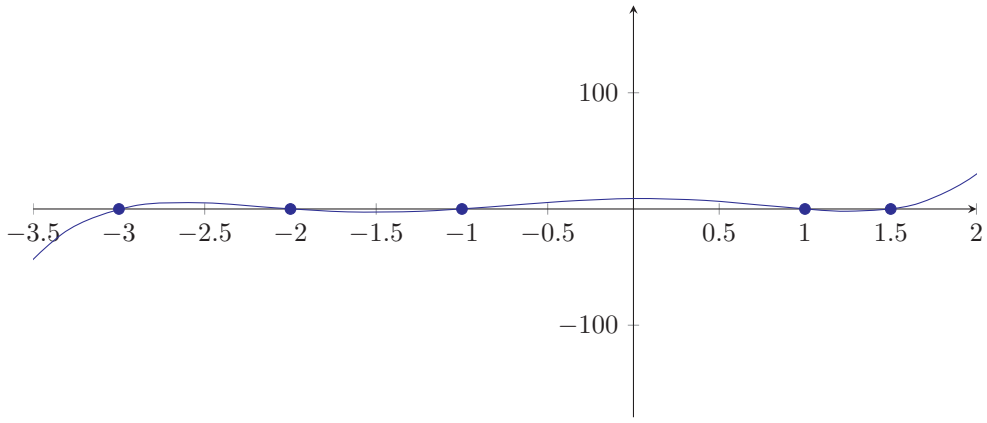
Slika 2.8: Ugani število

V tem programu uporabljamo vgrajene funkcije `Randomize()`, `Rnd()` in `Int()`. Funkcija `Randomize()` zažene generator zaporedja naključnih realnih števil iz intervala $[0, 1]$. Generator si lahko predstavljamo kot bobenček za izbiro števil na loteriji. Potem, ko nam bobenček dodobra premeša števila lahko uporabimo funkcijo `Rnd()`, ki nam vsakič vrne novo realno število iz generiranega zaporedja. Sedaj moramo poskrbeti še za preslikavo realnih števil iz intervala $[0, 1]$ v cela števila iz intervala $[a, b]$. Do formule za preslikavo pridemo z upoštevanjem naslednjega razmerja: $(1-0)/(b-a) = (1-x)/(b-y)$, kjer je x naključna realna vrednost iz intervala $[0, 1]$, spremenljivka y pa predstavlja celo število iz intervala $[a, b]$ v katero se preslika. Na ta način izpeljemo funkcijo $y = a + x * (b-a)$. Nazadnje moramo poskrbeti za pretvorbo realnega števila y v celo število. V ta namen uporabimo funkcijo `Int()`, ki pretvori realno število v celo število tako, da odreže vse decimalke.

Zanka Do-Loop-While se izvaja vse dokler uporabnik ne ugane generiranega števila. Ustavitveni pogoj za to zanko je izjava $izbira = stevilo$.

2.9 Izračun ničle funkcije z bisekcijo

Denimo, da je podana funkcija $f(x) = x^5 + 3.5 * x^4 - 2.5 * x^3 - 12.5 * x^2 + 1.5 * x + 9$, katere potek je prikazan na sliki 2.9. Funkcija ima pet ničel, za katero vemo, da so v točkah $(-3, 0)$, $(-2, 0)$, $(-1, 0)$, $(1, 0)$, $(1.5, 0)$, vendar bi želeli izdelati program, ki jih zna poiskati po metodi bisekcije.



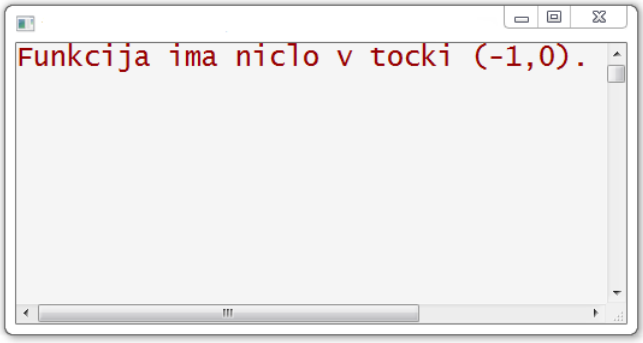
Slika 2.9: $f(x) = x^5 + 3.5 * x^4 - 2.5 * x^3 - 12.5 * x^2 + 1.5 * x + 9$

Program za bisekcijo izgleda takole.

```
Module Module1

    Sub Main()

        Dim a As Double = -1.5 ' Zgornja meja intervala
        Dim b As Double = 0.5 ' Spodnja meja intervala
        Dim eps As Double = 0.000000000000001 ' Natančnost izračuna ničle funkcije
        Dim x, fa, fb, fx As Double ' Vrednosti ničle x ter vrednosti funkcije v
        točkah a, b in x

        Do
            x = (a + b) / 2 ' Razdelimo interval na polovico
            fa = Math.Pow(a, 5) + 3.5 * Math.Pow(a, 4) - 2.5 * Math.Pow(a, 3) - 12.5 _
            * Math.Pow(a, 2) + 1.5 * a + 9
            fb = Math.Pow(b, 5) + 3.5 * Math.Pow(b, 4) - 2.5 * Math.Pow(b, 3) - 12.5 _
            * Math.Pow(b, 2) + 1.5 * b + 9
            fx = Math.Pow(x, 5) + 3.5 * Math.Pow(x, 4) - 2.5 * Math.Pow(x, 3) - 12.5 _
            * Math.Pow(x, 2) + 1.5 * x + 9

            If fa * fx <= 0 Then
                b = x
            Else
                a = x
            End If
```

```
        Loop While (b - a) > eps

        ' dokončni izračun ničle funkcije
        x = (a + b) / 2
        fx = Math.Pow(x, 5) + 3.5 * Math.Pow(x, 4) - 2.5 * Math.Pow(x, 3) - 12.5 * _
        Math.Pow(x, 2) + 1.5 * x + 9

        Console.WriteLine("Funkcija ima niclo v tocki (" & x.ToString("F0") & "," & _
        fx.ToString("F0") & ").")
        Console.ReadKey()

    End Sub

End Module
```

Program deluje tako, kot prikazuje naslednja slika 2.10.



Slika 2.10: Bisekcija

Predpostavimo, da je na izbranem intervalu $[a, b]$ le ena ničla funkcije. Če je na spodnji meji a vrednost funkcije negativna, je na zgornji meji funkcija pozitivna ali obratno. Algoritem je zasnovan tako, da najprej izračunamo srednjo točko intervala x in za ta x izračunamo vrednost funkcije. Če je produkt $f_a \cdot f_x < 0$ to pomeni, da se ničla funkcije nahaja na intervalu $[a, x]$, sicer pa se nahaja na intervalu $[x, b]$. V prvem primeru x postane zgornja meja intervala, v drugem primeru pa x postane spodnja meja intervala, v katerem iščemo ničlo funkcije. Postopek bisekcije se nadaljuje vse dokler nismo dovolj blizu ničle funkcije - to pomeni, da je razlika med zgornjo in spodnjo mejo intervala na katerega se nahja ničla funkcije poljubno majhna. Problem lahko nastane, če v našem začetnem intervalu $[a, b]$ ne leži samo ena ničla, ampak dve ali celo več, vendar reševanje teh primerov preseže naše namene v tem poglavju. Opazimo še, da smo v naš program uporabili funkcijo `Pow()`, ki je vključena v knjižnico `Math`. Funkcija `Pow(x, a)` za podani realni števili x in a, vrne vrednost x^a.

Kako upoštevamo zahtevano natančnost?

opis funkcije .toString("F2")

3 Spomin

V prejšnjih poglavjih smo videli, da za vse podatke, ki jih želimo uporabljati pri programiranju potrebujemo spremenljivke. Deklariranje spremenljivk povzroči zasedbo prostora v računalniškem pomnilniku. Spoznali smo tudi nekaj osnovnih podatkovnih tipov: `Integer`, `Single`, `Double` in `String`. Določanje količine potrebnega spomina za podatke in poimenovanje spremenljivk je zelo pomembna naloga pri programiranju.

Velikokrat pri programiranju potrebujemo tudi večje podatkovne strukture, kot so na primer polja za delo z večdimenzionalnimi vektorji in matrike za reševanje sistemov linearnih enačb. Uporaba polj in matrik nam lahko v veliki meri olajša programersko izkušnjo, saj namesto da deklariramo dvajset spremenljivk za shranjevanje dvajset realnih števil, deklariramo eno samo podatkovno polje v katerega lahko shranimo vsa zahtevana števila. V nadaljevanju si bomo na različnih primerih ogledali uporabo takšnih kompleksnejših podatkovnih struktur.

3.1 Niveleta ceste s statičnim poljem

Niveleta ceste je črta, ki prikazuje višinski potek ceste v vzdolžnem prerezu. Glede na podatke o višini posameznih točk lahko določimo v kakšnem naklonu je nek odsek ter ali niveleta ceste pada oz. narašča. Za opis nivelete ceste tako potrebujemo podatke o višini posameznih točk, ki jih ponavadi definiramo po profilih, ki so razmaknjeni na 20 metrov. V tej nalogi torej hočemo shraniti niveleto odseka ceste v dolžini 400 metrov, za kar potrebujemo 20 višinskih točk. V nadaljevanju izračunamo, za koliko se niveleta ceste na tem odseku dvigne.

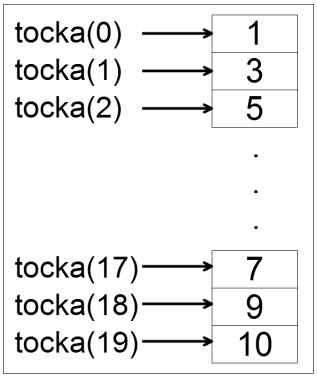
Za hranjenje podatkov potrebujemo podatkovno strukturo, v katero bomo shranili 20 višinskih točk. Višino bomo upoštevali na 1m natančno, kar pomeni, da za shranjevanje posamezne točke potrebujemo spremenljivko podatkovnega tipa `Integer`.

Za shranjevanje vseh podatkov uporabimo podatkovno strukturo polje. Obstajajo statična in dinamična polja. Pri statičnih poljih je število elementov polja določeno v trenutku deklaracije in se ne bo spreminjalo v času delovanja programa. V našem primeru imamo fiksno število točk, ki jih želimo obdelati, zato je najbolje, da uporabimo statično polje. Obstajajo tudi dinamična polja, pri katerih se lahko število elementov dinamično spreminja v času izvajanja programa. Teh bomo spoznali nekoliko pozneje v tem poglavju.

Torej, za shranjevanje 20 točk lahko deklariramo statično polje *tocka* takole:

```
Dim tocka(19)as Integer
```

Deklariramo torej na podoben način kot osnovne spremenljivke, le da ob imenu podamo v okroglih oklepajih zgornjo mejo (največji indeks elementa v polju pomnilniških celic). Indeksi elementov polja tečejo od 0 naprej. Na ta način se v spominu računalnika ustvari polje v katerega lahko shranimo natančno 20 števil: `tocka(0)`, `tocka(1)`, `...` , `tocka(19)`, tako kot prikazuje slika 3.1.



Slika 3.1: Polje števil

Če želimo posamezni točki dodeliti višino v metrih, to dosežemo s prireditvenim stavkom, na primer `tocka(2)= 5`, kar praktično pomeni, da je tretja točka v nizu na (relativni) nadmorski višini 5 m.

Pri programiranju moramo najprej poskrbeti zato, da uporabnik vpiše višine posameznih točk. To najbolje naredimo z uporabo iterativenga stavka For-Next:

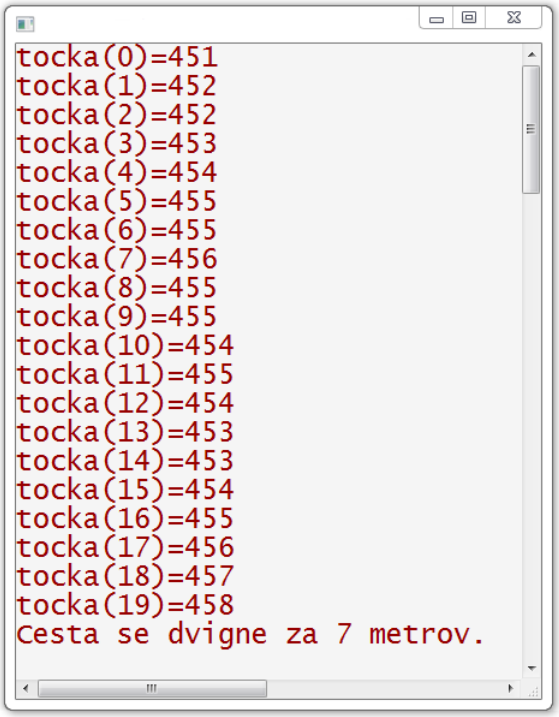
```
        ' vpis podatkov
        For i = 0 To 19
                Console.Write("tocka(" & i & ")=")
                tocka(i) = Console.ReadLine()
        Next
```

Sedaj zapišemo celoten program, ki najprej omogoči vnos podatkov o višini posameznih točk, temu pa sledi izračun in izpis velikosti dviga ali padca nivelete.

Niveleta ceste s statičnim poljem

```
Module Module1

    Sub Main()
        Dim tocka(19) As Integer ' niveleta ceste
        Dim najnizja, najvisja As Integer ' najvišja in najnižja točka
        'vpis podatkov
        For i = 0 To 19
            Console.Write("tocka(" & i & ")=")
            tocka(i) = Console.ReadLine()
        Next
        ' izračun najvišje in najnižje točke
```

```
        najnizja = tocka(0)
        najvisja = tocka(0)
        For i = 1 To 19
            If najnizja > tocka(i) Then
                najnizja = tocka(i)
            End If
            If najvisja < tocka(i) Then
                najvisja = tocka(i)
            End If
        Next
        Console.WriteLine("Cesta se dvigne za " & najvisja - najnizja & " metrov.")
        Console.ReadKey()
    End Sub

End Module
```



Slika 3.2: Niveleta s statičnim poljem

Izdelani program deluje tako kot prikazuje slika 3.2.

3.2 Niveleta ceste z dinamičnim poljem

Pred izdelavo programa ponavadi ne moremo vedeti, koliko točk bo uporabnik želel podati. Če želimo dopustiti možnost spremembe količine rabljenega spomina, potem deklariramo dinamično polje na naslednji način `Dim tocka()as Integer`. Ker pri tem količina zahtevanega pomnilnika ni specificirana, ne bo prišlo do rezervacije spomina v naprej. Pred samim shranjevanjem podatkov, pa moramo še vedno rezervirati prostor v spominu, saj drugače program ne bo imel na razpolago mesta, kam shraniti spremenljivke. Prostor za 10 spremenljivk rezerviramo s programskim stavkom `ReDim tocka(9)`, ki polju *tocka* dodeli nove in prazne pomnilniške celice. Sedaj predelajmo prejšnji program tako, da uporabniku omogočimo, da sam določi število točk.

Niveleta ceste z dinamičnim poljem

```
Module Module1

    Sub Main()
        Dim tocka(), n As Integer ' niveleta ceste in število točk
        Dim najnizja, najvisja As Integer ' najvišja in najnižja točka
        Console.Write("Vpisi stevilo tock ") : n = Console.ReadLine
        ReDim tocka(n - 1) ' rezervacija pomnilniških celic za polje tocka
        'vpis podatkov
        For i = 0 To n - 1
            Console.Write("tocka(" & i & ")=")
            tocka(i) = Console.ReadLine()
        Next
        ' izračun najvišje in najnižje točke
        najnizja = tocka(0)
        najvisja = tocka(0)
        For i = 1 To n - 1
            If najnizja > tocka(i) Then
                najnizja = tocka(i)
            End If
            If najvisja < tocka(i) Then
                najvisja = tocka(i)
            End If
        Next
        Console.WriteLine("Cesta se dvigne za " & najvisja - najnizja & " metrov.")
        Console.ReadKey()
    End Sub

End Module
```

Kot vidimo, sedaj program opravi izračun za poljubno število podanih točk. Opazimo, da to ni najbolj ugodna rešitev, saj moramo predhodno vedeti število vseh točk in tudi sporočiti programu pred samim začetkom vnašanja točk. Veliko boljša rešitev zagotavljanja zadostnega števila polj

je uporaba dinamičnih polj, saj se nam tako število polj posamezne spremenljivke tekom vnosa spreminja.

3.3 Niveleta ceste s sprotnim podaljševanjem

V tej zadnji različici Nivelete ceste uporabniku omogočimo še večjo prožnost pri vnašanju posameznih točk. Uporabnik vnaša višinske točke ne da bi moral skrbeti glede števila teh. Novo izdelani program izgleda takole:

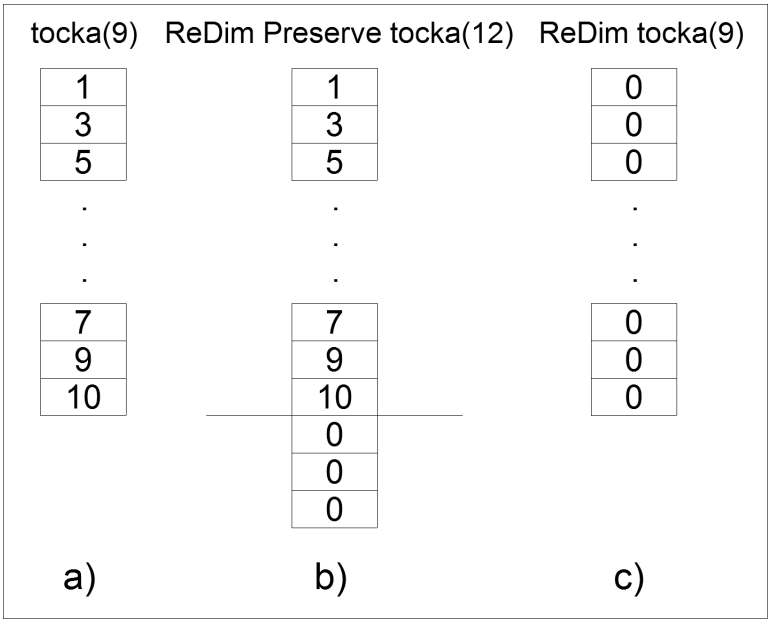
Niveleta ceste s sprotnim podaljševanjem

```
Module Module1

    Sub Main()
        Dim tocka() ' niveleta ceste
        Dim n As Integer = 0 ' števec
        Dim najnizja, najvisja As Integer ' najvišja in najnižja točka
        Dim odgovor As String ' odgovor uporabnika
        'vpis podatkov
        Do
            ReDim Preserve tocka(n)
            Console.Write("tocka(" & n & ")=")
            tocka(n) = Console.ReadLine()
            n = n + 1
            Console.WriteLine("Napisi da, ce zelis nadaljevati: ")
            odgovor = Console.ReadLine
        Loop While odgovor = "Da" Or odgovor = "da" ' izračun najvišje in najnižje
        točke
        najnizja = tocka(0)
        najvisja = tocka(0)
        For i = 1 To n - 1
            If najnizja > tocka(i) Then
                najnizja = tocka(i)
            End If
            If najvisja < tocka(i) Then
                najvisja = tocka(i)
            End If
        Next
        Console.WriteLine("Cesta se dvigne za " & najvisja - najnizja & " metrov.")
        Console.ReadKey()
    End Sub

End Module
```

Za začetek v programu rezerviramo spremenljivko z imenom $tocka()$, ki pa še ne vsebuje nobene celice. Zmeraj, ko uporabnik vnese novo točko, se spremenljivki $tocka()$ doda nova celica, v katero se

nato shrani vnešena točka. To storimo z ukazom `ReDim Preserve tocka(n)`. Razlika med tem ukazom in ukazom `Redim` je v tem, da ukaz `Redim Preserve` ohrani vrednosti v prejšnjih poljih spremenljivke ter doda samo nova prazna polja. Pozorni moramo biti le na stanje števca n, da si slučajno ne prepišemo starih vrednosti.

