Uvod v programski jezik Visual Basic.NET

doc. dr. Iztok Kovačič in dr. Matevž Dolenc

11. december 2007

DRAFT

DRAFT

Kazalo

1 Uvod v VB.NET **7**
1.1 Računalniški program . . . 7
1.2 VB.NET . . . 8
1.3 Platforma .NET . . . 8
1.4 Vrste programov . . . 9
1.5 Konzolni programi . . . 9
1.6 Okenski programi . . . 10
1.7 Programske komponente . . . 10
1.8 Spletni programi . . . 11
1.9 Spletne storitve . . . 11

2 Kodiranje programa **13**
2.1 Uvod v kodiranje . . . 13
2.2 Potek izdelave programa . . . 13
2.3 Razvojno programsko okolje . . . 14
2.4 Stavki v VB.NET . . . 15
2.5 Komentarji . . . 16
2.6 Imena . . . 17

3 Konzolni programi **19**
3.1 Konzolni projekt . . . 19
3.2 Konzolni program . . . 19
3.3 Skelet konzolnega programa . . . 20
3.4 Enostaven konzolni program . . . 21
3.5 Vnos podatkov . . . 22
3.6 Prikaz rezutatov . . . 23
3.7 Oblikovanje rezultatov . . . 24
3.8 Dober primer . . . 25

4 Spremenljivke **29**
4.1 Koncept spremenljivke . . . 29
4.2 Tipi spremenljivk . . . 30
4.2.1 Koncept tipa . . . 30
4.2.2 Vrednostni tipi . . . 31
4.2.3 Referenčni tipi . . . 31
4.2.4 Integer . . . 32
4.2.5 Single . . . 32
4.2.6 Double . . . 33

4.2.7 Boolean . . . 34
4.2.8 String . . . 34
4.2.9 Date . . . 35
4.2.10 Object . . . 35
4.3 Vrste spremenljivk . . . 36
4.4 Deklaracije spremenljivk . . . 37
4.5 Začetne vrednosti spremenljivk . . . 38
4.5.1 Začetne vrednosti vrednostnih tipov . . . 38
4.5.2 Začetne vrednosti referenčnih tipov . . . 39

5 Logični izrazi **41**
5.1 Izjave . . . 41
5.2 Primerjalni operatorji . . . 42
5.3 Logični operatorji . . . 43

6 Aritmetika **45**
6.1 Aritmetični izrazi . . . 45
6.2 Mešana aritmetika . . . 46
6.3 Matematične funkcije . . . 46

7 Kontrolni stavki **49**
7.1 Uvod v kontrolne stavke . . . 49
7.2 Pogojni stavki . . . 49
7.3 Zanke . . . 51
7.3.1 Neskončna zanka . . . 51
7.3.2 Do While/Until . . . 52
7.3.3 Do ... Loop While/Until . . . 53
7.4 Stavek For . . . 54
7.4.1 Uvod v stavek For . . . 54
7.4.2 Korak pri stavku For . . . 55
7.5 Stavek For Each . . . 55
7.6 Izbirni stavek Select . . . 56
7.7 Gnezdenje kontrolnih stavkov . . . 56
7.8 Primeri . . . 58
7.8.1 Izračun n! . . . 58
7.8.2 Primer: Ocene . . . 58
7.8.3 Ugotavljanje praštevil . . . 59

8 Spremenljivke z indeksi **61**
8.1 Uvod v polja . . . 61
8.2 Deklaracije polj . . . 61
8.3 Uporaba polj . . . 62
8.4 Začetne vrednosti . . . 62
8.5 Dinamična polja . . . 62
8.6 ReDim Preserve . . . 63
8.7 Vektorji in matrike . . . 63
8.8 Primeri . . . 64
8.8.1 Skalarni produkt vektorjev . . . 64
8.8.2 Produkt matrike z vektorjem . . . 64
8.8.3 Vektorski produkt . . . 64

DRAFT

9 Procedure in funkcije **65**
9.1 Koncept procedure . . . 65
9.1.1 Uvod v procedure . . . 65
9.1.2 Namen procedure . . . 66
9.1.3 Imena . . . 67
9.1.4 Procedure . . . 68
9.1.5 Funkcija . . . 69
9.2 Argumenti . . . 70
9.2.1 Uvod v argumente . . . 70
9.2.2 Prenos argumentov . . . 71
9.2.3 Klicni argumenti . . . 72
9.2.4 Argument sprem. z indeksi . . . 73
9.3 Procedure . . . 74
9.3.1 Definicija . . . 74
9.3.2 Podatki . . . 75
9.3.3 Rezultati . . . 75
9.4 Funkcije . . . 76
9.5 Definicija funkcije . . . 76
9.6 Primer funkcije . . . 77

10 Programiranje datotek **79**
10.1 Koncept datoteke . . . 79
10.1.1 Vrste datotek . . . 79
10.1.2 StreamReader in StreamWriter . . . 80
10.2 Delovna mapa . . . 81
10.3 Čitanje datotek . . . 81
10.4 Pisanje datotek . . . 84
10.5 Kodirni sistemi znakov . . . 85
10.6 Oblikovanje rezultatov . . . 86

11 Okensko programiranje **89**
11.1 Uvod . . . 90
11.2 Kontrole . . . 92
11.3 Glavno okno . . . 92
11.4 Kontrolni gradniki . . . 92
11.5 Primer Krog . . . 92
11.5.1 Opis . . . 92
11.5.2 Okno . . . 92
11.5.3 Koda . . . 92

DRAFT

Poglavje 1

Uvod v VB.NET

1.1 Računalniški program

Koncept programa Računalniški program je zbirka navodil (inštrukcij), ki opisujejo nalogo (sklop nalog), ki jo lahko izvede (reši) računalnik. Beseda program se lahko nanaša na izvorno kodo zapisano v določenem programskem jeziku ali na izvedljiv program, ki ustreza izvornemu programu. Pomen je običajno razviden iz konteksta.

Izvorni program Izvorni program je zapis v obliki niza stavkov, ki ustrezajo določenemu algoritmu oz. postopku (proceduralno ali imperativno programiranje) bodisi vsebuje niz informacij (dejstev) o problemu in metodo reševanja določi sistem samodejno (deklarativno programiranje). Programiranje z VB.NET sodi v prvo, proceduralno paradigmo.

Izvorni program lahko izdela (ustvari) bodisi razvijalec programa (programer) bodisi ga ustvari drug računalniški program (programski generator). Izvorni program v jeziku VB.NETje običajno zapisan v datoteki s končnico vb.

Izvedljiv program Izvorni program je zapisan v programskem jeziku razumljivem razvijalcu programa, ne pa tudi procesorju računalnika. Izvorni program poseben prevajalni program (compiler) prevede v obliko primerno za izvajanje, v izvedljiv program. Izvedljiv program je bil še pred nekaj leti vedno zapisan v strojnem jeziku. To je zapis niza ukazov, ki jih zna izvesti procesor računalnika. Pravimo, da računalnik razume strojni jezik. Izvedljiv program je običajno zapisan v datoteki s končnico exe.

Ko uporabnik računalnika sproži izvajanje programa, izvedljiv program najprej poseben program operacijskega sistema naloži v glavni pomnilnik in nato sproži izvajanje inštrukcij po navodilih programa, dokler se program ne konča. Med izvajanjem program pogosto uporablja vhodne in izhodne naprave, ki se ob zagonu računalnika pripravijo za uporabo.

Modernejši programski jeziki, med katere sodi tudi VB.NET, zapišejo izvedljiv program v posebnem vmesnem jeziku (CIL). Potrebno je dodatno izvajalno okolje (CLR), ki ta zapis med izvajanjem programa sproti spreminja v ukaze strojnega jezika, ki jih procesor tudi izvede. Kratica CIL pomeni Common Intemedate Language, CLR pa Common Language Runtime.

1.2 VB.NET

Lastnosti jezika VB.NET

- Spada med jezike platforme .NET. Druga dva sta C++ in C#. Vsem je skupna knjižnica programskih komponent Framework.NET.
- Podpira proceduralni in objektni način izdelave programov. Proceduralni način je lažji, ker je bolj podoben običajnemu načinu reševanja problemov (po danem postopku, po korakih). Primeren je za učenje osnov programiranja. Objektni način je bolj zmogljiv od proceduralnega, je pa konceptualno težji, ker običajno problemov ne rešujemo z objekti.
- Stavki jezika VB.NET pogosto vsebujejo angleške besede, zato je zapis programa lahko razumljiv in pregleden.
- Rezultat prevajanja programa v jeziku VB.NET ni program v strojnem jeziku (strojna koda). Preveden program je zapisan v posebnem vmesnem jeziku MSIL (Microsoft Intermediate Language). Podoben sistem pozna programski jezik Java.
- Za izvajanje programa skrbi posebno izvajalno okolje (Framework Runtime), ki je standarden del operacijskega sistema Windows XP s popravki (Service Pack 2). Ko sprožimo izvajanje programa, izdela izvajalni sistem učinkovito strojno kodo, ki je prilagojena zmogljivostim procesorja.

Primerjava z drugimi jeziki Predhodnik jezika VB.NET je še vedno zelo razširjen jezik Visual Basic 6, ki je enostavnejši in lažji od VB.NET. VB 6 omejeno podpira objektno programiranje. Uveljavil je enostaven način izdelave okenskih programov z vizualnim programiranjem in tehnologijo COM programskih komponent. VB.NET je težji od VB 6, ker temelji na ogrodju Framework.NET, ki vsebuje veliko število programskih komponent. Komponente so sicer razdeljene v imenske prostore, ampak jih je v tudi v posameznem imenskem prostoru za manj izkušene programerje preveč.

Na VB.NET nekateri profesionalni programerji še vedno gledajo zviška, češ da je zaradi mnogih vključenih angleških besed preveč dolgovezen in da ne upošteva uveljavljenih rešitev, ki jih spoštujejo jeziki C++, Java in C# . Očitajo mu tudi omejevanje svobode pri kodiranju, saj se določeni elementi zapišejo samodejno, na primer presledki v izrazih.

Zelo malo je elementov, ki se jih v primerjavi z jezikom C# ne da zapisati v jeziku VB.NET. Glede hitrosti izvajanja programov praktično ni razlik med jeziki C++, C# in VB.NET, saj vsi uporabljajo iste programske komponente iz knjižnice Framework.NET.

1.3 Platforma .NET

Platforma .NET je skupni pojem za zbirko produktov in tehnologij firme Microsoft, ki omogočajo izdelavo in uporabo programov z operacijskim sistemom Windows.

DRAFT

1.4 Vrste programov

Tako kot imamo različne vrste računalnikov (prenosne, namizne, delovne postaje, domače računalnike, strežniške računalnike, ...), so tudi programi lahko različni:

- okenski programi,
- konzolni programi,
- spletni programi,
- knjižnice programskih komponent,
- spletne storitve (XML Web Services),
- servisni programi, ...

Programiranje različnih vrst programov zahteva različna znanja. Enostavno je izdelati konzolne in okenske programe, ki jih namestimo in izvajamo na delovni postaji ali prenosnem računalniku. Izdelava spletnih programov je bila do nedavna domena specialistov, vendar so razvojna orodja vse bolj dostopna tudi neprofesionalnim programerjem.

1.5 Konzolni programi

Konzolni programi so programi, ki komunicirajo z uporabnikom s pomočjo tipkovnice in konzolnega okna, ki ga uporabnik vidi na zaslonu. Informacije so zapisane v obliki besedil. Grafičnih elementov konzolno okno ne pozna. Lastnosti konzolnih programov so:

- Navodila (podatke, zahteve) programu vnaša uporabnik s tipkovnico, program prikazuje odgovore (rezultate) v obliki besedil v konzolnem oknu.
- Za večje pogovorne programe tak uporabniški vmesnik ni primeren, ker temelji na besedilih (ni grafični vmesnik).
- Konzolni programi so zaradi enostavnosti zelo koristni za učenje programiranja.
- Konzolni programi so primerne rešitve v primerih reševanja nalog brez uporabniškega vmesnika (strežniški programi).
- Konzolni programi lahko komunicirajo z drugimi programi tudi s pomočjo datotek ali z izmenjavo sporočil po uveljavljenih mrežnih protokolih (TCP/IP, HTTP, ...).
- Tipkovnica je standardna vhodna naprava konzolnega programa, smemo definirati drugo napravo, na primer datoteko.
- Konzolno okno je standardna izhodna naprava konzolnega programa, smemo definirati drugo napravo, na primer datoteko.

Razvojno okolje Visual Studio (Visual Basic Express) omogoča s posebnim tipom projekta Console Application učinkovito izdelavo konzolnih programov.

1.6 Okenski programi

Za okenske programe je značilno glavno okno pravokotne oblike z glavnim menijem in drugimi kontrolnimi elementi, ki predstavljajo grafični uporabniški vmesnik. Okno lahko povečamo, zmanjšamo ali spremenimo v ikono v opravilni vrstici. Uporabljamo lahko tudi dodatna okna, najpogosteje pogovorna okna, za katere veljajo specifična pravila. Določena pogovorna okna za izbiranje datotek, map in barv so standardzirana.

V okno se zapisujejo informacije o stanju naloge v obliki besedil in slik. Okno lahko vsebuje tudi elemente za vnos podatkov, ki spominjajo na obrazce (formularje). Kontrolo nad programom izvajajo posebni kontrolni gradniki (programske komponente). Pogosti so gumbi, ki so razporejeni v nizu - orodjarne. Zelo uporaben kontrolni element je tudi kontekstni (pojavni) meni, ki ga praviloma sprožimo z desno tipko. Poleg tipkovnice je pomembna naprava tudi miška. Obnašanje okenskih programov je standardizirano, zato se uporabe okenskih programov naučimo zelo hitro.

Razvoj novih okenskih programov s sodobnimi razvojnimi okolji (na primer VB.NET 2005 Express) in tehniko vizualnega programiranja se je v zadnjih letih zelo poenostavil.

Razvojno okolje Visual Studio (Visual Basic Express) s posebnim tipom projekta Windows Application omogoča učinkovito izdelavo okenskih programov.

1.7 Programske komponente

Knjižnica komponent ali knjižnica razredov napisanih v jeziku VB.NET (C#, C++) je poseben tip projekta (Class Library) v razvojnem okolju Visual Studio. Komponente lahko večkrat uporabimo v različnih programih. Z uporabo že izdelanih programskih komponent in/ali izdelavo lastnih komponent lahko zelo povečamo učinkovitost izdelave programov.

- Knjižnica komponent je fizično datoteka s končnico dll. Vsebuje binarne programske komponente.
- Zapisi komponent v programskem jeziku so definicije razredov, ki so temelj objektnega (predmetnega) programiranja. Med programske komponente spadajo tudi kontrolni gradniki (kontrole), ki se uporabljajo v okenskih programih v povezavi s tehniko vizualnega programiranja.
- Knjižnico komponent lahko uporabimo v poljubnem okenskem ali konzolnem programu tako, da dodamo ustrezno sklicevanje na knjižnico (Add Reference).
- Knjižnica je vedno zapisana v binarni obliki, tako da je avtorstvo izvorne kode primerno zaščiteno.
- Kvalitetne programske komponete so iskano tržno blago.

Razvojno okolje Visual Studio (Visual Basic Express) s posebnim tipom projekta Class Library omogoča učinkovito izdelavo knjižnice programskih komponent.

DRAFT

1.8 Spletni programi

Spletni programi podpirajo različne tehnike porazdeljenega procesiranja. Najbolj pogost način je sistem odjemalec/strežnik. Uporabnik komunicira s strežniškim programom bodisi s pomočjo brskalnika bodisi s posebnim odjemalskim praviloma okenskim programom (fat or rich client). Izmenjava sporočil med odjemalskim in strežniškim programom teče praviloma po spletnem protokolu HTTP.

Prednosti spletnih programov

- Strežniški program se lahko poveže z več različnimi odjemalci (hkrati).
- Uporabniku ni treba skrbeti za nameščanje in posodabljanje programa na strežniku.
- Če je odjemalski program brskalnik, uporabnik nima nobenega dodatnega dela, saj je brskalnik del operacijskega sistema.
- Podatki povezani z obravnavano nalogo so lahko shranjeni samo enkrat v skupnem skladišču - relacijski podatkovni bazi. To nalogo običajno opravlja poseben strežniški program, ki se pogosto izvaja na drugem strežniškem računalniku. V takem primer govorimo o trislojni arhitekturi (brskalnik, program, ki reši nalogo in strežnik podatkov).
- Usluge spletnih programov je enostavno tržiti, saj so strežniški programi varno nameščeni na oddaljenem računalniku.

Pomanjkljivosti spletnih programov

- Izdelava spletnih programov je zahtevna in do nedavna ni bila dostopna širokemu krogu razvijalcev. Razvojna orodja obstajajo, vendar so zahtevna.
- Odzivnost odjemalcev zaradi komuniciranja po omrežju je pogosto počasna in odvisna od prometa v omrežju. Dostop do interneta je občasno moten.
- Težave so povezane tudi z varnostjo podatkov (napadi virusov, prestrezanje podatkov, predstavljanje z lažnimi identitetami, namerno motenje delovanja programa). Izdelava varnega spletnega programa zahteva veliko znanja.

Razvojno okolje Visual Studio 2005 omogoča izdelavo spletnih programov s projektno predlogo ASP.NET Website.

1.9 Spletne storitve

Spletne storitve XML (XML Web Services) Spletne storitve XML (XML Web Services) je novejša tehnologija spletnih programov, ki se uspešno razvija od leta 2001 dalje. Za razliko od običajnih spletnih programov, temeljijo XML spletne storitve na uveljavljenih standardih: XML, SOAP in WSDL. SOAP je

XML-jezik za prenos podatkov, WSDL pa je XML-jezik za specifikacijo spletnih storitev.

XML spletna storitev je spletni program, določen z URL (Universal Resource Locator), ki temelji na sporočilih. Sporočila so vhodna in izhodna. Spletno storitev lahko poenostavljeno definiramo kot program, ki je sposoben rešiti dano nalogo s pomočjo sistema definiranih sporočil, ki jih lahko zapišemo v standardnem XML-jeziku WSDL (Web Services Definition Language). Sporočilo je zelo podobno definiciji funkcije: ima argumente določenega tipa (parametre), ki so vhodni in/ali izhodni) in rezultat določenega tipa.Programski jezik, ki se uporablja za izvedbo reševanja naloge ni posebej predpisan, saj je interne narave. Uporaba spletnih storitev lahko poteka med različnimi operacijskimi sistemi, če so standardi (SOAP in WSDL) pravilno upoštevani!

Programiranje spletnih storitev Razvojno okolje Visual Studio 2005 s s projektno predlogo ASP.NET WebService podpira razvoj XML spletnih storitev.

2. generacija XML spletnih storitev je od 2003 dalje poznana pod imenom arhitektura WS-* (WS-* architecture). Temelji na dodatnih industrijskih standardih z oznako WS-*, ki so jih definirala velika računalniška podjetja: IBM, Microsoft, Sun, ...). Standardi določajo postopke varnega komuniciranja in izmenjave zahtevnejših sporočil (na primer izvajanje transakcij). Podjetje Microsoft je razvilo poseben podsistem WCF (Windows Communication Foundation) znotraj operacijskih sistemov Windows XP in Windows Vista, ki temelji na XML spletnih storitvah 2. generacije.

Razvojno okolje Visual Studio 2005 s s projektno predlogo WCF Service podpira razvoj WCF spletnih storitev, ki podpirajo arhitekturo WS-*.

DRAFT

Poglavje 2

Kodiranje programa

Poglavje uvodoma razloži pojem kodiranje programa. Sledi pregled dostopnih razvojnih okolij za izdelavo programov. Osrednji del se ukvarja z zapisom programa v jeziku VB.NET v obliki niza stavkov. Razložena so leksična pravila za zapis stavkov (zapis stavka v eni vrstici, zapis preko več vrstic). Predstavljen je koncept sestavljenega stavka. Razložen je zapis komentarjev v programu. Na koncu so opisana pravila za imena. Dodan je tudi seznam napotkov za dobra imena.

