Visual Studio 2013 je objektno orijentisano programski integrisano razvojno okruženje, koje u sebi ima više programski jezika: Visual Basic, Visual C#, Visual C++, Visual F#.

U ovoj knjizi će najveća pažna biti posvećena programskom jeziku Visual Basic. Zahvaljujući Microsoft-u postao je profesinalni razvojni alat i standard za razvoj aplikacija u Windows okruženju.

Zbog svoje jednostavnosti u pisanju programa i lakom praćenju toka izvršenja programa, ovaj programski jezik može da posluži početnicima u programiranju za savladavanje prvih koraka u programiranju.

Cilj ovog izdanja je da se prilagodi potrebama studenata u početnoj fazi učenja programiranja, kao i svima onima koji su se ranije bavili programiranjem ali u nekom drugom programskom jeziku.

Izdavanjem ove knjige nadomještena je literatura, koja nedostaje za izvođenje predavanja i vježbi iz predmeta Uvod u programiranje i Programski jezici na Fakultetu poslovne ekonomije u Bijeljini i Pedagoškom fakultetu u Bijeljini.

Takođe je evidentan nedostatak literature, koji se odnosi na pravljenje algoritma. Autori su, uvidjevši tu prazninu, brižljivo razradili primjere algoritama. Na osnovu toga su studentima i ostaloj stručnoj javnosti ponudili jedan ovakav nastavni sadržaj.

ISBN 978-99955- 45-23- 9
COBISS.RS-ID 6641176
004.432.2(075.8)

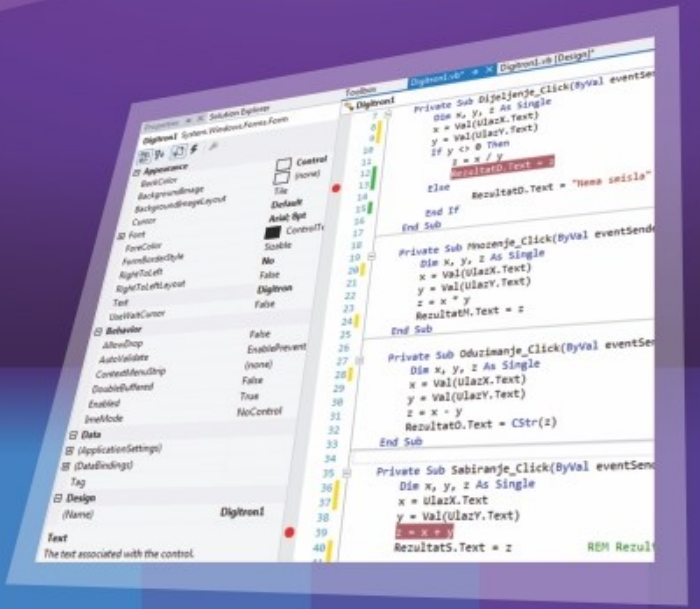
Integrisano razvojno okruženje
Visual Studio
2013

Visual Studio 2013
Integrisano razvojno okruženje

Dr Srđan Damjanović
Dr Predrag Katanić

Integrisano razvojno okruženje

Visual Studio 2013



FAKULTET POSLOVNE EKONOMIJE
BIJELJINA, 2017.

Dr Srđan Damjanović
Dr Predrag Katanić

INTEGRISANO RAZVOJNO OKRUŽENJE *VISUAL STUDIO* 2013

FAKULTET POSLOVNE EKONOMIJE
BIJELJINA, 2017.

INTEGRISANO RAZVOJNO OKRUŽENJE *VISUAL STUDIO* 2013

Autori:
Prof. dr Srđan Damjanović
Doc. dr Predrag Katanić

Recenzenti:
Prof. dr Rade Stankić
Prof. dr Slobodan Obradović

Izdavač:
UNIVERZITET U ISTOČNOM SARAJEVU
FAKULTET POSLOVNE EKONOMIJE BIJELJINA

Odgovorno lice za izdavača:
Prof. dr Miladin Jovičić

Odgovorni urednik:
Vanja Đurić, dipl. ek.

Lektor i korektor:
Mr Suzana Marković

Likovni urednik:
Borislav Drakul, dipl. ing. el.

Štampa:
GRAFIKA GOLE
Bijeljina

Za štampariju:
Gole Nikola

Tiraž:
200 primjeraka

Godina izdanja:
2017.

ISBN: 978-99955-45-23-9

Posveta

Ovu knjigu posvećujem dedi
Iliji Damjanoviću i dedi Risti Škrbi.

Srđan Damjanović

Ovu knjigu posvećujem dedi
Milanu Kataniću i dedi Milivoju Lakiću.

Predrag Katanić

S A D R Ž A J

UVOD

Visual Studio 2013 je objektno orijentisano programski integrisano razvojno okruženje, koje u sebi ima više programski jezika: *Visual Basic, Visual C#, Visual C++, Visual F#.* U ovoj knjizi će najveća pažna biti posvećena programskom jeziku *Visual Basic*. Zahvaljujući *Microsoft*-u postao je profesinalni razvojni alat i standard za razvoj aplikacija u *Windows* okruženju. Zbog svoje jednostavnosti u pisanju programa i lakom praćenju toka izvršenja programa, ovaj programski jezik može da posluži početnicima u programiranju za savladavanje prvih koraka u programiranju. Cilj ovog izdanja je da se prilagodi potrebama studenata u početnoj fazi učenja programiranja, kao i svima onima koji su se ranije bavili programiranjem ali u nekom drugom programskom jeziku. Izdavanjem ove knjige nadomještena je literatura, koja nedostaje za izvođenje predavanja i vježbi iz predmeta Uvod u programiranje i Programski jezici na Fakultetu poslovne ekonomije u Bijeljini i Pedagoškom fakultetu u Bijeljini.

Knjiga je napisana zbog nedostatka na tržištu literature na srpskom jeziku za integrisano razvojno okruženje *Visual Studio* 2013. Naime, postoje samo uputstva za rad sa raznim verzijama *Visual Studio* 2013, koja su napisana na engleskom jeziku i dostupna su u elektronskom obliku. To su, uglavnom, obimne dokumentacije izložene na više stotina, čak i oko hiljadu stranica, a uglavnom se odnose na jedan segment problema, koji se obrađuje. U ovoj knjizi je na relativno skromnom broju stranica izložena jedna konzistentna cjelina pogodna za edukaciju u obrazovnim ustanovama. Knjiga je posebno aktuelna za studente i čitaoce, koji se samostalno bave programiranjem u programskom jeziku *Visual Basic* 2013. Ponuđeni primjeri su, zbog konciznosti izlaganja, djelimično uprošćeni, ali se nevelikim trudom mogu brzo i efikasno dovesti do profesionalne aplikacije. Obrađene oblasti su raznovrsne i brižljivo birane, kako bi čitaoci mogli da uvide višestruke primjene obrađenog materijala u praksi. Knjiga je nastala nadogradnjom knjige „Programski jezik Visual Basic Zbirka zadataka“ od istih autora.