Denimo, da ima spremenljivka *tocka* že dodeljenih 9 pomnilniških celic od 0 do 8. V trenutku, ko uporabimo ukaz `ReDim Preserve tocka(9)` se ustvari nova pomnilniška celica $tocka(9)$, vse stare vrednosti pa se ohranijo (primer, slika 3.3 (b)). Če bi uporabili ukaz `ReDim tocka(9)` pa dobili povsem novo prazno polje, ki bo imelo 10 pomnilniških celic od 0 do 9 (primer, slika 3.3 (b))



Slika 3.3: ReDim Preserve

Poimenovanje spremenljivk ter uporaba velikih in malih črk

Vse spremenljivke, procedure in funkcije moramo poimenovati dosledno. Naj imena leteh izražajo pomen podatkov za katere jih uporabljamo. Na izbiro imamo poimenovanje v angleščini ali slovenščini. Obstajajo tudi pravila glede uporabe majhnih in velikih črk. Na splošno velja pravilo, da se spremenljivke oziroma imena posameznih predmetov pišejo z malo začetnico, na primer, spremenljivka *ime* ali polje $tocka(9)$, medtem, ko se imena procedur in funkcij pišejo z veliko začetnico, na primer, `WriteLine()` in `ReadLine()`.

3.4 Razvrščanje števil

Izdelajmo program, ki uredi n podanih števil po vrstnem redu in jih nato izpiše. Uporabnik poda dolžino niza števil, ki jih nato vnese v program, ta pa nato izračuna in izpiše števila, ki si sledijo v naraščajočem vrstnem redu.

Števila po naraščajočem vrstnem redu

```
Module Module1

    Sub Main()

        Dim s() As Integer ' niz celih števil
        Dim n As Integer ' dolžina niza
        Dim a As Integer ' pomožna spremenljivka

        Console.Write("Dolzina niza stevil: ") : n = Console.ReadLine
        ReDim s(n)

        'vnos števil
        For i = 0 To n - 1
            Console.Write("Podaj stevilo " & i + 1 & ": ")
            s(i) = Console.ReadLine
        Next

        'razvrščanje
        For i = 0 To n - 2
            For j = i + 1 To n - 1
                If s(j) < s(i) Then
                    a = s(j)
                    s(j) = s(i)
                    s(i) = a
                End If
            Next
        Next

        ' izpis
        For i = 0 To n - 1
            Console.WriteLine(s(i))
        Next
        Console.ReadKey()

    End Sub

End Module
```

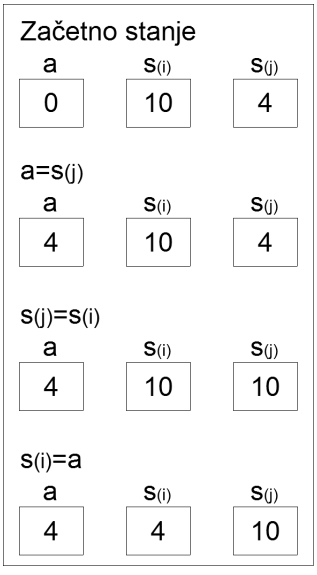
Delovanje programa prikazuje naslednja slika 3.4.

Slika 3.4: Razvrščanje števil

V programu najprej deklariramo spremenljivke: polje s in število elementov polja n. Glede na podano vrednost n ustvarimo polje celih števil s z ukazom `ReDim s(n-1)`. Uporabniku nato omogočimo vnos vseh n števil, ki so zaradi deklaracije omejena na cela števila.

Sedaj se posvetimo algoritmu za razvrščanje števil. Začeli bomo s prvim številom v nizu. Predpostavimo, da je prvo število ($i = 0$) najmanjše število v nizu, nato sistematično pregledamo vsa preostala števila z notranjo zanko `For j= i+1 To n-1`. Ko najdemo katero število, ki je manjše od prvega, ti števili zamenjamo.

Za zamenjavo dveh števil v nizu potrebujemo neko pomožno spremenljivko. V našem primeru je to spremenljivka a, ki jo deklariramo kot `Integer`. Denimo, da je $i = 0$, $j = 1$, $s(i) = 10$, $s(j) = 4$. Postopek zamenjave poteka tako, kot prikazuje slika 3.5. Spremenljivka a najprej pridobi vrednost 2, nato v celico $s(5)$ vpišemo vrednost celice $s(0)$, nakar v celico $s(0)$ vpišemo vrednost 2. Na ta način smo zamenjali vrednosti dveh pomnilniških celic.



Slika 3.5: Menjava števil

Postopek nadaljuje vse dokler ne dosežemo zadnji element niza $s(n-1)$. Šele sedaj smo lahko popolnoma prepričani, da bo na prvem mestu polja s najmanjše število v nizu.

Postopek nadaljujemo s podnizom $s(1), s(2), ..., s(n)$. V tem primeru je $i = 1$, j pa se spreminja od 2 do $n-1$. Po končanem postopku pregledovanja in zamenjave, se na drugem mestu v nizu nahaja drugo najmanše število celotnega niza.

Postopek nadaljujemo sistematično vse dokler ne pridemo do predzadnjega števila, ko je $i = n-2$. Predpostavimo, da je predzadnje število $s(n-2)$ manjše od zadnjega števila v nizu. Če temu ni tako, si bosta ti dve števili zamenjali mesta v nizu, če je vrstni red teh števil pravilen program ne bo naredil nič in bo izstopil iz zunanje zanke.

V programu nam ostane samo še naloga, da izpišemo podana števila v naraščajočem vrstnem redu.

3.5 Transponiranje matrike

V naslednji nalogi si bomo ogledali še možnosti in način dela z matrikami. Naredili bomo preprost program, ki uporabniku omogoči vnos matrike A velikosti 10×5. Program matriko A transponira in izpiše vrednost transponirane matrike. Kot vemo iz matematike, transponiranje matrike pomeni zamenjavo podatkov v vrsticah s podatki iz stolpcev.

Izdelani program vsebuje nekaj pomembnih korakov: deklaracijo matrik in drugih spremenljivk, vnos podatkov, transponiranje ter izpis rezultatov. Posamezni deli programa so prikazani v nadaljevanju.

Deklaracija matrik in drugih spremenljivk

```
' deklaracija spremenljivk in matrik
  Dim m, n As Integer
  Dim A(,) As Single ' originalna matrika
  Dim At(,) As Single ' transponirana matrika

  Console.WriteLine("Transponiranje matrike")
  Console.WriteLine("Podaj stevilo stolpcev "): m = Console.ReadLine
  Console.WriteLine("Podaj stevilo vrstic ") : n = Console.ReadLine
  ReDim A(n - 1, m - 1)
  ReDim At(m - 1, n - 1)
```

V tem delu programske kode, podobno kot pri poljih, deklariramo dve dinamični matriki $A(,)$ za matriko, ki jo poda uporabnik ter $At(,)$ za transponirano matriko. Uporabniku omogočimo vnos števila stolpcev m in vrstic n na kar z uporabo ukaza `ReDim` zasedemo ustrezno število celic v pomnilniku.

Vnos podatkov v matriko

```
' vnos matrike
   For i = 0 To n - 1
       For j = 0 To m - 1
           Console.Write("Podaj clen A(" & i & "," & j & ") = ")
           A(i, j) = Console.ReadLine()
       Next
   Next
```

Vnos elementov matrike je realiziran z dvojno zanko $For - Next$, ki nam ustvari vse kombinacije stevil i iz množice $0, 1, ..., n - 1$ ter j iz množice $0, 1, ..., m - 1$. Po poteku teh programskih stavkov bo celotna matrika shranjena v spremenljivki A.

Naslednji izsek programa predstavlja glavni del - transponiranje matrike. Pri tem upoštevamo osnovno pravilo, da element matrike, ki se nahaja v i-ti vrstici in j-tem stolpcu, je v transponirani matriki v j-ti vrstici in i-tem stolpcu. Po poteku teh programskih stavkov, je v spremenljivki At shranjena transponirana matrika.

Algoritem transponiranja matrike

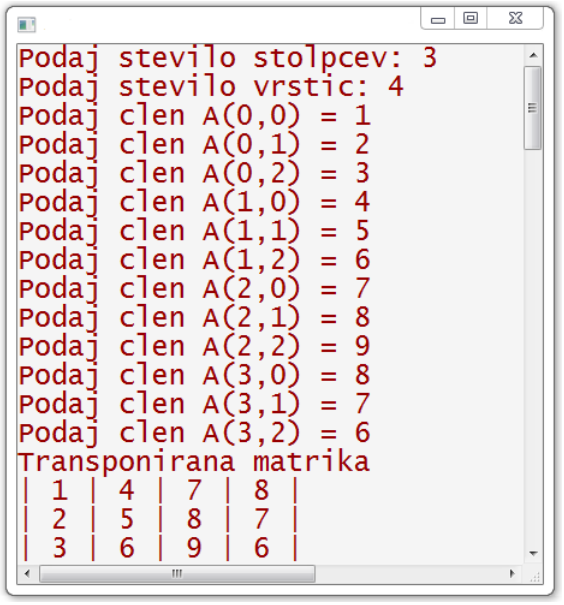
```
   ' transponiranje
   For j = 0 To m - 1
       For i = 0 To n - 1
           At(j, i) = A(i, j)
       Next
   Next
```

Ostane nam samo še izpis transponirane matrike, ki ga opravi naslednji del programske kode.

Izpis matrike

```
' izpis
   Console.WriteLine("Transponirana matrika")
   For j = 0 To m - 1
       Console.Write("| ")
       For i = 0 To n - 1
           Console.Write(At(j, i) & " | ")
       Next
       Console.WriteLine()
   Next
```

Primer delovanja programa je prikazan na sliki 3.6.



Slika 3.6: Transponiranje matrike

4 Procedure in funkcije

Ob izdelavi bolj kompleksnih programov se hitro zgodi, da postane programska koda dolga in nepregledna, zato jo je smiselno razdeliti na posamezne podprograme. Podprogrami so lahko v obliki procedur ali funkcij.

Procedure zapišemo znotraj programskega modula z uporabo programske besede `Sub`, kar pomeni podprogram (Subroutine), zaključi pa se z programskimi besedami `End Sub`. Nato sledi ime podprograma ter morebitni argumenti, ki jih lahko spročimo proceduri ob njenem klicu, na primer:

```
Sub Delaj1(ByVal n As Integer, ByRef A() As Single)
    '...  programska koda
End Sub
```

Funkcije zapišemo na zelo podoben način. Edina razlika med procedurami in funkcijami je v tem, da funkcije vrnejo določen rezultat. Primer funkcije, ki vrne realno število in ima en sam argument, definiran kot celo število, je naslednji:

```
Function Delaj2(ByRef a As Integer) As Single
    Dim rezultat As Single
    '...  programska koda, na primer:
    rezultat = 3.3
End Function
```

Kot smo že opazili, proceduram in funkcijam podajamo argumente. Pri deklaraciji argumentov funkcije lahko uporabljamo ključni besedi `ByVal` in `ByRef`. Ti dve ključni besedi nam omogočata takoimenovan prenos argumentov podprograma po vrednosti ali po referenci.

Vsak programski modul lahko ima svoje globalne in lokalne spremenljivke, tako kot prikazuje naslednji konzolni program.

```
Module Module1
Dim z, v, w As Single

Sub Main()
Dim x as Single
' programski stavki
End Sub

Sub Delaj3(ByRef x as Single)
' ...  programski stavki
End Sub

End Module
```

Vse spremenljivke, ki so deklarirane znotraj podprogramov so lokalne. Na primer spremenljivka x v proceduri Main() ni enaka spremenljivki x, ki je deklarirana kot argument procedure Delaj3(). Ti dve spremenljivki zasedata povsem drugi pomnilniški celici. Spremenljivke, ki so deklarirane zunaj podprogramov pa so globalne spremenljivke. Primer globalnih spremenljivk so z, v in w v zgornjem programu.

Globalne spremenljivke lahko spreminjajo vse procedure in funkcije. Če je program obsežnejši in večina izdelanih procedur in funkcij spreminja vrednosti globalnih spremenljivk lahko programer izgubi kontrolo nad svojim programom. Zato je v programerski praksi ustaljena navada pisati programe brez uporabe globalnih spremenljvk. Odgovor na to, kaj je boljše, pa je vsekakor nekoliko bolj zapleten, saj nekatere algoritme zapišemo najlažje z uporabo globalnih spremenljivk.

Ključna beseda ByVal določa prenos spremenljivke "po vrednosti". To pomeni, da se ob klicu procedure ustvari identična kopija argumenta, s katero naprej podprogram dela. Ključna beseda ByVal pomeni, da podprogram dobi dostop do originalne spremenljivke, ki je bila uporabljena ob klicu procedure. V slednjem primeru, se moramo zavedati, da v podprogramu delamo z originalno spremenljivko. Če jo spremenimo, se sprememba prenese tudi nazaj na podprogram, ki je to proceduro poklical.

V nadaljevanju si oglejmo nekaj primerov procedur in funkcij.

4.1 Izpis prvih n števil

Za začetek izdelajmo program, ki uporablja proceduro za izpis prvih n naravnih števil.

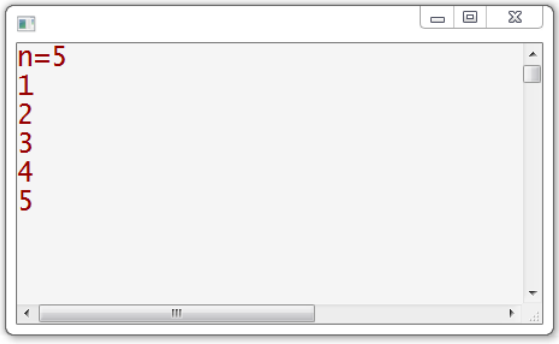
Izpis prvih n števil

```
Module Module1

    Sub Main()
        Dim n As Integer ' uporabnik poda število n
```

```
        Console.Write("n=") : n = Console.ReadLine()
        Izpisi(n) ' klic procedure Izpisi()
        Console.ReadKey()
    End Sub
    ' procedura za izpis prvih n števil
    Sub Izpisi(ByVal n As Integer)
        For i = 1 To n
            Console.WriteLine(i)
        Next i
    End Sub

End Module
```

Ta procedura ima en sam argument, spremenljivko n, ki je celo število. Ključna beseda `ByVal` pomeni, da se ob klicu procedure ustvari identična kopija poslanega argumenta n, ki jo potem uporablja procedura `Izpisi()`. Če želimo to proceduro uporabljati potem jo moramo poklicati iz glavnega podprograma `Main()`. To praktično pomeni, da moramo v glavnem podprogramu napisati ime procedure, ki jo kličemo ter podati vse argumente, ki jih procedura potrebuje za svoje delovanje. Argumenti morajo biti podani v enakem vrstnem redu kot so deklarirani v proceduri, prav tako se morajo ujemati po številu in podatkovnih tipih.

Na koncu si oglejmo na sliki 4.1 delovanje programa.



Slika 4.1: Izpis N števil

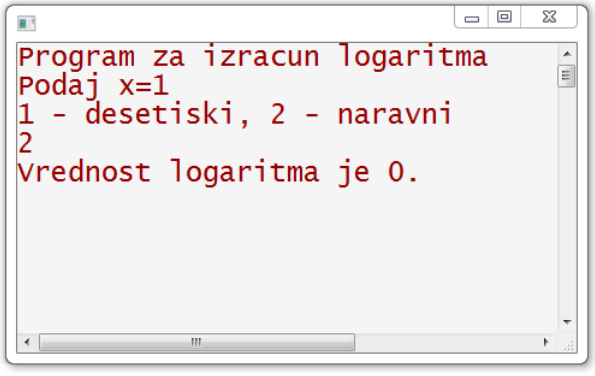
4.2 Program za izračun logaritma

Naslednji program si zamislimo tako, da uporabnik vpiše določeno pozitivno realno število, program izračuna in izpiše vrednost logaritma.

Izračun logaritma

```
Module Module1

    Sub Main()
        Dim x As Single ' podana vrednost
        Dim izbira As Integer  ' tip logaritma
        Console.WriteLine("Program za izracun logaritma")
        Console.Write("Podaj x=") : x = Console.ReadLine
        Console.WriteLine("1 - desetiski, 2 - naravni")
        izbira = Console.ReadLine()
        Console.WriteLine("Vrednost logaritma je " & Logaritem(x, izbira) & ".")
        Console.ReadKey()
    End Sub

    Function Logaritem(ByVal x As Single, ByVal c As Integer) As Single
        Dim rezultat As Single
        If x > 0 Then
            Select c
                Case 1
                    rezultat = Math.Log10(x)
                Case 2
                    rezultat = Math.Log(x)
            End Select
        End If
        Return rezultat
    End Function

End Module
```

Delovanje programa je prikazano na sliki 4.2.



Slika 4.2: Izračun vrednosti logaritma

Vse spremenljivke, ki jih uporabljata procedura `Main()` in funkcija `Logaritem()` so lokalne, kar pomeni, da če tudi v obeh procedurah nastopa spremenljivka x v resnici gre za dve povsem ločeni spremenljivki. Kot smo naredili že večkrat doslej, uporabnika vprašamo, da vpiše eno pozitivno realno število, za katerega bomo izračunali vrednost logaritma. Ponudimo tudi možnost izbire dveh različnih logaritemskih tipov - desetiški in naravni logaritem.

Sledi programski stavek `Console.WriteLine("Vrednost logaritma je "& Logaritem(x, izbira)& ".")`. Program mora pred začetkom izpisovanja rezultata najprej ovrednotiti posamezne elemente besedila med katerimi spada tudi klic funkcije `Logaritem()` z argumentoma x in $izbira$.

Ob klicu funkcije `Logaritem()`, program ustvari identični kopiji spremenljivk x in $izbira$ in ju dodeli spremenljivkama x in c iz funkcije `Logaritem()`. Nato se začne izvajanje funkcije, ki najprej deklarira lokalno spremenljivko $rezultat$. Takoj zatem funkcija preveri izbiro tipa logaritma (glede na vrednost spremenljivke c), izračuna ter vrne izračunano vrednost logaritma. Pri tem se mora podatkovni tip spremenljivke $rezultat$ ujemati s podatkovnim tipom `Single`, ki ga vrne funkcija.

Po uspešno izvedeni funkciji program nadaljuje na mestu, kjer je poklical funkcijo. Sestavi celotno besedilo za izpis, ki vsebuje tudi vrednost logaritma ter ga izpiše.

4.3 Niveleta ceste z uporabo procedur

Sedaj se lotimo izdelave programa za izračun nivele.

```
Module Module1
' tukaj deklariramo globalne spremenljivke, ki jih potrebujejo podprogrami
⋮

    ' glavna procedura
    Sub Main()
        vpisTock() ' klic podprograma za vpis točk
        izracunMaxInMin() ' klic podprograma za izračun najvišje in najnižje točke
        izpisDviga() ' klic podprograma za izpis dviga
        Console.ReadKey()
    End Sub

    Sub vpisTock() 'vpis podatkov
        ⋮
    End Sub

    Sub izracunMaxInMin()
        ⋮
    End Sub

    Sub izpisDviga()
```

```
        ⋮
    End Sub

End Module
```

Mesto deklariranja spremenljivk je zelo pomembno. Če spremenljivko deklariramo znotraj procedure ali funkcije, ji rečemo lokalna spremenljivka. Drugi deli programa je namreč ne vidijo in je zato ne morejo uporabljati. Če pa želimo, da spremenljivko uporablja več podprogramov ali funkcij, jo moramo deklarirati kot globalno. V našem primeru moramo nujno uporabljati globalne spremenljivke, saj morajo vsi trije podprogrami (procedure) dostopati in spreminjati vrednosti spremenljivk: $tocke()$, $najvecji$ in $najmanjsi$.

Podajanje argumentov proceduram in funkcijam

Če procedura (ali funkcija) uporablja argumente, potem moramo natančno definirati tudi njihovo število, podatkovne tipe ter vrstni red podajanja.