2.1 Uvod v kodiranje

Kodiranje programa je pisanje programa v izbranem programskem jeziku, na primer v jeziku VB.NET.

- Sinonim besede kodiranje je šifriranje: zapis v nerazpoznavni obliki.
- Zapis programa v programskem jeziku (programska koda) običajnim uporabnikom praviloma ni razumljiv.
- Med kodiranjem programa sproti preiskušamo program (testiranje programa). Med testiranjem odkrijemo napake in ustrezno popravimo kodo.

2.2 Potek izdelave programa

Izdelava večjih programov poteka po fazah. Vsebina vsake faze je določena, določeni so tudi njeni rezultati. Postopek za izdelavo programa lahko zapišemo z naslednjimi koraki:

- Najprej se izdela analiza problema, problem se podrobno prouči in razišče.
- Sledi načrt programa. V načrtu se določijo lastnosti programa, arhitektura programa, algoritmi za določene dele programov.
- Če je načrt dobro narejen in dokumentiran, je kodiranje programa, ki sledi načrtovanju lažji del celotne izdelave.

- Testiranje (preiskušanje) izdelanega programa je zadnja faza pred namestitvijo. Cilj testiranja je odkriti čim več napak in slabosti. Včasih je koristno v testiranje vključiti tudi končne uporabnike (beta-testiranje).
- Namestitvi programa sledi najdaljša faza - uporaba in vzdrževanje (odpravljanje napak in posodabljanje) programa.

Pogosto je potrebno prejšnje faze popraviti, kar naredi izdelavo programa za evolucijski (iteracijski) sistem.

2.3 Razvojno programsko okolje

Za učinkovit razvoj programov se uporabljajo celovita razvojna okolja (ang. Integrated Development Environment, IDE). Razvojno okolje je v celoto povezan sklop programov, ki razvijalcu omogoča izdelavo novih in urejanje obstoječih programov. Mnoga razvojna okolja so prosto dostopna (zastonj), druga so komercialni produkti. Primeri uveljavljnih razvojnih okolj so:

- Visual Studio 2005 (VB.NET, C#, C++, spletni programi, spletne storitve, ...), komercialni produkt, obstajajo različice glede zmogljivosti.
- Visual Basic 2005 Express..
- Visual C# 2005 Express.
- MonoDevelop (programski jeziki: C#, VB.NET, platforme Windows, Linux).
- Eclipse (programski jezik Java).
- JCreator (programski jezik Java).
- NetBeans IDE (programski jezik Java).

Preprost program v jeziku VB.NET lahko zapišemo (skodiramo) z urejevalnikom Notepad (shranimo s končnico vb) in prevedemo s prevajalnikom (vbc), ki je vgrajen v prosto dostopnem razvojnem sistemu .NET Framework 2.0 SDK (Software Development Kit). Na primer z ukazom:

```
vbc PozdravSvetu.vb
```

dobimo izvedljiv program PozdravSvetu.exe. Program zapisan v datoteki PozdravSvetu.vb je lahko zelo kratek:

```
Module Module1
  Sub Main()
      MsgBox("Pozdrav svetu!")
  End Sub
End Module
```

DRAFT

2.4 Stavki v VB.NET

Programski jezik, slovnica in slovnično pravilen program Program je zapis, ki določa računalniku potek reševanja določene naloge.

Program najprej zapišemo v izbranem programskem jeziku (izvorni jezik, source). Jezik je na splošno sredstvo za komuniciranje. Jezik pogosto določa pravila, kako pravilno zapisati (oblikovati) sporočilo (informacijo) v obliki niza stavkov. Pravila pogosto imenujemo slovnica (gramatika) oz. skladnja (sintaksa). Tudi programski jeziki imajo pravila, kako zapisati slovnično pravilen program. V primerjavi z naravnimi jeziki so pravila enostvana in se jih lahko hitro naučimo.

Ker procesor računalnika ne razume izvornega programskega jezika, ga mora poseben prevajalni program (prevajalnik) prevesti v procesorju razumljivo obliko - v preveden program. Prevajalnik slovnično nepravilnih programov ni pripravljen prevesti, zato razvijalcu programa sporoči podrobne informacije o slovničnih napakah (lego in vrsto nepravilnosti).

VB.NET velja za jezik z majhnim obsegom slovničnih pravil. Mnoga so enostavna in inuitivna.

Stavki: enote zapisa programa Tako kot pri naravnih jezikih je tudi zapis programa VB.NET sestavljen iz stavkov. Stavek je enota programa, ki predstavlja zaključeno operacijo. Na primer, deklaracijski stavek, prireditveni stavek, pogojni stavek ponavljalni stavek, klicni stavek (klic procedure).

Vsak stavek v VB.NET zapišemo v svojo vrstico, zato znak za konec stavka ni potreben in VB.NET ga ne pozna.

Stavek preko več vrst Če je stavek dolg, ga lahko zapišemo preko več vrst. Stavek nadaljujemo v naslednjo vrsto tako, da na koncu stavka dodamo znak presledek in znak podčrtaj. Na primer:

```
d = Math.Sqrt( (x2 - x1)^2 + _
               (y2 - y1)^2 )
```

V dobro sestavljenem programu so zapisi stavkov preko več vrsti pogosti, saj omogočajo celovit pogled na daljši stavek.

Več stavkov v isti vrsti Več stavkov smemo zapisati v isto vrsto, če jih med seboj ločimo z znakom : (dvopičje). V splošnem tega ne počnemo, ker stiskanje lahko pomeni nepreglednost, razen v primeru kratkih stavkov, ki so med seboj logično močno povezani. Na primer:

```
x = 0: y = 0: z = 0
```

Enostavni in sestavljeni stavki Stavki v VB.NET so enostavni in sestavljeni. Med enostavne spadajo: prireditveni stavki, deklaracijski stavki, klici procedur, ...

Primeri sestavljenih stavkov so: bločni pogojni stavki, izbirni stavek Select, stavki Do, stavki For, ... Sestavljeni stavki so vedno napisani preko več vrst in praviloma smejo vsebovati druge stavke: enostavne ali sestavljene.

2.5 Komentarji

Komentarji so pojasnila vgrajena v kodo programa. Namenjeni so razvijalcem programa, da hitreje razume delovanje programa. Sintaksa je preprosta:

- Komentar se začne z znakom ' (enojni narekovaj) in se nadaljuje do konca vrstice.
- Izjema je znak 'zapisan znotraj konstantnega besedila, na primer: Založba O'Reilly".
- Prevajalnik komentarjev ne upošteva (preprosto jih preskoči).

Primeri komentarjev:

```
' Program dimenzionira linijsko konstrukcijo.
'
' Podatki: ...
'
' Rezultati: ...
' ...
Sub Main
   Dim ne As Integer ' Število elementov.
   Dim nv As Integer ' Število vozlišč.
   ...
End Sub
```

Napotki za izdelavo dobrih komentarjev

- Komentarjev ne sme biti preveč. Ne smejo zmanjševati obseg kode, ki jo hkrati vidimo na zaslonu.
- Razumljiv program z malo komentarji je samodokumentiran. Pomembna so dobra imena spremenljivk in procedur.
- Komentarji naj bodo na začetku procedur in funkcij. Igrajo naj vlogo specifikacije procedur.
- Komentarji naj bodo kratki vendar celoviti. Dovoljene so smiselne okrajšave.
- Nepotrebni so komentarji, ki pojasnjujejo delovanje stavkov programskega jezika. Na primer, stavku If je nesmiselno dodati komentar: ' To je pogojni stavek.
- Pogosto napišemo komentar na desni strani izvedljivega stavka.
- Pametno je komentirati vse deklaracije globalnih spremenljivk v konzolnem programu tako, da vsako deklariramo posebej v svoji vrstici.

Celovita razvojna okolja za izdelavo programov vsebujejo možnost, da izbrane vrstice s klikom miške zakomentiramo in obratno.

DRAFT

2.6 Imena

Imena imajo naslednji elementi v programu VB.NET:

- spremenljivke,
- konstante,
- procedure Sub inprocedure Function, lastnosti Property,
- formalni argumenti procedur,
- tipi spremenljivk, ...

Pravila za imena

- V imenih smejo nastopati črke, števke in znak _ (podčrtaj). Črke določa znakovni standard Unicode.
- Ime se mora začeti s črko. Podčrtaj igra vlogo črke.
- Praktične omejitve glede dolžine imen ni.
- Nekatere besede za imena niso dovoljene, ker imajo poseben pomen (rezervirane besede). Na primer: For, Each, While, Until, Sub, Function, Do, Next, Loop, Exit, If, Then, Nothing, ... Če jih na vsak način želimo uporabiti v programu, jih moramo obdati z oglatimi oklapaji, na primer [Exit].

Napotki za primerna imena

- Ime naj spominja na element, ki ga predstavlja. To je temeljni koncept samodokumentiranih programov.
- Ime je lahko kratko ali dolgo.
- Kratka imena so primerna za indekse (i,j,k, ...) in podatke, ki velikokrat nastopajo v izrazih. Kratkim imenom moramo posvetiti dovolj pozornosti, sicer bomo izdelali kriptičen, težko razumljiv program.
- Dolga imena pogosteje priredimo elementom, ki se v programu pojavijo bolj redko. Sicer dolga imena niso praktična, ker jih moramo tipkati in ker zasedejo veliko prostora na zaslonu.
- Imena velikokrat sestavimo iz dveh besed. Na primer: obsegKroga, dolžinaKonzole.Če so besede predolge, jih primerno okrajšamo.Na primer: plElipse, obElipse.
- Velike in male črke v imenih prevajalniku ne pomenijo nobene razlike. Male črke so za oko lepše; časopisi in knjige s samimi velikimi črkami ne obstajajo.
- Samo velike črke v imenih uporabljamo samo za posebne namene. Kombinacija malih in velikih črk je najbolj primerna za zapis razumljivih programov.
- Obstajajo napotki, kdaj naj se ime začne z veliko črko (javni elementi) in kdaj z malo (zasebni elementi). Podrobnosti so v drugih poglavjih.

DRAFT

Poglavje 3

Konzolni programi

Poglavje obravnava programiranje konzolnih programov. Najprej je predstavljen koncept konzolnega programa. Sledi informacija o uporabnosti konzolnih programov. Nato je predstavljen in podrobno razložen primer enostavnega konzolnega programa. Sledi razlaga pogovornega vnosa podatkov in prikaza rezultatov v konzolnem oknu. Na koncu je podrobno razložen primer tipičnega konzolnega programa.

3.1 Konzolni projekt

Postopek: nov konzolni projekt Izdelava novega konzolnega programa poteka kot nov projekt po naslednjih korakih.

1. Poženi Visual Studio ali VB 2005 Express.
2. V meniju File izberi ukaz New in nato ukaz Project.
3. Na levi strani v rubriki Project Types izberi Visual Basic in nato Windows.
4. Na desni strani oziroma v sredini izberi Console Applications.
5. Poleg Name: določi ime projekta. Ime projekta skrbno izberi.
6. Če obstaja podatek Location, določi mapo, v kateri se bo ustvarila mapa projekta. Mapo lahko izbereš z gumbom Browse.
7. Zapri vsa druga področja razen tistega, ki vsebuje Module ... End Module.

Vse je pripravljeno za razvoj konzolnega programa.

3.2 Konzolni program

Koncept konzolnega programa Konzolni program je program, pri katerem lahko uporabnik med izvajanjem programa komunicira s programom s pomočjo tipkovnice in ukaznega okna (Command Prompt Window). Starejši operacijski sistemi so to okno imenovali DOS okno.

- Izdelava konzolnih programov je bolj enostavna od izdelave okenskih programov.
- Konzolni programi so zato zelo primerni za učenje osnov programiranja.
- Pogosto pa konzolni programi nimajo uporabniškega vmesnika. Drugi programi lahko konzolni program poženejo in tudi ustavijo. Konzolni programi so torej lahko gradniki integriranih sistemov, pri katerih reševanje naloge porazdelimo preko več samostojnih programov. Izdelava takega sistema je lažja, sistem je bolj robusten za vzdrževanje. Hkrati ga lahko razvija več programerjev.

Uporaba konzolnih programov na strežnikih

- Konzolni programi so primerni za uporabo na strežnikih, kjer ni uporabnikov (tam okenski programi nimajo kaj delati).
- Ob zagonu programa lahko z zagonskimi parametri določimo začetne podatke (parametre) takega programa.
- Komunikacija pogosto poteka preko datotek (vhodne in izhodne datoteke). Konzolni programi lahko med seboj komunicirajo tudi po internetskem protokolu TCP/IP ali spletnem protokolu HTTP.

3.3 Skelet konzolnega programa

- Konzolni program sestavlja eden ali več modulov. Enostaven program ima en sam modul.
- Modul pogosto vsebuje globalne podatke in niz procedur.
- Procedure so pogosto brez parametrov (argumentov).
- Modul lahko vsebuje še druge elemente (razrede, lastnosti, dogodke, ...)
- Globalni podatki so dostopni v vseh procedurah. Deklariramo jih izven procedur, običajno na vrhu modula.
- Izvajanje programa se privzeto začne v proceduri Main.

```
Module Module1
    Deklaracije globalnih podatkov ...

    Sub Main
        ...
    End Sub
    ...
    Druge procedure (vrste Sub ali Function)
    ...
End Module
```

Na primer, vzorec skeleta računskega programa lahko zapišemo:

```
Module Module1
   Dim a,b As Single, ...
   ...

   Sub Main()
       PreberiPodatke()
       Izračunaj()
       PrikažiRezultate()
       Console.ReadLine()
   End Sub

   Sub PreberiPodatke()
       Console.Write("Podaj a: ")
       ...
   End Sub

   Sub Izračunaj()
       ...
   End Sub

   Sub PrikažiRezultate()
      ...
   End Sub
End Module
```

3.4 Enostaven konzolni program

Opis Enostaven pogovorni konzolni program naredimo takole:

- Vsebuje en sam modul.
- Podatke vnašamo pogovorno preko tipkovnice s pomočjo odzivnikov (promptov). Na primer:

```
   Dim a As Single
   ...
   Console.Write("Podaj a: ")
   a = Console.ReadLine    ' Prebrana vrstica se
                           ' pretvori v realno število
```

- Rezultate prikazujemo v konzolnem oknu z: Console.Write(...) in/ali Console.WriteLine(...).
- Podatke in rezultate naloge definiramo kot globalne spremenljivke (pred proceduro Main).
- Program, ki pri danih podatkih izračuna rezultate, praviloma razdelimo na 3 procedure (poleg procedure Main): Sub Preberi, Sub Izračunaj, Sub Prikaži.

Koda

```
Module Module1
   Dim ... ' Globalni podatki
   ...

   Sub Main()
       Preberi()
       Izračunaj()
       Prikaži()

       Console.ReadLine()   ' Zadrži prikaz konzolnega okna.
   End Sub

   ' Procedura prebere ...
   Sub Preberi()
       ...
   End Sub

   ' Procedura izračuna ...
   Sub Izračunaj()
       ...
   End Sub

   ' Procedura prikaže ...
   Sub PrikažiRezultate()
      ...
   End Sub
End Module
```

3.5 Vnos podatkov

Podatke v konzolnem programu vnašamo pogovorno z odzivniki. Odzivnik je sporočilo, na katerega je potrebno takoj (promptno) odgovoriti. Odgovorimo tako, da odtipkamo zahtevani podatek.

Vnos števila

```
Dim število As Single, vrsta As String

Console.Write("Podaj število: ") ' Prikaz odzivnika.
vrsta = Console.ReadLine         ' Vnos vrstice, na koncu pritisnemo Enter.
število = vrsta                  ' Pretvorba besedila v število
```

Dovoljen je tudi krajši zapis, če velja Option Strict Off. Besedilo se v število tipa Single pretvori samodejno.

```
Console.Write("Podaj število: ") ' Prikaz odzivnika.
r = Console.ReadLine             ' Vnos vrstice in hkratna pretvorba v število.
```

DRAFT

Vnos niza podatkov v vrstici Postopek je sledeč:

1. Preberi vrstico.
2. Razcepi jo na besede.
3. Besede pretvori v ustrezne podatke.

Izvedba je preprosta in je predstavljena s primerom čitanja oštevilčene točke (ind x y):

```
Dim ind As Integer, x,y As Single
Dim vrsta As String, b() As String

Console.Write("Podaj ind x y: ")
vrsta = Console.ReadLine
b = vrsta.Split
ind = b(0)
x = b(1)
y = b(2)
```

Metoda Split razreda String razcepi vrsto na besede (b = vrsta.Split). Med besedami mora biti natanko po en presledek. Začetni element polja b ima indeks 0. Rezultat je polje besed tipa String. b.Length določa število prebranih besed.

Dovoljen je krajši zapis:

```
Console.Write("Podaj ind x y: ")
b = Console.ReadLine.Split
ind = b(0)
x = b(1)
y = b(2)
```

3.6 Prikaz rezutatov

Rezultate prikazujemo v konzolnem oknu z:

- Console.Write(izraz)
- Console.WriteLine(izraz)

Najprej se izračuna izraz, rezultat se pretvori v besedilo na standarden način. Razlika med Write in WriteLine je samo v tem, da WriteLine sproži preskok v novo vrsto, Write pa ne. Write in WritLine sta metodi razreda Console. Tipične vzorec izpisa je:

```
Console.WriteLine("Napetost je " & sigma)
```

Prazno vrstico zapišemo z: Console.WriteLine(). Če se pogosto pojavlja izpis več praznih vrstic, izdelamo proceduro s parametrom, ki določa število praznih vrstic:


```
Sub IzpisPraznihVrst( ByVal nv As Integer)
   For i As Integer = 1 To nv
       Console.WriteLine(nv)
   Next i
End Sub
```

3.7 Oblikovanje rezultatov

Oblikovanje rezultatov je pomembna vendar razmeroma enostavna tema, saj izpisane rezultate vidimo in popravimo, kar nam ni všeč.

Oblikovanje realnih vrednosti Realne vrednosti spremenljivk pogosto želimo izpisati na podano število decimalk v normalni obliki. To naredimo tako, da ob pretvorbi realnega števila v besedilo z metodo ToString s kontrolnim argumentom določimo način oblikovanja. Sledeči primer izpiše realno vrednost na dve decimalki (F2).

```
Console.WriteLine("ploščina = " & pl.ToString("F2"))
```

Izpis v eksponentni obliki ni nič težji, le namesto črke F oz. f uporabimo črko E oz. e. Število poleg črke E pomeni število števk izpisanega števila poleg desetiškega eksponenta.

```
Console.WriteLine("Zgornja meja = " & zgMeja.ToString("E3"))
```

Tretja možnost je generaliziran format G, Gn, kjer se format izpisa določi v odvisnosti od velikosti vrednosti: normalna oblika z decimalkami ali eksponentna oblika na podano število mest (kombinacija formata F in E).

Izpis rezultatov v obliki tabele Ideja je preprosta. Z metodama razreda String PadLeft (dodaj presledke na levi) oziroma PadRight (dodaj presledke na desni), lahko sprožimo dodajanje presledkov danemu besedilu, da bo končno besedilo predpisane dolžine znakov. PadLeft realizira desno poravnavo, presledki se dodajajo na levi. PadRight pa izdela levo poravnava, presledki se dodajajo na desni. Desno poravnavo uporabljamo za števila, levo pa za imena.