Takođe je evidentan nedostatak literature, koji se odnosi na pravljenje algoritma. Autori su, uvidjevši tu prazninu, brižljivo razradili primjere algoritama. Na osnovu toga su studentima i ostaloj stručnoj javnosti ponudili jedan ovakav nastavni sadržaj.

Knjiga je u osnovi podijeljena na osam poglavlja, koja čine dvije zasebne cjeline. Prvu cjelinu čini poglavlje jedan, u kome su date osnove za nastanak programiranja. Drugu cjelinu čine poglavlja dva do osam, u kojima su date osnove integrisanog razvojnog okruženja *Visual Studio* 2013, kroz teorijska razmatranja i praktične primjere programa.

Za svakog programera je bitno da poznaje kako funkcioniše hardver na kome se njegov program pokreće. To je razlog zašto su u prvom poglavlju predstavljene

osnovne hardverskih komponente računara. Zatim je ukratko opisan pojam softvera. Dati su osnovni koraci, koji čine proces pisanja programa. Na kraju ovog poglavlja opisan je postupak i značaj pravljenja algoritma, kroz nekoliko praktičnih primjera.

Postupak instaliranja *Visual Studio* 2013 predstavljen je u drugom poglavlju. Na kraju je opisano, kako može izgledati postupak pravljenja prvog programa u programskom jeziku *Visual Basic*.

U trećem poglavlju opisani su osnovni tipovi podataka, koji se koriste u *Visual Studio* 2013. Uz svaki tip podatka navedene su gotove funkcije, koje se mogu koristiti za taj tip podataka. Prikazani su i primjeri programa, kaji opisuje kako se koriste ove funkcije i tipovi podataka.

Naredbe selekcije i iteracije opisane su u četvrtom poglavlju. Detaljno su opisani razni postupci pravljenja grananja u programu kroz primjere, kako bi se što lakše mogli koristiti za pisanje programa. Predstavljen je i način višestrukog ponavljanja nekih naredbi, kako da se značajno smanjio broj kodnih linija u programu. Dati su primjeri sa rješenim ispitnim zadacima.

U petom poglavlju predstavljene su funkcije i procedure, koje omogućuju da program bude konstruisan od ranije napravljenih dijelova. Predstavljen je samo jedan mali dio gotovih funkcija, koje postoje u *Visual Studio* 2013, ali je opisano i kako se mogu isprogramirati nove. Navedene su vrste procedura, način pravljenja novih i način njihovog pozivanja iz glavnog programa.

Objektno orijentisano programiranje opisano je u šestom poglavlju. Objektno orijentisano programiranje je, kao što ime kaže, orijentisano ka objektu, koji je osnova ovakvog programiranja. Obezbjeđena je jednistvena kontrola pristupa podacima, pošto objekat nije prost tip podatka, poput cijelog broja ili niza znakova, već skup primitivnijih vrsta podataka, kojima je, po potrebi, dodato određeno ponašanje.

U sedmom poglavlju predstavljeni su najčešće korišćeni gotovi objekti, koji postoje u standardnoj paleti objekata u programskom jeziku *Visual Basic* 2013. Za svaki objekat su dati primjeri programa za korišćenje ovih objekata.

Primjeri programa napisanih u programskom jeziku *Visual Basic* 2013 predstavljeni su u osmom poglavlju. Prvo su predstavljeni primjeri programa, koji služe za komunikaciju sa *Access* bazom podataka i *Excel* dokumentom. U ovom poglavlju su dati zadaci za samostalan rad studenata, a slični zadaci se pojavljuju na praktičnom dijelu ispita.

Nadamo se da će ova kniga biti od koristi svima onima, koji se bave programiranjem u programskom jeziku *Visual Studio* 2013. Sugestije i pitanja mogu se slati na adresu srdamjan@yahoo.com i predrag@telrad.net.

1. OSNOVE PROGRAMIRANJA

Svjedoci smo da živimo u informatičkom dobu i najjednostavnije odluke se ne mogu donijeti, ako se ne posjeduje prava informacija[1]. Za rješavanje kompleksnih problema neophodno je obraditi ogromnu količinu podataka, kako bi se došlo do pravih informacija. Svi ovi poslovi oko dobijanja informacije (prikupljanje, ažuriranje, obrada, prenošenje podataka, itd.) su nezamislivi bez upetrebe računarskih sistema.

Računari su mašine za obradu podataka. One na osnovu programa vrše obradu ulaznih podataka, koje zadaje korisnik i generišu odgovarajući skup izlaznih podataka. Ulazni podaci redovno predstavljaju veličine, na osnovu kojih se obavlja rješavanje nekog problema, a izlazni podaci predstavljaju rezultate obrade datog problema. Stoga se podrazumijeva da svaki računar posjeduje jedinicu za ulaz (unos) podataka i jedinicu za izlaz (prikazivanje) rezultata. U slučaju kada korisnik računara interaktivno komunicira sa računarom onda je najčesće ulazna jedinica tastatura, a izlazna jedinica ekran. Naravno, postoje i brojni drugi ulazni i izlazni uređaji.

U svakodnevnom životu ljudi za međusobno sporazumijevanje upotrebljavaju cifre i slova, koje jednim imenom nazivamo alfanumjerički znaci. Za prenos informacija ljudi koriste slike i zvuk, kao i neke posebne znakove. Računar ne radi sa decimalnim brojevima. To su uobičajeni brojevi sa kojima računamo i brojimo. U računaru ne postoje elektronska kola, koja mogu raditi sa tim brojevima. Da bi se u računaru mogli obrađivati podaci, koji su opisani pomoću cifara, slova, slike, zvuka itd. potrebno ih je binarno predstaviti. Binarni brojevi su, barem na prvi pogled, čudni. Oni postaju uobičajeni i razumljivi tek onda kada se nauči njihovo pretvaranje u decimalne brojeve.

Binarni brojni sistem ima bazu dva, što znači da koristi samo skup cifara 0 i 1. Stoga se često naziva i "dualni" sistem brojeva. Koliko god je decimalni brojni sistem značajan i uobičajen za naš svakodnevni život, toliko je i binarni brojni sistem značajan za predstavljanje i shvatanje principa rada računara. Ovo dolazi otuda što elementarne elektronske i magnetne komponente računara u stvari mogu prikazati (tj. zauzeti) samo dva moguća stanja, uključeno (*On*) ili isključeno (*Off*). Primjeri funkcionisanja u binarnom obliku prisutni su i u našem svakodnevnom životu kod električnog zvonca na vratima, sijalice u kući itd. Čak i samo porijeklo riječi ("bis" - dva puta, "binaris" - dvojni) upućuje na bazu dva.

Da bi korisnik mogao da komunicira sa računarom kao mašinom, potrebno je da postoje programi, koji to omogućuju. Današnja situacija je takva da iste hardverske komponente mogu izvršavati programe pisane na različitim višim programskim

[1] Srđan Damjanović, Predrag Katanić, Borislav Drakul, *Zbirka zadataka iz poslovne informatike*, Fakultet spoljne trgovine, Bijeljina, 2008, str.17.

jezicima. Ovo je dovelo do toga, da pojam računar nedovoljno opisuje kompleksnost ovakvog sistema, pa se često pojam računara uopštava pojmom računarski sistem. Računarski sistem[2] se sastoji od pet komponenti: hardvera (*hardware*), softvera (software), podataka (*data*), procedura (*procedures*) i ljudi (*people*).