Program lahko še nekoliko izboljšamo tako, da ga razbijemo na posamezne funkcionalne enote, kot so recimo funkcije. Izdelani program izgleda takole:

```
Module Module1
    ' tukaj deklariramo globalne spremenljivke, ki jih potrebujejo podprogrami
    Dim tocka(19) As Integer ' niveleta ceste
    Dim najnizja, najvisja As Integer ' najvišja in najnižja točka

    ' glavna procedura
    Sub Main()
        vpisTock() ' klic podprograma za vpis točk
        izracunMaxInMin() ' klic podprograma za izračun najvišje in najnižje točke
        izpisDviga() ' klic podprograma za izpis dviga
        Console.ReadKey()
    End Sub

    Sub vpisTock() 'vpis podatkov
        For i = 0 To 19
            Console.Write("tocka(" & i & ")=")
            tocka(i) = Console.ReadLine()
        Next
    End Sub

    Sub izracunMaxInMin()
```

```
        najnizja = tocka(0)
        najvisja = tocka(0)
        ' izračun najvišje in najnižje točke
        For i = 1 To 19
            If najnizja > tocka(i) Then
                najnizja = tocka(i)
            End If
            If najvisja < tocka(i) Then
                najvisja = tocka(i)
            End If
        Next
    End Sub

    Sub izpisDviga()
        Console.WriteLine("Cesta se dvigne za " & najvisja - najnizja & " metrov.")
    End Sub

End Module
```

Poimenovanje spremenljivk

Vse spremenljivke moramo poimenovati dosledno. Naj ime spremenljivk izraža pomen podatkov, za katere jih uporabljamo. Na izbiro imamo bodisi poimenovanje v angleščini ali v slovenščini. Imena spremenljivk v programih ponavadi pišemo z malo začetnico.

Lokalna spremenljivka obstaja le v času izvajanja podprograma in jo je mogoče uporabiti le v funkciji/proceduri v kateri je bila definirana. Vse lokalne spremenljivke se začnejo izvajati (dobijo začetne vrednosti) vsakič, ko se znova zažene podprogram. Če za svoje delovanje podprogram potrebuje določeno vrednost, ki je dostopna v glavnem programu (v podprogramu `Main()`), potem jo moramo uporabiti kot argument. Temu ustrezno mora biti deklariran podprogram.

Globalne spremenljivke deklariramo zunaj podprogramov in obstajajo v času celotnega delovanja programa. Dosegljive so iz vseh funkcij/procedur. Lahko jih deklariramo s ključno besedo `Private` ali `Dim` ali celo iz funkcij/procedur definiranih v drugih modulih (če jih deklariramo s ključno besedo `Public`).

4.4 Ploščina mnogokotnika

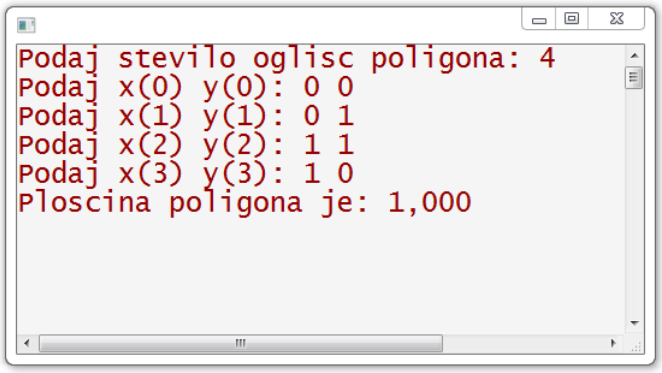
V tej nalogi naredimo program, ki omogoča izračun ploščine poljubnega mnogokotnika glede na podane koordinate. Program razdelimo na več podprogramov: proceduro za vnos podatkov, funkcijo za izračun ter proceduri za izpis in čakanje.

Ploščina poljubnega mnogokotnika

```
Module Module1

    Sub Main()
        'n je število oglišč; x() in y() sta koordinati in p je ploščina
        Dim n As Integer
        Dim x(), y(), pl As Single

        PreberiPodatke(x, y, n)
        pl = Izracunaj(x, y, n)
        Izpisi(pl)
        Pocakaj()
    End Sub

    Sub PreberiPodatke(ByRef x() As Single, ByRef y() As Single, ByRef n As Integer)
        Dim vrsta As String, b() As String, i As Integer
        Console.Write("Podaj stevilo oglisc poligona: ")
        n = Console.ReadLine
        ReDim x(n - 1), y(n - 1)

        For i = 0 To n - 1
            Console.Write("Podaj x(" & i & ") y(" & i & "): ")
            vrsta = Console.ReadLine
            b = vrsta.Split()
            x(i) = b(0)
            y(i) = b(1)
        Next
    End Sub

    'Funkcija izracuna ploscino poligona.
    Function Izracunaj(ByVal x() As Single, ByVal y() As Single, ByVal n As Integer) _
    As Single
        Dim p As Single = 0

        For i = 0 To n - 2
            p = p + Math.Abs((x(i) - x(i + 1)) * (y(i) + y(i + 1)))
        Next
        p = p + Math.Abs((x(n - 1) - x(0)) * (y(n - 1) + y(0))) 'pristejemo zadnji
        clen
        p = p / 2
        Return p
    End Function

    'Procedura izpise izracunano vrednost ploscine
    Sub Izpisi(ByVal ploscina As Single)
        Console.Write("Ploscina poligona je: ")
        Console.Write(ploscina.ToString("F3"))
```

```
    End Sub

    Sub Pocakaj()
        Console.ReadKey()
    End Sub

End Module
```

Program deluje tako kot prikazuje naslednja slika 4.3.



Slika 4.3: Izračun ploščine poljubnega mnogokotnika

Že na samem začetku obravnave programa opazimo, da so vse spremenljivke lokalne. Pri izdelavi programa najprej zapišemo glavno proceduro `Sub Main()`. Tu najprej definiramo spremenljivke za število in koordinate ogljišč. Nadaljujemo s klicanjem naslednjih procedur za branje podatkov, izračun, izpis ter čakanje.

Proceduri `PreberiPodatke(...)` pošljemo vse argumente (x in y koordinate ter spremenljivko za hranjenje števila ogljišč) "po referenci". To praktično pomeni, da potem, ko uporabnik vpiše vrednosti x in y koordinat mnogokotnika ter število točk n, se spremembe vidijo tudi v spremenljivkah x, y in n v proceduri Main().

Funkcija `Izracunaj()` temelji na zelo preprosti formuli za izračun ploščine mnogokotnika, rezultat, ki ga vrne pa je vrednost ploščine.

$$\begin{aligned} p &= p + |(x_i - x_{i+1}) \cdot (y_i - y_{i+1})|;\ i = \{0, 1, 2, \ldots, n-1\} \\ p &= p + |(x_{n-1} - x_0) \cdot (y_{n-1} - y_0)| \\ p &= \tfrac{p}{2} \end{aligned} \tag{4.1}$$

Proceduri `Izpisi()` in `Pocakaj()` sta preprosti proceduri za izpis ploščine in čakanje. Procedura Izpis() prek argumenta pridobi realnost vrednost, ki jo mora izpisati.

Kako organiziramo program?

Program organiziramo v podprograme, ki predstavljajo določene logične enote, ki jih lahko več krat pokličemo. Na primer, procedure so lahko namenjene izvajanju operacij branja ali pisanja, medtem, ko so funkcije lahko namenjene različnim izračunom.

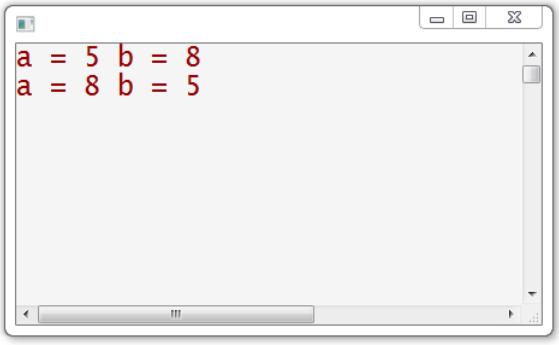
4.5 Zamenjaj

V glavnem programu imamo dve spremenljivki a in b. Želimo izdelati proceduro, ki nam na preprost način zamenja vrednosti teh dveh spremenljivk.

Menjava vrednosti spremenljivk

```
Module Module1

    Sub Main()
        Dim a, b As Integer
        a = 5
        b = 8
        Zamenjaj1(a, b)
        Console.WriteLine("a = " & a & " b = " & b)
        Zamenjaj2(a, b)
        Console.WriteLine("a = " & a & " b = " & b)
        Console.ReadKey()
    End Sub

    Sub Zamenjaj1(ByVal a, ByVal b)
        Dim c As Integer
        c = a
        a = b
        b = c
    End Sub

    Sub Zamenjaj2(ByRef b, ByRef a)
        Dim c As Integer
        c = a
        a = b
        b = c
    End Sub

End Module
```

Slika 4.4 prikazuje delovanje tega programa.



Slika 4.4: Zamenjaj

Naredili smo poskus z dvema procedurama `Zamenjaj1()` in `Zamenjaj2()`. Vse spremenljivke v programu so lokalne. Kot vidimo, procedura `Zamenjaj1()` nima nobenega učinka v glavni program zaradi prenosa argumentov po vrednosti (`ByVal`). Druga procedura `Zamenjaj2()` uspešno zamenja vrednosti obeh spremenljivk zaradi prenosa argumentov po referencah (`ByRef`).

V proceduri `Zamenjaj1()`, se v spominu ustvarita novi lokalni spremenljivki a in b, ki se jima pripišeta vrednosti začetnih spremenljivk a in b. Tako nam zaporedni ukazi `c=a`, `a=b`, `b=c` za zamenjavo obeh spremenljivk spremenijo vrednosti samo novo ustvarjenih spremenljivk. Ko se procedura konča nima nobenega učinka na glavni program.

V proceduri `Zamenjaj2()` pridobi spremenljivka b dostop do pomnilniške lokacije, ki je rezerviran za spremenljivko a iz procedure `Main()`, spremenljivka a iz procedure `Zamenjaj2()` pa dostop do pomnilniške lokacije, ki je rezerviran za spremenljivko b iz procedure `Main()`. Tako lahko procedura `Zamenjaj2()` zamenja vrednosti obeh originalnih spremenljivk a in b iz procedure `Main()`.

4.6 Fibonaccijevo zaporedje

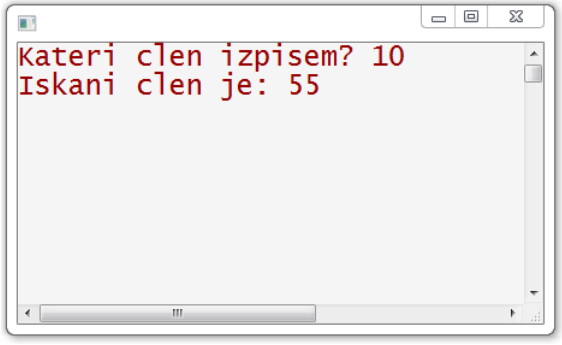
Fibonaccijevo zaporedje (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...), je utemeljil italijanski matematik Leonardo Pisano-Fibonacci. Prva dva člena sta enici, vsak nadaljnji člen pa je seštevek prejšnjih dveh števil iz zaporedja. Program za izračun n-tega števila je podan v nadaljevanju.

Fibonacci

```
Module Module1

    Sub Main()
        Dim n As Integer
        Console.Write("Kateri clen izpisem? ")
        n = Console.ReadLine()
        Console.WriteLine("Iskani clen je: " & Fibonacci(n))
        Console.ReadKey()
    End Sub
```

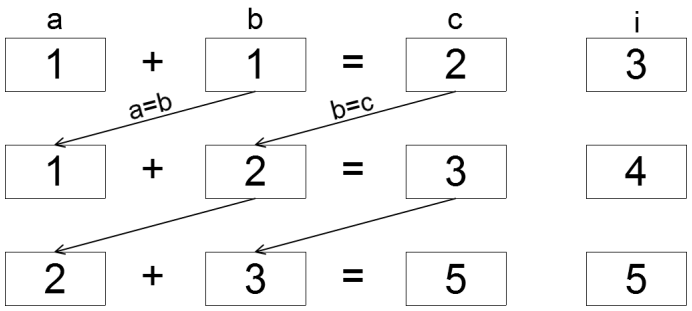
```
    Function Fibonacci(ByVal n As Integer) As Integer
        Dim a, b, c As Integer
        If n = 1 Or n = 2 Then
            Return 1
        Else
            a = 1
            b = 1
            For i = 3 To n
                c = a + b
                a = b
                b = c
            Next
            Return c
        End If

    End Function

End Module
```

Program deluje tako kot je prikazano na sliki 4.5
Slika prikazuje delovanje tega programa.



Slika 4.5: Fibonacci

Pomemben del programa predstavlja funkcija Fibonacci. Najprej razrešimo primer, ko je n=1 ali n=2. Vrednost prvih dveh členov Fibonaccijevega zaporedja je ena, zato funkcija vrne vrednost 1 brez nadaljnjih izračunov. Drugi del stavka If-Then pa se nanaša na možnost, da je n>2. V tem primeru najprej dodelimo začetne vrednosti spremenljivkam a in b, nato pa z For-Next zanko računamo naslednje člene zaporedja po definiciji (`c=a+b$`). To pomeni, da je vsak naslednji člen Fibonaccijevega zaporedja enak seštevku prejšnjih dveh členov. Potem, ko smo izračunalni naslednji člen c, lahko pripravimo spremenljivki a in b za nadaljnje izračune tako, da v spremenljivki a shranimo vrednost zadnjega števila b, v spremenljivko b pa shranimo vrednost sedanjega števila c.



Slika 4.6: Dogajanje v spominu

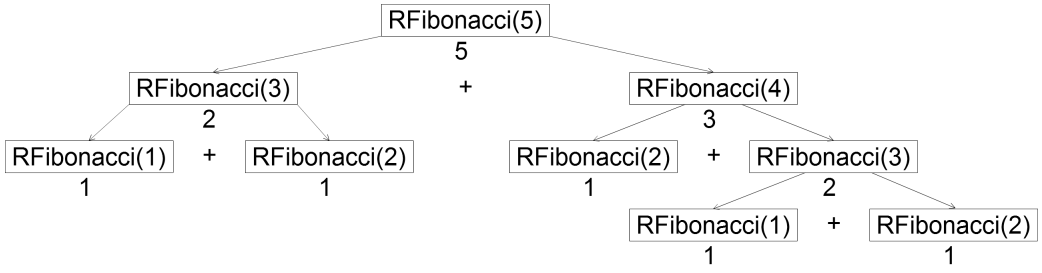
4.7 Rekurzivni Fibonacci

Naslednji primer prikazuje možnost realizacije funkcije za Fibonacci z uporabo rekurzije, kar pomeni, da določena funkcija kliče samo sebe. Funkcijo za rekurzivni Fibonacci preprosto dodamo prejšnjemu programu in jo že lahko začnemo uporabljati.

Rekurzivni Fibonacci

```
Function RFibonacci(ByVal n As Integer) As Integer
    If n = 1 Or n = 2 Then
        Return 1
    Else
        Return RFibonacci(n - 2) + RFibonacci(n - 1)
    End If
End Function
```

Kot vidimo, funkcija sledi definiciji Fibonaccijevega zaporedja. V drugem delu If-Then-Else stavka pokliče samo sebe. Pri izdelavi funkcije smo morali biti zelo pozorni na ustavitveni pogoj. Torej vrednost argumenta n se bo sproti zmanjševala in v enem trenutku bo dosegla vrednosti 1 ali 2. V tem primeru, bo funkcija `RFibonacci()` vrnila vrednost 1.



Slika 4.7: Dogajanje v spominu

Kako preverimo pravilnost delovanja programa?

Pri programiranju je zelo pomembno, da se naši programi ne sesujejo ob "napačnih"vnosih uporabnika. Pravtako moramo sproti odkrivati in odpravljati napake pri njihovem delovanju.

5 Datoteke

Notranji pomnilnik računalnika (angl. Random Access Memory) nam ne omogoča trajnejšega shranjevanja podatkov. Za to potrebujemo drugačno vrsto medija, kot so trdi diski, zgoščenke in podobno. Z drugimi besedami lahko temu rečemo tudi zunanji pomnilnik. Podatki na zunanjem pomnilniku so shranjeni v datotekah. Datoteka predstavlja zaporedje podatkov, ki ima začetek in konec. Vsaka datoteka ima ime in je zapisana v nekem direktoriju na zunanjem pomnilniku. Polno ime datoteke dobimo tako, da združimo pot do datoteke, npr. `"D:\delajtu\"` in njeno ime `"podatki.txt"`. Tako dobimo: `"D:\delajtu\podatki.txt"`. Na ta način operacijski sistem točno ve, kje se začne zaporedje podatkov. Konec datoteke označuje ukaz `EndOfStream`, ki pomeni konec zaporedja podatkov.

Osnovni operaciji z datotekami sta branje podatkov iz datoteke ter pisanje podatkov v datoteko.

Podatki v datotekah so lahko shranjeni v različnih oblikah - v navadni (tekstovni ali XML obliki) ali pa so zakodirani na drugačne načine. Če želimo shranjevati navadno besedilo, potem nam ponavadi zadošča tekstovni tip datoteke. V primeru shranjevanja večje količine podatkov pa je bolj ustrezna tako imenovana binarna datoteka, za katero je značilno, da zakodira podatke na način, da zasedejo čim manj prostora v pomnilniku.

5.1 Matrike in tekstovne datoteke

V nadaljevanju bomo izdelali program, ki bo omogočal shranjevanje matrike v tekstovno datoteko in njeno poznejše branje iz datoteke. Predstavljajmo si, da je naš program del večjega programa, ki omogoča različne operacije z matrikami, kot sta recimo množenje in seštevanje matrik. Problem bomo poenostavili tako, da bomo upoštevali, da matrika vsebuje samo cela števila in je velikosti 5x5.

Razvoja programa se bomo lotili po korakih. Najprej bomo deklarirali globalne spremenljivke in glavno proceduro programa, potem pa se bomo po korakih lotili razvoja pomožnih procedur. Program bo imel meni, ki bo omogočal štiri operacije:

- vpis matrike prek konzole
- izpis matrike v konzolo
- shranjevanje matrike v tekstovno datoteko
- branje matrike iz datoteke

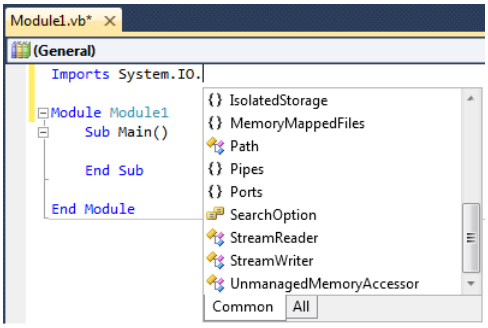
Vsaka od teh operacij je zaključena celota, zato jih bomo izdelali v obliki štirih procedur. Osnutek programa za branje in pisanje matrik v datoteko je predstavljen v nadaljevanju.