Sistem je uporaben, če so vsi izpisani znaki enako široki (pisava Courier New). Na primer, tabelo točk na 3 decimalke pregledn o izpišemo z:

```
Sub IzpišiTočke(x() As Single, y() As Single)
  Dim nt As Integer = x.Length

  Console.WriteLine("      i     x(i)      y(i)  ")
  Console.WriteLine()

  For i As Integer= 1 To nt
     Console.WriteLine(i.ToString().PadLeft(8) & x(i).ToString("F3").PadLeft(10) & y(i).ToStrir
  Next i
End Sub
```

DRAFT

3.8 Dober primer

Predstavljena je dobro narejena samodokumentirana koda programa, ki izračuna vrednost polinoma pn(x) reda n za več vrednosti argumenta x. Polinom podamo pogovorno s stopnjo polinoma n in nizom koeficientov a0, a1, ... , an. Nato v zanki pogovorno beremo vrednosti x, niz končamo s tipko Enter. Rezultat se sproti prikazuje v konzolnem oknu.

Testni primer Pri podatkih za n=2 in koeficientih 3, -2, 1 za a2, a1, a0 in nizu vrednosti 1,2,3 za x, dobimo naslednji prikaz:

```
Stopnja polinoma: 2
Koeficienti polinoma: a0 a1 ... a2: 3 -2 1
p2(x) = 3-2*x+3*x^2

Podaj x: 1
p2(1) = 2

Podaj x: 2
p2(2) = 9

Podaj x: 3
p2(3) = 22

Podaj x:
```

Koda programa

```
' Naloga: Izračun vrednosti polinoma po Hornerjevem algoritmu.
'
' Podatki: red polinoma n, koeficienti polinoma a0, a1, ... an.
' argument x, pri y = pn(x).
'
' Rezultat: vrednost polinoma pn(x). Zaporedoma lahko podamo več
'           podatkov za x.
'
Module Module1
   Dim n As Integer  ' Red polinoma.
   Dim a() As Single ' Koeficienti polinoma.
   Dim p As Single   ' Vrednost polinoma pri x.
   Dim x As Single   ' Argument za izračun vrednosti polinoma.

   Sub Main()
      PreberiPolinom()
      PrikažiPolinom()

      Do
         PreberiX()
         IzračunajVrednostPolinoma()
         PrikažiRezultat()
      Loop
```



```
   End Sub

   ' Prebere stopnjo in koeficiente polinoma.
   Sub PreberiPolinom()
      Dim i As Integer, b() As String

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

      ReDim a(n)  ' Dinamično določi velikost polja.

      Console.Write("Koeficienti polinoma: a0 a1 ... a" & n & ": ")
      b = Console.ReadLine.Split ' Razcepi prebrano vrsto na besede.

      For i=0 To n              ' Besede spremeni v realna števila.
         a(i) = b(i)
      Next i
   End Sub

   ' Prikaže polinom pri x.
   Sub PrikažiPolinom()
      Dim i As Integer

      Console.Write("p" & n & "(x) = " & a(0))

       For i = 1 To n
          If a(i)>=0 Then Console.Write("+")

          If i = 1 Then
             Console.Write(a(i) & "*x")
          Else
             Console.Write(a(i) & "*x^" & i)
          End If
      Next i

      Console.WriteLine()
      Console.WriteLine()
   End Sub

   ' Pogovorno prebere x.
   Sub PreberiX
      Dim vrsta As String

      Console.Write("Podaj x: ") ' Pogovorno preberi x.
      vrsta = Console.ReadLine

      If vrsta = "" Then End
      x = vrsta
   End Sub

   ' Izračuna vrednost polinoma pri x.
   Sub IzračunajVrednostPolinoma()
       Dim i As Integer

```

DRAFT

```
        p = a(n)

        For i = n - 1 To 0 Step -1
            p = p * x + a(i)
        Next i
    End Sub

    ' Prikaže rezultat.
    Sub PrikažiRezultat()
        Console.WriteLine("p" & n & "(" & x & ") = " & p)
        Console.WriteLine()
    End Sub
End Module
```


DRAFT
DRAFT

Poglavje 4

Spremenljivke

Uvodoma je v poglavju razložen ključni element programiranja - spremenljivka. Opisane so razlike med vrednostnimi in referenčnimi spremenljivkami. Sledi pomembno poglavje o deklaracijah spremenljivk s stavkom Dim.

Spremenljivka je element programa, ki predstavlja bodisi podatek programa bodisi predmet (objekt) v programu. Predmeti v programu so elementi v programu, ki predstavljajo sklop podatkov (o stanju predmeta) in sklop procedur (metod), ki spreminjajo stanje predmeta.

4.1 Koncept spremenljivke

Spremenljivke so elementi, ki s svojimi vrednostmi predstavljajo podatke v programu. Vrednost spremenljivke se lahko med izvajanjem programa spreminja.

Lastnosti spremenljivk

- Spremenljvka ima ime. Po imenu se v programskih stavkih na spremenljivko sklicujemo, kadar potrebujemo njeno trenutno vrednost.
- Spremenljivka je določenega tipa. Tip določa niz lastnosti spremenljivke, na primer vrednost. Tip - podatkovni tip - je sklop informacij o podatkih določene vrste v programu. Je neke vrste zapis, ki je dostopen prevajalniku in našemu programu. Tipi so vgrajeni v razvojno okolje, lahko pa razvijalci definirajo in dodajajo svoje tipe.
- Spremenljivka ima vrednost določenega tipa, ki je med izvajanjem programa zapisana v glavnem (RAM) pomnilniku.
- Spremenljivke moramo pred uporabo napovedati (deklarirati) s posebnimi napovednimi oz. deklaracijskimi stavki. Deklaracija jim določi ime in tip.

Kako spremenljivki določimo vrednost Spremenljivki lahko določimo oz. spremenimo vrednost na različne načine:

- S prireditvenim stavkom (najpogosteje), na primer: s = (a + b + c) / 2. To ni enačba znak = (enačaj) imenujemo prireditveni operator.

- Z izhodnimi parametri procedur, na primer: IzračunajKrog(r, ob, pl). ob in pl sta tu izhodna parametra, rezultata procedure.
- Ob deklaraciji lahko spremenljivki določimo začetno vrednost, na primer: Dim naslov As String = "FGG".

4.2 Tipi spremenljivk

Najprej je razložen koncept tipa, ki je tesno povezan s pojmom spremenljivka. Razložene so razlike med referenčnimi in vrednostnimi tipi. Sledi predstavitev vgrajenih primitivnih tipov: Integer, Single, Double, Boolean, String, Date. Na koncu je opisan tip Object, ki je nekaj posebnega.

4.2.1 Koncept tipa

Tip je element programskega jezika, ki določa lastnosti spremenljivk določene vrste.

- Celoštevilske spremenljivke so tipa Integer ali Long.
- Realne spremenljivke so tipa Single (enojna natančnost) ali Double (dvojna natančnost).
- Logične spremenljivke so tipa Boolean. Predstavljajo logično vrednost (True/False).

Tipi se delijo na vrednostne in referenčne tipe. Vgrajeni numerični tipi, tipa Date in Boolean spadajo med vrednostne tipe, večina drugih tipov definiranih v ogrodju Framework.NET je referenčnih. Tudi tip String spada med referenčne tipe.

Tip določa naslednje lastnosti spremenljivk:

- Vrsto tipa (referenčni ali vrednostni tip).
- Konstante (literale).
- Množico dovoljenih vrednosti.
- Velikost prostora v pomnilniku za zapis vrednosti.
- Dovoljene operacije.

Na primer, tip Integer spada med vrednostne tipe. Določa dovoljene celoštevilske vrednosti na intervalu –2147483648 do 2147483647, ki zavzamejo po 4 bajte z dovoljenimi operatorji: +, -, *, /, \, ^ . Konstante (literale) zapišemo tako kot smo navajeni zapisovati cela števila v matematiki (-133, 0, 2630, ...).

DRAFT

4.2.2 Vrednostni tipi

Vrednostni tipi imajo naslednje lastnosti:

- Ime spremenljivke v programu neposredno predstavlja njeno vrednost v pomnilniku.
- Med vrednostne tipe spadajo vsi vgrajeni numerični tipi: Byte, Short, Integer, Long, Single, Double, Decimal, ... Tudi Char, Boolean in Date spadata med vrednostne tipe.
- Vrednostne spremenljivke (spremenljivke vrednostnega tipa) veljajo za zelo učinkovite. Hitro lahko nastanejo in hitro se lahko spet odstranijo.
- Nov (svoj) vrednostni tip lahko definira razvijalec programa v jeziku VB.NET tako, da zapiše stavek Structure Ime-tipa ... End Structure. Tak tip velja samo v programu, v katerem je definiran.

Vrednostni tipi so zelo varni, kadar jih uporabimo kot argumete procedur. Ob klicu procedure se praviloma izdela kopija vrednosti argumenta, tako da pravi podatek v proceduri ni dostopen in ga procedura tudi ne more pokvariti. Del pomnilnika, kjer nastajajo delovne spremenljivke procedur in delovne kopije klicnih argumentov, se logično imenuje sklad (ang. stack).

4.2.3 Referenčni tipi

Referenčni tipi imajo naslednje lastnosti:

- Vrednost spremenljivke referenčnega tipa je kazalec na vrednost, ki je zapisana na drugem mestu v pomnilniku. Vrednosti pogosto pravimo objekt, referenčne spremenljivke pa objektne spremenljivke.
- Kazalec je vedno dolg 4 bajte in predstavlja naslov začetnega bajta vrednosti v glavnem pomnilniku.
- Za nastajanje in odstranjevanje vrednosti referenčnih tipov skrbi poseben smetarski sistem (program, ki je del izvajalnega okolja), ki se imenuje Garbage Collector; GC na kratko.
- Vrednosti referenčnih tipov praviloma ustvarimo z operatorjem New. Tedaj GC ustvari prostor za vrednost in zapiše sklicevanje na ta prostor. Določi se tudi začetna vrednost.
- Večina tipov v ogrodju Framework.NET je referenčnih. Med referenčne tipe spada tudi tip String.
- Sklicevanje na vrednost odstranimo tako, da refeerenčni spremenljivki priredimo vrednost Nothing. Vrednost kasneje fizično odsrani sistem GC. Sistem GC odstranjuje kampanjsko, samodejno se sproži ob primernem času, ko se nabere dovolj nereferenciranih predmetov.
- Na isti predmet se lahko sklicujeta dve ali referenčnimih spremenljivk. Objekt ne bo odstranjen, doker obstaja nanj vsaj ena referenca (sklicevanje).

Referenčni tipi predstavljajo predmete (objekte) v programu. So tesno povezani z objektnim programiranjem. Razvijalec v jeziku VB.NET definira nov referenčni tip tako, da v programu zapiše stavek Class Ime-tipa... End Class.

Primer uporabe vgrajenega referenčnega tipa Random, ki skrbi za generacijjo psevdoslučajnih števil:

```
Dim kocka As New Random
Dim meti(100) As Integer
...
For i=1 To 100
   meti(i) = Random.Next(1,7)
Next i
...
kocka = Nothing
```

4.2.4 Integer

Vrednosti tipa Integer so cela števila. Podrobnosti so naslednje:

- Vrednost v RAM-u zaseda 4 bajte.
- Dovoljene vrednosti so v intervalu: –2147483648 do 2147483647 (od -2,1 do 2,1 milijarde)
- Dovoljene operacije so: + - * / \ ˆ Mod
  - + določa seštevanje.
  - - določa odštevanje.
  - * določa množenje.
  - / pomeni deljenje, kjer je rezultat realno število.
  - \ pomeni deljenje, kjer je rezultat celo število.
  - Mod izračuna ostanek pri deljenju, na primer n Mod m izračuna ostanek pri deljenju m z n. Na primer, 5 Mod 2 je 1.

4.2.5 Single

Vrednosti tipa Single so realna števila. Beseda Single izhaja iz sintagme Single Precision, kar pomeni realno število z enojno natančnostjo (obstaja tudi tip Double). Podrobnosti so naslednje:

- Vrednost v RAM-u zaseda 4 bajte.
- Število točnih mest je 7-8.
- Dovoljene vrednosti poleg 0 so:
  - pozitivne vrednosti so v intervalu: 1.4E-45 do 3.4E+38,
  - negativne vrednosti so v intervalu: -3.4E+38 do -1.4E-45,
- Dovoljene operacije so: + - * / ˆ Mod

DRAFT

- + določa seštevanje (a + b),
- - določa odštevanje (a - b),
- * določa množenje (a*b),
- / določa deljenje(a/b),
- ˆ določa potenciranje (aˆb),
- m Mod n izračuna ostanek pri deljenju m z n. Na primer, 5 Mod 2 je 1, 11 Mod 3 je 2.

Zapis konstant tipa Single V konstanti tipa Single znotraj programa je decimalno ločilo vedno pika (.). Konstante tipa Single smemo napisati v normalni ali eksponentni obliki:(e oziroma E pomeni desetiški eksponent). Na primer:

- 3.14159265
- -10.335
- 12.6E-8, 1.e9

Razredne lastnosti tipa Single Tip Single ima določene tudi naslednje uporabne razredne lastnosti:

- Single.NaN - konstanta Not a Number.
- Single.PositiveInfinity - konstanta, ki predstavlja pozitivno neskončnost.
- Single.NegativeInfinity - konstanta, ki predstavlja negativno neskončnost.
- Single.MinValue, Single.MaxValue - konstanti, ki določata namanjšo in največjo vrednost tipa Single.
- Single.Epsilon - najmanjše pozitivno število tipa Single.

4.2.6 Double

Vrednosti tipa Double so realna števila. Beseda Double izhaja iz sintagme Double Precision, kar pomeni realno število z dvojno natančnostjo (obstaja tudi tip Single). Podrobnosti so naslednje:

- Vrednost v RAM-u zaseda 8 bajte.
- Število točnih mest je 18.
- Dovoljene vrednosti poleg 0 so:
  - pozitivne vrednosti so v intervalu: 4.9E-324 do 1.8E+308
  - negativne vrednosti so v intervalu: -1.8E+308 do -4.9E-324,
- Dovoljene operacije so: + - * / ˆ Mod
  - + določa seštevanje (a + b),
  - - določa odštevanje (a - b),
  - * določa množenje (a*b),

  - / določa deljenje(a/b),
  - ˆ določa potenciranje (aˆb),
  - m Mod n izračuna ostanek pri deljenju m z n. Na primer, 5 Mod 2 je 1, 11 Mod 3 je 2.

Decimalno ločilo konstant tipa Double je vedno pika. Števila lahko zapišemo v normalni ali eksponentni obliki. Črki E oz. e pomenita osnovo za desetiški eksponent. Na primer:

- -2.532
- 0.123E-3
- 2.5e10

4.2.7 Boolean

- Tip Boolean določa logično vrednost True ali False.
- Določa stanje z dvema vrednostima: Da/Ne, Res/Ni res, Prav/Narobe.
- Pogosto se uporablja v pogojnih stavkih.
- Privzeta začetna vrednost je False.
- Tip Boolean zasede 4 bajte.

Primeri deklaracij

```
Dim niNapak, stanjeSpremenjeno, stereo As Boolean

Dim vRedu As Boolean ' Deklaracija spremenljivke tipa Boolean
...
vRedu = True
...
If vRedu Then
...
Else
...
End If
```

4.2.8 String

Tip String omogoča programiranje besedil (nizov znakov). Vrednost tipa String je besedilo. Tip String spada med referenčne tipe. Na primer:

Dim naslov As String Dim ocena As String = "Odlično"Dim podatki, rezultati As String

Lastnosti tipa String:

- Vrednosti tipa String so besedila (nizi znakov).
- Konstanta (literal) tipa String je besedilo med dvojnimi narekovaji. Na primer: "Univerza v Ljubljani"

DRAFT

- Vsak znak je zakodiran po standardu Unicode v 2 bajtih (32 bitov).
- Tip String je referenčni tip. Spremenljivka tipa String vsebuje naslov besedila (kazalec, ki kaže na mesto besedila).
- Privzeta začetna vrednost spremenljivke tipa String ni določena (Nothing).
- Velikost prostora vrednosti tipa String je odvisna od števila znakov prirejenega besedila.
- & je edini operator tipa String. Imenujemo ga operator združevanja besedil. Dve besedili združi v eno besedilo. Na primer: "Računalništvo"& "in"& "informatika".

4.2.9 Date

Tip Date spada med kompleksne tipe. Omogoča programiranje časovnih točk: datum in čas.

- Vrednost tipa Date je kombiniran datumsko časovni podatek.
- Vrednost je celo število, ki obsega 8 zlogov (bajtov).
- Enota se imenuje tik in je dolga 100 nanosekund.
- Privzeta vrednost 0 tikov določa začetek leta 1 (1. Januar, opoldne po gregorijanskem koledarju).
- Konstanta tipa Date ima obliko: #mesec/dan/leto ura:minuta:sekunda PM|AM#, na primer: #8/24/2003 1:25 PM#.

Tip Date ima mnogo uporabnih lastnosti (na primer: Date, Day, DayOfWeek, DayOfYear, Hour, Minute, Month, Second, TimeOfDay, Ticks) in metod (na primer: Add, Subtract, ToLocalTime, ToUniversalTime). Uporabne so tudi razredne lastnosti (na primer: Now, ToDay, UtcNow) in razredne metode (na primer: IsLeapYear).

Tip Date dovoljuje tudi uporabo operatorjev: +, +=, >, >=, <, <=, =, <>. Metode upoštevajo krajevne nastavitve operacijskega sistema, na primer za prikaz datumov.

Za programiranje časovnih intervalov je primernejši tip TimeSpan, za koledarske operacije skrbita tipa Calendar in GregorianCalendar.

4.2.10 Object

Tip Object je nekaj posebnega. Je referenčni tip. Vsi tipi so izpeljani iz tipa Object. To pomeni, da je vsaka spremenljvka tudi tipa Object. Ta lastnost je močno povezana s tehniko objektnega programiranja, pri kateri je dedovanje tipov nekaj običajnega.

Smemo deklarirati spremenljivke tip Object. Spremenljivki tipa Object smemo prirediti vrednost kakršnegakoli drugega tipa. Praktično bi to pomenilo, da nam ni spremenljivkam določiti tip. To je sicer res, ampak ni priporočljivo, saj ni več kontrole, ali imajo spremenljivke smiselne vrednosti. premenljivke tipa Object uporabljamo samo v posebnih primerih.

4.3 Vrste spremenljivk

V platformi .NET obstajata dve vrsti spremenljivk, vrednostne in referenčne, ki sta si med seboj zelo različni. Vrednostne spremenljivke neposredno predstavljajo vrednosti (podatke), referenčne spremenljivke pa posredno (indirektno) predmete (sklop podatkov in operacij).

Referenčne spremenljivke so kompleksne, zato za njihovo upravljanje skrbi poseben sistem GC.

Vrednostne spremenljivke Vrednostna spremenljivka neposredno določa vrednost, ki je zapisana v glavnem pomnilniku. Vrednostne spremenljivke določajo vrednostni tipi, ki so definirani s strukturo

```
Structure ImeTipa
...
End Structure
```

Pogosto določa vrednostna spremenljivka vrednost sestavljeno iz več podatkov. Razvijalec programa lahko definira svoje vrednostne tipe. Vgrajeni vrednostni tipi so: Char, Byte, Short, Integer, Long, Decimal, Single, Double, Boolean, Date, Color, Point, PointF,... .

Čeprav je vrednost najpomembnejši element vrednostnih spremenljivk, imajo vrednostne spremenljivke tudi predmetni značaj. Za predmete je značilno stanje, ki ga določajo lastnosti in spreminjajo metode (operacije). Tip Date ima izmed naštetih najbolj pestro predmetno naravo - veliko podatkov, ki določajo njegovo stanje in precej operacij, ki to stanje spreminjajo.

Podatkovni tipi lahko določajo spremenljivkam tudi podatke, ki so skupni vsem istega tipa. Te podatke imenujemo deljene (ang. Shared Values) oziroma razredne (ang. Class Values) podatke. Na primer vsaka spremenljivka tipa Single ima deljeno lastnost MaxValue (MinValue), ki določa največjo (najmanjšo) vrednost tipa Single.