Jedinstveno djelovanje ovih pet komponenti računarskog sistema, definiše uslove rada i korišćenja računarskog sistema, odnosno definiše okruženje u kome korisnik radi.

Osnovni cilj ovog teksta je da definiše osnovne principe i mehanizme pomoću kojih se jedno takvo okruženje stvara i kako ono funkcioniše. Bolje razumijevanje okruženja, omogućava i efikasnije korišćenje mogućnosti koje ono pruža.

1.1. H A R D V E R

Pod hardverom podrazumijevamo fizičke komponente računarskog sistema kao što su: centralni procesor koji vrši obradu podatak, operativna memorija u kojoj se nalaze programi i podaci i razne vrste ulaznih i izlaznih uređaja. Ulazni uređaji su oni uređaji preko kojih sistem dobija naredbe ili podatke. U računarskom sistemu srećemo ulazne uređaje kao što su: tastatura, miš, džojstik, skener, mikrofon, digitalne kamere, olovka za crtanje, barkod čitači i sl. Izlazni uređaji su oni putem kojih sistem proslijeđuje informacije o izvršenoj naredbi, obrađenim podacima korisniku sistema. U računarskim sistemima srećemo sljedeće izlazne uređaje: monitor, štampač, ploter, zvučnik, projektor, razni računarski kontrolisani uređaji, kao, razne vrste sekundarnih i spoljnih memorija.

U ovomj knjizi ćemo se zadržati samo na centralnom procesoru i njegovim komponentama.

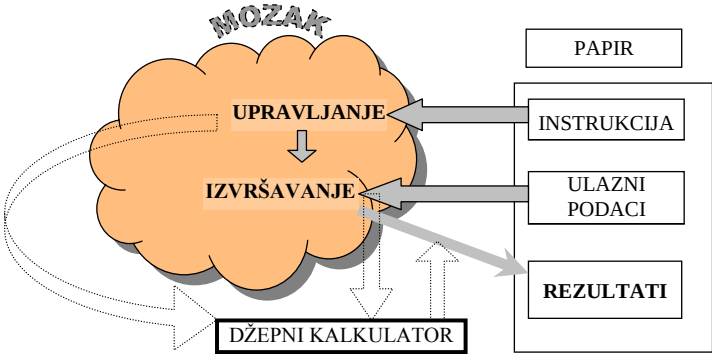
1.1.1. Osnovne komponente računarskog sistema

U cilju definisanja osnovnih hardverskih komponenti računarskog sistema i njegovog funkcionisanja, poći ćemo od poređenja ručne i automatske obrade podataka.

U opštem slučaju, ručna obrada podataka odvija se tako što se prethodno na papiru pripremi uputstvo (opis postupka obrade, instrukcije) i podaci sa kojima se počinje obrada. U samom izvršavanju obrade, čovjek čita jednu po jednu instrukciju (korak obrade) sa papira i potom obavlja dvije radnje. Prva radnja je vezana za upravljanje, odnosno tumačenje instrukcije i donošenje odluke o operaciji, koja će se izvršiti. Druga radnja je samo izvršavanje operacije, pri čemu

[2] Slobodan Obradović, *Veština dobrog programiranja*, Viša elektrotehnička škola Beograd, 1997. str. 1

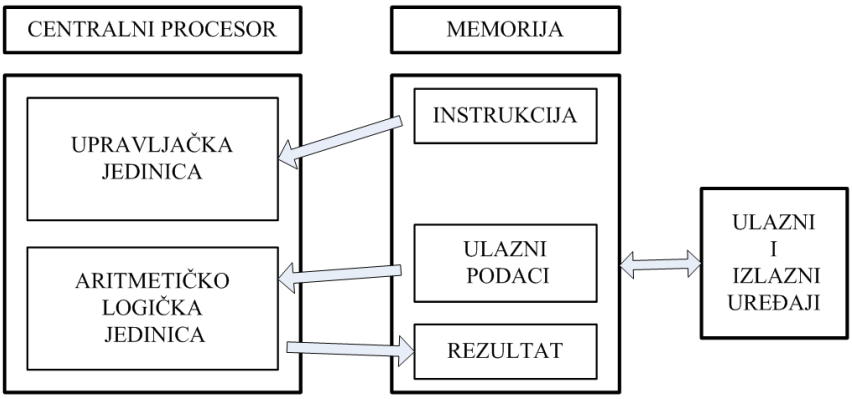
se najćešće podaci potrebni za njeno izvršenje čitaju sa papira, a rezultat operacije se takođe zapisuje na papir. Blok dijagram koji predstavlja opisani postupak prikazan je na slici 1.1.



Slika 1.1. Ručna obrada podataka

Na slici 1.1. takođe je prikazana i uloga kalkulatora u obradi podataka, pri čemu se može uočiti da kalkulator može zamijeniti čovjeka samo u izvršenju operacija, a ne i u upravljanju obradom. Naime, čovjek mora da unese podatke u kalkulator i da mu zada operaciju, koju treba izvršiti sa tim podacima. Po izvršenju jedne zadate operacije i dobijanju rezultata, kalkulator miruje. Da bi se obrada nastavila, čovjek mora da unese nove podatke u kalkulator i da mu zada novu operaciju itd.

Očigledno, kalkulator nije automat, odnosno mašina koja može da po izvršenju jedne operacije automatski, bez intervencije čovjeka, pređe na izvršavanje sljedeće operacije. Automatska mašina bi očigledno morala da zamijeni čovjeka, ne samo u izvršavanju operacija sa podacima, već i u upravljanju obradom. Takva automatska mašina jeste centralni procesor, koji ima upravljačku jedinicu i aritmetričko logičku jedinicu.