Branje in pisanje matrik v datoteko

```
Imports System.IO ' uvozimo knjižnico z razredoma StreamReader in StreamWriter
Module Module1
    Dim matrika1(4, 4) As Integer
    Dim potDoDatoteke As String = "D:\Delajtu\matrika1.txt"
    Sub Main()
        Dim izbira As Integer

        Do
            Console.WriteLine("Program za delo z matriko velikosti 5x5")
            Console.WriteLine("1. vpis matrike prek konzole")
            Console.WriteLine("2. izpis matrike na konzolo")
            Console.WriteLine("3. shranjevanje matrike v datoteko")
            Console.WriteLine("4. branje matrike iz datoteke")
            Console.WriteLine("5. izhod")
            Console.Write("Vpisi izbiro: ")
            izbira = Console.ReadLine

            If izbira = 1 Then VpisMatrikePrekKonzole()
            If izbira = 2 Then IzpisMatrikeNaKonzolo()
            If izbira = 3 Then ShraniMatrikoVDatoteko()
            If izbira = 4 Then BeriMatrikoIzDatoteke()
        Loop Until izbira = 5

        Console.WriteLine("Konec dela z matriko!")
    End Sub

    Sub VpisMatrikePrekKonzole()
       '...
    End Sub

    Sub IzpisMatrikeNaKonzolo()
       '...
    End Sub

    Sub ShraniMatrikoVDatoteko()
       '...
    End Sub

    Sub BeriMatrikoIzDatoteke()
       '...
    End Sub
End Module
```

Če želimo delati z navadnimi tekstovnimi datotekami, potrebujemo določeno programsko orodje, ki rešuje problema branja in pisanja tekstovnih datotek. Taka programa (strokovno poimenovana ra-

zreda) sta StreamReader in StreamWriter. Nahajata se v knjižnici System.IO, ki vsebuje tudi druge razrede namenjene delu z vhodno-izhodnimi operacijami. Če želimo v našem programu uporabiti omenjena razreda moramo v program uvoziti knjižnjico System.IO, tako, da na samem začetku programa zapišemo `Imports System.IO`. Del vsebine te sistemske knjižnice je predstavljen na sliki 5.1.



Slika 5.1: Sistemska knjižnica System.IO

Za delovanje našega programa bomo potrebovali matriko v spominu (notranjem pomnilniku) računalnika, ki jo deklariramo kot statično matriko `Dim matrika1(5,5)As Integer`. Spremenljivka $matrika1$ bo globalna spremenljivka v našem programu, ki jo bodo uporabljali vsi podprogrami.

Za izdelavo programa za zapis matrike v tekstovno datotoko, se moramo odločiti tudi za ime datoteke in za direktorij na disku, kjer bodo podatki shranjeni, na primer: `"D:\Delajtu\matrika1.txt"`. Nato deklariramo globalno spremenljivko $potDoDatoteke$, ki jo bomo uporabljali v izdelanem programu in ji dodelimo začetno vrednost takole:

```
Dim potDoDatoteke As String = "D:\Delajtu\matrika1.txt".
```

Tako zasnovan program lahko v prihodnje izboljšamo na več načinov. Na primer, posamezne procedure bi lahko omogočale shranjevanje poljubno velikih matrik ali da bi omogočali shranjevanje več matrik s katerimi lahko izvajamo tudi določene računske operacije (seštevanje, odštevanje ali množenje).

5.2 Branje matrike iz konzole

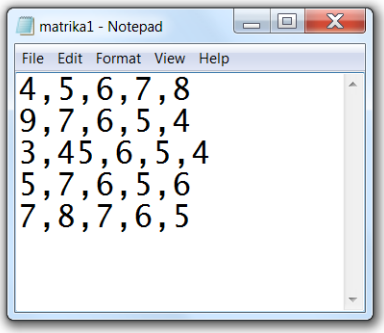
Potem ko smo izdelali glavno proceduro se lahko po vrsti lotimo izdelave pomožnih procedur. Najprej bomo izdelali proceduro, ki omogoča vpis matrike preko konzole. Ker gre za matriko velikosti 5x5, je procedura za vpis podatkov prek konzole zelo preprosta in izgleda takole.

Branje in pisanje matrik v datoteko (Vpis matrike prek konzole)

```
Sub VpisMatrikePrekKonzole()
        For i = 0 To 4
            For j = 0 To 4
                Console.Write("x(" & i & ", " & j & ")= ")
                matrika1(i, j) = Console.ReadLine
            Next j
        Next i
End Sub
```

5.3 Zapis matrike v tekstovno datoteko

Še preden začnemo pisati podatke v datoteko se moramo določiti organizacijo tekstovne datoteke. V našem primeru bo v vsaki vrstici tekstovne datoteke zapisana po ena vrstica iz matrike, podatki pa bodo ločeni z vejicami - slika 5.2.



Slika 5.2: Organizacija podatkov v tekstovni datoteki

V nadaljevanju ustvarimo spremenljivko, ki predstavljala datoteko:
`Dim datotekaZaPisanje As StreamWriter.`
Nato odpremo to datoteko, saj bomo vanjo zapisali podatke:
`datotekaZaPisanje = New StreamWriter(potDoDatoteke).`

Novost pri tem prireditvenem stavku predstavlja uporaba ključne besede `New`. Gre za posebno operacijo, ki v pomnilniku računalnika ustvari predmet tipa `StreamWriter`. Po izvršitvi tega ukaza predmet `datotekaZaPisanje` kaže na začetek povsem prazne datoteke. Več o uporabi ukaza `New` si bomo pogledali v naslednjem poglavju.

Če želimo zapisati število "10"v datoteko, to storimo preprosto z ukazom:
`datotekaZaPisanje.WriteLine(10).`
V primeru, da pa želimo zapisati celotno matriko potrebujemo iterativna stavka `For-Next`, ki nam omogočita zapis vseh podatkov ločene z vejicami takole:

```
For i = 0 To 4
            For j = 0 To 3
                datotekaZaPisanje.Write(matrika1(i, j) & ",")
            Next j
            datotekaZaPisanje.WriteLine(matrika1(i, 4)) ' zaključimo vrstico brez
            vejice
Next i
```

Datoteko zapremo z ukazom `datotekaZaPisanje.Close()`. Ta ukaz postavi znak za konec datoteke `EndOfStream` na konec datoteke in zaključi s pisanjem v datoteko.

Končana procedura za zapis matrike v datoteko izgleda takole:

Branje in pisanje matrik v datoteko (Shrani matriko v datoteko)

```
Sub ShraniMatrikoVDatoteko()
     ' deklariramo spremenljivko, ki bo predstavljala datoteko v notranjem
     pomnilniku
     Dim datotekaZaPisanje As StreamWriter
     ' odpremo datoteko za pisanje
     datotekaZaPisanje = New StreamWriter(potDoDatoteke)
     ' zapišemo podatke
     For i = 0 To 4
         For j = 0 To 3
             datotekaZaPisanje.Write(matrika1(i, j) & ",")
         Next j
         datotekaZaPisanje.WriteLine(matrika1(i, 4)) ' zaključimo vrstico brez
         vejice
     Next i
     ' zapremo datoteko
     datotekaZaPisanje.Close()

 End Sub
```

5.4 Branje matrike iz tekstovne datoteke

Za branje matrike iz tekstovne datoteke potrebujemo predmet, ki nam bo omogočal branje iz datoteke. Najprej deklariramo spremenljivko `Dim datotekaZaBranje As StreamReader`, nato pa z uporabo operacije `New` ustvarimo predmet in ga dodelimo spremenljivki.

`datotekaZaBranje = New StreamReader(potDoDatoteke).`

Sedaj je datoteka pripravljena za branje.

Deklarirajmo še spremenljivko `prebranaVrstica`, ki predstavlja prebrano vrstico iz datoteke in je podatkovnega tipa `String`. Če datoteka ni prazna potem prvo vrstico iz datoteke preberemo na naslednji način

`prebranaVrstica = datotekaZaBranje.ReadLine().`

Problem nastopi, ker je v posamezni vrstici v datoteki shranjeno več podatkov, ločenih z vejico. Prva prebrana vrstica je na primer takšna `prebranaVrstica = "4,5,6,7,8"`. To so po vrsti elementi `matrika1(0,0), matrika1(0,1), matrika1(0,2), matrika(0,3)in matrika(0,4)`. Toda, kako po vrsti dodelimo prebrane vrednosti posameznim celicam v matriki? Prebrano vrstico moramo najprej razbiti na posamezne elemente z operacijo `Split()` glede na ločilo (v našem primeru vejico). Operacija `prebranaVrstica.Split(",")` torej razbije prebrano vrstico na več tekstovnih nizov in sicer "4", "5", "6", "7"in "8". Da bi te podatke shranili v notranjem pomnilniku potrebujemo polje podatkovnega tipa String s petimi celicami, ki jo deklariramo takole `Dim razbitje(4)As String`. Sedaj lahko posamezne vrednosti dodelimo polju razbitje(4) z enim samim prireditvenim stavkom `razbitje = prebranaVrstica.Split(",")`. Po izvedbi tega stavka bo situacija v pomnilniku naslednja `razbitje(0)="4"`, `razbitje(1)="5"`, `razbitje(2)="6"`, `razbitje(3)="7"`, `razbitje(4)="8"`.

Preostane nam samo dodeliti posamezne prebrane vrednosti celicam matrike. Na primer, s prireditvenim stavkom `matrika1(0,0)= razbitje(0)`, v celico matrika1(0,0) vpišemo vrednost "4". Na tem mestu velja opozoriti, da je na levi strani enačbe podatek tipa String, na desni strani enačbe pa celica tipa Integer. To pomeni, da najprej pride do pretvorbe iz podatkovnega tipa String v Integer, torej iz "4"v 4, nato pa se lahko vrednost vpiše v celico `matrika1(0,0)`. Če bi v celici `razbitje(0)` pisalo kaj drugega, na primer `"Dobro jutro!"`, bo program ob pretvorbi iz podatkovnega tipa `String` v `Integer` gotovo javil napako. Procedura za branje podatkov iz datoteke izgleda takole:

Branje in pisanje matrik v datoteko (Beri matriko iz datoteke)

```
Sub BeriMatrikoIzDatoteke()
       ' deklariramo spremenljivko, ki bo predstavljala datoteko v notranjem
       pomnilniku
       Dim datotekaZaBranje As StreamReader
       Dim prebranaVrstica As String
       Dim razbitje(4) As String
       ' odpremo datoteko za branje
       datotekaZaBranje = New StreamReader(potDoDatoteke)
       ' branje iz datoteke
       For i = 0 To 4
           prebranaVrstica = datotekaZaBranje.ReadLine()
           razbitje = prebranaVrstica.Split(",")
           For j = 0 To 4
               matrika1(i, j) = razbitje(j)
           Next
       Next i
   End Sub
```

6 Numerična matematika

6.1 Reševanje sistema linearnih enačb

V inženirstvu pogosto pridemo do situacije, ko moramo reševati sisteme linearnih enačb. Pogosto je potrebno reševati tudi sisteme nelinearnih enačb, vendar to preseže zahteve našega tečaja programiranja. Pred izumom računalnikov so matematiki in inženirji sisteme linearnih enačb reševali na roko, danes pa lahko napišemo programe s katerimi avtomatiziramo celoten postopek reševanja. Pri reševanju kompleksnejših problemov je seveda možnih več načinov reševanja, v vsakem primeru pa moramo biti zelo sistematični, saj lahko hitro pride do napake. Za začetek si oglejmo kako je zastavljen problem.

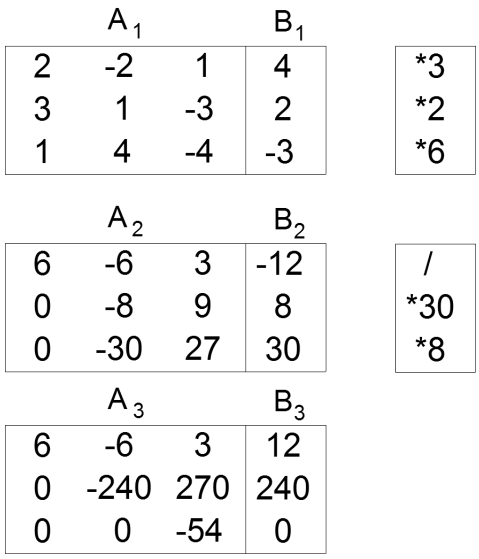
Podan je sistem n linearnih enačb:

$$\sum_{j=0}^{n-1} a_{ij} \cdot x_j = b_i, i = 0, 1, ..., n-1 \tag{6.1}$$

Če so podani vsi a_{ij} in b_i, lahko izračunamo neznane vrednosti $x_0, ..., x_{n-1}$.

Eno izmed metod za izračun teh vrednosti je razvil in uporabljal nemški matematik Carl Friedrich Gauss in je znana kot metoda postopnega eliminarnja neznank oziroma Gaussova eliminacijska metoda. Metoda je bila implementirana v veliko različnih programskih jezikov, kot je naprimer Pascal (Niklaus Wirth, Računalniško programiranje 1. del). Da bi lažje razumeli celoten postopek Gaussove eliminacije, si najprej oglejmo reševanje sistema treh linearnih enačb po korakih.

Kot je prikazano na sliki 6.1, moramo najprej vse enačbe zapisati v matrično obliko z matrikama A_1 in B_1. V principu moramo dobiti zgornjo trikotno matriko matrike A, da bomo lahko rešili sistem enačb. Najprej množimo prvo vrstico matrik s 3, drugo vrstico z 2 in tretjo vrstico s 6. Nato od druge in tretje vrstice odštejem prvo vrstico ter dobim stanje matrik A_2 in B_2. Kot lahko vidimo, smo v prvem stolpcu druge in tretje vrstice dobili ničli. Nadaljujemo tako, da drugo vrstico matrike A_2 in B_2 pomnožimo s 30, tretjo vrstico pa z 8. Nato odštejemo drugo vrstico od tretje in dobimo stanje matrik A_3 in B_3. Tako smo prišli do konca eliminacijskega postopka, saj imamo v prvem stolpcu matrike A_3 dve ničli, v drugem pa eno. Iz tega lahko gremo na proces povratne substitucije.



Slika 6.1: Postopek postopne eliminacije neznank

S procesom povratne substitucije določimo vrednosti neznank. Iz matrik A_3 in B_3 zapišemo enačbe 6.2. Vidimo, da lahko najhitreje pridemo do rešitve, če začnemo s tretjo enačbo, iz katere dobimo rešitev $x_3 = 0$. Postopek nadaljujemo in dobimo še preostali rešitvi $x_2 = -1$ in $x_1 = 1$.

$$\begin{aligned} 6 \cdot x_1 + -6 \cdot x_2 + 3 \cdot x_3 &= 12 \\ 0 \cdot x_1 + 240 \cdot x_2 - 270 \cdot x_3 &= -240 \\ 0 \cdot x_1 + 0 \cdot x_2 + 54 \cdot x_3 &= 0 \end{aligned} \tag{6.2}$$

Sedaj si lahko ogledamo delovanje Gaussove metode na splošno. V prvem koraku izrazimo neznanko x_0 iz enačbe 6.1 pri $i = 0$. Dobimo naslednji izraz:

$$x_0 = \left(b_0 - \sum_{j=1}^{n-1} a_{0j} \cdot x_j \right) / a_{00} \tag{6.3}$$

Nato zamenjamo x_0 v preostalih $n-1$ enačbah, pri čemer dobimo sistem $n-1$ linearnih enačb z $n-1$ neznankami x_1, x_2, ... , x_{n-1}. Temu postopku pravimo eliminacijski korak. Če ga ponovimo $n-1$ krat, bomo dobili reducirani sistem ene enačbe z eno neznanko, ki ga lahko rešimo neposredno.

V nadaljevanju bomo sestavili program, ki bo natančno opisoval ta postopek.
Da lahko to dosežemo moramo najprej določiti splošen zapis za k-ti eliminacijski korak.

Imamo sistem $n - k + 1$ linearnih enačb

$$\sum_{j=k}^{n-1} a_{ij}^{(k-1)} \cdot x_j = b_i^{(k-1)} i = k-1, ..., n-1 \tag{6.4}$$

iz tega sistema izračunamo nove koeficiente $a_{ij}^{(k)}$ ter $b_i^{(k)}$ in tako dobimo sistem $n-k$ enačb.

$$\sum_{j=k}^{n} a_{ij}^{(k} \cdot x_j = b_i^{(k)} i = k, ..., n-1 \tag{6.5}$$

Te koeficiente dobimo kot linearne kombinacije k-te in i-te enačbe (vrstice) in sicer

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - (a_{k-1j}^{(k-1)} / a_{k-1k-1}^{(k-1)}) \cdot a_{ik-1}^{(k-1)} \\ b_i^{(k)} &= b_i^{(k-1)} - (b_{k-1}^{(k-1)} / a_{k-1k-1}^{(k-1)}) \cdot a_{ik-1}^{(k-1)} \end{aligned} \tag{6.6}$$

pri $i, j = k, ..., n-1$. V naslednjem koraku k-to enačbo odštejemo od i-te enačbe potem, ko jo pomnožimo s faktorjem, ki ga izberemo tako, da velja za $j = k-1$ in za vse i naslednja formula

$$a_{ik}^{(k)} = a_{ik-1}^{(k-1)} - (a_{k-1k-1}^{(k-1)} / a_{k-1k-1}^{(k-1)}) \cdot a_{ik-1}^{(k-1)} = 0 \tag{6.7}$$

To pomeni, da so v novem sistemu vsi koeficienti neznanke x_{k-1} nič, s čimer x_{k-1} dejansko izločimo. Potem je torej dejanski izračun koeficientov $a_{ik-1}^{(k)}$ nepotreben. Po $n-1$ korakih reducirani sistem enačb izgleda takole

$$a_{n-1n-1}^{(n-1)} \cdot x_{n-1} = b_{n-1}^{(n-1)} \tag{6.8}$$

Iz sistema 6.8 lahko takoj izračunamo x_{n-1}. Preostale neznanke dobimo z zamenjavo že dobljenih neznank v prej izračunanih enačbah. npr. x_{n-2} dobimo z zamenjavo x_{n-1} v enačbi

$$a_{n-2,n-2}^{(n-2)} \cdot x_{n-2} + a_{n-2,n}^{(n-2)} \cdot x_n = b_{n-2}^{n-2} \tag{6.9}$$

Temu postopku pravimo korak povratne substitucije. Na splošno ima k-ti korak naslednjo obliko

$$x_k = \left(b_1^{(k)} - \sum_{j=k+1}^{n} a_{ij}^{(k)} \cdot x_j \right) / a_{ik}^{(k)} \tag{6.10}$$

pri vseh takih i, za katere velja $k \leq i \leq n$. (Iz razlogov, ki bodo razumljivi nekoliko kasneje ponavadi izberemo $i = k$.) Vendar si zapomnimo, da je zaporedje korakov povratne substitucije določeno s tem, da so za izračun x_k potrebne že izračunane vrednosti $x_{k+1}, ..., x_n$.

Pri sestavljanju primernega programa za ta postopek je zelo pomembno, da se zavedamo naslednje lastnosti opisanega algoritma - pri računanju koeficientov $a^{(k+1)}$ in $b^{(k+1)}$ potrebujemo le koeficiente $a^{(h)}$ in $b^{(h)}$ pri $h = k$ in nobenega pri $h \leq k$.

Ali potem zadostujeta le dve spremenljivki A in B za vse $a^{(k)}$ in $b^{(k)}$? Če želimo odgovoriti na to vprašanje, moramo pogledati, kateri koeficienti so potrebni v kasnejšem koraku povratne substitucije. Toda videli smo že, da lahko uporabimo katero koli enačbo z indeksom $i = k, ..., n$. Če uporabimo k-to $(i = k)$ za izračun x_k, potem so na razpolago natanko tiste vrstice A in B, ki vsebujejo koeficienta

$a_{ij}^{(k+1)}$ in $b_j^{(k+1)}$ po vrsti. Torej po ena spremenljivka A in B zadostujeta zato, da hranimo po vrsti $a_{ij}^{(1)}$, $a_{ij}^{(2)}$, ... , $a_{ij}^{(n)}$ in $b_i^{(1)}$, $b_i^{(2)}$, ..., $b_i^{(1)}$.

S tako odločitvijo lahko znatno zmanjšamo potrebe po pomnilniku. Namesto $n^2+(n-1)^2+...+2^2+1^2 = 1/6(2n^3+3n^2+n)$ pomnilniških enot za $a^{(1)}$, $a^{(2)}$, $a^{(n)}$ in $n+(n-1)+...+2+1 = 1/2(n^2+n)$ za $b^{(1)}$, $b^{(2)}$, ..., $b^{(1)}$, na le n^2+n enot za oboje, A in B.