Referenčne spremenljivke Referenčne ali sklicne spremenljivke so kazalci na predmete (objekte) v RAM pomnilniku. Za predmete je značilna življenska doba: nastanejo, obstajajo in preminejo. Njihovo stanje določajo podatki (lastnosti), njihovo spreminjanje pa metode (operacije).

Vrednost referenčne spremenljivke je kazalec - celoštevilski naslov 1. bajta podatkov objekta v glavnem pomnilniku, ki določajo tekoče stanje predmeta. Bajti so v glavnem pomnilniku oštevilčeni, naslovi so številke bajtov. Naslovi so pri 32-bitnem operacijskem sistemu dolgi 32 bitov (4 bajte). 32 bitov omejuje naslovni prostor bajtov na cca. 4 milijarde.

Referenčna spremenljivka torej predstavlja predmet (kaže na predmet) določenega referenčega tipa, ki je definiran kot razred s stavkom

```
Class ImeTipa
...
End Class
```

DRAFT

Vgrajenih referenčnih tipov je mnogo več kot vrednostnih. Razvijalec programa pa si lahko definira svoje referenčne tipe. Tehnika objektnega programiranja temelji na tej možnosti.

Tip String je tipičen predstavnik referenčnih tipov, ki se v programih zelo pogosto uporabljaja. Drugi zelo uporabni vgrajeni referenčni tipi so: Timer, Random, Font, Pen, Brush, PictureBox, Form, Process, Thread, File, StreamReader, StreamWriter, ...

4.4 Deklaracije spremenljivk

Stavek Dim V konzolnem programu deklariramo spremenljivke s stavkom Dim. Deklaracija spremenljivki določi ime, tip in prostor v pomnilniku. Vrednostnim spremenljivkam določi tudi prostor v glavnem pomnilniku in začetno vrednost (definicija spremenljivke).

```
Dim a, b, c As Single
```

Referenčni spremenljivki stavek Dim predmeta ob deklaraciji ne določi, razen če ne dodamo operatorja New. Pripravi prostor dolžine 4 bajte za kazalec na predmet (naslov predmeta). Predmet lahko referenčni spremenljivki določimo/spremenimo kasneje z operatorjem New.

```
Dim ura As Timer
Dim ura1 As New Timer
```

Beseda Dim pride od angl. besede Dimension, saj lahko s stavki Dim deklariramo tudi spremenljivke z indeksi, ki imajo podano dimenzijo - število elementov. Poleg stavka Dim obstajajo še deklaracijski stavki Public, Private, Friend, Protected, za katere tudi veljajo pravila obravnavana v tem odseku. Primer stavka Dim, ki določa polje 11-elementov.

```
Dim p(10) As Single
Primeri deklaracij vrednostnih spremenljivk

Dim širina, višina As Single
Dim a, b, c As Double, x1, x2 As Double, napaka As Boolean
Dim naslov As String
```

Splošno pravilo za deklaracije spremenljivk Besedi Dim sledi seznam imen ločenih z vejicami z dodano sintagmo As Tip, kjer je Tip eden izmed dovoljenih tipov. Pri vrednostnih tipih stavek Dim napove in hkrati definira spremenljivko. Definicija pomeni, da se ustvari prostor za vrednost spremenljivke (v RAMu) in njena začetna vrednost.

```
Dim ime1,ime2 ... As Tip, ime3 As Tip ...
```

Stavek Dim smemo napisati:

- Znotraj modula (običajno na vrhu) izven procedur. Spremenljivke veljajo (so dostopne proceduram) tedaj za vse procedure modula (glej Primer 1). Imenujemo jih tudi globalne spremenljivke.
- Kjerkoli znotraj procedure običajno na vrhu v zapisu procedure. Spremenljivke veljajo tedaj samo v proceduri.
- Ob vsakem klicu procedure notranje spremenljivke nastanejo na novo (glej primer 2), po končani izvedbi jih izvajalni sistem odstrani.

Primer 1

```
Module Module1
   Dim p1,p2 As Integer ' Globalne spremenljivke
   ...
   Sub Main()
       ...
   End Sub
   ...
End Module
```

Primer 2

```
Module Module1
   Sub Main()
       Dim x,y,z As Single ' Notranje spremenljivke.
       ...
   End Sub
   ....
End Module
```

4.5 Začetne vrednosti spremenljivk

Pravila glede začetnih vrednosti spremenljivk se razlikujejo glede na referenčne in vrednostne tipe. Začetne vrednosti so bodisi privzete bodisi so eksplicitno določene ob deklaraciji (stavek Dim).

4.5.1 Začetne vrednosti vrednostnih tipov

Začetno vrednost spremenljivke eksplicitno določimo ob njeni deklaraciji. Izraz izračuna vrednost, ki postane začetna vrednost spremenljivke. Ni dovoljeno z izrazom hkrati določiti vrednost več spremenljivk.

```
Dim ime As Tip = izraz
```

Primeri deklaracij z začetnimi vrednostmi vrednostnih tipov:

```
Dim širina As Single = 10, višina As Single = 10, ploščina As Single = širina * višina
Dim barva As Color = Color.Blue
```

DRAFT

Privzete začetne vrednosti

- Privzete začetne vrednosti spremenljivk numeričnih tipov Byte, Short, Integer, Long, Single, Double, Decimal so 0 (nič).
- Privzeta začetna vrednost spremenljivk tipa Boolean je False.
- Privzeta začetna vrednost tipa String je Nothing (vrednost ni definirana), ker je String referenčni tip.

Določanje začetnih vrednosti s konstruktorji Pri nekaterih vrednostnih tipih določimo začetne vrednosti tako, da pokličemo posebno konstruktorsko funkcijo, ki ji rečemo na kratko konstruktor. Ima isto ime kot je ime tipa. Pogosto ima parametre. Na primer:

```
Dim tA As New PointF(2.5, -1.5)     ' Klic konstruktorja
Dim tB As PointF = New PointF(2,-2) ' Daljša različica uporabe konstruktorja.
```

Narobe je:

```
Dim a, b, c As Single = 1
```

4.5.2 Začetne vrednosti referenčnih tipov

Privzete vrednosti referenčnih tipov Referenčne spremenljivke so kazalci na predmete. Privzeta začetna vrednost spremenljivke referenčnega tipa je Nothing (ničelni naslov, naslov z vrednostjo 0), kar pomeni, da spremenljivka ne predstavlja nobenega predmeta.

```
Dim ura As Timer, naslov As String
```

Določanje predmetov ob deklaraciji predmetnih spremenljivk Ekcplicitno določimo začetno vrednost ob deklaraciji referenčne spremenljivke tako, da hkrati ustvarimo z operatorjem New predmet, ki ga predstavlja.

```
Dim ime As New Predmet(p1,p2, ...)
```

Operator New izvede klic konstruktorske funkcije, ki ima isto ime kot tip (razred). Konstruktorski funkciji rečemo krajše konstruktor. Konstruktor sme imeti parametre (argumente). Konstruktorjev pri določenem tipu sme biti več, razlikovati se morajo po številu in tipih parametrov.

```
Dim ura As New Timer, pisava As New Font("Courier Now", 12)
Dim kocka As New Random ' Daljši način
```

Posebno pravilo velja za vgrajeni referenčni tip String, kjer smemo prirediti konstanten niz znakov.

```
Dim naslov As String = "Uvod v Visual Basic.NET"
```


DRAFT

Poglavje 5

Logični izrazi

Logični izrazi so tesno povezani z izjavami. Logične vrednosti so elementi, ki v kontrolnih stavkih določajo vrstni red izvajanja posameznih delov programa. Najprej so razloženi primerjalni operatorji, ki izračunajo logično vrednost. Sledi podrobna razlaga logičnih operatorjev And, Or in Not. Poglavje zaključi niz razloženih primerov.

5.1 Izjave

Izjava je v VB.NET logični izraz, ki izračuna logično vrednost True ali False. Izjava lahko vsebuje:

- logične konstante (True, False),
- logične spremenljivke tipa Boolean (njihova vrednost sme biti bodisi True bodisi False),
- logične operatorje (And, Or, AndAlso, OrElse, Not, Xor),
- primerjalne izraze (rezultat primerjalnega izraza je logična vrednost) in
- oklepaje.

Primeri izjav:

```
dežuje Or sneži Or naliva
x >= xMin And x <= xMax And y >= yMin And y <= yMax
napaka < eps Or n > nMax
```

Primer uporabe logičnih spremenljivk pri vnosu podatkov:

```
Dim p As Single, napaka As Boolean, pKrit As Single = 2500
...
Do ' Ponavljaj vnos, dokler ni podatek pravilen.
    Console.Write("p : ")
    p = Console.ReadLine
    napaka = False
```



```

    If p <= 0 Or p>pKrit Then
        napaka = True
        Console.WriteLine("Napaka, ponovi vnos!")
    End If
While napaka
...
```

5.2 Primerjalni operatorji

Primerjalni operatorji primerjajo dva numerična izraza.VB.NET pozna operatorje:

- > : večji (a > b),
- < : manjši (a < b),
- >= : večji ali enak (a >= b),
- <= : manjši ali enak (a <= b).

Primerjalni operatorji imajo običajen pomen, vrnejo logično vrednost True/False. So dvomestni operatorji. Prvi operand napišemo pred izrazom, drugega pa za njim. Oba operanda morata biti numeričnega tipa. Pogosto uporabljamo primerjalne operatorje v pogoju v stavku If ali v stavku Do While. Na primer:

```
If x>0 And y>=x Then ...

Do While n<0
    ...
Loop
```

Med primerjalne operatorje lahko štejemo tudi operatorje ugotavljanja enakosti in neenakosti.

- = je operator ugotavljanja enakosti (a=b).
- <> je operator ugotavljanja neenakosti (a<>b>.

= se enako enako zapiše kot prireditveni operator, vendar je razlikovanje razvidno iz kode. Na primer:

```
Dim enako As Boolean, a,b As Single
...
enako = a = b ' Prvi operator je prireditveni, drugi ugotavlja enakost
```

Drug primer uporabe pri vnosu podatkov:

```
Do
  vrsta = datoteka.ReadLine
  If vrsta = "" Then End
     ...
Loop While ocena <> 10
```

DRAFT

Operatorja smemo uporabiti pri vseh tipih. Pri vrednostnih tipih se primerajo vrednosti pri referenčnih tipih pa naslovi predmetov.

5.3 Logični operatorji

Logični operatorji računajo z izjavami - logičnimi izrazi, ki izračunajo logično vrednost (True/False). Logični operatorji so:

- And : logična konjunkcija, logični in,
- Or : logična disjunkcija, logični ali,
- Not : negacija, ne,
- Xor : ekskluzivna logična disjunkcija, izključitveni logični ali,
- AndAlso : izvedbena različica operatorja And,
- OrElse : izvedbena različica operatorja Else.

Operator And, AndAlso Logični operator And je dvomestni operator, ki vrne rezultat True, kadar sta oba operanda True, sicer pa False. Definira ga naslednja tabela, ki določa rezultat Op1 And Op2:

```
Op1/Op2        True        False
True         True        False
False         False        False
```

Točko x,y znotraj kroga s središčem v xc,yc in radijem r v 1. kvadrantu kartezičnega koord. sistema ugotovi izjava:

```
(x-xc)^2 + (y-yc^2) < r^2 And x > 0 And x > y
```

Logični operator AndAlso računa enako kot And, le da v primeru, ko rezultat določi že prvi operand, drugega operanda ne izračuna (če je 1.operand False, je razultat False). Tak operator imenujemo kratkostični operator.

Operator Or, OrElse Logični operator Or je dvomestni operator, ki vrne rezultat True, kadar je vsaj en operand True, sicer pa False. Definira ga naslednja tabela, ki določa rezultat Op1 Or Op2:

```
Op1/Op2        True        False
True         True        True
False         True        False
```

Logični operator OrElse je kratkostična različica operatorja Or. Če je prvi operand True, se drugi ne izračuna. Primer iteracije, ki se konča, če je število korakov preseženo ali je dosežena zahtevana natančnost:


```
Do
  n = n + 1
  xStari = xNovi
  xNovi = ...
  ...
Loop While n < nMax Or (xNovi-xStari) < eps
```

Operator Not Operator Not spremeni logično vrednost: iz True naredi False in obratno.

```
Izjava        Not Izjava
True        False
False        True
```

Operatorja And in Or dovoljujeta tudi logične operacije celoštevilskih operandov (izvedene po bitih), AndAlso in OrElse pa ne.

DRAFT

Poglavje 6

Aritmetika

Poglavje obravnava računanje s števili v programu. Razložen je koncept izraza. Podrobno so predstavljeni aritmetični operatorji in druga pravila za zapis in računanje izrazov. Pravila povezana s samodejnimi pretvorbanmi niso enostavna. Na koncu so opisane vgrajene matematične funkcije, ki smejo igrati vlogo členov v zrazih.

6.1 Aritmetični izrazi

Aritmetični izraz je izračunljiv zapis (formula), čigar rezultat je celo ali realno število. Rezultat je določenega celoštevilskega ali realnega tipa. Izraz sme vsebovati aritmetične operatorje, konstante, spremenljivke, oklepaje in klice funkcij, ki izračunajo numerično vrednost. Primeri zapisa aritmetičnih izrazov v VB.NET:

- 1
- a + b - 1
- -x
- (a + b) * (x^2 - 2)
- 2*Sin(omega*t - fi)

Aritmetični operatorji so:

- + (seštevanje),
- - (odštevanje),
- * (množenje),
- / (deljenje),
- ^ (potenciranje) in
- Mod (ostanek po modulu).

Veljajo običajne prioritete operatojev: operator potenciranja ^ ima najvišjo prednost, * in / imata prednost pred + in -. Privzetih operatorjev ni, zato je naslednji zapis neveljaven: (a + b)(a - b). Rezultat pri deljenju z operatorjem / je vedno realno število, tudi če sta oba operanda tipa Integer. Operator določa deljenje, pri čemer je rezultat operacije celo število. Dvomestni operator Mod izračuna ostanek pri deljenju. Definiran je za vse numerične tipe. Na primer: a Mod b izračuna 0.5, če je a=4.5 in b=2.

6.2 Mešana aritmetika

VB.NET zna računati s celimi in realnimi števili. Pri računanju dvomestnih aritmetičnih operacij x+y, x-y, x*y, x/y, x^y, x Mod y veljajo naslednja pravila, ki temeljijo na zahtevi, da se informacija ne sme izgubiti. Če je en operand realnega tipa Single ali Double, potem mora biti drug operand celoštevilski ali realen. Računanje poteka tedaj po naslednjem postopku.

1. Če je en operand celoštevilski, se le-ta najprej pretvori v realni tip drugega operanda.
2. Če je eden izmed operandov tipa Double se operacija izvede v obsegu in natančnosti tega tipa. Rezultat je tudi tipa Double. Sicer se izvede v obsegu in natančnosti tipa Single. Rezultat je tedaj tipa Single.
3. Če je operacija neveljavna, sistem ne sproži izjeme (neulovljena izjema pomeni zaustavitev programa zaradi usodne napake), ampak je rezultat operacije bodisi 0 (nič), neskončno ali NaN (Not A Number). Pomen teh vrednosti je sledeč:
   - Če je rezultat operacije premajhen (manjši od najmanjšega dovoljenega realnega števila), je rezultat operacije 0.
   - Če je rezultat operacije prevelik (večji od največjega dovoljenega števila, deljenje z 0), je rezultat operacije bodisi PositiveInfinity bodisi NegativeInfinity, kot določa predznak rezultata.
   - Če je rezultat operacije neveljaven, kadar kateri izmed operandov ni sprejemljivega tipa, je rezultat operacije NaN (Not A Number).
   - Če ima vsaj en od operandov vrednost NaN, je rezultat operacije NaN.

6.3 Matematične funkcije

Vgrajene matematične funkcije Matematične funkcije so deljene (razredne) metode razreda Math, ki se nahaja v imenskem prostoru System.

```
Math.Sqrt(x)
Math.Sin(fi)
```

Imports System.Math omogoča uporabo funkcij brez imena razreda Math.

```
Sqrt(x)
Sin(fi)
```

DRAFT

Pomembnejše matematične funkcije

- Trigonometrične funkcije: Sin(fi), Cos(fi), Tan(fi)
- Obratne trigonometrične funkcije: ASin(x), Atan(x), Atan2(y,x)
- Hiperbolične trig. funkcije: SinH(fi), CosH(fi), TanH(fi)
- Potenciranje: Pow(x,y),
- Kvadratni koren: Sqrt(x),
- Naravni in desetiški logaritem: Log(x), Log10(x)
- Eksponentna funkcija: Exp(x)
- Absolutna vrednost: Abs(x)
- Spodnja in zgornja meja: Floor(x), Ceiling(x)
- Manjše (večje) od dveh števil: Min(x1, x2), Max(x1, x2)
- Okrnitev: Truncate(x)
- Zaokroževanje: Round(x)

DRAFT
DRAFT

Poglavje 7

Kontrolni stavki

Poglavja obravnava pogojne stavke, ponavljalne stavke (zanke) in izbirne stavke.

7.1 Uvod v kontrolne stavke

Izvedljivi stavki v programu so zapisani v procedurah. Velja preprosto pravilo, da se stavki izvajajo po vrsti od zgoraj navzdol, tako kot so zapisani v proceduri. Razen, če s kontrolnimi stavki ne zahtevamo drugače.

Najbolj preprosti in razumljivi so pogojni stavki, ki omogočajo, da se del stavkov izvede ali pa ne, odvisno od določenega pogoja (izjave). Pogoj je izračunljiv logični izraz, ki vrne logično vrednost True ali False. Ponavljalni stavki omogočajo ponavljanje dela stavkov znotraj procedure. Nekateri ponavljalni stavki imajo števec korakov (stavek For) in drugi pa ne (stavek Do). Med kontrolne stavke sodijo tudi večvejni izbirni stavki, kjer je izbira odvisna od določene (izračunane) vrednosti (stavek Select). Kontrolni stavki so sestavljeni stavki. Vsebujejo stavke, pogosto tudi druge kontrolne stavke. Zamikanje gnezdenih kontrolnih stavkov povečuje preglednost programa. Razvojno okolje zagotavlja samodejno zamikanje gnezdenih kontrolnih stavkov (običajno je zamik 4 znake).

7.2 Pogojni stavki

Pogojni stavki omogočajo pogojno izvajanje niza stavkov programa. Bistveni element pogojnega stavka je pogojni izraz (pogoj, izjava).

Pogojni izraz Pogojni izraz je logični izraz, ki izračuna logično vrednost True ali False. Pogojni izraz lahko vsebuje:

- logične operatorje (And, Or, Not, AndAlso, OrElse, Xor),
- primerjalne izraze (>, >=, <, <=, =, <>),
- funkcije, ki izračunajo logično vrednost,
- logične spremenljivke (tipa Boolean),
- logični konstanti (True/False) in
- okrogle oklepaje.

Vrstični pogojni stavek

```
If pogoj Then stavek ' Stavek se izvede ob izpolnjenem pogoju.
...
```

Vrstični pogojni stavek zapišemo v eni vrstici. Če je pogoj izpolnjen, se stavek poleg besede Then izvrši, sicer pa ne. Dovoljena je tudi različica z Then ... Else ... .

```
If pogoj Then stavek1 Else stavek2
...
```

Če je pogoj izpolnjen, se izvrši stavek1, sicer pa stavek2.