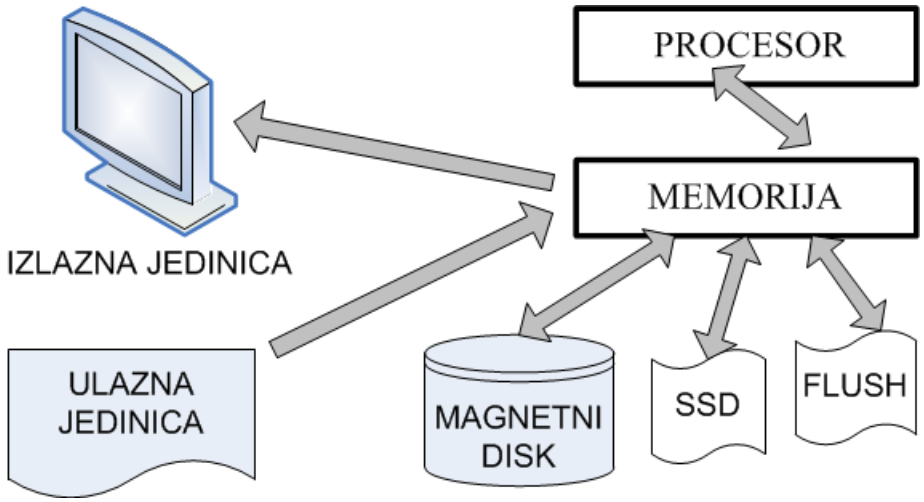
Slika 1.2. Osnovna blok šema računara

Kako je to prikazano na slici 1.2., osnovna komponenta hardvera računarskog sistema je centralni procesor (*central processing unit*), čiji je zadatak da uzima instrukcije iz memorije, analizira ih i potom izvršava. Ram memorija računara (*RAM memory*) sa slučajnim, proizvoljnim pristupom čuva (pamti, skladišti) instrukcije i podatke (početne podatke, međurezultate i rezultate obrade). Ulazni i izlazni uređaji omogućavaju komunikaciju računara sa njegovom okolinom, kao i spoljne, sekundarne memorije koje služe za trajno skladištenje programa i podataka.

Nešto detaljnija konfiguracija računarskog sistema prikazana je na slici 1.3. Na njoj se može uočiti, da su u cilju bolje preglednosti razdvojeni ulazni i izlazni uređaji, dok su, sa druge strane prikazana i dva nova memorijska uređaja - magnetni diskovi, SSD i *Flash* kao eksterne memorije.

Naime, kapacitet radne memorije, odnosno, kako se često naziva, operativne memorije, uprkos tome što se stalno povećava, nije dovoljno veliki da uskladišti sve programe i podatke, koji se koriste u automatskoj obradi podataka. Usljed toga, praktično svaka konfiguracija računarskog sistema posjeduje i dodatne memorijske uređaje, koji se nazivaju sekundarnim memorijama i na kojima se trajno čuvaju programi i podaci. Kako u toku rada centralni procesor pristupa samo onim instrukcijama i podacima, koji se nalaze u operativnoj memoriji, programi koji se trenutno izvršavaju i podaci nad kojima se vrši obrada, prebacuju se po potrebi iz sekundarnih memorija u operativnu memoriju.

U nastavku izlaganja, kako bismo lakše shvatili suštinu funkcionisanja računarskog sistema, ograničićemo se na osnovni opis strukture i funkcije dvije ključne komponente računarskog sistema: centralnog procesora i operativne memorije.



Slika 1.3. Detaljna konfiguracija računarskog sistema

1.1.2. Centralni procesor

Tipična, iako još uvijek nedovoljno detaljna struktura centralnog procesora, ili jednostavnije procesora, prikazana je na slici 1.4. Osnovna funkcija upravljačke jedinice je da uzima instrukcije iz memorije, analizira ih i šalje odgovarajuće komande za njihovo izvršavanje ostalim komponentama.

Slika 1.4. Struktura centralnog procesora

Aritmetičko-logička (A/L) jedinica, kako samo ime govori, sastoji se od elektronskih kola, koja obavljaju različite aritmetičke, logičke, kao i neke druge operacije nad podacima. Za A/L jedinicu se kaže da je jedina aktivna komponenta računarskog sistema, u smislu da je jedina komponenta, koja može da stvarno obrađuje podatke, u najužem smislu te riječi. Jedna od bitnih karakteristika procesora je skup instrukcija, koje on može da izvrši. Detaljniji opis tipova intrukcija prevazilazi ovaj kurs.

Pored navedenih komponenti, centralni procesor sadrži i skup posebnih hardverskih jedinica, registara, obično 8 do 16, koji igraju ulogu interne, brze memorije samog procesora. Kako je brzina pristupa registrima približno za red ili dva reda veličine veća od brzine pristupa memoriji, procesor u toku izvršavanja programa u ovim registrima opšte namjene i registrima posebne namjene, čuva podatke kojima mora često da pristupa.

Napredak u razvoju računara je na početku tekao sporo, a zatim je došlo do postepenog ubrzavanja razvoja, da bi danas imali takvu situaciju da se razvoj računarske tehnike ne može ni pratiti tj. gotovo svakodnevno se pojavi novo rješenje u cilju napretka računarskih sistema. Razvoj tehnologije posebno se ogleda na polju razvoja procesora. Tako da pojedine komponente procesora poprimaju nove oblike. Uvode se nove memorijske jedinice ("keš" memorija L1, L2, L3, ...). Ovo omogućava mnogo rjeđe pristupe sporijoj *RAM* memoriji. Na ovaj način se ubrzao pristup podacima, a samim tim i obrada podataka. Današnji računarski

sistemi imaju i do 130000 mikroprocesora (“srce računara”). Na ovaj način se paralelno može obrađivati veliki broj podataka, u najsloženijim sistemima kao što je na primjer praćenje satelita i obrada meteoroloških podataka, koji se prikupljaju sa njih.

1.1.3. Operativna memorija

Operativna (primarna) memorija, ili kraće memorija, je pasivna komponenta računarskog sistema, čija je funkcija da skladišti programe i podatke. Memorija se sastoji od velikog broja registara (memorijskih lokacija) jednake dužine, pri čemu svaki registar ima svoj redni broj u okviru memorije. Numerisanje memorijskih registara počinje od 0 (nule). Redni broj registra predstav1ja njegovu jedinstvenu adresu. Jedan memorijski registar moža da sadrži jednu i samo jednu instrukciju ili samo jedan podatak. Možemo da izvedemo zaključak da svaka instrukcija, odnosno, svaki podatak ima sopstvenu adresu - adresu lokacije u kojoj je smješten.

Uvođenjem koncepta adresa i adresiranja stvoren je bitan mehanizam za funkcionisanje računara. Stvorena je mogućnost da se svaka instrukcija i svaki podatak smiješta u određenu memorijsku adresu, da bi mu se kasnije, posredstvom te adrese moglo pristupiti. Pri tome su nad memorijom moguće dvije operacije: upisivanje i čitanje sadržaja neke adrese. Realizacija ovih operacija omogućena je posredstvom dva registra, od kojih se jedan naziva adresnim registrom memorije (MAR), a drugi prihvatnim registrom memorije (MBR). Operacija čitanja sadržaja određene memorijske lokacije, obavlja se tako, što se adresa lokacije kojoj se želi pristupiti, upisuje u adresni registar memorije. Zatim se inicira izvršenje same operacije, da bi se kao rezultat njenog dejstva sadržaj posmatrane lokacije prekopirao u prihvatni registar memorije.

Operacija upisivanja podatka u određenu memorijsku lokaciju odvija se na sličan način. Adresa lokacije u koju se vrši upisivanje unosi se u adresni registar memorije, a podatak koji treba da se upiše, unosi se u prihvatni registar memorije. Zatim se inicira operacija upisivanja. Rezultat operacije upisivanja je da se podatak iz prihvatnog registra memorije upisuje u posmatrani registar, pri čemu se, naravno, njen prethodni sadržaj trajno uništava.