Ko gre za reševanje večjih sistemov linearnih enačb, je prihranek spomina zelo pomemben, zato je zgornji premislek izredno koristen. Podobno lahko sklenemo, da lahko vrednosti x_j in b_i uporabljajo skupne pomnilniške celice. Iz tega razloga ni potrebno uvesti spremenljivke X.

Na podlagi pridobljene teoretične podlage lahko naredimo več izvedb programa za reševanje sistema linearnih enačb. V nadaljevanju si oglejmo nekaj teh načinov.

Prvi korak

Prva izvedba programa bi bila nekako takole.

```
Module Module1

    Sub Main()
   ' deklaracije

   ' preberi vrednosti A in B

   For k = 1 to n-1
      ' izračunaj a^(k+1) in b^(k+1) iz a^(k) in b^(k) glede na (6.6)
   Next k
   k=n
   Do
      ' izračunaj x_k glede na 6.10
      k=k-1
   Loop While k>0 ' ali Loop Until k=0, preveri!

    End Sub

End Module
```

Drugi korak

Program lahko napišemo tudi tako, da začnemo s pisanjem izračuna $a^{(k+1)}$ in $b^{(k+1)}$ iz $a^{(k)}$ in $b^{(k)}$ glede na (6.6):

```
For i = k+1 to n
   ' izračunaj i-to vrstico
Next i
```

Sedaj opazimo, da je pri računanju vrednosti $a_{ij}^{(k+1)}$ (pa tudi $b_i^{(k+1)}$ na podlagi 6.6 faktor $a_{kj}^{(k)}/a_{kk}^{(k)}$ (oziroma $b_k^{(k)}/a_k^{(k)}$) neodvisen od kontrolne spremenljivke i. Upoštevati moramo tudi pravilo, da se moramo izogibati večkratnemu računanju izrazov z nespremenljivimi deli, zato potegnemo deljenje z $a_{kk}^{(k)}$ izven For stavka.

Rezultate deljenja moramo tudi nekje spraviti. Ena od možnosti je, da preprosto zamenjamo $a_{kj}^{(k)}$ in $b_k^{(k)}$ z $a_{kj}^{(k)}/a_{kk}^{(k)}$ in $b_k^{(k)}/a_{kk}^{(k)}$ po vrsti. Odločitev lahko sprejmemo šele potem, ko pogledamo kakšen bo učinek tega na korak povratne substitucije. Zavedamo se, da množenje vseh koeficientov neke enačbe z istim faktorjem ne spremeni vrednosti neznank. Poleg tega pri deljenju z $a_{kk}^{(k)}$ dobimo v primeru samega $a_{kk}^{(k)}$ vrednost 1, torej, postane deljenje pri koraku povratne susbstitucije nepotrebno. Predlagana sprememba je zato dopustna, istočasno pa tudi koristna.

S tem dobimo tretjo izvedbo programa za postopek eliminacije.

Tretji korak

```
For k = 1 to n-1
   p = 1 / A(k, k)
   For j = k+1 to n
      A(k, j)= p * A (k, j)
      B(k) = p * B(k)
      For i = k+1 to n
         ' izračunaj ai na k+1 in bi na k+1 glede na enačbo ...
      Next i
   Next j
Next i
```

Manjkajoči del programa razvijemo na podlagi ugotovitve, da $A(k, j)$ predstavlja vrednost $a_{kj}^{(k)}/a_{kk}$ na k-tem koraku.

Četrti korak

```
For i = k+1 to n-1
   For j = k+1 to n
      A(i, j)= A(i, j) - A(i, k) * A (k, j)
   Next j
   B(i) = B(i) - A(i, k) * B(k)
Next i
```

Sedaj bom naprej razvijali fazo povratne substitucije v prvi izvedbi programa (glej zgoraj). Ta ima obliko zaporednega odštevanjana podlagi (6.10). Deljenje z $a_{kj}^{(k)}$ je bilo že narejeno v fazi eliminacije.

Peti korak

```
t=B(k)
   For j = k+1 to n
      t= t - A(k, j) * X (j)
   Next j
   X(k) = t
Next i
```

Opazimo še, da smo tukaj upoštevali pravilo, da stavek For nima nobenega vpliva, če je zapisana zgornja meja manjša od začetne vrednosti kontrolne spremenljivke. Celotni program za reševanje sistema linearnih enačb je naslednji... V tem programu se izvedejo n namesto $n-1$ eliminacijskih korakov - n ti korak potrebujemo, ker vsebuje deljenje $b_n^{(n)}$ z $a_{nn}^{(n)}$.

Program za reševanje sistemov linearnih enačb po Gaussovi metodi

```
Module Module1

    Sub Main()
        Dim n As Integer = 10 ' število enačb
        Dim i, j, k As Integer
        Dim p, t As Double
        Dim A(10, 10) As Double
        Dim b(10), X(10) As Double
        ' določi vrednosti vhodne matrike A in vektorja b

        ' Gaussov eliminacijski postopek
        For k = 1 To 10
            p = 1 / A(k, k)
```

```
            For j = k + 1 To n
                A(k, j) = p * A(k, j)
            Next
            b(k) = p * b(k)
            For i = k + 1 To n
                For j = k + 1 To n
                    A(i, j) = A(i, j) - A(i, k) * A(k, j)
                Next
                B(i) = B(i) - A(i, k) * b(k)
            Next
            k = n
            Do
                t = b(k)
                For j = k + 1 To n
                    t = t - A(k, j) * X(j)
                Next
                X(k) = t
                k = k - 1
            Loop While k > 0
        Next
        ' Izpiši vrednosti X(1), X(2), ..., X(n)
    End Sub

End Module
```

V tem algoritmu na določenih mestih nastopa tudi operacija deljenja. Če je kateri delitelj enak nič program javi napako. Tudi v primerih, ko gre za deljenje z zelo majhnimi vrednostmi, je lahko rezultat operacije tako velik, da ga ni možno shraniti v razmeroma majhnem rezerviranem prostoru v pomniliniku.

Tovrstne težave pri reševanju sistemov linearnih enačb rešujemo s pivotiranjem. Pivotiranje je postopek s katerim se izognemo ničelnim deliteljem oziroma v vsakem naslednjem koraku izbiramo enačbe, ki imajo neničelne delitelje. Neničelne delitelje imenujemo pivote (oziroma opore). Na vsakem eliminacijskem koraku izberemo tisto enačbo (iz preostalih enačb), ki nam omogoči največjega možnega pivota. Če na določenem eliminacijskem koraku ne moremo najti neničelnega delitelja potem pravimo, da je sistem enačb singularen oz. da nima ene same rešitve. Če na določenem eliminacijskem koraku ne moremo najti zadosti velikega neničelnega delitelja potem pravimo, da je sistem enačb slabo pogojen.

Numerični algoritmi

Z razvojem algoritmov za natančno računanje in reševanje različnih matematičnih problemov s pomočjo računalnikov se ukvarja področje numerične matematike. Nekaj dobro raziskanih problemov, ki jih na ta način računamo so:

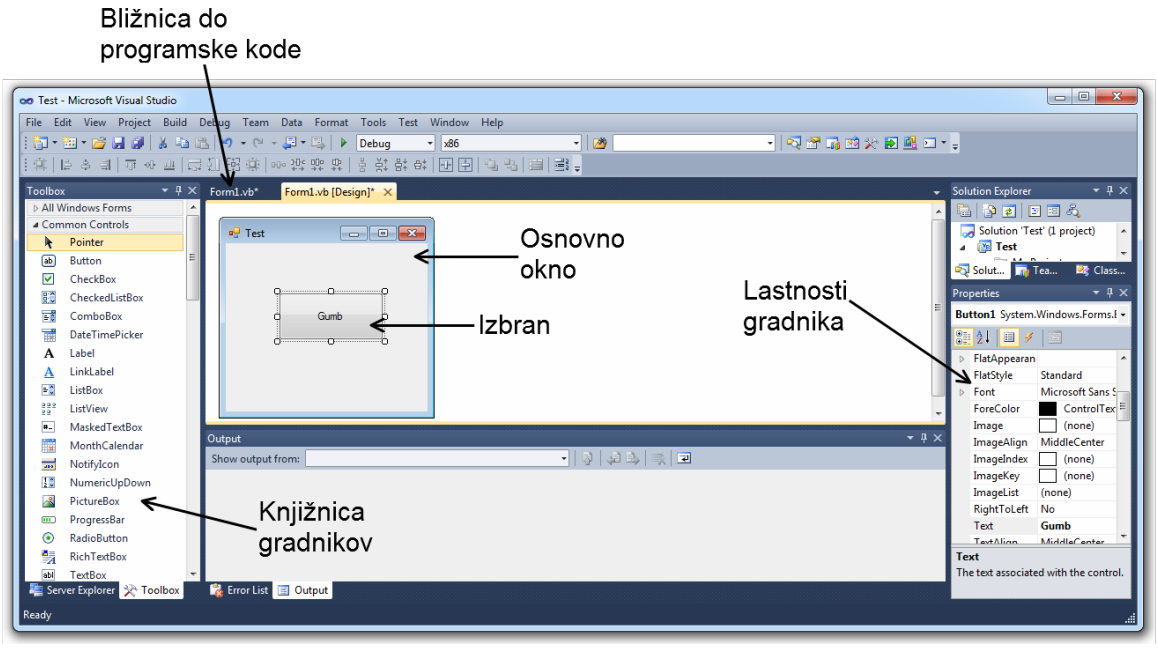
- sistemi linearnih enačb,
- lastne vrednosti matrik,
- nelinearne enačbe,
- interpolacija, odvajanje in integriranje,
- aproksimacija funkcij,
- običajne in parcialne diferencialne enačbe itd.

7 Okenske aplikacije

Do sedaj smo pisali programe, s katerimi smo komunicirali preko konzolnega vmesnika. Pri kompleksnejših programih takšen vmesnik ne zadostuje, saj je omejena komunikacija z uporabnikom, poleg tega pa ne more prikazovati bolj kompleksnih grafičnih elementov.

Tehnologija izdelave okenskih aplikacij poteka v dveh fazah. V prvi fazi izdelamo vizualno podobo aplikacije, določimo imena in druge lastnosti grafičnih elementov oziroma gradnikov (npr. gumbov, polj za vnos podatkov ipd.) iz katerih je sestavljena aplikacija, v drugi fazi določimo kako naj aplikacija reagira na različne dogodke (premik miške, klik na gumbu, zahteva po risanju grafa in podobno).

Za izdelavo aplikacij uporabljamo predlogo Windows Forms Application. Ob izbiri te predloge, nam program Visual Studio sam zgenerira osnovno uporabniško okno, v katerega nato dodamo in razporedimo potrebne gradnike.



Slika 7.1: Oblikovanje uporabniškega okna

Osnovni gradniki so, na primer: Label, TextBox, Button, CheckBox, PictureBox itd. Teoretično

gledano, gradniki predstavljajo razrede predmetov. Na primer, gradnik Button je razred vseh gumbov, ki jih lahko uporabljamo v aplikacijah. Posamezne primerke gradnikov ustvarimo tako, da jih preprosto povlečemo iz orodjarne (ToolBox) in jih postavimo na samo predlogo okna grafičnega vmesnika. S klikom na predmet le tega izberemo in se nam tako omogoči spreminjanje njegovih lastnosti. Le-te lahko uporabnik spreminja in s tem prilagaja podobo same aplikacije.

Zelo pomembno je, da primerke različnih predmetov primerno poimenjujemo. Ime je lahko sestavljeno iz oznake tipa gradnika (na primer, btn za Button ter oznako za vsebino, npr. Izracun). Tako dobimo enolično poimenovanje za določen gumb, v našem primeru $btnIzracun$. Seznam vseh gradnikov dobimo, če v meniju View izberemo Toolbox. Lastnosti gradnikov nastavljamo v oknu Properties.

Primerno poimenovanje spremenljivk nam tako olajša drugo fazo programiranja, izdelavo dogodkovne procedure, ko moramo določiti kako mora aplikacija reagirati na različne dogodke. Razvoj okenskih programov si bomo v nadaljevanju ogledali tako, da bomo začeli z najbolj osnovnimi gradniki, ki omogočajo zajemanje podatkov in izpisovanje rezultatov in bomo postopoma stopnjevali tako, da bomo omogočili risanje linijskih konstrukcij in grafov ter dinamično risanje oziroma simulacije na podlagi fizikalnih zakonov.

7.1 Ploščina pravokotnika

Naš cilj je izdelati program za izračun ploščine pravokotnika z uporabo predloge Windows Forms Application namesto z uporabo konzolnega okna. Grafični uporabniški vmesnik izdelamo tako, da na pripravljeno okno (Form1) postavimo ustrezne gradnike, ki jim spremenimo lastnosti, da dosežemo boljši izgled ter si olajšamo delo pri kasnejšem programiranju. Za izdelavo našega programa uporabimo naslednje gradnike, ki smo jim ustrezno spremenili lastnosti:

Tip gradnika	*Lastnost*	*Privzeta vrednost*	*Naša nastavitev*
Form	Text	Form1	Ploščina pravokotnika
Label	Text	Label1	Dolžina stranice a
Label	Text	Label2	Dolžina stranice b
Label	Text	Label3	Ploščina pravokotnika je:
Label	Text	Label4	(prazno)
	(name)	Label4	lbl4
TextBox	(name)	TextBox1	tbStranicaA
	Text	(prazno)	[cm]
TextBox	(name)	TextBox2	tbStranicaB
	Text	(prazno)	[cm]
Button	(name)	Button1	btnIzracun
	Text	Button1	Izračunaj

Vsak gradnik poimenujemo tako, da bomo kasneje med programiranjem jedra programa natančno vedeli, za kateri gradnik gre in kje leži. Na ta način si olajšamo samo programiranje.

Za program, ki bo omogočal izračun ploščine pravokotnika potrebujemo napise (Labels), ki bodo sporočali kam vnesti posamezno dolžino stranice pravokotnika in izračunano ploščino pravokotnika.

Vedno stremimo k temu, da uporabimo minimalno število gradnikov in ne pretiravamo z njihovo velikostjo, saj lahko le tako izdelamo pregleden in uporabniku prijazen grafični vmesnik. Ko so gradniki grafičnega uporabniškega vmesnika postavljeni, izdelamo še tako imenovane dogodkovne procedure za dogodke na katere želimo odgovoriti v programu. V našem primeru je to le en dogodek, ko uporabnik pritisne na gumb $btnIzracun$ za izračun ploščine.

Dogodkovno proceduro izdelamo tako, da v modelu grafičnega vmesnika dvakrat kliknemo na gradnik btnIzracun. V razvojnem okolju se odpre novo okno v katerem se vidi prazna procedura, ki jo sedaj lahko dopolnimo. Najprej deklariramo potrebne spremenljivke, nato iz gradnikov TextBox določimo vrednosti tem spremenljivkam. Nadaljujemo z izračunom ploščine, ki jo izpišemo v gradnik `lbl4`.

```
Public Class PloscinaPravokotnika

    Private Sub btnIzracun_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnIzracun.Click
        Dim a, b As Single
        Dim p As Single

        a = tbStranicaA.Text
        b = tbStranicaB.Text
        p = a * b

        lbl4.Text = p & " cm^2"

    End Sub
End Class
```

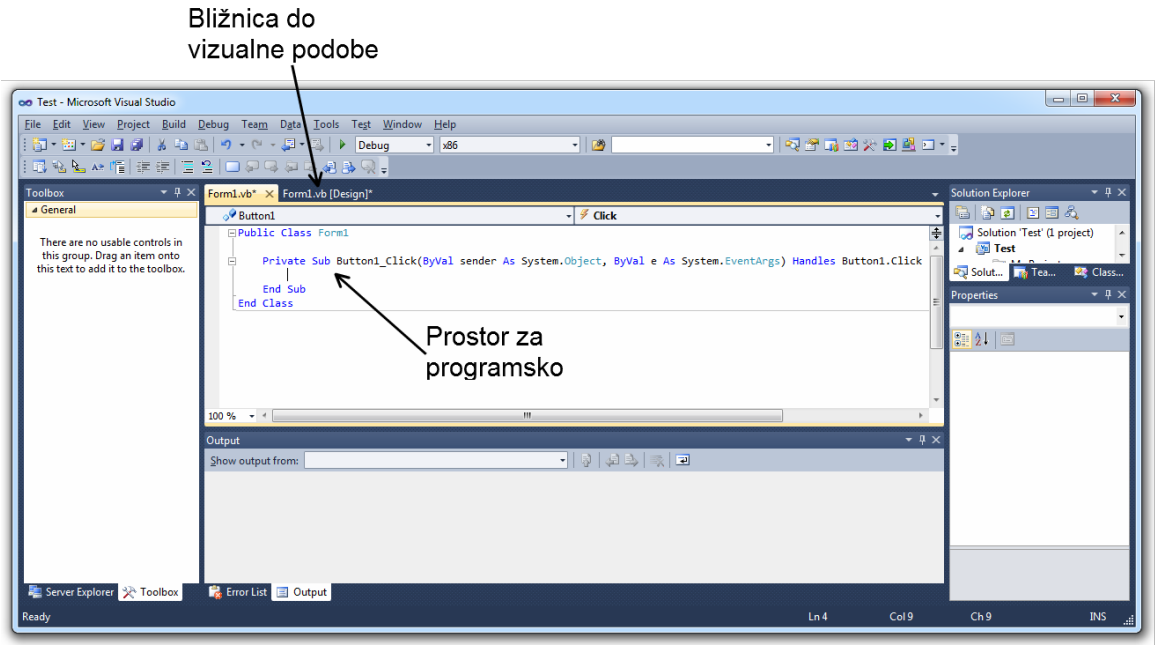
Dogodkovna procedura

Procedura `Private Sub btnIzracun_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnIzracun.Click` je takoimenovana dogodkovna procedura. Procedura se začne izvajati vsakič, ko uporabnik klikne na $btnIzracun$, torej ob dogodku $btnIzracun.Click$. Procedura ima dva argumenta, ki se prenašata po vrednostih (`ByVal`). Prvi argument, poimenovan $sender$, predstavlja predmet, ki je sprožil dogodek. Drugi argument, e, pa predstavlja podatke o samem dogodku. Včasih sta lahko oba argumenta $sender$ in e uporabna pri programiranju.

Tako končamo osnovno oblikovanje programa in se lahko lotimo pisanja programske kode, ki nam bo posamezne gradnike povezala v delujočo celoto.

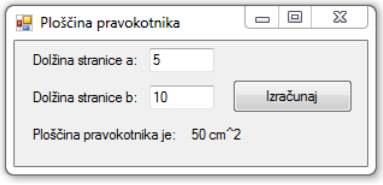
Prehod iz načina izdelave vizualne podobe aplikacije ter načina izdelave dogodkovnih procedur lahko dosežemo na več načinov, na primer, v meniju lahko spreminjamo med View Code in View Design. Za programiranje drugih dogodkov lahko označimo gradnik in potem v nastavitvah gra-

dnika poiščemo ikono Events, ki je označena s strelo. Ko jo odpremo vidimo vse možne dogodke, ki jih izbran gradnik podpira. V našem primeru smo izbrali gradnik Button, ki naj se zažene, ko uporabnik klikne nanj. Zato med dogodki izberemo *Click* ter določimo ime dogodka.



Slika 7.2: Programiranje gradnikov

Ko končamo z urejanjem grafične podobe in imamo sprogramirano dogodkovno proceduro, lahko program zaženemo. Na sliki 7.3 tako vidimo končno obliko delujočega programa.



Slika 7.3: Ploščina pravokotnika

7.2 Izračun geometrijskih oblik

Naš naslednji program bo omogočal izračun več geometrijskih oblik in vsaka geometrijska oblika bo izračunana v svojem zavihku. Pri tem bomo uporabili gradnik TabControl. Izberemo ga v orodjarni ToolBox in povlečemo na pripravljeno okno. Nadaljujemo z urejanjem nastavitev samega gradnika.