Primer vrstičnih pogojnih stavkov

```
If a > aMax Then aMax = a
If start Then x = 0: y = 0: z = 0
If uspeh Then status = "Uspeh" Else status = "Usodna napaka"
```

Bločni pogojni stavek If ... Then ... EndIf

```
If pogoj Then
   stavki
End If
...
```

Bločni pogojni stavek lahko pogojno izvede niz stavkov. Če je pogoj izpolnjen, se stavki izvršijo, sicer pa ne. Na primer:

```
If ocena>5 And ocena <= 10 Then
  štPoz = štPoz + 1
  vsota = vsota + ocena
End If
...
```

Bločni pogojni stavek If ... Then ... Else ... End If

```
If pogoj Then
   stavki1
Else
   stavki2
End If
...
```

Če je pogoj izpolnjen, se izvršijo stavki1, sicer pa stavki2.

DRAFT

Primer stavka If ... Then ... End If

```
If diskriminanta >= 0 Then
   ' Realna korena
   x1 = (-b + Math.Sqrt(diskriminanta))/(2*a)
   x2 = (-b - Math.Sqrt(diskriminanta))/(2*a)
Else
   ' Kompleksna korena
   re = -b/(2*a)
   im = Math.Sqrt(-diskriminanta))/(2*a)
End If
```

Večvejna razvejitev pri stavku If Splošna sintaksa bločnega pogojnega stavka ima dodano še možnost Else If, kar omogoča nadaljnjo večvejno razvejitev v primeru, da pogoj ni izpolnjen. To je pregledno, če je dodatnih razvejitev malo, sicer je pametneje uporabiti izbirni stavek Select ... End Select.

```
If pogoj Then
   stavki1
Else If pogoj1 Then
   stavki2
Else If pogoj2 Then
   stavki2
   ...
Else
   stavki3
End If
```

7.3 Zanke

Zanke omogočajo ponavljanje izvedljivih stavkov. Ponavljalni stavki so: stavek For, stavek Do in stavek For Each.

Ponavljanje se ustavi bodisi ob izstopu iz zanke (Exit Do, Exit For) bodisi ob zaključku izvajanja procedure (Return) bodisi zaustavitvi programa. Stavek Continue preskoči preostale stavke in začne nov cikel. Izjemoma uporabimo za izstop iz zanke stavek GOTO.

Stavek For je značilen za programiranje spremenljivk z indeksi, ker ima kontrolno spremenljivko.

7.3.1 Neskončna zanka

Neskončna zanka omogoča ponavljanje niza stavkov. Zapišemo jo s stavkom Do ... Loop.

```
Do
   stavki
Loop
stavek1
```


Stavki se ponavljajo v nedogled, dokler bodisi ne ustavimo programa bodisi ne prekinemo izvajanje stavkov znotraj zanke z Exit Do ali stavkom Return.

Primer uporabe stavka Exit Do Stavek Exit Do prekine izvajanje zanke, izvajanje se nadaljuje s stavkom, ki sledi besedi Loop. Praviloma ga uporabimo v pogojnem stavku. Na primer:

```
Do
  stavki
  If pogoj Then Exit Do
  stavki
Loop
stavek1
```

Primer končanja neskončne zanke s stavkom Return Stavki se izvajajo, dokler ni izpolnjen pogoj; tedaj se izvajanje stavkov znotraj zanke prekine in nadaljuje na stavku stavek1, ki sledi besedi Loop. Primer uporabe stavka Do v kombinaciji s stavkom Return:

```
Sub IzračunX()
   Do
       x1 = x2
       x2 = ...
       If Math.Abs(x2 - x1) < eps Then Return
       ...
   Loop
End Sub
```

Primer neskončne zanke, ki jo prekine stavek End Sledi primer neskončne zanke. Program se konča znotraj procedure PreberiPrimer, ko zmanjka podatkov v vhodni datoteki:

```
Sub Main
   Do
       PreberiPrimer()
       IzračunajRezultate()
       Izpiši()
   Loop
End Sub
```

7.3.2 Do While/Until

Do While Do While pogoj ... Loop je zanka Do ... Loop s pogojem na začetku. Če je pogoj izpolnjen (True) se izvajanje nadaljuje, sicer se izvede stavek, ki sledi Loop.

Do Until Do Until pogoj ... Loop ima pogoj obrnjen. Če pogoj ni izpolnjen (False) se izvajanje nadaljuje, sicer se izvede stavek, ki sledi Loop. Pomembno se je zavedati, da se jedro zanke (stavki med Do in Loop) nikoli ne izvrši, če pogoj že na začetku ni izpolnjen!

DRAFT

Primer uporabe pri branju podatkov Na primer, iz datoteke, ki jo določa spremenljivka vhod (tipa StreamReader) beremo po vrsti podatke, dokler so (vhod.Peek()>=0). Metoda Peek() vrne -1, če znakov ni več, sicer pa kodo naslednjega še ne prebranega znaka:

```
...
Dim vhod As New StreamReader("c:\Podatki\Primer1.txt")
...
Do While vhod.Peek() >=0 ' Zanka se konča, ko zmanjka podatkov.
   vrsta = vhod.ReadLine()
   ...
Loop
...
```

Primer ponavljanja zanke, dokler ne pride do napake Pri naslednjem primeru se zanka ponavlja, dokler ni napake.

```
Dim napaka As Boolean
...
Operacija(q, napaka)
...
Do Until napaka ' Dokler ni napake izvajaj niz operacij.
   Operacija(p, napaka)
Loop
...
```

7.3.3 Do ... Loop While/Until

Do ... Loop While Do ... Loop While pogoj je zanka Do ... Loop s pogojem na koncu. Če je pogoj izpolnjen (True) se izvajanje nadaljuje, sicer se izvede stavek, ki sledi Loop.

Do ... Loop Until Do ... Loop Until pogoj ima pogoj obrnjen. Če pogoj ni izpolnjen (False) se izvajanje nadaljuje, sicer se izvede stavek, ki sledi Loop. Pomembno je vedeti, da se jedro zanke (stavki med Do in Loop) vedno vsaj enkrat izvrši!

Primer Na primer, pogovorni vnos podatka ponavljamo, dokler je podatek napačen.

```
Dim napaka As Boolean
...
Do
   Console.Write("Podaj a :")  ' Odzivnik za a
   a = Console.ReadLIne()

   napaka = false
   If a <=0 Or a>100 Then napaka = True
Loop While napaka
...
```


7.4 Stavek For

Stavek For je ponavljalni stavek s kontrolno spremenljivko, ki igra vlogo števca korakov. Števec sme biti poljubnega numeričnega tipa. Kontrolno spremenljivko smemo deklarirati v glavi stavka For. Korak je običajno pozitiven, njegova privzeta vrednost je 1. Dovoljen je tudi negativen korak, tedaj zavzame kontrolna spremenljivka po vrsti padajoče vrednosti. Stavek For pogosto uporabljamo v povezavi s spremenljivkami z indeksi. Izvajanje znotraj stavka For prekine stavek Exit For. Izvajanje se nadaljuje s stavkom, ki sledi Next.

7.4.1 Uvod v stavek For

Koncept stavka For Stavek For omogoča ponavljanje stavkov, pri katerem se uporablja posebna kontrolna spremenljivka, ki se spreminja od začetne vrednosti do končne vrednosti s predpisanim korakom. Privzeto je korak enak 1. Sintakso stavka For s kontrolno spremenljivko k predstavimo z:

```
For k = kZačetni To kKončni              ' Korak je privzeto 1.
   stavki
Next k

For k = kZačetni To kKončni Step korak ' Tu korak podamo
   stavki
Next k

```

kZačetni, kKončni, korak smejo biti poljubni numerični izrazi. Kontrolna spremenljivka je vedno numerična spremenljivka, bodisi celoštevilska (Integer, Long, ...) bodisi realna (Single, Double).

Primer stavka For Pri naslednjem primeru teče kontrolna spremenljivka od 1 do 10. Jedro zanke (niz-stavkov) se 10 x izvrši. Nato se izvajanje nadaljuje s stavkom, ki sledi besedi Next.

```
For i=1 To 10
  niz-stavkov
Next i
stavek1
```

Algoritem za stavek For For k = kZačetni To kKončni Step korak stavki Next k

Izvajanje stavka For za pozitiven korak določa naslednji algoritem.

k = kZačetni Ponavljaj Če je k > kKončni Potem Izstopi iz zanke Izvedi jedro zanke (niz stavkov) k = k + korak Do Sem Konec

Izvajanje stavka For za negativen korak določa naslednji algoritem.

k = kZačetni Ponavljaj Če je k < kKončni Potem Izstopi iz zanke Izvedi jedro zanke (niz stavkov) k = k + korak Do Sem Konec

DRAFT

7.4.2 Korak pri stavku For

Privzeto je korak pri stavku For enak 1. Na primer, v spodnjem primeru zavzame kontrolna spremenljivka k vrednosti: 1, 2, ..., 10.

```
For k As Integer = 1 To 10
...
Next k
```

Če korak ni enak 1, dodamo besedo Step in izraz, ki določa korak. Na primer:

```
For j=-3 To 3 Step 2
  ...
Next j

```

Kontrolna spremenljivka zavzame vrednosti: -3, -1, 1 in 3.

Dovoljen je tudi negativen korak, tedaj velja zahteva, da mora biti začetna vrednost večja ali enaka končni, sicer se jedro stavka For nikoli ne izvrši. Na primer:

```
For j=n To 1 Step -1
  ...
Next j
```

Kontrolna spremenljivka zavzame vrednosti: n, n-1, n-2, ... 2, 1.

7.5 Stavek For Each

Stavek For Each omogoča izvajanje niza stavkov za vsek element v podani zbirki elementov. Prednost stavka For Each je razumljiv in enostaven zapis zanke. Zbirka (kolekcija) je lahko vsak referenčni tip, ki izpolnjuje določene pogoje. Primeri takih razredov so: polja tipa Array (spremenljivke z indeksi), razredi ArrayList, Dictionary, StringArray, ... in parametrični razredi List, ... Definiramo lahko svoj kolekcijski razred, pri katerem smemo uporabiti stavek For Each. Sintaksa stavka For Each je naslednja:

```
For Each element In Zbirka
  ....
Next element  ' element smemo tu spustiti
```

Element je bodisi referenčnega bodisi vrednostnega tipa odvisno od narave elementov v zbirki.

Primer obravnave vseh ocen zapisanih v polju ocene, začenši z elementom ocene(0). Izračunati je potrebno povprečno oceno:


```
Dim ocena, povpOcena, ocene() vsota As Integer
...
n = Console.ReadLine ' Število ocen
ReDim ocene(n-1)
...
For Each ocena In ocene
   vsota = vsota + ocena
Next el

povpOcena = vsota / ocene.Length
```

7.6 Izbirni stavek Select

Izbirni stavek Select omogoča izbiranje izmed več podanih možnosti. Zapis je pregleden in enostaven. Sintakso stavka Select pojasnjuje naslednja predloga.

```
Select ind
   Case 1
       stavki1
   Case 2
       stavki2
   Case 3,4,5
       stavki345
   CaseElse
       drugiStavki
End Select
stavki
```

Če je ind 1, se izvedejo stavki1, če je ind 2,se izvedejo stavki2, če je 3 ali 4 ali 5, se izvedejo stavki345, sicer pa drugiStavki. Izvajanje se potem pri vsakem primeru nadaljuje s stavki.

Primer uporabe stavka za vsak dan v tednu.

```
Dim dan As String
...
Select dan
  Case "Pon", "Tor", "Sre", "Čet", "Pet"
      DelovniDan
  Case "Sob"
      ŠportneAktivnosti
  Case "Ned"
      Počitek
End Select
```

7.7 Gnezdenje kontrolnih stavkov

Kontrolni stavki pogosto vsebujejo druge kontrolne stavke. Pravimo, da so gnezdeni. Gnezdeni kontrolni stavki so vedno zamaknjeni. Za zamik poskrbi

DRAFT

razvojno okolje (Visual Studio/VB 2005 Express). Velikost zamika lahko nastavimo (Options), privzeto je 4 presledke.

Sledeči program prebere kvadratno matriko reda n ter vektor reda n in nato izračuna produkt matrike z vektorjem (y = Ax) in ga prikaže v konzolnem oknu:

- Zaporedoma reši več primerov, kar je realizirano z zanko Do ... Loop, ki vsebuje skoraj vse stavke programa.
- Ko za podatek n pritisnemo tipko Enter, se izvajanje programa ustavi.
- Program vsebuje stavek For, ki vsebuje drug stavek For.

```
Sub Main
   Dim n, i, j As Integer, vrsta As String
   Dim a(,), x(), y() As Single

   Do
       ' Vnos matrike reda n.
       ' ====================
       Console.Write("Podaj n: ")
       vrsta = Console.ReadLine
       If vrsta = "" Then End     ' Ustavi program.

       n = vrsta ' Red matrike in vektorja.

       ReDim a(n,n), x(n), y(n)  ' Določi velikosti polj a, x, in y.

       For i=1 To n               ' Vnos elementov matrike a po vrsticah.
           For j=1 To n
               Console.Write("Podaj a(" & i & "," & j & "): ")
               a(i,j) = Console.ReadLine
           Next j
       Next i

       Console.WriteLine()

       ' Vnos vektorja x.
       ' ================
       For i=1 To n
           Console.Write("Podaj x(" & i & "): ")
           x(i) = Console.ReadLine
       Next i

       ' Izračun y = Ax.
       ' ===============
       For i=1 To n
           y(i) = 0

           For j=1 To n
               y(i) = y(i) + a(i,j)*x(j)
           Next j
       Next i

       ' Prikaz rezultatov.
```



```
       ' ==================
       Console.Write("y = Ax : ")

       For i=1 To n
           Console.Write( y(i) & " " )
       Next i

       Console.WriteLine()
       Console.WriteLine()
   Loop
End Sub
```

7.8 Primeri

7.8.1 Izračun n!

n! oz. n fakulteta izračunamo z n*(n-1)*(n-2), ... 2, 1.

```
Sub Main
   Dim n, i As Integer, fakulteta as Decimal

   Do
       Console.Write("Podaj n: ")
       n = Console.ReadLine
       If n=0 Then End

       fakulteta = 1

       For i=1 To n
          fakulteta = fakultata * i
       Next i

       Console.WriteLine(n & "! = " & fakulteta)
   Loop
End Sub
```

7.8.2 Primer: Ocene

Izmed podanih ocen med 1 in 10 določi število pozitivnih (>5) in negativnih (<=5). Število ocen je omejeno na 100. Podatke o ocenah določi pogovorno. Podajanje ocen končamo s tipko Enter.

```
Sub Main
   Dim ocene(100), štPozitivnih, štNegativnih, štOcen, i As Integer
   Dim vrsta As String

   Do
       Console.Write("Podaj oceno: ")
       vrsta = Console.ReadLine

```

DRAFT

```
        If vrsta="" Then Exit Do

        štOcen = štOcen + 1
        ocene(štOcen) = vrsta
    Loop

    štPozitivnih = 0

    For i=1 To štOcen
        If ocene(i) > 5 Then štPozitivnih = štPozitivnih + 1
    Next i

    štNegativnih = štOcen - štPozitivnih

    Console.WriteLine("Število pozitivnih ocen je: " & štPozitivnih )
    Console.WriteLine("Število negativnih ocen je: " & štNegativnih )
End Sub
```

7.8.3 Ugotavljanje praštevil

Program pogovorno prebere celo število in nato ugotovi ali je praštevilo ali ni. Algoritem je enostaven in vendar neučinkovit.

```
Sub Main
   Dim število, d As Integer
   Dim izjava As Boolean

   Console.Write(" Podaj število: ")
   število = Console.ReadLine

   d = 2
   izjava = " je praštevilo"

   Do While d<število
       If število Mod d = 0 Then
           izjava = " ni praštevilo"
           Exit Do
       End If
       d = d + 1
   Loop

   Console.WriteLine("Število " & število & izjava)
   Console.ReadLine()
 End Sub
```


DRAFT

Poglavje 8

Spremenljivke z indeksi

Spremenljivke z indeksi so zelo pomemben koncept računalniških programov. Razložena so pravila povezana z deklaracijami spremenljivk z indeksi in njihovo uporabo v izrazih. Dinamična polja so ustrezno obravnavana. Sledi poglavje o programiranju vektorjev in matrik. Poglavje se zaključi z nizom primerov.

8.1 Uvod v polja

Spremenljivke z indeksi predstavljajo niz podatkov, ki imajo skupno ime in se med seboj ločijo po indeksu. Večinoma se uporabljajo spremenljivke z enim indeksom, sme pa jih biti več. Matrike uporabljajo dva indeksa, tenzorji pa še več.

Spremenljivke z indeksi imenujemo tudi polja, tabele, nizi. V računalniških programih so nepogrešljivi elementi. Primeri spremenljivk z indeksi v inženirskih nalogah so:

- Element matrike: ai,j
- Vektor: x1, x2, x3
- Koeficienti polinoma: c0, c1,c2

Spremenljivke z indeksi v programu predstavljajo spremenljivke z indeksi v nalogah.

8.2 Deklaracije polj

Spremenljivke z indeksi (polja) deklariramo z običajnimi deklaracijskimi stavki (Dim, Private, Public, Friend, ...). Poleg Dim napišemo ime polja, nato najvišji dovoljeni indeks v okroglih oklepajih, sledi As in ime tipa, ki določa vrsto elementov polja. Najnižji indeks je vedno 0 (nič).

```
Dim p(5) As Single
```

Zgornji stavek torej definira polje z naslednjimi elementi: p(0), p(1), p(2), p(3), p(4), in p(5). Število elementov polja je vedno za 1 večje od najvišjega indeksa.

8.3 Uporaba polj

Elemente polj (spremenljivke z indeksi) uporabljamo kot navadne spremenljivke. Indekse pišemov okrogli oklepajih. Indeks sme biti poljuben numeričen izraz, ki se izračuna, pretvori v celo število, ki potem določa indeks. V primeru neveljavnega indeksa se sproži izjema, ki v primeru, da se ne ujame, provzroči prekinitev izvajanja programa. Nikako pa ni mogoče zaradi neveljavnega indeksa povoziti kakšne druge vrednosti v glavnem pomnilniku. Priprost primer uporabe spremenljivk z indeksi je izračun skalarnega produkta dveh vektorjev:

```
Dim x(3), y(3), skalarniProdukt As Single
...
skalarniProdukt = x(1)*y(1) + x(2)*y(2) + x(3)*y(3)
```

Skalarni produkt lahko izračunamo tudi z:

```
skalarniProdukt = 0
For i=1 To 3
   skalarniProdukt += x(i)*y(i)
Next i
```

8.4 Začetne vrednosti

Privzete začetne vrednosti elementov numeričnih polj so ničle. Na primer:

```
Dim x(100) As Single, c(3) As Integer
```

x(0),... x(100) imajo začetno vrednost 0, tako kot tudi c(0), c(1),c(2) in c(3).

Eksplicitno določimo začetne vrednosti tako, da jih naštejemo v zavitih oklepajih. Najvišjega indeksa tedaj v deklaraciji polja ne smemo predpisati, saj se določi samodejno. Na primer:

```
Dim praštevila() As Single = {1, 2, 3, 5, 7, 11, 13}  ' Najvišji indeks je 6.
```

Dneve v tednu (najvišji indeks je 6) predstavimo z:

```
Dim dnevi() As String = { "Ponedeljek", "Torek", "Sreda", _
                          "Četrtek", "Petek", "Sobota", "Nedelja" }
```

8.5 Dinamična polja

Dinamična polja se od običajnih razlikujejo po tem, da se velikost polja spreminja med izvajanjem programa. Spremembo najvišjega dovoljenega indeksa polja določa izvedljivi stavek ReDim. Vse druge lastnosti polja ostajajo nespremenjene. ReDim torej določi tekoči najvišji indeks polja. Lahko ga poveča ali zmanjša. Tipičen primer uporabe dinamičnih polj je naslednji:

DRAFT

```
Dim a() As Single   ' Polje a nima nobenega elementa, kot predmet tipa Array sploh ne obstaja.
Dim n As Integer    ' Najvišji indeks polja.

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

ReDim a(n)          ' Določi novo velikost polja.

For i=0 To n
   Console.Write("Podaj a" & "(" & i & "): ")
   a(i) = Console.ReadLine
Next i
...
```

8.6 ReDim Preserve

V nekaterih primerih velikost polja v programu diha: lahko se večkrat poveča in spet zmanjša.