Vrijeme koje protekne od trenutka iniciranja jedne operacije nad memorijom, bez obzira da li je u pitanju čitanje ili upisivanje, do završetka posmatrane operacije, naziva se vremenom memorijskog ciklusa. Jednostavnije rečeno, vrijeme memorijskog cilusa je vrijeme, koje mora da protekne između dva uzastopna pristupa memoriji. Pri tome, treba napomenuti da je vrijeme pristupa jednako za sve lokacije operativne memorije. Zbog toga se primarna memorija često naziva memorija sa slučajnim pristupom. Ovaj naziv je zapravo doslovan prevod originalnog izraza (*random access memory - RAM*).

1.2. S O F T V E R

Softver (*eng. software*) računara predstavlja skup programa, smještenih u memoriju računara, koji kao cjelina, povremeno ili stalno, kontroliše i upravljanja radom računara. Ovdje se pod memorijom podrazumijeva sva dostupna radna i spoljna memorija računaru. Računarski programi se prave sa ciljem, da se pomoću njih rješavaju računarski rješivi problemi. Pretraživanje baza podataka, izračunavanje plata, upravljanje mašinama, samo su neki primjeri uobičajene primjene računara. Broj računarski rješivih problema je beskonačan, a broj praktičnih primjena računara je ograničen samo našom maštom, znanjem i vještinom. Grubo softver možemo podijeliti na:

- sistemski softver i,
- korisnički softver (aplikativni).

Sistemski softver se dijeli na:

- operativni sistem (OS) i
- uslužne programe.

Operativni sistemi su se neprestano razvijali sa razvojem računarskih sistema. OS se posmatra na njegovom najnižem nivou, kao nadzornik koji pazi na disk, procesor, memoriju, štampače i dr. OS ima ulogu posrednika između korisničkih programa i sredstava potrebnih za njihovo izvođenje (hardvera). Ovakva filozofija računarskog sistema je prikazana na slici 1.5., gdje se jasno može vidjeti mjesto i zadatak operativnog sistema.

Slika 1.5. Struktura operativnog sistema

Kompleksnost operativnog sistema otežava preciznu definiciju. Sa zadržanom uopštenošću, može se reći, da pod operativnim sistemom podrazumijevamo skup sistemskih programa, koji upravljaju radom svih hardverskih komponenti, dodeljuju programu koji se izvršava sve potrebne resurse i djeluju kao posrednik između hardvera, programa i korisnika. Upravlja dodjeljivanjem (alokacijom) resursa računarskog sistema, u cilju njegovog efikasnog rada.

Kraće rečeno, operativni sistem je organizovan skup sistemskih programa, koji upravlja radom različitih komponenti računarskog sistema svim programima koji se istovremeno izvršavaju i efikasno korišćenje resursa samog sistema. Pri tome se poslovi (funkcije) operativnog sistema mogu grubo podijeliti u četiri grupe:

- Upravljanje procesorom,
- Upravljanje memorijom,
- Upravljanje uređajima,
- Upravljanje podacima.

Resursi računarskog sistema o kojima OS mora voditi računa su: procesori, memorija, ulazno-izlazni uređaji i podaci. Navedimo neke od poslova koje obavljaju komponente operativnog sistema:

1) Jezgro (kernel ili nucleus**)** operativnog sistema obezbjeđuje realizaciju sljedećih funkcija:

- upravljanje sistemom prekida računara i obradu prekida,
- dodela procesora (dispečer),
- manipulaciju nad procesima (proces je program, koji je prenijet u operativnu memoriju i izvršava se),
- sinhronizaciju procesa,
- komunikaciju među procesima.

2) Upravljanje memorijom podrazumijeva upravljanje operativnom memorijom računara. Obuhvata sljedeće funkcije:

- realizaciju određene strategije dodjele memorije,
- dodjelu memorije,
- realizaciju određene strategije oslobađanja memorije.

3) Upravljanje uređajima obuhvata sljedeće funkcije:

- obezbjeđenje nezavisnosti uređaja,
- obezbjeđenje efikasnosti rada uređaja,
- realizaciju određene strategije dodjele uređaja,
- dodjelu uređaja,
- realizaciju određene strategije oslobađanja uređaja.

4) Upravljanje podacima treba da obezbijedi softverska sredstva za organizovanje i pristup podacima na način koji odgovara korisniku računarskog sistema. Funkcije koje se realizuju na ovom nivou su:

- kreiranje i brisanje fajlova,
- otvaranje i zatvaranje fajlova,
- čitanje i upisivanje,
- upravljanje prostorom na sekundarnim memorijskim jedinicama,
- obraćanje fajlovima po imenu,
- zaštita podataka od namjernog i nenamjernog uništenja,
- zaštita podataka od neovlašćenog pristupa i korišćenja,
- zajedničko korišćenje fajlova.

5) Planiranje obuhvata aktivnosti u vezi sa uvođenjem novih poslova u sistem i određivanje poretka u kojem će se oni izvršavati. Funkcije koje se realizuju na ovom nivou su:

- izbor novog posla za izvršenje,
- dodjela prioriteta poslovima,
- realizacija strategije dodjele resursa.

6) Evidencija obuhvata vođenje evidencije korišćenja svih resursa sistema po korisnicima.

Očigledno je da akcenat na riječi upravljanje, u navođenju funkcija operativnog sistema nije slučajan. Naime, u situaciji, kada se u računaru odvija više aktivnih procesa (programi koji se izvršavaju), jasno je da će ti procesi konkurisati jedan drugom u pogledu korišćenja resursa računara (procesora, memorije, različitih uređaja, datoteka tj. podataka itd.). Zadatak operativnog sistema je da omogući neometano odvijanje svih procesa na takav način, da se svi resursi sistema što efikasnije iskoriste.

Uslužni ili korisnički programi koriste se na računaru za obavljanje nekih specifičnih poslova. Danas postoji veliki broj uslužnih programa, koji se razlikuju po namjeni i po veličini. Recimo *Word* se koristi za pisanje i uređivanje teksta, *Excel* za razne matematičke proračune i grafički prikaz podataka, *Access* za pravljene baze podataka itd. Uslužni programi se mogu pisati u raznim programskim jezicima, ali se pomoću prevodioca svi moraju prevesti na mašinski jezik. U Windows operativnom sistemu se prepoznaju, u odnosu na druge podatke na računaru, po tome što uglavnom imaju ekstenziju ".EXE".

1.3. PISANJE PROGRAMA

Programi se pišu, kako bi pomogli čovjeku da pomoću računara riješi neki problem. Instrukcije i programi zahtjevaju neki redoslijed, kojim će ih računar izvršavati. Projektovanjem programa određujemo koje su programske instrukcije i kakva struktura podataka je potrebna da bi mašina-računar obavila neki željeni zadatak. Najznačajnije je to što projektant programa određuje redoslijed, kojim će se instrukcije izvršavati u cilju uspješnog odvijanja i završetka programa. Postoje dva pristupa u razvoju programa[3]:

- od računara ka problemu i
- od problema ka računaru.