Da bo program ličen in prijazen do uporabnika, najprej nastavimo ustrezno pozicijo nato pa nastavimo ustrezno število in imena zavihkov. Po končanem urejanju začnemo v zavihke dodajati

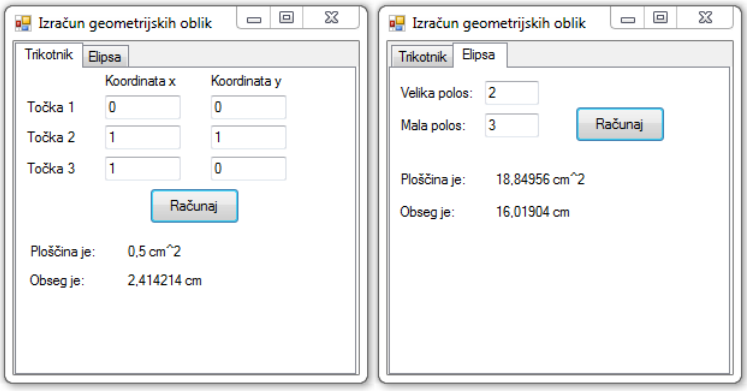
gradnike, ki jih potrebujemo vnos podatkov in prikaz rezultatov.

Enak postopek ponovimo v vseh zavihkih. Izberemo gumb, ki bo sprožil dogodkovno proceduro in zapišemo dogodkovno proceduro. Ko smo končali s tem, lahko naredimo še zadnje izboljšave.

V nadaljevanju sta obe izdelani dogodkovni proceduri.

```
Public Class Form1

    Private Sub btnRacunaj1_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnRacunaj1.Click
        'program drugega zavihka
        Dim vpo, mpo As Single
        Dim pl, ob As Single

        vpo = tbVelika.Text()
        mpo = tbMala.Text
        pl = vpo * mpo * Math.PI
        ob = Math.PI * (2 * (vpo ^ 2 + mpo ^ 2)) ^ 0.5

        lblPloscina1.Text = pl & " cm^2"
        lblObseg1.Text = ob & " cm"

    End Sub

    Private Sub btnRacunaj2_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnRacunaj2.Click
        'program prvega zavihka
        Dim x(2), y(2) As Single
        Dim pl, ob As Single

        'koordinate ogljišč ter izračun
        x(0) = tbX1.Text : y(0) = tbY1.Text
        x(1) = tbX2.Text : y(1) = tbY2.Text
        x(2) = tbX3.Text : y(2) = tbY3.Text

        pl = Math.Abs((x(0) * (y(1) - y(2)) + x(1) * (y(2) - y(0)) + x(2) * (y(0) - _
        y(1))) / 2)

        For i = 0 To 1
            ob = ob + ((x(i + 1) - x(i)) ^ 2 + (y(i + 1) - y(i)) ^ 2) ^ (1 / 2)
        Next

        lblPloscina2.Text = pl
        lblObseg2.Text = ob
    End Sub
End Class
```

Izdelani program deluje tako kot prikazuje naslednja slika.



Slika 7.4: Izračun ploščine

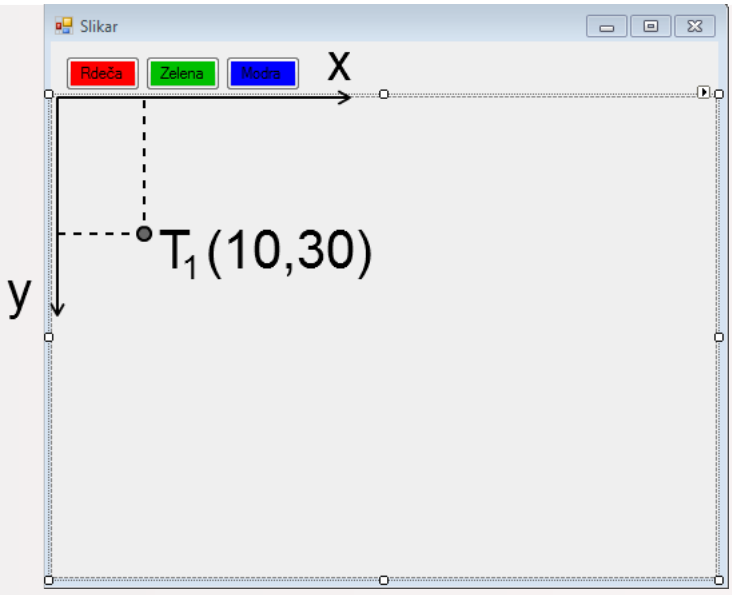
7.3 Slikar

Sedaj se lotimo programa, ki nam bo omogočal risanje pokljubnih krivulj na principu Slikarja, ki ga vsi dobro poznamo. Ob tem bomo spoznali gradnik PictureBox, ki nam omogoča risanje grafike.

Najprej izdelamo vizualno podobo aplikacije, kot je prikazano na sliki 7.6. Aplikacijo zasnujemo tako, da bo vsebovala tri gumbe v treh različnih barvah: rdečo, zeleno in modro. Aplikaciji bomo dodali en primerek gradnika PictureBox, kar bo predstavljalo površino za risanje. Prednastavljena barva za risanje naj bo modra. Če želi uporabnik risati v drugi barvi potem klikne na ustrezen gumb v aplikaciji in na ta način zamenja barvo svinčnika. Ko uporabnik pritisne gumb na miški in se le-ta nahaja nad risalno površino se začne zajemanje x in y koordinat miške in risanje prostih črt med koordinatami. Zajemanje podatkov se konča, ko uporabnik sprosti gumb na miški. Nato uporabnik lahko nadaljuje risanje druge črte z drugo barvo.

Gradnik PictureBox

Gradnik PictureBox vsebuje prosojnico (to je predmet tipa Graphics), ki jo lahko uporabljamo za risanje. Koordinatni sistem prosojnice poteka tako kot prikazuje naslednja slika.



Slika 7.5: Položaj koordinatnega sistema v gradniku PictureBox

Še preden začnemo z izdelavo posameznih dogodkovnih procedur deklariramo posamezne globalne spremenljivke. Celoten izdelan program je predstavljen proti koncu tega podpoglavja. Spremenljivko $risi$ podatkovnega tipa Boolean potrebujemo zato, da lahko ločimo trenutke, ko rišemo od trenutkov, ko premikamo miško ne da bi imeli pritisnjen gumb (za risanje). Spremenljivka k bo štela tekoče krivulje. Na začetku ima vrednost 0, ko uporabnik nariše prvo krivuljo pa se njena vrednost poveča za ena in tako naprej vse dokler ne doseže največje število dovoljenih krivulj. Zaradi lažjega dela bomo v tej aplikaciji omejili število točk na posamezni črti na 1001 točko ter število posameznih črt na 101 črto, kar naj bi zadoščalo za risanje manj zahtevnih slik. Tako, dvodimenzionalni statični polji x(100,1000) in y(100,1000) predstavljata x in y koordinate vseh točk na vseh krivuljah, polje n(100) predstavlja število točk v posameznih krivuljah ter polje barva(100) predstavlja barvo vsake posamezne krivulje.

Ko uporabnik premakne miško nad risalno površino in pritisne na levi gumb miške se začne risanje. Ob pritisku na gumb se sproži dogodek MouseDown, ki zažene dogodkovno proceduro Zacetek. V tej proceduri najprej zapišemo relativno pozicijo miške na zaslonu glede na gradnik pbSlikar. Ta podatek je vsebovan v argumentu e procedure Zacetek, to sta koordinati $e.X$ in $e.Y$ miške. Tekoča krivulja je k, tekoča točka znotraj krivulje pa $n(k)$. Zato imamo dva prireditvena stavka `x(k,n(k))=e.X` in `y(k,n(k))=e.Y`. V primeru, da uporabnik še ni izbral nobene barve je prednastavljena modra barva, sicer je prednastavljena barva tista, ki je bila uporabljena nazadnje.

Naslednja možna serija dogodkov je premikanje miške po zaslonu, to so dogodki MouseMove. Za to serijo dogodkov smo izdelali proceduro Premik. Podatke zajemamo le če je uporabnik predhodno pritisnil na levi gumb na miški, zato mora imeti Boolean spremenljivka risi vrednost $True$. Vsakič povečamo število točk na krivulji za ena in zapišemo trenutno lokacijo miške. Graf narišemo znova tako, da pokličemo proceduro `pbSlikar.Refresh()`, ki sproži dogodek `Paint` za ta predmet.

Ko uporabnik preneha s pritiskom na gumb od miške se sproži dogodek `MouseUp`. V tem primeru je

treba zajeti še zadnjo točko, tako, kot smo storili pri proceduri Premik ter povečati števec krivulj k za ena.

Naslednja skupina dogodkov za katere moramo izdelati dogotkovne procedure je tista, ki se nanaša na klik na posameznih gumbih, ki spreminjajo barvo krivulj. Na primer, klik na rdeči gumb sproži proceduro `IzberiRdeco`, ki nastavi barvo tekoče krivulje na rdečo s prireditvenim stavkom `barva(k)="rdeca"`.

Navsezadnje moramo sprogramirati tudi proceduro `RisiRisbo`, ki se sproži ob dogodku `Paint`. Ta procedura nariše vse dosedaj zajete krivulje glede na njihovo barvo. Pri tem lahko uporabljamo predmet e, ki je argument dogotkovne procedure. Predmet kot je $pbSlika$ ima določeno vizualno podobo, ki jo lahko spreminjamo. Predstavljajmo si, da je ta predmet narisan na eni prosojnici (poimenovani $e.Graphics$), ki jo računalnik nariše na zaslonu. Ko se sproži procedura `RisiRisbo`, kot argument dobi tudi prosojnico na katero lahko rišemo. Za risanje na prosojnici $e.Graphics$ so na voljo najbolj različne procedure. Na primer, procedura `DrawLine()` omogoča risanje črt v različnih barvah. Programski stavek

```
e.Graphics.DrawLine(Pens.Red, x(i, j), y(i, j), x(i, j + 1), y(i, j + 1))
```

nariše rdečo črto od točke s koordinato (x(i,j), y(i,j)) do točke s koordinato (x(i,j+1), y(i, j+1)). Večina procedur za risanje ima tudi nekaj različic, kar nam omogoča, da prilagodimo postopek risanja našim trenutnim željam in potrebam. Le slediti moramo natančno navodilom in proceduri moramo podati podatke v predpisani obliki.

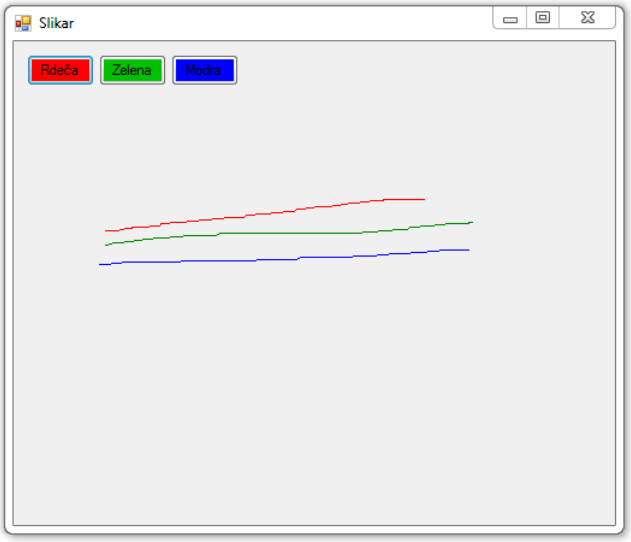
```
Public Class Slikar
    Dim risi As Boolean
    Dim k As Integer = 0 ' tekoča krivulja, maksimalno število je 100
    Dim x(100, 1000) As Single
    Dim y(100, 1000) As Single
    Dim n(100) As Integer ' število točk v posamezni krivulji
    Dim barva(100) As String ' barva posamezne krivulje je lahko rdeča, zelena ali
    modra

    Private Sub Zacetek(ByVal sender As System.Object, ByVal e As _
    System.Windows.Forms.MouseEventArgs) Handles pbSlikar.MouseDown
        x(k, n(k)) = e.X
        y(k, n(k)) = e.Y
        ' če barva ni izbrana potem določi modro črto
        If barva(k) <> "modra" And barva(k) <> "rdeca" And barva(k) <> "zelena" Then
            If k = 0 Then
                barva(k) = "modra"
            Else
                barva(k) = barva(k - 1)
            End If
        End If
        risi = True ' začnemo z risanjem črte
    End Sub

    Private Sub Premik(ByVal sender As System.Object, ByVal e As _
```

```
System.Windows.Forms.MouseEventArgs) Handles pbSlikar.MouseMove
    If risi = True Then
        n(k) = n(k) + 1 ' povečamo število točk
        x(k, n(k)) = e.X 'vpišemo novo točko
        y(k, n(k)) = e.Y
    End If
    pbSlikar.Refresh() ' znova narišemo graf
End Sub

Private Sub Konec(ByVal sender As System.Object, ByVal e As _
System.Windows.Forms.MouseEventArgs) Handles pbSlikar.MouseUp
    n(k) = n(k) + 1 ' povečamo število točk
    x(k, n(k)) = e.X
    y(k, n(k)) = e.Y
    risi = False
    k = k + 1 ' naslednjič bo uporabnik risal novo krivuljo
    pbSlikar.Refresh()
End Sub

Private Sub IzberiRdeco(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles btnRdeca.Click
    barva(k) = "rdeca"
End Sub

Private Sub IzberiZeleno(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles btnZelena.Click
    barva(k) = "zelena"
End Sub

Private Sub IzberiModro(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles btnModra.Click
    barva(k) = "modra"
End Sub

Private Sub RisiRisbo(ByVal sender As System.Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles pbSlikar.Paint
    ' narišemo vse črte glede na izbrane barve
    For i = 0 To k
        Select Case barva(i)
            Case "rdeca"
                For j = 0 To n(i) - 1
                    e.Graphics.DrawLine(Pens.Red, x(i, j), y(i, j), x(i, j + 1), _
                    y(i, j + 1))
                Next
            Case "zelena"
                For j = 0 To n(i) - 1
                    e.Graphics.DrawLine(Pens.Green, x(i, j), y(i, j), x(i, j + _
                    1), y(i, j + 1))
```

```
                    Next
                Case "modra"
                    For j = 0 To n(i) - 1
                        e.Graphics.DrawLine(Pens.Blue, x(i, j), y(i, j), x(i, j + _
                        1), y(i, j + 1))
                    Next
            End Select
        Next
    End Sub

End Class
```

V trenutku, ko se zažene aplikacija, le ta postopoma zgradi v pomnilniku vse primerke predmetov. Že na samem začetku se avtomatično sproži dogodek `Paint`, ki povzroči formiranje celotne grafične podobe aplikacije. V naši aplikaciji vidimo, da dogodek Paint lahko sprožimo tudi programsko, tako, da zaženemo proceduro `Refresh()` za posamezni predmet, na primer, `pbSlikar.Refresh()`.

Delovanje aplikacije je predstavljeno na naslednji sliki.



Slika 7.6: Risanje v programu Slikar

7.4 Postopno risanje linijske konstrukcije

V tej nalogi se bomo lotili izdelave programa, ki bo omogočal postopno risanje linijske konstrukcije. Program smo si zamislili tako, da bo uporabniku omogočal vpis x in y vrednosti dveh točk ter izbiro podpore v vsaki posamezni točki (členek, pomična, nepomična, brez). Nato, ko uporabnik klikne na gumb Dodaj se bo posamezen element izrisal na zaslonu. Poleg tega, smo si zamislili tudi okence za kontrolo vpisanih podatkov. Velikost površine za risanje bo 350x350 pik, naše realne enote pa bodo

od 0 do 5 metrov, zato bomo v programu morali poskrbeti za ustrezno pretvorbo realnih koordinat točk v koordinate na sliki.

Potem, ko smo si ustvarili predstavo o želeni aplikaciji jo lahko izdelamo. Na prazno okno dodamo nekaj napisov, štiri predmetov tipa TextBox za zajemanje podatkov o koordinatah, gumb za vnos podatkov, en predmet tipa ListBox za prikaz podatkov ter predmet tipa PictureBox za risanje konstrukcije. Takoj zatem se lotimo izdelave dogodkovnih procedur. Ena procedura zajema podatke, ki jih vpiše uporabnik, druga procedura postopno riše konstrukcijo. Pri risanju moramo biti pozorni na različne oznake podpor (členek, pomična, nepomična ali brez), ki jih narišemo na ustrezno mesto. Za risanje krogov uporabljamo proceduro `DrawEllipse()` iz razreda `Graphics`. Izdelan program je predstavljen v nadaljevanju.