- Ob vsaki spremembi s stavkom ReDim se dosedanje polja vrednosti izgubijo. ReDim vedno ustvari novo polje, staro pa pobriše.
- Če poleg besede ReDim dodamo besedo Preserve, se dosedanje vrednosti elementov ohranijo. Če se polje zmanjša, se ohranijo samo obstoječe.

Na primer, 10 krat zapored polju p povečamo število elementov za 10, njihove vrednosti pa sproti določamo tako, da mečemo kocko s 6 pikami. Želimo, da se te vrednosti ohranijo. p(0) ne upoštevamo.

```
Dim kocka As New Random
Dim p(-1) As Integer, i, j, k As Integer

For i=1 To 10
   ReDim Preserve p(p.Length + 10)

   For j=1 To 10
       k = k + 1
       p(k) = kocka.Next(1,7) ' Met kocke s 6 pikami
   Next j
Next i

For i=1 To 100
  Console.Write(p(i) & " ")
Next i

Console.ReadLine
```

8.7 Vektorji in matrike

Matrike se v tehniki pogosto uporabljajo. Predstavljajo podatke urejene v dvorazsežne tabele z dvema indeksoma. Prvi indeks šteje vrstice, drugi indeks pa

šteje stolpce.

```
Dim a(10,10) As Single, x(10) As Single
```

Elementi dvodimenzionalnega polja a so a(0,0), a(0,1), ... a(0,10), a(1,0), ... a(1,10), ... a(10,0), ... a(10,10). Število elementov je torej 11x11 = 121.

8.8 Primeri

8.8.1 Skalarni produkt vektorjev

```
Dim x(3), y(3), skalarniProdukt As Single, i As Integer
...
skalarniProdukt = 0

For i=1 To 3
   skalarniProdukt = skalarniProdukt + x(i)*y(i)
Next i
...
```

8.8.2 Produkt matrike z vektorjem

```
' Funkcija AkratX izračuna produkt matrike "A" z vektorjem "b".
' Red matrike "A" mora biti enak reu vektorja "b".
Function AkratX(ByVal A(,) As Single, ByVal b() As Single) As Single()
   Dim n As Integer = b.Length - 1
   Dim i,j As Integer, x(n) As Single

   For i=1 To n
       For j=1 To n
           x(i) = x(i) + a(i,j)*b(j)
       Next j
   Next i

   Return x
End Function
```

8.8.3 Vektorski produkt

```
Dim u(3), v(3), vektorskiProdukt(3) As Single, i As Integer
...

vektorskiProdukt = New Single() { 0, u(2)*v(3)-u(3)*v(2), u(3)*v(1)-u(1)*v(3), u(1)*v(2)-u(2)*v
...
```

DRAFT

Poglavje 9

Procedure in funkcije

Procedura je imenovan element programa, ki vsebuje izvedljive stavke. Procedura reši določeno nalogo programa. Program razdeljen na procedure je lahko pregleden in razumljiv. Izvajanje procedure se sproži s klicem procedure po imenu. Procedure imajo lahko argumente: vhodne in/ali izhodne. Vhodni argumenti predstavljajo podatke procedure, izhodni pa rezultate. Procedure so sestavni deli modulov (pri konzolnih programih) in sestavni deli razredov (pri objektnem programiranju). Procedure v razredih imenujemo metode.

Procedure so dveh vrst; procedure Sub in funkcijske procedure. Pri objektnem programiranju obstajajo posebne vrste procedur, ki določajo lastnosti objektov (ang. Property procedures).

Namen procedur je dvojen: urejenost (preglednost) programa in ponovna uporaba. Procedure lahko zmanjšajo kompleksnost programa. Procedure smemo bodisi večkrat uporabiti v istem programu bodisi ponovno uporabiti v drugih programih.

Procedure v modulih (konzolni programi) podatkov in postopkov ne obravnavajo enakovredno; podatki so upoštevani slabše kot postopki. Proceduralno programiranje zato pogosto ni uspešno pri kompleksnih programih (možnost skritih napak, nezanesljiva ponovna uporaba).

9.1 Koncept procedure

V tem poglavju je razložen koncept procedure. Obravnavana je uporabnost procedur. Dani napotki za ustrezna imena procedur. Predstavljene so značilnosti procedur Sub in procedur Function (funkcije).

9.1.1 Uvod v procedure

Programi in procedure

- Računalniški programi rešujejo naloge (probleme).
- Problemi so običajno kompleksni, zato nalogo razdelimo na več manjših nalog.
- Algoritem je zapis, ki po korakih določa, kako lahko rešimo dano nalogo.

- Procedure so elementi računalniških programov, ki predstavljajo postopke - algoritme.
- Procedure praviloma vsebujejo izvedljive stavke.
- Procedure so zato bistveni elementi vsakega programa.

Koncept procedure

- Procedura je del programa, ki ima ime in vsebuje izvedljive stavke.
- Procedura zato nekaj naredi. Izvede postopek, ki reši del naloge programa. Reši podnalogo programa.
- Proceduro izvedemo tako, da jo pokličemo po imenu.
- Ob klicu procedure se izvajanje programa nadaljuje z izvedljivimi stavki znotraj procedure.
- Po končani izvedbi procedure, se izvede stavek, ki sledi klicu procedure.
- Procedura sme imeti tudi vhodne in izhodne argumente (parametre), ki igrajo vlogo podatkov in rezultatov procedure.

Procedure so sestavni deli modulov (proceduralno programiranje) in razredov (objektno programiranje). Procedure so dveh vrst: procedure Sub in funkcijske procedure (procedure Function), ki jim rečemo tudi funkcije.

9.1.2 Namen procedure

Procedure uporabljamo v programih, ker omogočajo pregledne programe in ponovno uporabo.

- Procedure omogočajo razdelitev programa na manjše, obvladljive dele.
- Program razdeljen na procedure je lahko pregleden in razumljiv.
- S procedurami zmanjšujemo kompleksnost programa.
- Proceduro lahko tudi večkrat uporabimo bodisi v istem bodisi v drugem programu.
- Pogosto uporabimo ali izdelamo knjižnico procedur za reševanje problemov na določenem področju.

Dobre procedure

- Dobra procedura reši samo eno dobro definirano nalogo.
- Povezave med procedurami morajo biti čim šibkejše. Povezave so učinki ene procedure na delovanje drugih procedur.
- Drugače povedano, procedura mora biti čimbolj samostojna in čim manj odvisna od drugih procedur.
- Možnosti za skrite napake v programu lahko z uporabo dobrih procedur zelo zmanjšamo.

DRAFT

Modeliranje in abstraktni nivoji

- Modeliranje na splošno pomeni uporabo modela naloge pri njenem reševanju. Odstranimo (abstrahiramo) vse, kar ne spada k reševanju naloge. Osredotočimo se samo na elemente povezane z reševanjem.
- Procedure omogočajo realizacijo abstraktnih nivojev naloge.

Vsak abstraktni nivo zapišemo v programu z ustreznimi procedurami:

- Program pogosto razvijamo od zgoraj navzdol.
- Na vrhu odstranimo vse podrobnosti (detajle).
- Vsak naslednji nivo se ukvarja z več podrobnostmi.

Primer uporabe procedur na dveh abstraktnih nivojih znotraj modula

```
Module Module1
    Sub Main()
        Preberi()
        Izračunaj()
        Izpiši()
    End Sub

    Sub Preberi() ' Podroben zapis branja podatkov.
        ...
    End Sub

    Sub Izračunaj()
       ...
    End Sub

    Sub Izpiši()
       ...
    End Sub
End Module
```

9.1.3 Imena

Imena so pomembna - upoštevaj naslednje nasvete:

- Ime mora spomniti, kaj procedura ali funkcija naredi, čemu je namenjena.
- Ime funkcije je lahko samostalnik, ki pove, kaj je rezultat funkcije.
- Procedura Sub naj v imenu vsebuje glagol, ki pove, kaj procedura naredi.
- Procedure, ki se redko uporabljajo, ali opravljajo manj običajne naloge, imajo lahko daljša bolj točna imena.
- Imena argumentov procedur naj spominjajo na podatke, ki jih argumenti določajo.
- Dobra imena so temelj samodokumentiranih programov.

Primeri dobrih imen

```
Sub IzračunajKrog(ByVal r As Single, ByRef ob As Single, ByRef pl As Single)
Sub IzpišiSporočiloONapaki(ByVal sporočilo As String)
Function DolžinaVektorja(ByBal v As Vektor) As Single
```

Primeri slabih imen

```
Sub Z3()
Sub Reši(ByVal z2 As String, ByRef u5 As String)
Function Ocena(ByVal p1 As Integer, ByVal p2 As Integer, ByVal r() As Single) As Single()
```

9.1.4 Procedure

Procedure Sub predstavljajo rešitev dane naloge (problema). Vsebujejo izvedljive stavke.

- Procedure Sub imajo obvezno ime.
- Lahko imajo argumente, ki predstavljajo podatke in rezultate procedure.
- Procedure v modulih imajo lahko dostop do globalnih podatkov modula.
- Procedure v razredih imajo dostop do podatkov razreda.
- Definicija procedure Sub vsebuje glavo in telo.

Primer definicije procedure

```
Sub IzračunajKrog( ByVal r As Single, ByRef ob As Single, ByRef pl As Single )
    ob = 2 * Math.PI * r
    pl = Math.Pi * r ^2
End Sub
```

Izvajanje procedure

- Izvajanje stavkov procedure sproži klic procedure, ki je samostojen stavek.
- Proceduro Sub pokličemo tako, da napišemo ime procedure in v okroglih oklepajih navedemo dejanske argumente, ki jih ločimo z vejicami.
- Če procedura nima argumentov napišemo poleg imena samo predklepaj in zaklepaj: ().

```
IzračunajKrog(radij, obseg, ploščina)
IzračunajRezultate()        ' Klic procedure Sub brez argumentov.
NarišiSliko(širina, višina) ' Klic procedure Sub z argumenti.
```

DRAFT

9.1.5 Funkcija

Funkcija je programska enota, ki reši nalogo z enim samim rezultatom.

- Funkcije vedno izračunajo in vrnejo en rezultat.
- Funkcija ima obvezno ime. Po imenu funkcijo pokličemo.
- Funkcija sme imeti argumente, praviloma samo vhodne, izjemoma pa tudi izhodne.
- Funkcija ima glavo in telo.
- Glava funkcije določa ime funkcije, argumente in tip rezultata.
- Telo funkcije vsebuje izvedljive stavke.
- Funkcija se mora končati z End Function.

Primer funkcije

```
' Funkcija "Večji" vrne večje izmed števil p in q.
Function Večji(ByVal p As Single, ByVal q As Single) As Single ' Glava
    If p > q Then
        Return p
    Else
        Return q
    End If
End Function
```

Izvajanje funkcij

- Izvajanje funkcije sproži klic funkcije.
- Klic funkcije praviloma ni samostojen stavek.
- Klic funkcije je element v izrazu.
- Izraz je zapis, ki po izvedbi izračuna določeno vrednost.
- Klic vsebuje ime funkcije in dejanske argumente v okroglih oklepajih med seboj ločene z vejico.
- Če funkcija nima argumentov, napišemo samo ime funkcije (brez okroglih oklepajev).
- Funkcijo smemo poklicati tudi s samostojnim stavkom (tako kot proceduro Sub). Rezultata funkcije tedaj ne uporabimo nikjer.

```
y = 2*F(x)/G(x) ' Primer klicev funkcij F in G v izrazu.
F(x)            ' Klic funkcije s samostojnim stavkom.
```


9.2 Argumenti

Poglavje podrobno obravnava argumente procedur Sub in funkcij, ki so temelj procedur, saj eksplicitno določajo podatke in rezultate naloge, ki jo rešuje procedura. Argumenti so težji del programiranja procedur, saj je pravil precej in nekatera so precej zapletena. Razloženi so formalni in klicni argumenti procedur. Obravnavana sta prenosa argumentov po vrednosti in referenci. Na koncu so predstavljena pravila za primere, ko so argumenti procedur polja (spremenljivke z indeksi). Opcijski (neobvezni) argumenti so posebnost jezika VB.NET. Med težje elemente spada variabilno število argumentov, ki je opisano na koncu.

9.2.1 Uvod v argumente

Argumenti procedur so temelj procedur, saj eksplicitno določajo podatke in rezultate procedur. Procedure, ki komunicirajo z ostalimi deli programa potom argumentov veljajo za najbolj varne glede uporabe, saj predstavljajo zaključeno celoto - modul.

```
Sub Ime (ByVal a As Tip, ByVal b As Tip, ... )
    ...
End Sub
```

- Argumenti ali parametri procedur so podatki in rezultati procedur, ki so definirani v glavi procedure.
- Imenujemo jih tudi formalni argumenti oz. formalni parametri.
- Ob klicu procedure se morajo klicni argumenti ujemati z istoležnimi formalnimi argumenti po tipu in številu.

Splošna pravila za zapis argumentov

- Število argumentov ni omejeno. Procedura je pogosto brez argumentov.
- Formalne argumente zapišemo v obliki linearnega seznama, ki sledi imenu procedure. Argumente med seboj ločimo z vejicami.
- Vrstni red argumentov je pomemben.
- Formalni argument vsebuje naslednje dele: določilo prenosa ByVal ali ByRef, ime argumenta in tip argumenta.
- ByVal določa prenos po vrednosti (ang. By Value), ByRef določa prenos po referenci (ang. By Reference)
- Argument mora imeti enolično ime, da ga lahko uporabimo v izvedljivih stavkih procedure.
- Vsak argument predstavlja podatek določenega tipa. Sme biti referenčni ali vrednostni.
- Tip argumenta moramo eksplicitno zapisati s sintagmo: As ImeTipa.

DRAFT

Primer procedure s parametri vrste ByVal in ByRef

```
Sub ImeProceredure( ByVal p1 As Integer, ByVal p2 As Single, ...,
                    ByRef r1 As Date, ByRef r2 As Boolean, ... )
    ....
End Sub
```

p1, p2, ... so podatki procedure, r1, r2,... so rezultati procedure. Vhodni in izhodni argumenti smejo biti pomešani.

Primeri glav procedur

```
Sub RešiSistemLinEnačbPoGausu( ByVal a(,) As Single, ByVal b() As Single, _
                               ByVal x() As Single, ByRef uspeh As Integer )

Function SkalarniProdukt( ByVal u() As Single, ByVal v() As Single ) As Single

Function VektorskiProdukt( ByVal u() As Single, ByVal v() As Single ) As Single()
```

9.2.2 Prenos argumentov

Prenos argumentov določa določilo tipa prenosa v glavi procedure. Sme biti bodisi ByVal bodisi ByRef.

Pravila za prenos argumentov vrste ByVal

- Prenosni atribut ByVal določa, da se ob klicu procedure samodejno izdela kopija vrednosti argumenta, ki se potem uporabi v izvedljivih stavkih.
- Po končani izvedbi procedure se kopija argumenta samodejno odstrani.
- Sprememba vrednosti kopije z izvedljivim stavkom znotraj procedure ne spremeni vrednosti argumenta.
- To je varno pri vrednostnih tipih. Tudi zaradi skritih napak v izvedbi procedure vrednosti klicnega argumeta ni mogoče spremeniti (pokvariti).
- Izdelava kopije pri argumentih, katerih vrednost obsega veliko bajtov, je lahko prostorsko in procesno potratna.
- Če je argument referenčnega tipa, se izdela kopija naslova oz. sklicevanja (dolžine 4 bajte), kar omogoča spremembo vrednosti argumenta.
- Uporaba referenčnih argumentov torej ni varna glede možnih napačnih sprememb.

Pravila za prenos argumentov vrste ByRef

- Prenosni atribut ByRef določa neposredno uporabo argumenta v izvedljivih stavkih procedure.
- Izvedljivi stavki smejo vrednost argumenta vrednost spremeniti.

- Prenos po referenci ByRef je zato primeren za izhodne argumente vrednostnih tipov.
- Določilo ByRef moramo uporabiti tudi pri referenčnih tipih, če pripadajoči predmet referenčnega tipa nastane znotraj procedure.
- Določilo ByRef uporabimo tudi pri vhodnih argumetih vrednostnega tipa, če želimo učinkovit prenos (brez izdelave delovne kopije argumenta).Smisleno je to samo v primeru, če vrednost obsega veliko bajtov.
- Argumenti z določilom ByRef lahko pa igrajo vlogo vhodnih in izhodnih argumentov.

9.2.3 Klicni argumenti

Klicni ali dejanski argumenti so argumenti v klicu procedure Sub oz. Function. Zanje veljajo posebna pravila.

```
Sub IzračunKroga( ByVal r As Single, ByRef ob As Single, ByRef pl As Single )
    ...
End Sub
```

Pravila za klicne (dejanske) argumente

```
Dim a, b, obseg, ploščina As Single
...

' Klic procedure.
IzračunKroga(2*(a + b)/2, obseg, ploščina)
...
```

- Ob klicu procedure se mora število klicnih (dejanskih) in formalnih argumentov ujemati.
- Ujemati se morajo tipi enakoležnih argumentov.
- Vrednost klicnega argumenta sme biti izraz, ki izračuna vrednost tipa, ki je določen v formalnem argumentu.

Samodejne pretvorbe pri klicanju procedur

- Če velja Option Strict On, samodejne pretvorbe klicnih argumentov niso dovoljene.
- Samodejne pretvorbe so dovoljene, če velja nastavitev Option Strict Off.
- Na primer, če je formalni argument tipa Integer in dejanski argument tipa Single, se samodejno pretvori v celo število.

V naslednjem primeru je vrednost izraza, ki predstavlja 1. dejanski argument tipa Integer, delovna kopija pa je tipa Single. Vrednost $(a + b)/2$ se samodejno pretvori v tip Single.

DRAFT

```
Dim a, b As Integer
Dim obseg, ploščina As Single
...
IzračunKroga((a + b)/2, obseg, ploščina)
...
```

9.2.4 Argument sprem. z indeksi

Polje (ang. Array) je referenčni podatkovni tip, ki določa niz podatkov iste vrste. Elemente polj programiramo s spremenljivkami z indeksi. Na primer:

```
Dim p(5) As Single
```

Polje p določa 6 podatkov tipa Single: p(0), p(1), p(2), p(3), p(4) in p(5).

Pravila za argumente - spremenljivke z indeksi Če je argument procedure tipa polje (spremenljivka z indeksi, Array), veljajo pravila:

- Poleg imena argumenta moramo napisati: () As Tip. Tip določa tip elementov polja.
- Če ima polje dva indeksa med oklepaja dodamo vejico: (,) As Tip.
- Določilo pred imenom je praviloma ByVal, saj ime predstavlja naslov polja, ki se pri kopiranju nič ne spremeni (kopija naslova je isti naslov).
- Določilo ByRef je potrebno, če se polje ustvari znotraj procedure (na primer, če znotraj procedure uporabimo stavek ReDim).