U prvom pristupu se polazi od toga da korisnik najprije upozna računar i njegove mogućnosti, a zatim pristupi rješavanju problema uz pomoć računara. U drugom pristupu se prvo nastoji dobro razumjeti problem, zatim napraviti odgovarajući model kako da se taj problem riješi, a tek zatim se pristupa pisanju programa u odgovarajućem programskom jeziku na računaru.

Programiranje u užem smislu predstavlja proces pisanja programa za računar. U širem smislu to je proces pripreme, razrade i pisanja programa radi rješavanja nekog problema na računaru. Proces programiranja zavisi od problema, koji se rješava. Međutim, mogu se uvesti neki tipični postupci, koji se odvijaju u toku razvoja programa. U programiranju uočavamo sljedeće faze:

- Definisanje problema,
- Izrada algoritma,
- Pisanje i unošenje programa u računar,
- Testiranje programa i ispravke greške,
- Implementacija programa i obuka korisnika i
- Održavanje i nadogradnja programa.

Sve ove korake mora da prati odgovarajuća dokumentacija, u vidu tekstualnih zapisa, grafičkih prikaza i slika.

1.3.1. Definisanje problema

Prva i najbitnija faza u procesu pisanja programa je definisanje problema. Najbolje je da se glavni problem prvo podijeli na više manjih problema, koji se mogu lakše dalje rješavati. U ovoj fazi treba da se uoči problem, određuje se način rješavanja, vrši se analiza postojećeg stanja, analiziraju se iskustva u radu sa ovakvim i sličnim zadacima, biraju metode rada. U ovoj fazi rada često je neophodno definisati fizički sistem i matematički model sistema ili procesa čije ponašanje se želi implementirati na računaru. Fizički sistem uvijek predstavlja

[3] Dr Tihomir Latinović, *Osnove programiranja (Visual Basic)*, Biblioteka Informacione tehnologije, Banja Luka 2007, str. 2.

realni sistem ili proces, koji se odvija u prirodi. Matematički model je skup matematičkih postupaka i relacija kao i puteva njihovog rješavanja, koji moraju egzaktno odrediti postavljeni fizički sistem. Matematički model zapravo predstavlja skup matematičkih relacija, koje dovoljno tačno opisuju pojavu ili proces, koji želimo da opišemo programom. Najčešće se pri tome koristimo jednačinama i sistemima jednačina, nejednačinama i sistemima nejednačina, različitim transformacijama, funkcijama, vektorima, matricama i drugim matematičkim iskazima.

1.3.2. Izrada algoritma

Ljudi se svakodnevno nalaze u situaciji da rješavaju različite, manje ili više kompleksne, probleme (zadatke). To mogu biti zadaci na radnom mjestu, u kući ili u životnoj sredini, čovjeka uopšte. Čovjek ne bi dorastao ni najobičnijem problemu, da ne koristi razna pomagala tj. alate, mašine uređaje itd. Jedno od pomagala jeste i računar sa svim svojim prednostima. Međutim, iako je računar mašina, još uvijek ne posjeduje razum.

Značajna sposobnost ljudi jeste da uoče zadatak, da ga dobro postave, a zatim da ga riješe. Zadatak sadrži skup informacija, na osnovu kojih treba izvesti određene zaključke, koji predstavljaju rješenje zadatka. Ovakav skup informacija čini ulazne veličine zadatka. Kao rezultat zadatka dobijaju se izlazne veličine zadatka.

Izbor informacija koje čine ulazne veličine zadatka, kao i tačno formulisanje samog zadatka možemo zvati postavka zadatka. Jasno je da prije nego što se pristupi rješavanju, mora se izvršiti tačna postavka zadatka. Rad na postavci zadatka zahtijeva dobro poznavanje oblasti, kojoj pripada zadatak.

Kada je izvršena postavka zadatka, može se pristupiti rješavanju istog, a prvi naredni korak je izrada algoritma. Algoritam je uređen skup pravila, koja se formulišu u cilju rješavanja zadatka. Ulazne veličine zadatka zovu se ulazne veličine algoritma, a izlazne veličine zadatka zovu se izlazne veličine algoritma. Svako pravilo, iz skupa pravila formulisanih u cilju rješavanja zadatka, zove se algoritamski korak. Prema tome, algoritam se sastoji od niza algoritamskih koraka, kroz koje se vrši transformacija ulaznih veličina algoritma, sve dok se ne dođe do izlaznih veličina algoritma tj. rješenja. U ovakvom nizu algoritamskih koraka mora se znati prvi (start) i zadnji algoritamski korak (kraj). Svi ostali algoritamski koraci određeni su izvršenjem predhodnog algoritamskog koraka. Veze između algoritamskih koraka određuju strukturu algoritama.

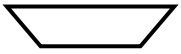
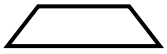
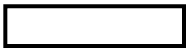
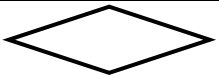
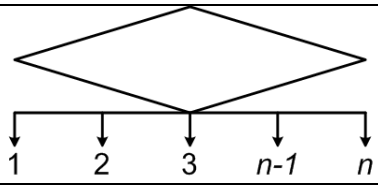
Osnovne osobine algoritma su: definisanost, konačnost, efikasnost, diskretnost i masovnost. Definisanost znači da se moraju tačno definisati: ulazi, izlazi, postupci i redoslijed obavljanja pojedinih postupaka. Konačnost i efikasnost proističu iz potrebe da se uz što manji broj operacija i koraka, dođe do prihvatljivog i željenog

rješenja. Diskretnost znači da se algoritam sastoji iz tačno određenog konačnog broja koraka. Masovnost proizilazi iz činjenice da je skup ulaznih i izlaznih veličina, kao i promjenjljivih praktično neograničen.

1.3.2.1. Grafički zapis algoritma

Za zapis algoritama pogodno koristiti njihovo grafičko prikazivanje[4]. Kod ovakvog prikazivanja algoritama svaki algoritamski korak je prikazan grafičkim simbolom, koji su između sebe povezani linijama saglasno strukturi algoritma. Oblik grafičkog simbola za algoritamski korak ukazuje na funkciju, koju obavlja određen algoritamski korak. U tabeli 1.1. dati su grafički simboli za pojedine algoritamske korake i opisana je njihova funkcija.

Tabela 1.1. Grafički simboli algoritamskih koraka

Grafički simbol algoritamskih koraka	**Funkcija algoritamskog koraka**
početak/kraj	Ukazuje na prvi ili zadnji algoritamski korak
	Definiše ulazne veličine algoritma
	Definiše izlazne veličine algoritma
	Definiše obradu podataka
	Uslovni algoritamski korak
	Definiše višestruku odluku
	Povezivanje ili konektor
	Tok algoritma
	Dvosmjerni prenos podataka

[4] Dr Lazar Miličević, Mr Lazar Radovanović, *Programiranje (Visual Basic)*, Ekonomski fakultet, Brčko, 2005, str.11.