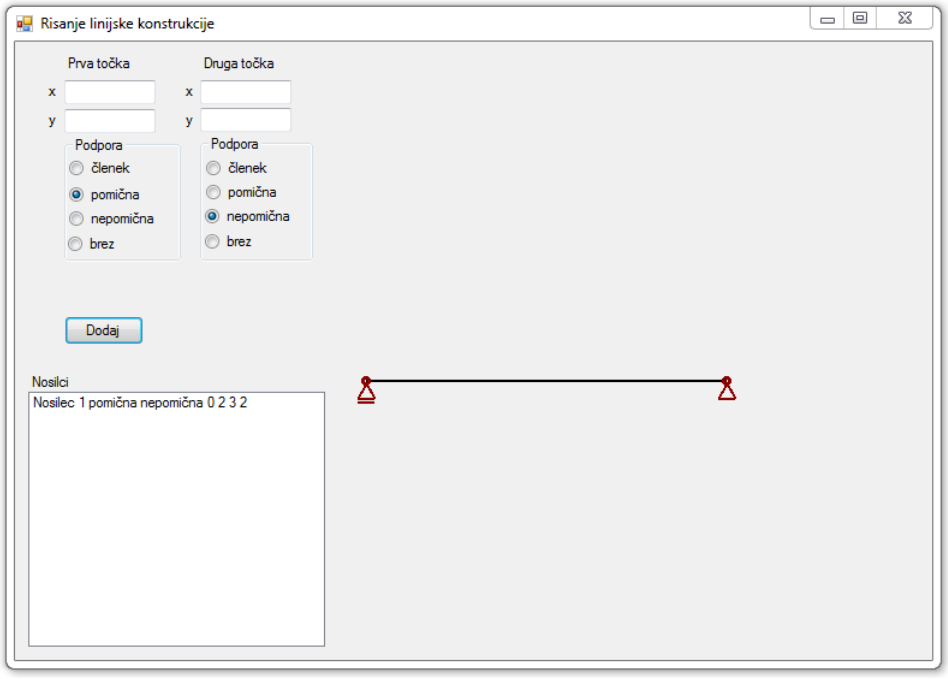
```
Public Class Form1

    ' Globalne spremenljivke
    Dim tockeAx(), tockeAy(), tockeBx(), tockeBy() As Single ' točke
    Dim tipA(), tipB() As String
    Dim k(), m() As Single
    Dim n As Integer = 0 ' števec točk

    Private Sub tbDodaj_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles tbDodaj.Click

        ReDim Preserve tockeAx(n), tockeAy(n), tockeBx(n), tockeBy(n), tipA(n), _
        tipB(n), k(n), m(n) ' povečamo število črt za ena, že določene
        ohranim

        tockeAx(n) = tbAx.Text * 100 '100 pikslov je 1 meter
        tockeAy(n) = tbAy.Text * 100
        tockeBx(n) = tbBx.Text * 100
        tockeBy(n) = tbBy.Text * 100
        If rbA1.Checked Then tipA(n) = "clenek"
        If rbA2.Checked Then tipA(n) = "pomicna"
        If rbA3.Checked Then tipA(n) = "nepomicna"
        If rbA4.Checked Then tipA(n) = "brez"
        If rbB1.Checked Then tipB(n) = "clenek"
        If rbB2.Checked Then tipB(n) = "pomicna"
        If rbB3.Checked Then tipB(n) = "nepomicna"
        If rbB4.Checked Then tipB(n) = "brez"

        lbPovezave.Items.Add("Nosilec " & n + 1 & " " & tipA(n) & " " & tipB(n) & " _
        " & tockeAx(n) / 100 & " " & tockeAy(n) / 100 & " " & tockeBx(n) / 100 & " " _
        & tockeBy(n) / 100)

        n = n + 1 ' povečaj števec za ena
        pbKonstrukcija.Refresh()
```

```
        tbAx.Clear()
        tbAy.Clear()
        tbBx.Clear()
        tbBy.Clear()
    End Sub

    Private Sub NarisiKonstrukcijo(ByVal sender As System.Object, ByVal e As _
    System.Windows.Forms.PaintEventArgs) Handles pbKonstrukcija.Paint
        Dim x, y As Single
        Dim crna, podpore, podpore1 As Pen
        crna = New Pen(Brushes.Black, 2)
        podpore = New Pen(Brushes.DarkRed, 5)
        podpore1 = New Pen(Brushes.DarkRed, 2)

        'prestaviteve ks kroga v središče
        x = CSng(Math.Sin(45 * Math.PI / 180) * 3)
        y = CSng(Math.Cos(45 * Math.PI / 180) * 3)

        For i = 0 To n - 1
            e.Graphics.DrawLine(crna, tockeAx(i) + 30, 470 - tockeAy(i), tockeBx(i) _
            + 30, 470 - tockeBy(i))

            Select Case tipA(i)
                Case "clenek"
                    e.Graphics.DrawEllipse(podpore, tockeAx(i) + 30 - x, 470 - _
                    tockeAy(i) - y, 3, 3)
                Case "pomicna"
                    e.Graphics.DrawEllipse(podpore, tockeAx(i) + 30 - x, 470 - _
                    tockeAy(i) - y, 3, 3)

                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30, 470 - tockeAy(i) _
                    + y, tockeAx(i) + 30 + 7, 470 - tockeAy(i) + y + 12)
                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30, 470 - tockeAy(i) _
                    + y, tockeAx(i) + 30 - 7, 470 - tockeAy(i) + y + 12)

                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30 + 7, 470 - _
                    tockeAy(i) + y + 12, tockeAx(i) + 30 - 7, 470 - tockeAy(i) + y + _
                    12)
                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30 + 7, 470 - _
                    tockeAy(i) + y + 15, tockeAx(i) + 30 - 7, 470 - tockeAy(i) + y + _
                    15)
                Case "nepomicna"
                    e.Graphics.DrawEllipse(podpore, tockeAx(i) + 30 - x, 470 - _
                    tockeAy(i) - y, 3, 3)

                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30, 470 - tockeAy(i) _
                    + y, tockeAx(i) + 30 + 7, 470 - tockeAy(i) + y + 12)
                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30, 470 - tockeAy(i) _
```

```
                    + y, tockeAx(i) + 30 - 7, 470 - tockeAy(i) + y + 12)

                    e.Graphics.DrawLine(podpore1, tockeAx(i) + 30 + 7, 470 - _
                    tockeAy(i) + y + 12, tockeAx(i) + 30 - 7, 470 - tockeAy(i) + y + _
                    12)
                Case "brez"
            End Select

            Select Case tipB(i)
                Case "clenek"
                    e.Graphics.DrawEllipse(podpore, tockeBx(i) + 30 - x, 470 - _
                    tockeBy(i) - y, 3, 3)

                Case "pomicna"
                    e.Graphics.DrawEllipse(podpore, tockeBx(i) + 30 - x, 470 - _
                    tockeBy(i) - y, 3, 3)

                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30, 470 - tockeBy(i) _
                    + y, tockeBx(i) + 30 + 7, 470 - tockeBy(i) + y + 12)
                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30, 470 - tockeBy(i) _
                    + y, tockeBx(i) + 30 - 7, 470 - tockeBy(i) + y + 12)

                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30 + 7, 470 - _
                    tockeBy(i) + y + 12, tockeBx(i) + 30 - 7, 470 - tockeBy(i) + y + _
                    12)
                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30 + 7, 470 - _
                    tockeBy(i) + y + 15, tockeBx(i) + 30 - 7, 470 - tockeBy(i) + y + _
                    15)
                Case "nepomicna"
                    e.Graphics.DrawEllipse(podpore, tockeBx(i) + 30 - x, 470 - _
                    tockeBy(i) - y, 3, 3)

                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30, 470 - tockeBy(i) _
                    + y, tockeBx(i) + 30 + 7, 470 - tockeBy(i) + y + 12)
                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30, 470 - tockeBy(i) _
                    + y, tockeBx(i) + 30 - 7, 470 - tockeBy(i) + y + 12)

                    e.Graphics.DrawLine(podpore1, tockeBx(i) + 30 + 7, 470 - _
                    tockeBy(i) + y + 12, tockeBx(i) + 30 - 7, 470 - tockeBy(i) + y + _
                    12)

                Case ("brez")
            End Select

        Next

    End Sub
```

```
End Class
```

Primer delovanja programa je prikazan na naslednji sliki 7.7.



Slika 7.7: Primer risanja prostoležečega nosilca

7.5 Prostoležeč nosilec

Izdelali bomo aplikacijo, ki za podane podatke izračuna momente vzdolž prostoležečega nosilca in jih tudi grafično prikaže. Izbrali bomo nosilec, ki ima eno pomično in eno nepomično podporo. Uporabnik najprej pripravi naslednji program.

```
Public Class Form1
    Dim L, F, a, n As Single
    Dim Az, Bz, Ax, M() As Single
    Dim x, t As Single
    Dim risi As Boolean = False

    Private Sub btnRacun_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles btnRacun.Click
```

```
'Brisanje vsebine iz prejsnjih izracunov
Izpis.Items.Clear()

'Branje zacetnih podatkov
L = tbDolzina.Text
If L < 0 Then
    MsgBox("Popravi dolzino nosilca")
    L = 1
End If
F = tbSila.Text
a = tbPolozaj.Text
If a < 0 Or a > L Then
    MsgBox("Popravi mesto sile")
    a = 0.5
End If
n = tbDelitev.Text()
If n < 0 Then
    MsgBox("Popravi stevilo delitev")
    n = 2
End If

ReDim M(n + 1)

'Formule za izracun reakcij in momentov
Ax = 0
Bz = F * a / L
Az = F - Bz

t = L / n
For i = 0 To n
    x = i * t
    If x <= a Then
        M(i) = Az * x
    ElseIf x > a Then
        M(i) = Az * x - F * (x - a)
    End If
Next
M(n + 1) = Az * a

'podatki in rezultati se izpisejo v ListBox
Izpis.Items.Add("Podatki:")
Izpis.Items.Add("Dolzina:        " & L)
Izpis.Items.Add("Sila:              " & F)
Izpis.Items.Add("Polozaj sile:  " & a)
Izpis.Items.Add("Razdelitev:   " & n)
Izpis.Items.Add("")
Izpis.Items.Add("Reakcije:")
Izpis.Items.Add("Az= " & Az)
```

```
        Izpis.Items.Add("Ax= " & Ax)
        Izpis.Items.Add("Bz= " & Bz)
        Izpis.Items.Add("")
        Izpis.Items.Add("Razporeditev momentov:")
        Izpis.Items.Add("i       |       x       |       M")
        For i = 0 To n
            Izpis.Items.Add(i & "      |      " & i * t & "      |       " & M(i))
        Next
        Izpis.Items.Add("")
        Izpis.Items.Add("Maksimalni moment")
        Izpis.Items.Add("x= " & a)
        Izpis.Items.Add("M= " & M(n + 1))
        Izpis.Items.Add("-----------------------------------------------")
        risi = True
        pbIzris.Refresh()
    End Sub


    Private Sub pbIzris_Paint(ByVal sender As Object, ByVal e As _
    System.Windows.Forms.PaintEventArgs) Handles pbIzris.Paint
        If risi = True Then
            Dim prosojnica As Graphics
            prosojnica = e.Graphics
            Dim crna, crna1, rdeca, modra, zelena As Pen
            crna = New Pen(Brushes.Black, 5)
            crna1 = New Pen(Brushes.Black, 2)
            rdeca = New Pen(Brushes.Red, 3)
            modra = New Pen(Brushes.DarkBlue, 3)
            zelena = New Pen(Brushes.Green, 3)

            prosojnica.DrawString("Obtezba", New Font("Arial", 10, _
            FontStyle.Regular), Brushes.DarkBlue, New PointF(30, 5))
            'podpore
            prosojnica.DrawEllipse(crna1, 42, 61, 8, 8) : _
            prosojnica.DrawEllipse(crna1, 449, 61, 8, 8)
            prosojnica.DrawLine(crna1, 46, 70, 58, 86) : prosojnica.DrawLine(crna1, _
            453, 70, 465, 86)
            prosojnica.DrawLine(crna1, 46, 70, 34, 86) : prosojnica.DrawLine(crna1, _
            453, 70, 441, 86)
            prosojnica.DrawLine(crna1, 34, 86, 58, 86) : prosojnica.DrawLine(crna1, _
            441, 86, 465, 86)
            prosojnica.DrawLine(crna1, 441, 90, 465, 90)
            'linija nisilca
            prosojnica.DrawLine(crna, 50, 65, 450, 65)
            'sila
            prosojnica.DrawString("F", New Font("Arial", 10, FontStyle.Bold), _
            Brushes.DarkBlue, New PointF(50 + (a / L) * 400 - 20, 22))
            prosojnica.DrawLine(modra, 50 + (a / L) * 400, 15, 50 + (a / L) * 400, 60)
```

```
            prosojnica.DrawLine(modra, 50 + (a / L) * 400, 60, 50 + (a / L) * 400 - _
            7, 45)
            prosojnica.DrawLine(modra, 50 + (a / L) * 400, 60, 50 + (a / L) * 400 + _
            7, 45)


            prosojnica.DrawString("Reakcije", New Font("Arial", 10, _
            FontStyle.Regular), Brushes.DarkBlue, New PointF(30, 120))
            'reakcije
            prosojnica.DrawLine(crna, 50, 170, 450, 170)

            'reakcije A
            prosojnica.DrawString("Ax", New Font("Arial", 10, FontStyle.Bold), _
            Brushes.DarkBlue, New PointF(15, 150))
            prosojnica.DrawLine(rdeca, 5, 170, 50, 170)
            prosojnica.DrawLine(rdeca, 50, 170, 35, 163)
            prosojnica.DrawLine(rdeca, 50, 170, 35, 177)

            prosojnica.DrawString("Az", New Font("Arial", 10, FontStyle.Bold), _
            Brushes.DarkBlue, New PointF(55, 190))
            prosojnica.DrawLine(rdeca, 50, 170, 50, 215)
            prosojnica.DrawLine(rdeca, 43, 185, 50, 170)
            prosojnica.DrawLine(rdeca, 57, 185, 50, 170)

            'reakcije B
            prosojnica.DrawString("Bz", New Font("Arial", 10, FontStyle.Bold), _
            Brushes.DarkBlue, New PointF(426, 190))
            prosojnica.DrawLine(rdeca, 450, 170, 450, 215)
            prosojnica.DrawLine(rdeca, 443, 185, 450, 170)
            prosojnica.DrawLine(rdeca, 457, 185, 450, 170)


            'momenti
            prosojnica.DrawString("Potek momentov", New Font("Arial", 10, _
            FontStyle.Regular), Brushes.DarkBlue, New PointF(30, 250))
            prosojnica.DrawLine(crna, 50, 290, 450, 290)
            If M(n + 1) > 0 Then
                prosojnica.DrawLine(zelena, 50, 290, 50 + (a / L) * 400, 370) : _
                prosojnica.DrawLine(zelena, 50 + (a / L) * 400, 370, 450, 290)
                prosojnica.DrawString("Mmax", New Font("Arial", 10, FontStyle.Bold), _
                Brushes.DarkBlue, New PointF(50 + (a / L) * 400 - 21, 380))
                For i = 1 To n
                    prosojnica.DrawLine(zelena, 50 + 400 / n * i, 292, 50 + 400 / n _
                    * i, 292 + (M(i) / M(n + 1)) * 80)
                Next
                Dim aa As Integer
                aa = CInt(a / L * 400)
                prosojnica.DrawLine(zelena, 50 + aa, 292, 50 + aa, 292 + 80)
```

```
            End If
        End If

    End Sub
End Class
```

Pri izdelavi te naloge bomo tudi pozorni na komunikacijo z uporabnikom. Zato bomo uporabili programski stavek: MessageBox.Show("besedilo", "naslov", MessageBoxButtons.izberes, MessageBoxIcon.izberes). Za proceduro risanje kliknemo na paintbox ter v properties -> events dvakrat kliknemo na ukaz paint. To pomeni da ko bo program prišel do te točke bo začel v paintboxu risati kar mu bomo ukazali.

V proceduri moramo najprej deklarirati element,ki nam bo risal oz pisal. Ta element imenujmo prosojnica. Deklariramo jo as Graphics. Deklarirati moramo še pisala, ki jih bo prosojnica uporabljala za risane. Imenujmo jih svinčniki. Deklariramo jih as Pen. Prosojnici priredimo e.Graphics. Vsakemu posameznemu svinčniku pa priredimo njegove lastnosti z New Pen(brushes.barva, debelina).

Sedaj imamo pripravljene vse pripomočke za risanje, zato začnemo risati. Najprej narišemo podpore. Eno členkasto pomično in eno členkasto nepomično. Členek je v obliki kroga, zato moramo risati z ukazom prosojnica.DrawEllipse(pisalo, koordinate, 1.polos, 2.polos). Pri tem se moramo zavedati, da x koordinata poteka normalno od leve proti desni, y pa od zgoraj navzdol. Zaradi tega je koordinatno izhodišče zgoraj levo. Paziti moramo seveda tudi na to da imamo dovolj velik paintbox da lahko vanj vse narišemo. Lahko se namreč zgodi da kaj narišemo izven njega in zato ne bo vidno. Nato narišemo še preostali del kostrukcije z ukazom prosojnica.

Za konec pa z ukazom prosojnica.DrawString("Nosilec", New Font("Arial", 10, FontStyle.Regular), Brushes.DarkBlue, New PointF(30, 50)) v paintbox nekaj napišemo.

Gradnik CheckBox, ki ga predhodno uredimo v zavihku Design. Del procedure, na katero hočemo vplivati z checkboxom damo v if zanko. Logični test pa izgleda takole: if checkbox.Checked = True Then ter vrednost ob izpolnjenem pogoju. Ostane nam še procedura za računanje notranjih momentov. V zavihku design uredimo vse potrebne stvari, ki jih bomo potrebovali, dodamo pa še nov predmet, ki se imenuje ListBox. Ta nam bo prikazal podatke ter izračunane reakcije in momente. V ustvarjeno proceduro začnemo vpisovati algoritem za računanje. Podatke, reakcije in momente v ListBox vpišemo tako: LBMomenti.Items.Add("besedilo, ki ga hočemo prikazati").

8 Naloge in rešitve

8.1 Shranjevanje podatkov v notranjem pomnilniku

Naloga

Koliko bajtov spomina potrebujemo, če želimo v programu uporabljati 5 celih števil? Ustrezno deklariraj pet spremenljivk.

Rešitev

Za hranjenje poljubnih petih celih števil moramo kreirati 5 različnih spremenljivk. Za shranjevanje celih števil ima Visual Basic podatkovni tip `Integer`. Sama deklaracija izgleda takole:

```
Dim a, b, c, d, e As Integer
```

Za vsako spremenljivko deklarirano na ta način se porabijo štirje bajti spomina, torej se za pet spremenljivk porabi skupaj 20 bajtov.

8.2 Prireditveni stavki

Naloga

Kaj izpiše naslednji program in zakaj?

Prireditveni stavek

```
Module Module1

Sub Main()
Dim a As Integer = 10
Dim x As Single = 10.224
Dim y As Double = 10.224232242424222
Dim u As String = "Rezultat"

u = u & " je "
```

```
    a = x
    y = a
    u = u & y

    Console.WriteLine(u)
    Console.ReadKey()
    End Sub

    End Module
```

Rešitev

V nalogi imamo deklarirane 4 različne spremenljivke, ki jih nato med sabo enačimo in združujemo. Zelo moramo paziti, kako je deklarirana posamezna spremenljivka, saj to pomeni, kako natančno se vanjo shrani določeno število in kasneje je od tega odvisen tudi sam izpis rešitve.

Razumevanje sestave spremenljivke u nam ne bi smelo delati problemov. Spremenljivka u je podatkovnega tipa `String`, operacija & v prireditvenem stavku `u = u \& "je "` pa obstoječi vrednosti spremenljivke u na konec prilepi še štiri znake "je ". Tako je nova vrednost spremenljivke u v spominu `"Rezultat je "`. Prireditveni stavek `a = x` povzroči, da se vrednost spremenljivke x vpiše v pomnilniško celico rezervirano za spremenljivko a. Spremenljivka a je na žalost deklarirana kot celo število (`Integer`). Posledično se v spremenljivko a lahko shrani le celi del vrednosti 10.224, to je 10. Podatkovni tip `Integer` ne omogoča shranjevanja vrednosti za decimalno piko. Nova vrednost spremenljivke a je tako 10. Naslednji prireditveni stavek `y = a` prepiše vrednost spremenljivke a (to je 10) v pomnilniško celico rezervirano za spremenljivko y. Spremenljivka y je sicer podatkovnega tipa `Double` in bi jo lahko uporabili za shranjevanje realnih števil z velikim številom decimalnih mest, vendar v konkretnem primeru ostanejo te možnosti ne izrabljene. Končna vrednost spremenljivke y je tako 10. Zadnji prireditveni stavek je `u = u \& y`, kar vrednosti spremenljivke u pripiše vrednost spremenljivke y na konec. Nova vrednost spremenljivke u je `"Rezultat je 10"` in program izpiše točno to.

Slika 8.1: Prireditveni stavek

8.3 Ostanek pri deljenju

Naloga

Napiši program, ki ob vnosu deljenca in deljitelja izračuna ostanek pri deljenju. Spremenljivke poimenuj pomenljivo, na primer: deljenec, deljitelj in ostanek.

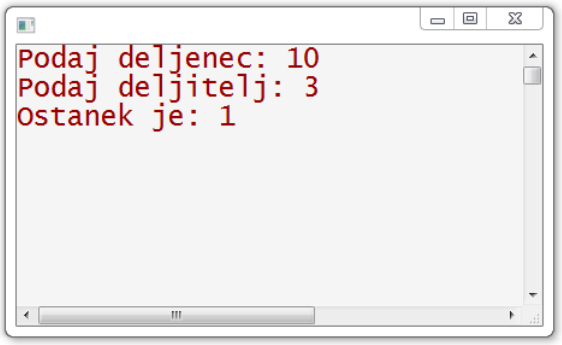
Rešitev

Za izračun ostanka pri deljenju lahko izkoristimo spremenljivke različnih podatkovnih tipov. Rezultat deljenja dveh celih števil je lahko celo število, če gre za deljenje brez ostanka. V vseh ostalih primerih pa pri deljenju dveh celih števil nastane racionalno število, ki vsebuje določeno število decimalk. Program izdelamo tako, da deklariramo spremenljivki a in c, ki sta deklarirani kot realni števili ter spremenljivko b, ki je deklarirana kot celo število.

Prireditveni stavek `a = deljenec / deljitelj` učinkuje tako, da se rezultat deljenja priredi spremenljivki a vključno z vsemi decimalkami. Prireditveni stavek `b = deljenec / deljitelj` pa učinkuje tako, da se spremenljivki b priredi rezultat deljenja pri katerega so odrezane vse decimalke. Z naslednjim prireditvenim stavkom `c = a - b` v spremenljivki c shranimo samo decimalke. V nadaljevanju lahko z uporabo spremenljivke c izračunamo ostanek pri deljenju takole: `ostanek = _ deljitelj * c`, ki ga potem tudi izpišemo.