Primer Funkcijo, ki izračuna in vrne največjo vrednost niza števil podanih s poljem p, zapišemo z:

```
Function NajvečjeŠtevilo( ByVal p() As Single) As Single
    NajvečjeŠtevilo = Single.MinValue

    For i As Integer=0 To p.Length-1
       If p(i) > NajvečjeŠtevilo Then NajvečjeŠtevilo = p(i)
    Next i
End Function
```

Primer klica zgornje funkcije predstavlja sledeča koda, ki v neskončni zanki najprej ustvari slučajno celo število n med 1 in 10. Nato ustvari niz n realnih slučajnih števil na intervalu (-1,1) ter izračuna in prikaže največje izmed njih. Po prikazu se program ustavi in počaka, da uporabnik pritisne tipko Enter. Uporabljen je vgrajeni razred Random, ki je sposoben generirati psevdoslučajna cela (metoda Next) in realna števila (metoda NextDouble).


```
Sub Main()
    Dim gsš As New Random ' Generator psevdoslučajnih števil.
    Dim i, n As Integer, naj, števila() As Single, niz As String

    Do
        n = gsš.Next(1, 11) ' Ustvari celo število med 1 in 10.
        niz = ""

        ReDim števila(n - 1)

        For i = 0 To n - 1
            meritve(i) = 2 * gsš.NextDouble - 1 ' Ustvari št. na int. (-1,1)
            niz &= meritve(i).ToString("F2") & " "
        Next i

        naj = NajvečjeŠtevilo(števila) ' Klic funkcije.

        Console.WriteLine("V nizu " & niz & "je največje št. " & naj.ToString("F2"))
        Console.ReadLine()
    Loop
End Sub
```

9.3 Procedure

Poglavje podrobno obravnava procedure Sub. Vsebuje razlago pravil za zapis procedure in pravila za klicanje procedure. Razloženi so koncepti podatkov in rezultatov procedure. Uporaba procedur je ilustrirana z nizom primerov.

9.3.1 Definicija

Proceduro Sub definiramo v konzolnem programu tako, da jo zapišemo znotraj modula. Po vrsti naslednje dele:

- Neobvezen komentar, ki razlaga uporabo procedure.
- Glava procedure, ki definira ime in formalne argumente (parametre) procedure.
- Telo procedure, ki vsebuje izvedljive stavke.
- Konec zapisa procedure določa End Sub.

Telo procedure

- Telo procedure sledi glavi procedure.
- Telo procedure vsebuje izvedljive stavke.
- Telo procedure ima lahko tudi krajevne spremenljivke, ki jih napovemo s stavkom Dim. Krajevne spremenljivke veljajo samo znotraj procedure.
- Poseben samostojen stavek Return zaključi izvajanje procedure. Izvajanje programa se nadaljuje s stavkom, ki sledi klicu procedure.

DRAFT

- Stavkov Return sme biti več. Pogosto nastopajo v pogojnih stavkih.
- Če v telesu procedure ni nobenega stavka Return, se izvajanje procedure konča po izvedbi zadnjega stavka.

Primer

```
Module Module1
    Sub Main()
        ...
        PrazneVrste(5) ' Izpiše 5 praznih vrst.
        Console.WriteLine( ... )
        ...
    End Sub

    ' Procedura izpiše "nv" praznih vrstic v konzolno okno.
    Sub PrazneVrste( ByVal nv As Integer)
        Dim i As Integer ' Krajevna spremenljivka

        For i = 1 To nv           ' Telo vsebuje izvedljive stavke.
            Console.WriteLine()
        Next i
    End Sub
End Module
```

9.3.2 Podatki

Proceduri Sub lahko določimo podatke na naslednje načine:

1. Preko vhodnih parametrov (argumentov). To je najboljši način in tudi najbolj varen. Slaba stran tega načina je veliko dela s pisanjem argumentov in neučinkovitost pri prenosu argumentov.
2. Preko globalnih spremenljivk v modulih. Ta način je zelo primeren za konzolne programe. Procedura pa žal ni več samostojna enota, saj je odvisna od globalnih podatkov, ki so definirani na drugem mestu programa.
3. Pri procedurah v razredih (objektno programiranje) imajo procedure neposreden dostop do podatkov razreda. Take procedure so najboljše.
4. Znotraj procedure lahko pokličemo drugo proceduro, funkcijo ali objektno komponento, ki lahko proceduri določi podatke.Tipičen primer je uporaba vhodnih datotek znotraj procedure.
5. Procedura lahko komunicira z napravami povezanimi z računalnikom ali komunicira z drugimi računalniki po internetu.

9.3.3 Rezultati

Procedura Sub lahko drugim delom programa posreduje rezultate na različne načine.

- Preko izhodnih parametrov. To je najboljši način in tudi najbolj varen. Je samodokumentiran. Slaba stran tega načina je veliko dela s pisanjem argumentov in neučinkovitost pri prenosu argumentov.
- Preko globalnih spremenljivk v modulih. Ta način je hiter in primeren za konzolne programe. Procedura ni več samostojna enota, saj je odvisna od globalnih podatkov, ki so definirani na drugem mestu programa.
- Pri procedurah v razredih (objektno programiranje) imajo procedure neposreden dostop do podatkov razreda. Take procedure so najboljše.
- Znotraj procedure lahko procedura pokliče drugo proceduro, kateri posreduje rezultati preko njenih vhodnih parametrov. Naprimer, izpis v izhodno datoteko, krmiljenje z računalnikom povezane naprave, pošiljanje sporočil po internetu. Tudi take procedure niso več samostojni moduli.

9.4 Funkcije

Poglavje podrobno obravnava funkcije. Funkcije so procedure, ki vrnejo en sam rezultat. Praviloma imajo samo vhodne argumente. Klic funkcije je element izraza. Poglavje vsebuje podrobno razlago pravil za zapis funkcije in pravila za klicanje funkcije. Rezultat funkcije sme biti tudi polje (spremenljivke z indeksi). Uporabnost funkcij je predstavljena z nizom primerov.

9.5 Definicija funkcije

Primer funkcije

```
' Funkcija izračuna ploščino kvadrata s stranico "a".
Function PloščinaKvadrata( ByVal As Single a ) As Single
   Return a^2
End Function
```

Definicija Funkcija je posebne vrsta procedura, ki izračuna en sam rezultat. Definicija funkcije vsebuje po vrsti naslednje dele:

- Neobvezen komentar, ki razlaga uporabo funkcije. Poleg namena funkcije je primerno podrobno razložiti pomen argumentov funkcije. Tudi kakršni koli stranski učinki, ki jih povzroči izvedba funkcije morajo biti tu razloženi. Podrobnosti izvedbe funkcije pa sodijo v načrtno dokumentacijo programa.
- Glava funkcije, ki definira ime, formalne argumente funkcije in tip rezultata funkcije.
- Argumenti in rezultat funkcije smejo biti vrednostnega ali referenčnega tipa.
- Če je argument tipa polje (spremenljivke z indeksi), velja posebna sintaksa. Podobno pravilo velja tudi za tip rezultata funkcije.

DRAFT

- Telo funkcije vsebuje izvedljive stavke. Konča se z End Function.
- Pravila povezana z argumenti so ista kot pri proceduri Sub in so podrobno razložena tam.

Rezultat funkcije

- Rezultat funkciji določi stavek Return, ki obenem tudi zaključi izvajanje funkcije.
- Besedi Return obvezno sledi izraz.. Njegova vrednost se vrne kot rezultat funkcije.
- Stavkov Return znotraj funkcije sme biti več.
- Rezultat funkcije smemo prirediti tudi imenu funkcije, kar je uporabno pri funkcijah brez stavkov Return. Izvajanje take funkcije se konča z zadnjim stavkom funkcije. Ime funkcije igra vlogo spremenljivke tipa, ki je določen s tipom rezultata v glavi funkcije in določa rezultat funkcije.

Predloga za definicijo funkcije s stavki Return

```
Function Ime(By Val p1 As Tip1, By Val p2 As Tip2, ... ) As TipRezultata
    Dim d1 As Tip3, ... ' Deklaracije delovnih spremenljivk
    ...
    If pogoj Then Return izraz1
    ...
    Return izraz ' Ta izraz izračuna vrednost funkcije.
End Function
```

Predloga za definicijo funkcije, kjer rezultat priredimo imenu funkcije

```
Function Ime(By Val p1 As Tip1, By Val p2 As Tip2, ... ) As TipRezultata
    Dim d1 As Tip3, ... ' Deklaracije delovnih spremenljivk
    ...
    Ime = izraz ' Ime funkcije sprejme rezultat funkcije.
End Function
```

9.6 Primer funkcije

Izračun dolžine daljice Funkcija DolžinaDaljice izračuna in vrne dolžino daljice podano z začetno (x1,y1) in končno točko (x2,y2) v 2D kartezičnem koord. sistemu. Funkcije z eno samo vrstico so priljubljene med razvijalci - enovrstičnice.

```
' Funkcija izračuna in vrne razdaljo med točkama (x1,y1) in (x2,y2).
Function DolžinaDaljice( ByVal x1 As Single, ByVal y1 As Single, _
                         ByVal x2 As Single, ByVal y2 As Single) As Single
    Return Math.Sqrt((x2- x1)^2 + (y2 - y1)^2)
End Function
```


Izračun dolžine daljice s tipom PointF Za programiranje točk, obstaja že vgrajen tip PointF. Zato je koristna tudi naslednja različica:

```
' Funkcija izračuna in vrne razdaljo med točkama (x1,y1) in (x2,y2).
Function DolžinaDaljice( ByVal t1 As PointF, ByVal t2 As PointF) As Single
    Return Math.Sqrt((t2.x- t1.x)^2 + (t2.y2 - t1.y)^2)
End Function
```

DRAFT

Poglavje 10

Programiranje datotek

Poglavje obravnava programiranje datotek. Podrobno sta razložena razreda StreamReader in StreamWriter, ki omogočata čitanje in pisanje tekstovnih datotek v programih. Razložene so tudi podrobnosti glede uporabe različnih sistemov za kodiranje znakov.

10.1 Koncept datoteke

Datoteka je enota za trajni zapis podatkov v računalniku običajno na disk, CD ali v USB-ključ. Vsebina datoteke je v splošnem niz bitov. Datoteke delimo glede na tip zapisa na

- besedilne ali tekstovne (To je niz tekst ...) in
- dvojiške ali binarne datoteke (10110101 01110101 10110101 ...).

Tekstovne datoteke, ki vsebujejo niz znakov, lahko ljudje preberejo, binarne, ki vsebujejo z biti kodirane podatke, pa ne. To je predvsem pomembno pri izdelavi programov. Če razvijalec dobro pozna vsebino datoteke, bo hitreje sestavil pravilen program, ki bo tako datoteko uporabljal. Zato uporabljamo binarne datoteke v novih programih samo izjemoma, kadar so posebne zahteve glede učinkovitosti. Binarne datoteke napolnjene s števili računalniki obravnavajo hitreje kot tekstovne z enako vsebino.

Standard XML predpisuje zapise podatkov v obliki besedila - v tekstovne datoeke.

10.1.1 Vrste datotek

Datoteke so več vrst. Tekstovne so zapisane z nizi znakov, binarne pa z nizi bitov.

Dvojiške datoteke Dvojiška ali binarna datoteka vsebuje zapis podatkov v binarni obliki. Zapis numeričnih podatkov v binarni obliki je enak zapisu v RAM-pomnilniku. Procesiranje numeričnih podatkov v dvojiških datotekah je zato učinkovitejše od tekstovnih, ker pretvorba števil zapisanih z znaki v binarno število ni potrebna. Vsebina binarne datoteke je ljudem praktično nerazumljiva.

Tekstovne datoteke Vsebina tekstovne ali besedilne datoteke je niz znakov. Tudi numerični podatki so v tekstovni datoteki zapisani kot niz znakov v desetiškem sistemu. Podatke tekstovne datoteke lahko procesira računalnik in človek. Obravnavanje podatkov v besedilnih datotekah omogoča večjo varnost, saj lahko ljudje veliko lažje kontroliramo tekstovne podatke kot podatke v binarni obliki.. Tudi pri izdelava novih programov je hitrejša, če so podatki v obliki besedila. Podatkovni standard XML predpisuje zapis podatkov v obliki besedil. Ker pa se za zapis besedil uporabljajo različni kodirni sistemi, jih moramo poznati pri programiranju tekstovnih datotek.

10.1.2 StreamReader in StreamWriter

Razredi za programiranje tekstovnih datotek Za delo z datotekami in mapami lahko uporabljamo več specializiranih razredov (programskih komponent), ki podpirajo novejše standardne sisteme za kodiranje znakov: ANSI-kodne tabele, UTF-8, UTF-16, ...

Razreda za branje in pisanje besedilnih datotek sta: StreamReader, StreamWriter. Nahajata se v imenskem prostoru System.IO knjižnice razredov ogrodja Framework.NET. V določenih primerih potrebujemo tudi razred Encoding, ki skrbi za kodirne sisteme znakov (System.Text).

Vemo, da je datotečni sistem računalnika drevesasta struktura datotek in map. Vsebina map smejo biti datoteke in mape. Znana so pravila za imenovanje datotek in map. Za urejanje datotek in map so pripravljeni zmogljivi razredi File, FileInfo, Directory ter DirectoryInfo.

StreamReader Razred StreamReader določa branje podatkov predstavljenih s tokom (nizom) znakov, ki je značilen za tekstovne datoteke. Pred branjem je potrebno datoteko odpreti (pripraviti za branje), po končani operaciji pa moramo datoteko zapreti (sprostiti). Odprta datoteka namreč drugim programom ni dostopna.

Obstaja konstruktor razreda StreamReader, ki ustvari predmet tipa StreamReader in nato datoteko tudi odpre. S posebnim parametrom (encoding) lahko določimo tudi kodirni sistem, s katerim je bila izdelana datoteka.

Programiramo lahko čitanje bodisi posameznih znakov bodisi vrstic znakov bodisi celotne vsebine v enem kosu. Pravila za čitanje podatkov so enaka kot pri komponenti Console pri konzolnih programih.

Z metodo Close (nima parametrov) datoteko na koncu čitanja zapremo.

StreamWriter Z razredom StreamWriter lahko programiramo bodisi pisanje podatkov v nove tekstovne datoteke bodisi dodajanje v obstoječe. Konstruktor razreda StreamWriter najprej ustvari predmet tipa StreamWriter in nato novo prazno datoteko tudi ustvari. Določimo lahko tudi kodirni sistem izpisanih znakov.

StreamWriter ima operacije za zapis podatkov (spremenljivk, konstant) katerega koli osnovnega tipa (Integer, Single, ...). Vrednosti lahko tudi oblikujemo (na primer, izpis realnih števil na določeno število decimalk, ...).

Z metodo Close datoteko na koncu pisanja zapremo.

DRAFT

10.2 Delovna mapa

Lastnosti delovne mape Tekoča ali delovna mapa je mapa, ki je izhodišče relativnega sklicevanja na datoteke . Relativno sklicavanje se začne z imenom bodisi datoteke bodisi mape, ne določa pa naprave (področja), kjer se nahaja datoteka (C:\ime1\ime2 ...). na primer:

- podatki.txt,
- primeri\primer1.txt

Delovna mapa datoteke z izvedljivim programom. Privzeta delovna mapa progama je mapa, v kateri se nahaja datoteka z izvedljivim programom (program.exe). Delovno mapo lahko posebej določimo v bližnici na izvedljiv program.

Delovna mapa lahko zelo poenostavi programiranje (imenovanje) datotek v programu. Namesto da se na datoteke sklicujemo z absolutnimi imeni (naprava:pot.imedatoteke), določimo delovno mapo in se potem sklicujemo na datoteke relativno glede na delovno mapo. Delovno mapo lahko znotraj programa določimo in/ali spremenimo (jo programiramo)

Če poženemo program s klikom na doatoteko s končnico exe, je torej začetna delovna mapa enaka mapi, v kateri se nahaja datoteka exe. Znotraj programa smejo biti izvedljivi stavki, ki spremenijo delovno mapo.

Relativno sklicevanje na datoteke glede na delovno mapo Relativno sklicevanje na datoteko je niz imen map (podmap) ločenih z znakom "\", Na koncu je "\"in ime datoteke. Samo ime datoteke pomeni datoteko v delovni mapi. Če je datoteka v podmapi, zapišemo podmapa\imeDatoteke ali .\podmapa\imeDatoteke. Pika na začetku pomeni delovno mapo. Sklicevati se smemo tudi (z dvema pikama) na starševsko mapo delovne mape, na primer: "..\imeDatoteke". .. smemo uporabiti rekurzivno, na primer: "..\..\..\imeDatoteke".

Programiranje delovne mape Po imenu tekoče delovne mape (podatek tipa String) poizvemo z:

```
Dim delovnaMapa As String = IO.Directory.GetCurrentDirectory()
```

Delovno mapo spremenimo z:

```
IO.Directory.SetCurrentDirectory(imeMape)
```

10.3 Čitanje datotek

Izbrano datoteko povežemo z objektom tipa StreamReader, ki ga ustvarimo z operatorjem New, ki mu sledi beseda StreamReader (tedaj se pokliče konstruktor tipa StreamReader). Datoteka se odpre in pripravi za branje. Drugi programi je tedaj ne morejo uporabljati.

Za čitanje podatkov veljajo enaka pravila kot pri čitanju podatkov z razredom Console: pogosto uporabimo metodo ReadLine, ki prebere naslednjo vrstico. Datoteko moramo na koncu obvezno zapreti z metodo Close, sicer datoteka ni dostopna drugim programom.

```
Dim vhod As New StreamReader(imeDatoteke) ' Pokliče konstruktor razreda StreamReader.
Dim vrsta, b() As String

vrsta = vhod.ReadLine() ' Prebere znake naslednje vrstice.
b = vrsta.Split()
p1 = b(0): p2 = b(1)
...
vhod.Close() ' Vhodni tok zapre z metodo Close.
...
```

Prepoznavanje prebranih podatkov Vnos in izpis podatkov v platformi .NET nista simetrična. Pri izpisu podatkov smemo uporabiti formatna določila, pri vnosu podatkov pa ne. Prvi vtis je, da je sistem narejen na pol. Vendar temu ni tako. Čitanje (vnos) podatkov poteka konceptualno drugače kot izpis rezultatov. Po vrsti

1. najprej preberemo naslednjo neprebrano vrstico kot niz znakov,
2. nato jo razčlenimo na manjše dele,
3. ki jih potem pretvorimo v podatke določenih tipov.

Prebrano vrstico (tipa String) razdelimo na manjše dele tako, da uporabimo funkcijo (metodo) Split tipa (razreda) String.

- Split vrne polje elementov tipa String, ki vsebuje razcepljene dele v obliki besedil.
- Če pri klicu metode Split ne podamo nobenega parametra, se upošteva, da je ločilo med deli natanko 1 presledek.
- Ločilo smemo eksplicitno določiti, na primer ločilo vejica, b = vrsta.Split(",").
- Podamo lahko tudi več enakopravnih ločil, na primer vejico in presledek z: b = vrsta.Split(New Char() ",",).

Na primer, zapis sestavljen iz celega števila in dveh realnih vrednosti med seboj ločenih s po enim presledkom preberemo z:

```
Dim ind As Integer, x,y As Single
...

Dim b() As String = vhod.ReadLine.Split

ind = b(0)   ' Pretvorba v celo število
x = b(1)     ' Pretvorba v realno število
y = b(2)
```

DRAFT

Primer uporabe gradnika StreamReader Iz datoteke beremo podatke o 2D točkah. Koordinati x y posameznih točk so zapisane po vrsticah druga za drugo. Prvi podatek na začetku določa število točk.

```
Dim vhod As New StreamReader("podatki.txt")
Dim nTočk, i As Integer, x(), y() As Single, b() As String
...
nTočk = vhod.ReadLine()  ' Število točk: niz znakov pretvori v Integer
ReDim x(nTočk), y(nTočk) ' Določimo ustrezno velikost polja.

For i=1 To nTočk
    b = vhod.ReadLine().Split() ' Vrstico razcepimo na besedi.
    x(i) = b(0)                 ' Besedo pretvori v Single.
    y(i) = b(1)
Next i
...
```

Odstranjevanje večkratnih presledkov Če je med podatki več kot 1 presledek, je večkratne presledke najprej zamenjamo z enim samim. Zapišemo takole:

```
Dim vrsta As String = vhod.ReadLine

Do
   Dim nz As Integer = vrsta.Length
   vrsta = vrsta.Replace("  ", " ")
While vrsta.Length <> nz

Dim b() As String = vrsta.Split
...
```

Kako zvemo, če obstajajo podatki, ki jih še nismo prebrali? Uporabimo funkcijsko metodo Peek(), ki nima parametov. Vrne celoštevilsko kodo (kodno točko Unicode) naslednjega še neprebranega znaka, oziroma -1, če znakov ni več.