Grafički zapis algoritma zove se blok-šema algoritma ili dijagram toka. Dijagrami toka su definisani međunarodnim standardom *ANSI X3.5*. Ovakav zapis algoritma odlikuje se sljedećim osobinama:

- preglednije praćenje toka algoritma,
- obezbjeđuje lako otkrivanje grešaka u strukturi algoritma,
- omogućuje kraći i jasniji zapis algoritma, nego pisanim jezikom,
- daje preglednu vezu između detalja i cjeline algoritma, i
- algoritam zapisan u obliku blok-šeme nezavisan je od kasnijeg korišćenja algoritma u odnosu na hardversku podršku i programski jezik u kome će program biti pisan.

Posljednja osobina je posebno značajna, jer grafičko prikazivanje algoritma nije orijentisano na prenošenje algoritma na računar, što omogućuje da algoritam u obliku blok-šeme mogu koristiti i osobe koje ne poznaju računare i programiranje. Detaljisanje algoritma, pri grafičkom prikazivanju može biti različito i zavisi od namjene blok-šeme algoritma. Blok-šema algoritma treba da je toliko detaljna, da svaki algoritamski korak bude razumljiv za onog ko će koristiti algoritam. Za složenije algoritme pogodno je koristiti i blok-šeme različitog nivoa detaljisanja algoritamskih koraka.

Ako se za prikaz algoritma koriste različiti nivoi detaljisanja algoritma, onda se koristi opšta blok-šema i detaljna blok-šema algoritma. Opšta blok-šema algoritma sadrži algoritameke korake, koji predstavljaju veće funkcionalne ili logičke cjeline u složenom algoritmu. Detaljna blok-šema algoritma sadrži algoritamske korake, u kojima je jasno naznačena funkcija svakog algoritamskog koraka. Opšta blok-šema služi za uvid u logičku strukturu složenog algoritma, a detaljna za uvid u sve detalje algoritma sa gledišta njegovog izvršavanja.

1.3.2.2. Struktura algoritma

Veze izmedju algoritamskih koraka u algoritmu definišu strukturu algoritma. Strukture algoritama, na koje se može razložiti proizvoljna struktura algoritma, zovu se elementarne strukture algoritama.

Elementarne strukture algoritama su:

- linijska struktura,
- ciklična struktura.

Ove strukture se među sobom bitno razlikuju sa gledišta izvršavanja algoritma.

Linijska struktura algoritma

Niz algoritamskih koraka u kojem se svaki algoritamski korak može izvršiti najviše jedan puta, u toku jednog izvršavanja algoritma čini linijsku strukturu. Prema tome, karakteristika linijske strukture algoritma jeste da poslije izvršenog jednog algoritamskog koraka, može se preći samo na sljedeći algoritamski korak. Linijska struktura algoritma može biti prosta ili razgranata.

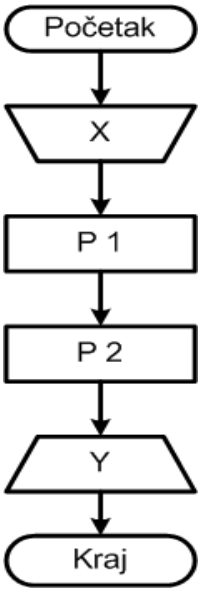
Prosta linijska struktura

Prosta linijska struktura algoritma je ona linijska struktura, kod koje se svaki algoritamski korak izvršava samo jedan put, u toku jednog izvršavanja algoritma.

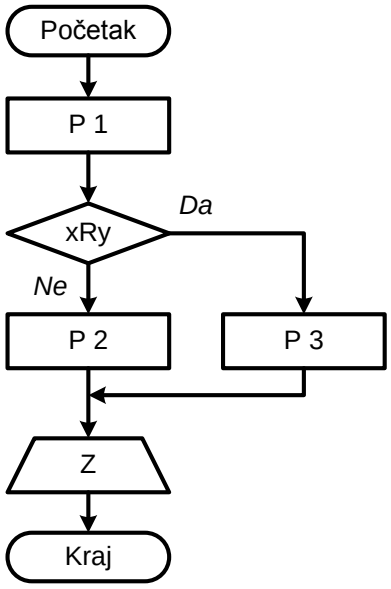
Algoritmi sa prostim linijskim strukturama, sastoje se isključivo od algoritamskih koraka ulaza, obrade ili izlaza. U ovakvim algoritamskim strukturama redoslijed izvršavanja algoritamskih koraka ne zavisi od ulaznih veličina, niti od među rezultata u procesu izvršavanja algoritma. Primjer ove strukture prikazan je na slici 1.6.

Razgranata linijska struktura

Razgranata linijska struktura prikazana je na slici 1.7. Ovo je linijska struktura algoritma, kod koje se svaki algoritamski korak izvršava najviše jednom, u toku jednog izvršavanja algoritma. To znači da u razgranatoj algoritamskoj strukturi postoje algoritamski koraci, koji se jednom izvrše, ali postoje i algoritamski koraci, koji se uopšte ne izvrše u toku jednog izvršavanja algoritma. U razgranatim algoritamskim strukturama mora postojati barem jedan uslovni algoritamski korak, koji omogućuje grananje algoritma na minimalno dva dijela. Jedan dio koji se dalje nastavlja da se izvršava i drugi dio koji se u tom izvršenju za date ulazne podatke neće izvršiti.



Slika 1.6. Prosta linijska struktura



Slika 1.7. Razgranata linijska struktura

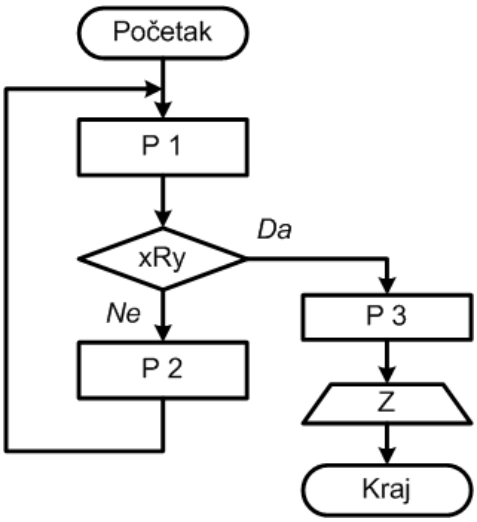
Elementarna razgranata struktura mora se sastojati od minimalno jednog uslovnog algoritamskog koraka, odnosno koraka grananja. Na slici 1.7. prikazana je elementarna razgranata struktura sa prostim algoritamskim koracima $\mathbf{P_1}$, $\mathbf{P_2}$ i $\mathbf{P_3}$ i

uslovnim algoritamskim korakom **xRy**. Gdje su **x** i **y** prethodno definisane veličine, a **R** relacija poređenja između veličina **x** i **y**. Relacija **R** (logički uslov) može biti ispunjena (Da) i tada se prelazi na prosti algoritamski korak $\mathbf{P_3}$ ili neispunjena (Ne) i tada se prelazi na prosti algoritamski korak $\mathbf{P_2}$. Važno je uočiti da se pri izvršavanju algoritma, na slici 1.7., izvršava uvijek samo jedan od prostih algoritamskih koraka $\mathbf{P_2}$ ili $\mathbf{P_3}$, u zavisnosti od relacije **xRy**.