Ostanek pri deljenju

```
Module Module1
Sub Main()
Dim deljenec, deljitelj, ostanek As Integer
Dim a As Single = 0
Dim b As Integer = 0
Dim c As Single = 0
Console.WriteLine("Izracun ostanka")
Console.Write("Podaj deljenec: ")
deljenec = Console.ReadLine()
Console.Write("Podaj deljitelj: ")
deljitelj = Console.ReadLine()
a = deljenec / deljitelj
b = deljenec / deljitelj
c = a - b
ostanek = deljitelj * c
Console.WriteLine("Ostanek je: " & ostanek)
Console.ReadKey()
End Sub
End Module
```



Slika 8.2: Ostanek pri deljenju

8.4 Natančnost izračuna in izpis rezultatov

Naloga

Izdelaj program za izračun prostornine krogle, ki izpiše rezultat na 9 decimalnih mest natančno.

Rešitev

Prostornino krogle izračunamo z uporabo ustrezne formule. Spremenljivka v je deklarirana kot realno število in izračunano vrednost hrani z natančnostjo velikega števila decimalk. Da bi izpis omejili na 9 decimalnih mest natančno uporabimo operacijo `ToString()`, kateri podamo kot argument vrednost "F9". Vrednost argumenta "F9"operaciji `ToString()` narekuje, da izpiše le prvih 9 decimalnih mest. Celoten program je prikazan v nadaljevanju.

Volumen krogle

```
Module Module1

Sub Main()

Dim r, v As Double

Console.WriteLine("Program za izracun volumna krogle")
Console.WriteLine("Podaj polmer krogle")
r = Console.ReadLine
v = 4 / 3 * Math.PI * r ^ 3

Console.WriteLine("Volumen krogle je: " & v.ToString("F9") & ".")

Console.ReadKey()
```

```
    End Sub

    End Module
```

8.5 Uporaba knjižnice Math

Naloga

V prejšnjih nalogah smo matematične konstante in matematične funkcije zapisovali kot `Math.Cos(x)`. Kako bi to poenostavili, da skrajšamo sam zapis programske kode?

Rešitev

Zapis lahko poenostavimo tako, da v našem programu uvozimo sistemsko knjižnico `Math`. To storimo tako, da na začetku programa zapišemo programski stavek: `Imports System.Math`. Potemtakem v našem programu lahko uporabljamo vse funkcije knjižnice takole `Cos(x)`.

8.6 Izpis poštevanke (nadgradnja)

Naloga

Napiši program, ki izpiše poštevanko na naslednji način:

$$\left|\begin{array}{l} 1\cdot 1=1 \\ 2\cdot 1=2 \\ 3\cdot 1=3 \\ \vdots \end{array}\right| \left|\begin{array}{l} 1\cdot 2=2 \\ 2\cdot 2=4 \\ 3\cdot 2=6 \\ \vdots \end{array}\right| \left|\begin{array}{l} 1\cdot 3=2 \\ 2\cdot 3=2 \\ 3\cdot 3=2 \\ \vdots \end{array}\right| \left|\begin{array}{l} \ldots \\ \ldots \\ \ldots \\ \end{array}\right.$$

Rešitev

Poštevanka - nadgradnja

```
Module Module1

    Sub Main()
        Dim dolzina As String
        'For zanka
        For i = 0 To 10 'vrstice
            For j = 0 To 10 'stolpci
                dolzina = i * j
                ' izpis z dodajanjem presledkov
                If i <> 10 And j <> 10 And dolzina.Length = 1 Then
                    Console.Write("|     " & i & "x" & j & "=" & i * j)
```

```
                ElseIf i <> 10 And j <> 10 And dolzina.Length = 2 Then
                    Console.Write("|   " & i & "x" & j & "=" & i * j)
                ElseIf i <> 10 And j <> 10 And dolzina.Length = 2 Then
                    Console.Write("|" & i & "x" & j & "=" & i * j)
                ElseIf i = 10 And dolzina.Length = 1 Then
                    Console.Write("|   " & i & "x" & j & "=" & i * j)
                ElseIf j = 10 And dolzina.Length = 1 Then
                    Console.Write("|   " & i & "x" & j & "=" & i * j)
                ElseIf j = 10 And dolzina.Length = 2 Then
                    Console.Write("|  " & i & "x" & j & "=" & i * j)
                ElseIf i = 10 And dolzina.Length = 2 Then
                    Console.Write("|  " & i & "x" & j & "=" & i * j)
                Else
                    Console.Write("|" & i & "x" & j & "=" & i * j)
                End If
            Next
            Console.WriteLine("|")
        Next
        Console.ReadKey()

    End Sub
End Module
```

8.7 Kaj dela naslednji program?

Naloga

Pri programiranju je včasih potrebno ugotoviti kaj dela programska koda, ki ni pravilno označena. Programer je poimenoval spremenljivke kot a, b, c, d in e. Ugotovi kaj dela program in predlagaj boljše poimenovanje spremenljivk.

Predhodno, smo izdelali program za izračun ostanka pri deljenju brez uporabe zank. Ta program izračuna ostanek pri deljenju z uporabo zanke `Do-Until`. Spremenljivke a, b in c bi lahko poimenovali $delitelj$, $deljenec$ ter $ostanek$. Ostale spremenljivke so pomožne in jih lahko poimenujemo z neutralnimi imeni.

Rešitev

Kaj dela naslednji program?

```
Module Module1

    Sub Main()

        Dim a, b, c As Integer
        Dim d As Single = 0
```

```
        Dim e As Integer = 1
        a = Console.ReadLine()
        b = Console.ReadLine()
        Do Until d > a
            d = b * e
            e = e + 1
        Loop
        e = e - 2
        c = a - b * e
        Console.WriteLine("Rezultat je: " & c)
        Console.ReadKey()

    End Sub

End Module
```

8.8 Kaj izpišeta programa?

Naloga

Ugotovi, kaj izpišeta naslednja programa in ju po potrebi popravi.

Program 1

```
Module Module1

Sub Main()
Dim a As Integer
a = 4

Do
If a > 3 Then
Console.WriteLine("a= " & a)
End If
Loop While a < 6
Console.ReadKey()

End Sub
End Module
```

Program 2

```
Module Module1
```

```
Sub Main()

i = 7

Do While i > 4

a = 0
Do
a = a + 1
Console.Write(a * i)
Loop While a < 2

i = i - 1
Loop

Console.ReadKey()

End Sub
End Module
```

Rešitev Poskusi na računalniku.

8.9 Največje in najmanjše število

Naloga

Napiši konzolni program v jeziku Visual Basic, ki najprej od uporabnika zahteva, da vnese 20 realnih števil (za vsakim vpisanim številom uporabnik pritisne tipko ENTER). Program nato ovrednoti števila in v konzolnem oknu izpiše, katero število je največje in katero najmanjše izmed prej vnešenih števil.

Rešitev

Naloge se lotimo tako, da najprej izdelamo načrt algoritma. V ta namen potrebujemo eno spremenljivko za hranjenje vrednosti, ki jo vpiše uporabnik. To bo spremenljivka $stevilo$. Nato potrebujemo še dve dodatni spremenljivki: $najvecje$ in $najmanjse$.

Uporabnik bo sproti vpisoval števila in na vsakem koraku bo naš algoritem moral preverjati ali je vpisano število večje od vrednosti spremenljivke $najvecje$. Pravtako bo algoritem moral preverjati ali je trenutno vpisano stevilo manjse od vrednosti spremenljivke $najmanse$.

Sedaj se lotimo pisanja programa pri katerega moramo biti pozorni tudi na začetne pogoje. Spremenljivki $najvecje$ in $najmanjse$ inicializiramo tako, da jim dodelimo vrednost prvega vpisanega števila. Če bi ti dve spremenljivki inicializirali z vrednostjo 0, ali katero drugo fiksno vrednost bi lahko program podajal napačne rezultate. Uporabnik bi se lahko, na primer, odločil, da vedno podaja vrednosti večje od števila 0. V slednjem primeru, bi program kot najmanjše podano število izpisal število 0, kar ne bi bilo pravilno.

Postopek podajanja števil in preverjanja moramo ponoviti še 19 krat. Zato smo uporabili zanko

`For-Next`.

Izdelani program je predstavljen v nadaljevanju.

Najdaljše naraščanje števil

```
Module Module1

    Sub Main()
        Dim stevilo, najvecje, najmanjse As Integer
        Console.WriteLine("Vnesi stevila")
        Console.Write("Vpisi stevilo ")
        stevilo = Console.ReadLine
        najmanjse = stevilo ' začetna pogoja
        najvecje = stevilo

        For i = 2 To 20
            Console.Write("Vpisi stevilo ")
            stevilo = Console.ReadLine
            If najvecje < stevilo Then
                najvecje = stevilo
            End If
            If najmanjse > stevilo Then
                najmanjse = stevilo
            End If
        Next

        Console.WriteLine("Najvecje vpisano stevilo je " & najvecje)
        Console.WriteLine("Najmanse vpisano stevilo je " & najmanjse)
        Console.ReadKey()

    End Sub

End Module
```

8.10 Razvrščanje lesa v trdnostne razrede

Naloga

Sodelavci gradbenega inštituta bi radi določili povprečno vrednost tlačne trdnosti neke vrste lesa in njeno standardno deviacijo. V ta namen so pridobili 20 testnih vzorcev, na katerih so opravili meritev tlačne trdnosti. V Tabeli (spodaj) so podani rezultati opravljenih meritev. Naloga je, da na čim lažji način uredite vnos podatkov ter sam izračun povprečne tlačne trdnosti in standardne deviacije.

Št. vzorca	Tlačna trdnost $[\frac{N}{cm^2}]$
1	1890
2	1985
3	1992
4	2006
5	2135
6	2089
7	1978
8	2113
9	1903
10	1999
11	2046
12	2156
13	1955
14	2023
15	2000
16	1856
17	2165
18	2099
19	2006
20	1979

Rešitev

Najprej poskrbimo za ustrezen vnos podatkov. Ker verjetno ne bomo vedno imeli natančno 20 podatkov o vzorcih, naredimo program fleksibilen. Omogočimo določitev števila podatkov, ki jih bomo vnesli. Nato z uporabo preproste for-zanke vnesem podatke o vzorcih in jih že hkrati med seboj seštevam. To nam bo naslednjem koraku služi, da lahko že izračunamo povprečje vnešenih vrednosti. Nato z zelo podobno for-zanko seštejem kvadrate razlike podanih vrednosti in povprečne vrednosti. Sledi še formula za izračun standardnega odklona in izpis rezultatov.

Povprečje in standardni odklon

```
Imports System.Math

Module Module1

    Sub Main()
        Dim n As Integer
        Dim podatek(), sum, povprecje, kvadrati, odklon As Single

        Console.Write("Podaj stevilo vnosov: ")
        n = Console.ReadLine()
        ReDim podatek(n)
```

```
        Console.WriteLine("Vnesi podatke meritev:")
        For i = 0 To n - 1
            podatek(i) = Console.ReadLine
            sum = sum + podatek(i)
        Next
        povprecje = sum / n

        For i = 0 To n - 1
            kvadrati = kvadrati + (podatek(i) - povprecje) ^ 2
        Next
        odklon = Sqrt(kvadrati / n)

        Console.WriteLine()
        Console.WriteLine("Povprecje meritev je: " & povprecje & ".")
        Console.WriteLine("Standardni odklon: " & odklon & ".")
        Console.ReadKey()

    End Sub

End Module
```

8.11 Najdaljše naraščajoče zaporedje

Naloga

Izdelaj program, ki za podana števila določi najdaljše naraščajoče zaporedje in ga izpiše. Na primer, če vpišemo števila: 1, 2, 8, 4, 5, 2, 3, 8, 11, 3, 5, 2, 5, 7, program izpiše: 2, 3, 8, 11, saj je to najdaljše naraščajoče zaporedje števil.

Rešitev

Najdaljše naraščanje števil

```
Module Module1

    Sub Main()
        Dim n As Integer = 10 ' dolžina podanega niza
        Dim s(n - 1) As Integer ' niz celih števil
        Dim ts, td, ns, nd As Integer 'pomožni spremenljivki

        ' ts - začetni element tekočega zaporedja
        ' td - tekoča dolžina
        ' ns - začetni element najdaljšega zaporedja
        ' nd - dolžina najdaljšega zaporedja
```

```
        'vnos števil
        For i = 0 To n - 1
            Console.Write("Podaj stevilo " & i + 1 & ": ")
            s(i) = Console.ReadLine
        Next i

        'razvrščanje števil po velikosti
        ts = 0 : td = 0 : ns = 0 : nd = 0 ' začetne vrednosti spremenljivk

        For i = 0 To n - 2
            If s(i + 1) >= s(i) Then
                td = td + 1
                If td > nd Then
                    nd = td
                    ns = ts
                End If
            Else
                If td > nd Then
                    nd = td
                    ns = ts
                    ts = i + 1
                    td = 0
                Else
                    ts = i + 1
                    td = 0
                End If
            End If
        Next

        'Izpis najdaljšega zaporedja
        For i = ns To ns + nd
            Console.WriteLine(s(i))
        Next
        Console.ReadKey()

    End Sub

End Module
```

8.12 Števila v padajočem vrstnem redu

Naloga

V nalogi 3.4 smo napisali program, ki podana števila uredi po vrstnem redu od najmanjšega do največjega. Za nalogo prilagodite program tako, da bo izpisal števila v padajočem vrstnem redu, torej od največjega do najmanjšega.

Rešitev

Menjava pogoja

```
'razvrščanje
For i = 0 To n - 2
    For j = i + 1 To n - 1
        If s(j) > s(i) Then
            a = s(j)
            s(j) = s(i)
            s(i) = a
        End If
    Next
Next
```

Izpis v obratnem vrstnem redu

```
' izpis
For i = n - 1 To 0 Step -1
    Console.WriteLine(s(i))
Next
```

8.13 Produkt matrik

Naloga

Napiši program, ki omogoča vnos dveh $n \times n$ matrik, nato pa izračuna njun produkt. Rezultate naj program tudi izpiše.

Rešitev

Najprej deklariramo tri matrike, od katerih sta dve namenjeni hranjenju vhodnih matrik, v tretjo pa potem shranjujemo vrednosti množenja. Osredotočimo se na $n \times n$ matrike zato, da ne rabimo primerjati ustreznost matrik za množenje.

Množenje matrik

```
Module Module1

    Sub Main()

        ' deklaracija spremenljivk in matrik
```

```
        Dim n As Integer
        Dim A(,), B(,) As Single ' matriki za vnos
        Dim C(,) As Single ' zmnožena matrika

        Console.Write("Podaj stevilo stolpcev in vrstic: ") : n = Console.ReadLine
        ReDim A(n - 1, n - 1)
        ReDim B(n - 1, n - 1)
        ReDim C(n - 1, n - 1)

        ' vnos matrik
        For i = 0 To n - 1
            For j = 0 To n - 1
                Console.Write("Podaj clen A(" & i & "," & j & ") = ")
                A(i, j) = Console.ReadLine()
            Next
        Next
        For i = 0 To n - 1
            For j = 0 To n - 1
                Console.Write("Podaj clen B(" & i & "," & j & ") = ")
                B(i, j) = Console.ReadLine()
            Next
        Next

        ' množenje
        For i = 0 To n - 1
            For j = 0 To n - 1
                C(i, j) = 0
                For k = 0 To n - 1
                    C(i, j) = C(i, j) + A(i, k) * B(k, j)
                Next

            Next
        Next

        ' izpis
        Console.WriteLine("Transponirana matrika")
        For i = 0 To n - 1
            Console.Write("| ")
            For j = 0 To n - 1
                Console.Write(C(i, j) & " | ")
            Next
            Console.WriteLine()
        Next
        Console.ReadKey()

    End Sub

End Module
```

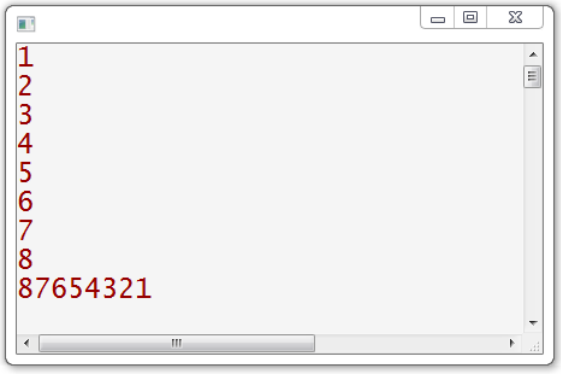
8.14 Kaj dela naslednji program?

Naloga
Podana je programska koda:

Kaj dela program?

```
Module Module1

Sub Main()
Dim a(7) As Integer
For i = 0 To 7
a(i) = Console.ReadLine()
Next

For i = 0 To 7
Console.Write(a(7 - i))
Next
Console.ReadKey()

End Sub

End Module
```

Ali lahko s pregledom programske kode ugotoviš kaj dela?
Rešitev



Slika 8.3: Izpis programa

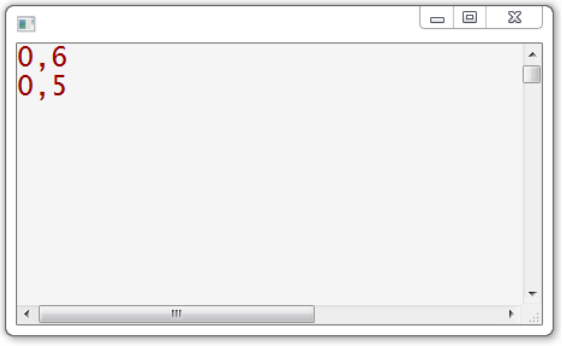
8.15 Kaj izpiše naslednji konzolni program?

Naloga

Kaj se izpiše?

```
Module Module1

Sub Main()
Dim b(1), c(3) As Double

c(1) = 0.3
c(3) = 0.4

For i = 0 To 1
b(i) = c(3 - 2 * i) + 0.2
Console.WriteLine(b(i))
Next
Console.ReadKey()
End Sub

End Module
```

Rešitev



Slika 8.4: Izpis programa

Vlado Stankovski

Visual Basic vaje za gradbenike

Visual Basic vaje za gradbenike je kakovostno pripravljen univerzitetni učbenik, ki na praktičen način študentom gradbeništva približa svet računalništva in informatike. Poudarek učbenika je na razumevanju računalnika, kot stroja, ki ga je možno programirati. Visual Basic je programski jezik, ki se ga lahko uporablja za hiter razvoj inženirskih programov. Predstavljen je postopek izdelave enostavnih programov ter nekaj osnovnih sestavin, nujnih pri programiranju, kot so: spremenljivke in aritmetično logični izrazi, algoritmi, možnosti za delo s pomnilnikom in datotekami, postopek razvoja procedur in funkcij, postopek razvoja zahtevnejših numeričnih programov in okenskih aplikacij. Podani primeri vključujejo programe za izračun in prikaz vrednosti sinusne funkcije, izračun ničle funkcije z bisekcijo, izračun nivelete ceste s sprotnim podaljševanjem, transponiranje matrik, reševanje sistemov linearnih enačb, razvrščanje lesa v trdnostne razrede in podobno. Po uspešno opravljenih vajah bo bodoči gradbenik vsekakor pridobil pri samozavesti ob uporabi in tudi razvoju računalniških programov.

prof. dr. Boštjan Brank

Vlado Stankovski je izredni profesor računalništva na Univerzi v Ljubljani. Ima več kot 15 let izkušenj s področja oblakovnega računalništva, porazdeljenih sistemih, semantike, programskega inženirstva, strojnega učenja in podatkovnega rudarjenja ter njihovih aplikacij na področju gradbeništva.

Dr. Stankovski ima izkušnje s področja integracije programske opreme in je v zadnjih 15 letih sodeloval z različnimi tehnologijami vmesne programske opreme. Delal je tudi pri več projektih EU, vključno z DataMiningGrid (2004–2006), InteliGrid (2004–2007), mOSAIC (2011–2013), ENTICE (2015–2018), SWITCH (2015–2018) in trenutno DECENTER (2018–2021). Sodeluje v nedavno oblikovanem konzorciju Superračunalniški center Slovenije in pri slovenskih projektih pametne specializacije, kot je projekt IQ DOM. Vlado Stankovski trenutno aktivno sodeluje v gruči projektov programskega inženirstva Obzorja 2020.

ISBN 978-961-6884-66-2