Na primer, podatki programa so ukazi zapisani vsak v svoji vrstici. Na začetku ukaza je ključna beseda, ki določa ukaz. Ukazni besedi sledijo v isti vrsti parametri v obliki niza števil in/ali imen. Program naj bere ukaze po vrsti, jih obdela in na koncu naj se ustavi. Za vsak ukaz naj se pokliče ustrezna procedura s prebranimi parametri.

```
Sub InterpretirajUkaze(vhod As StreamReader)
   Dim vrsta, b() As String

   Do While vhod.Peek() >= 0          ' Ali so še podatki ?
       vrsta = Console.ReadLine       ' Preberi naslednjo vrsto.
       b = vrsta.Split                ' Razcep na besede.

       Select Case b(0)               ' Ukaz določa 1. beseda.
           Case "Daljica"
```



```
               PreberiDaljico(b)
           Case "Trikotnik"
               PreberiTrikotnik(b)
           ...
           Case Else
               SporočiNapako("Neveljaven ukaz: " & b(0), vrsta)
               Return
       End Select
   Loop
End Sub
```

10.4 Pisanje datotek

V izbrano datoteko pišemo z objektom tipa StreamWriter, ki ga ustvarimo s konstruktorjem razreda StreamWriter.

```
Dim izhod As New StreamWriter(imeDatoteke)
```

Datoteko smemo podati z absolutno potjo ali relativno potjo glede na delovno mapo.

- Metoda Write(podatek) izpiše vrednost osnovnega tipa (Integer, Single, Double, String, Boolean,...) v privzeti obliki.
- Metoda WriteLine(podatek) izpiše vrednost osnovnega tipa v privzeti obliki in nato izvede preskok v novo vrsto.
- Metoda Write(oblikovniNiz, p1, p2,...) izpiše niz podatkov p1, p2, ... osnovnega tipa v obliki, ki jo določa oblikovni niz.
- Metoda WriteLine(oblikovniNiz, p1, p2,...) izpiše niz podatkov p1, p2, ... osnovnega tipa v obliki, ki jo določa oblikovni niz in nato izvede preskok v novo vrsto.

Spodnji primer v datoteko na enoti d izpiše n točk podanih s koordinatama x,y. Predstavljen je primer, kjer predmet razreda StreamWriter ustvarimo tik pred uporabo:

```
Dim izhod As StreamWriter, i,n As Integer, x(),y() As Single
...
izhod = New StreamWriter("d:\primeri\primer.txt")

For i=1 To n
    izhod.WriteLine("{0} {1}", x(i),y(i))
Next i

izhod.Close() ' Izhodni tok zapretemo z metodo Close.
```

DRAFT

Primer uporabe gradnika StreamWriter Niz 3D-točk izpišemo v datoteko rezultati.txt z:

```
Dim izhod As New StreamWriter("rezultati.txt")

For i=1 To nTočk
   izhod.WriteLine("{0,12:F3}{1,12:F3}{1,12:F3}", x(i), y(i), z(i))
Next i
...
izhod.Close()
```

Realne vrednosti koordinat so izpisane desno poravnane na 3 decimalke vsaka v svojem polju širine 12 znakov.

Dodajanje rezultatov v obstoječo datoteko Za dodajanje v obstoječo datoteko moramo uporabiti konstruktor razreda StreamReader z dodatnim parametrom::

```
StreamWriter(ime As String, dodaj As Boolean)
```

Argument dodaj lahko zavzame True ali False. True pomeni dodajanje v obstoječo, False pa izpis v novo datoteko. Na primer, v obstoječo datoteko rezultate dopisujemo z:

```
Dim out As New StreamWriter("rez.txt", True)
...
out.WriteLine(... )
```

10.5 Kodirni sistemi znakov

Gradnika StreamReader in StreamWriter privzeto upoštevata kodirni format UTF-8 sistema Unicode. UTF-8 je različica zapisa kodnih točk Unicode primerna (učinkovita) za zahodne civilizacije (ZDA, Evropa). Uporabljamo lahko vse znake sistema Unicode. Nekateri znaki se zapišejo v 1 bajt (ASCII znaki), drugi v dva, nekateri celo v tri bajte.

Za določanje kodirnih sistemov skrbi razred Encoding iz imenskega prostora System.Text. Pogosto velja v Sloveniji v operacijskem sistemu MS Windows privzeti kodirni sistem z oznako windows-1250 (kodna stran št. 1250). Za eksplicitno podajanje kodirnega sistema pri obravnavi besedilnih datotek moramo uporabiti konstruktorje razredov StreamReader in StreamWriter z dodatnim podatkom za format kodiranja znakov v datoteki:

```
StreamReader(ime As String, e As Encoding)
StreamWriter(ime As String, dodaj As Boolean, e As Encoding)
```

Podatek za kodirni format preprosto določimo z razredno metodo GetEncoding razreda Encoding. Na primer, kodirni sistem windows-1250 določimo z:


```
Dim vhod As StreamReader = New StreamWriter("pod.txt", _
   Encoding.GetEncoding("windows-1250"))

Dim izhod As StreamWriter = New StreamWriter("rez.txt", False, _
   Encoding.GetEncoding("windows-1250"))  ' Pišemo v novo datoteko.
```

10.6 Oblikovanje rezultatov

Uvod Več podatkov smemo izpisati skupaj z metodo WriteLine (Write) s pomočjo oblikovnega niza, ki je vedno 1. parameter pri WriteLine (Write). Zanj veljajo naslednja pravila:

- Oblikovni niz je besedilo, ki vsebuje elemente v zavitih oklepajih {...}, ki določajo mesto in obliko izpisanih podatkov (ang. placeholder, format).
- Zavitemu predklepaju { mora obvezno slediti celo število (kazalec), ki se sklicuje na podatek v seznamu. ki sledi kontrolnemu nizu. Podatke štejemo od 0 dalje (na prvi podatek se sklicujemo z 0, ...).
- Nato neobvezno sledi vejica in podatek o število izpisanih znakov.
- Nato spet neobvezno sledi podpičje in dodatna informacija o oblikovanju, sicer veljajo privzeti parametri oblikovanja, ki so odvisni od tipa izpisanega podatka.
- Med elemente {...} smemo napisati poljubno dodatno besedilo, ki se dobesedno izpiše kot rezultat.

Sledeči primer izpiše podatke o nizu točk, kjer sta sta vrednosti koordinat označeni (x=, y=) in dodatno ločeni z enim presledkom.

```
Dim out As StreamWriter, i,n As Integer, x(),y() As Single
...
out = New StreamWriter("a:\primeri\primer.txt")

For i=1 To n
    out.WritLine("x={0} y={1}", x(i), y(i))
Next i

out.Close() ' Izhodni tok moramo zapreti z metodo Close.
```

Splošna pravila za oblikovanje Za oblikovno določilo ... velja naslednje splošno pravilo, oglati oklepaji pomenijo neobvezne dele.

```
{IndP[,Poravnava][:FormatPodatka]}
```

- IndP je kazalec na podatek v seznam izpisanih vrednosti (0, 1, ...).

DRAFT

- Poravnava je celo število, ki določa število izpisanih znakov podatka. Pozitivno število pomeni poravnavo na levi rob, negativno število pa poravnavo na desni rob. Če je dolžina podatka merjena z znaki večja od podatka Poravnava, se poravnava ne upošteva.
- FormatPodatka določa podrobnosti povezane s tipom izpisanega podatka.

Oblikovanje besedil Besedila oblikujemo zelo preprosto. Lahko določimo samo poravnavo. Na primer:

```
WriteLine("{0,-12}{1,-12}", ime, priimek) ' Izpis s 12 znaki levo poravnano.
```

Oblikovanje celih števil Cela števila oblikujemo z določilom D. Na primer, D4 pomeni, da se celo število izpiše z vsaj 4 števkami, če je vrednost manjša se spredaj dodajo ničle.

Oblikovanje realnih števil Za oblikovanje realnih števil so na razpolago določila F(f), E(e) ali G(g), kot pri metodi ToString razreda String. Za decimalno ločilo veljajo tekoče nastavitve operacijskega sistema. Izpisana vrednost je vedno zaokrožena. Ostale podrobnosti so naslednje:

- F pomeni privzeto normalno obliko realnega števila na 2 decimalki.
- F3 oz. f3 pomeni izpis v normalni obliki na 3 decimalke.
- E pomeni privzeto eksponentno obliko z eno števko pred decimalko in 2 števkama za decimalko, nato sledi ekponent izpisan s 3 števkami. Na primer: 2,36E-003.
- E4 pomeni izpis v ekponentni obliki z eno števko pred decimalko in 4 števkami za decimalko, nato sledi eksponentni del. Na primer: -1,1234E-010.
- G oz. g pomeni generaliziran format. Zelo majhna in zelo velika števila se izpišejo v eksponentni obliki, sicer se uporabi normalna oblika.

Primeri:

- 0,10:F določa izpis na 2 decimalki, število je izpisano desno poravnano z 10 znaki.
- 3,10:F3 določa izpis 4. podatka na 10 znakov desno poravnano na 3 decimalke v normalni obliki.
- 0,-12:E6 določa izpis 1. podatka na 12 znakov levo poravnano s 6 ciframi koeficienta v eksponentni obliki.
- 1:F2 določa izpis 2. podatka brez poravnave na 2 decimalki v normalni obliki.

DRAFT

Poglavje 11

Okensko programiranje

Izdelava okenskih programov pomeni izdelavo samostojnih okenskih programov, ki jih namestimo na delovni postaji in nam koristijo pri delu, če so učinkoviti in imajo prijazen uporabniški vmesnik.

Pri okenskem programu uporabnik kontrolira program. Z izbiranjem ponujenih ukazov (meniji, orodne vrstice, kontektni meniji) sproži dogodek, na katerega se odzove program. Tehniko zato imenujemo dogodkovno programiranje. Program je stalno pripravljen, čaka, da se zgodi kak dogodek.

Visual Studio 2005 (Visual Basic 2005 Express) je odlično razvojno okolje za razvoj okenskih programov.

Kontrole in vizualno programiranje Učinkovitost izdelave okenskih programov temelji na vgrajenih gradnikih. Vidne gradnike znotraj okna imenujemo kontrole (kontrolniki). V glavno okno jih lahko dodamo z miško, ko jih izberemo v orodjarni razpoložljivih kontrol (ang. Toolbox). Tehniko imenujemo vizualno programiranje, ker sestavljamo program vizualno brez pisanja stavkov jezika VB.NET. Kontrole igrajo vlogo predmetov v programu. Njihove lastnosti lahko urejamo v posebnem oknu za lastnosti (Properties Window).

Del kode programa se ustvari samodejno, ko vizualno manipuliramo kontrolne gradnike. To je poleg dogodkovnega probramiranja bistvena značilnost izdelave okenskih programov. Generirana koda je vidna razvijalcu, ki jo lahko dodatno ureja (izboljša), vendar mora vedeti, kaj počne. Začetniki naj ne bi popravljali generirane kode.

Predmetno programiranje pri okenskih programih Okensko programiranje spada med predmetno (objektno) programiranje. Ni modulov in konzolnega okna. Glavno okno je predmet v programu tipa, ki ga določi razvijalec programa. Za vsak tip mora napisati strukturo Class ... End Class. Zato se običajno uporablja namesto besede tip beseda razred. Predmeti so določenega razreda. Privzeto ime razreda, ki predstavlja glavno okno programa je Form1. Pri predmetnem programiranju pogosto iz obstoječih razredov izpeljemo nove. Koncept se imenuje dedovanje razredov.

Za izdelavo ne preveč kompleksnih okenskih programov ni potrebno dosti vedeti o predmetnem programiranju. Zadošča razlaga na visokem nivoju in nekaj vzorčnih primerov. Na začetku kode, za vrstico Class Form1 deklariramo

podatke naloge (s stavki Dim ali Private), podobno kot globalne podatke v modulih pri konzolnih programih. Sledijo procedure, ki s vsebovanimi izvedljivimi stavki rešujejo nalogo. Podatki na vrhu razreda so dostopni vsem proceduram razreda. Procedure razreda imenujemo metode. Smejo biti tudi funkcije in lastnosti (Properties).

Koda v okenskem programu Z vizualnim programiranjem določimo zunanji izgled okna, gradnike (kontrole) ter njihovo postavitev v oknu. Za vse ostalo je potrebno napisati stavke v jeziku VB.NET.

Obstajata torej dva pogleda na program: načrtni pogled (ang. Design View) in kodni pogled (ang. Code View). Razvijalec lahko hitro preklaplja med obema načinoma.

Dogodkovno programiranje - izvedba Odzivanje na dogodke se v kodi izvede z dogodkovnimi procedurami, ki imajo posebno sintakso (na koncu je dodatek Handles ImeDogodka). V razvojnem okolju so pripravljeni generatorji odzivnih procedur. Najbolj pogoste odzivne procedure kontrol se ustvarijo z dvoklikom na kontrolo v pogledu načrta programa. Različnih dogodkov je lahko pri eni sami kontroli preko 30. Pripravljeni so v posebnem meniju na desni strani nad kodo v kodnem pogledu programa.

Izdelava okenskih programov vseeno ni enostavna: kontrol je veliko, imajo ogromno pripravljenih možnosti pa tudi razpoložljivih dogodkov. Dobro je vedeti, kje so dostopni nasveti za njihovo tipično uporabo.

11.1 Uvod

V tem poglavju je predstavljen okenski projekt. Opisane so lastnosti okenskih programov in sestavni deli okenskih projektov. Predstavljena je tudi funkcionalnost razvojnega okolja pri razvoju okenskih programov.

Okenski projekt Postopek začetka izdelave novega okenskega programa:

1. Poženi Visual Studio.NET (VB.NET 2005 Express).
2. V glavnem meniju ali oknu Start izberi ukaz New Project.
3. Izberi Visual Basic Projects, če ta opcija ni izbrana.
4. Desno izberi tip programa: Windows Application.
5. V okencu Location vpiši mapo, ki bo vsebovala mapo novega programa. Mapo lahko izbereš z gumbom Browse.
6. V okence Name zapiši ime novega programa.

Pritisni gumb OK in Visual Studio bo ustvaril projektno mapo, v katero bo zapisal datoteke programa. Prikazalo se bo več oken, obsežen meni in niz orodjarn. V sredi zaslona se bo pojavilo okno, ki predstavlja začetno stanje glavnega okna tvojega programa. V okno dodaj kontrolne gradnike, določi njihove dimenzije in lego z miško in v oknu Properties spremeni njihove lastnosti. Dodaj tudi ustrezne programske stavke (kodo). Privzete odzivne procedure ustvari z dvoklikom na ustrezno kontrolo.

DRAFT

Lastnosti okenskih programov Okenski program izdelan z orodjem Visual Studio.NET v programskem jeziku VB.NET ima naslednje lastnosti:

- Uporabniški vmesnik določajo kontrolni gradniki, to so vizualne programske komponente, ki omogočajo komunikacijo z uporabnikom. Kontrolnim gradnikom pravimo tudi kontrole ali kontrolniki. Kontrole so vedno postavljene znotraj glavnega okna programa. Omogočajo uporabniku nadzor nad programom. Nekatere skrbijo za vnos podatkov, druge za prikaz rezultatov, tretje pa upravljajo izvajanje operacij.
- Ob nastanku dobijo kontrole privzeta imena, ki pa jih je pametno čimpreje spremeniti, kar naredimo z določanjem vrednosti lastnosti (Name).
- Okenski program ima praviloma meni, ki uporabniku omogoča izbiranje operacij (ukazov). Najbolj pogosti ukazi so običajno dostopni na klik miške v orodni vrstici z gumbi.
- Informacija o tekočem stanju programa (sporočila o napakah) se pogosto prikazuje v statusni vrstici v spodnjem delu programa.
- Poleg stalno prisotnega glavnega okna, lahko komuniciranje uporabnika s programom prevzamejo pogovorna okna, ki se prikažejo na zahtevo uporabnika in se po uporabi spet skrijejo. Ostali deli programa so med operacijami z dialognim oknom začasno zamrznjeni.
- Posebni grafični gradniki omogočajo prikaze slik znotraj okna (PictureBox).
- Printerski gradniki omogočajo neposredno uporabo pisalnikov. Omogočajo kombiniran izpis slik in besedila.

Solution Explorer Solution Explorer je projektno podokno znotraj glavnega okna programa Visual Studio (VB 2005 Express), ki se običajno nahaja desno zgoraj. Vsebuje seznam datotek in drugih elementov, ki sestavljajo projekt. Omogoča nadzor nad izdelavo okenskega programa.

- Omogoča pregledovanje datotek s programsko kodo (imajo končnico .vb), dodajanje novih in odstranjevanje obstoječih.
- Omogoča izbor tekoče delovne datoteke .vb, njena vsebina se tedaj prikaže v urejevalnem delu programa.
- Omogoča tudi upravljanje z gradniki (programskimi komponentami). Knjižnice komponent (gradnikov) so praviloma vsebovane v datotekah s končnico dll. Nekaj datotek dll je programu prirejenih privzeto (odvisno od vrste projekta), druge lahko dodamo z ukazom Add Reference (dodaj sklicevanje na komponente).

11.2 Kontrole

11.3 Glavno okno

11.4 Kontrolni gradniki

11.5 Primer Krog

Vzorčni primer programa IzračunKroga je namenjen predstavitvi naslednjih konceptov okenskega in vizualnega programiranja z razvojnim okoljem VisualStudio 2005 ali VisualBasic 2005 Express:

- okenski projekt,
- urejanje kontrolnih gradnikov, določanje lastnosti kontrolnim gradnikom in oknu,
- predstavitev podatkov naloge s podatki znotraj razreda Form1,
- programiranje dogodkov z odzivnimi procedurami,
- programiranje podnalog s procedurami.

11.5.1 Opis

11.5.2 Okno

11.5.3 Koda

```
Imports System.Math

Public Class Form1
    'Podatki naloge.
    Dim radij, ploščina, obseg As Single

    ' Odzivna procedura na dogodek: sprememba vrednosti v kontroli tbRadij.
    Private Sub tbRadij_TextChanged(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles tbRadij.TextChanged

        PobrišiStatus()
        PobrišiRezultate()

        If tbRadij.Text = "" Then Return

       ' "tbRadij.Text" pretvori v realno število "radij". Ob neuspehu vrne False.
        If Single.TryParse(tbRadij.Text, radij) Then
            Izračunaj()
            PrikažiRezultate()
        Else
            PrikažiStatus("Neveljavno realno število")
        End If
    End Sub
```

```

    ' Izračuna obseg in ploščino kroga.
    Sub Izračunaj()
        ploščina = 2 * PI * radij
        obseg = PI * radij ^ 2
    End Sub

   ' Prikaže rezultate.
    Sub PrikažiRezultate()
        tbPloščina.text = ploščina
        tbObseg.text = obseg
    End Sub

    ' Pobriše rezultate.
   Sub PobrišiRezultate()
        tbObseg.Text = ""
        tbPloščina.Text = ""
    End Sub

    ' Pobriše status (informacijo o stanju programa).
    Sub PobrišiStatus()
       stOpis.Text = ""
    End Sub

    ' Prikaže sporočilo v statusni vrsatici.
    Sub PrikažiStatus(ByVal sporočilo As String)
       stOpis.Text = sporočilo
    End Sub
End Class
```

DRAFT