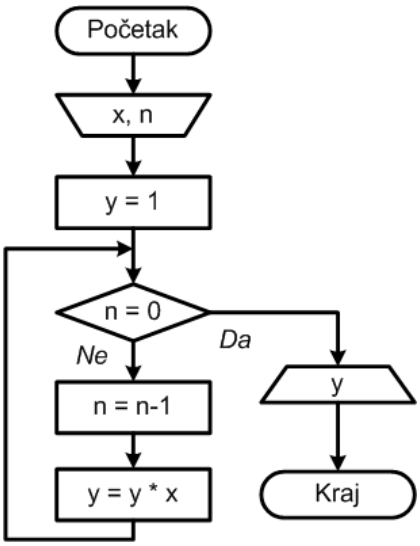
Veličine **x** i **y** mogu biti definisane sa ulaza ili to mogu biti međurezultati određeni u prostom algoritamskom koraku $\mathbf{P_1}$. Relacija između veličina **x** i **y** odgovara prirodi ovih veličina. Može se ispitivati da li su jednake ili koja je od ovih veličina manja, odnosno veća, dakle bilo koja relacija poređenja čiji rezultat je tačan ili netačan.

Ciklična struktura algoritma

Niz algoritamskih koraka u kojem se jedan ili više algoritamskih koraka može izvršiti više od jedan put, pri jednom izvršavanju algoritma, predstavlja cikličnu strukturu ili ciklus.



Slika 1.8. Konstantna ciklična struktura



Slika 1.9. Izračunavanje stepena

Ciklična struktura može biti konstantna i promjenljiva ciklična struktura. Svaka od ovih, cikličnih struktura u najopštijem obliku sastoji se od dva prosta linijska algoritamska koraka $\mathbf{P_1}$ i $\mathbf{P_2}$ (slika 1.8.) i uslovnog algoritamskog koraka **xRy**. Ako je relacija navedena u uslovnom algoritamskom koraku ispunjena (Da), vrši se izlazak iz ciklusa i prelazi se na prosti algoritamski korak $\mathbf{P_3}$, koja vodi do kraja cijelog algoritma. Ako uslov **xRy** nije ispunjen prelazi se na prosti algoritamski korak $\mathbf{P_2}$, poslije čega se ciklus ponavlja. Relacija koja definiše izlazak iz ciklusa

zove se izlazni kriterijum ciklusa. Naravno, da se može ciklus organizovati i tako da se izlazak iz ciklusa vrši ako uslov, koji definiše izlazni kriterijum, nije ispunjen, a ciklus nastavlja ako je uslov ispunjen.

Konstantna ciklična struktura

Ciklična struktura algoritma u kojoj ne dolazi do promjene zakona obrade u algoritamskim koracima, koji čine ciklus, zove se konstantna ciklična struktura. Izlazni kriterijum kod konstantnih cikličnih struktura najčešće je broj ponavljanja ciklusa ili dostignuta tačnost pri računanju po interativnim postupcima. Kod iterativnih ciklusa broj prolazaka kroz ciklus, obično nije unaprijed poznat.

Primjer 1.1. Sastaviti algoritam za izračunavanje stepena $y = x^n$ gdje je x nepoznata, a stepen može imati vrijednosti prirodnih brojeva $n = 0, 1, 2, ...$

Ovdje su x i n ulazne veličine, a y izlazna veličina. Pošto je eksponent cio broj, stepen se može izračunati uzastopnim množenjem. Prema tome, algoritam za rješavanje ovog zadatka sadrži ciklus u kojem će se vršiti n množenja osnove x, a izlazni kriterijum je broj izvršenih množenja (slika 1.9.).

Važno je uočiti da je u ovom zadatku vrijednost n promjenljiva (zadaje se kao ulazna veličina) te je nemoguće algoritam za ovaj zadatak sastaviti kao linijsku algoritamsku strukturu. U ovom zadatku korišćen je simbol, koji predstavlja operaciju dodjeljivanja vrijednosti promjenljivoj. Uočimo da ovaj simbol određuje proces izračunavanja vrijednosti na desnoj strani, a zatim ovako izračunata vrijednost se dodjeljuje promjenljivoj na lijevoj strani simbola. Zapis *" $y = x$ "* čita se "*vrijednost promjenljive* x *dodjeljuje se promjenljivoj* y ".

U ovom algoritmu je korišćen zapis $n = n - 1$, što znači da se vrijednost promjenljive n umanjuje za jedan i tako formirana vrijednost dodjeljuje, kao nova vrijednost promjenljive n. Da u posmatranom algoritmu nema ovog bloka, ovaj algoritam bi predstavljao beskonačnu petlju, koja se nikada ne bi završila.

Algoritam na slici 1.9. ima konstantnu cikličnu strukturu, jer zakon obrade u svim algoritamskim koracima, koji se nalaze u ciklusu, ne mijenja za vrijeme izvršavanja algoritma.

Za ovaj algoritam je takođe bitno da se uoči, da se ciklična struktura ne mora nijednom izvršiti, jer se uslov za ulazak u ciklus ispituje na početku ulaska u ciklus.

Promjenljiva ciklična struktura

Ciklična struktura u kojoj dolazi do promjene zakona obrade, u jednom ili više algoritamskih koraka, koji se nalaze u ciklusu zove se promjenljiva ciklična struktura. Promjena zakona obrade može se odnositi na promjene operacija, koje vrše obradu informacija ili na promjene promjenljivih, koje učestvuju u pojedinim algoritamskim koracima.

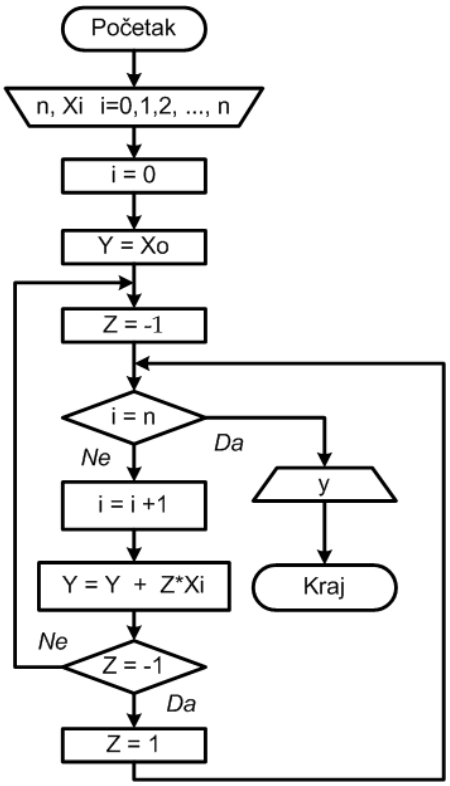
Primjer 1.2. Sastaviti algoritam, koji izračunava sljedeću sumu

$$y = \sum_{i=0}^{n} (-1)^i x_i \qquad (1.1.)$$

Rješenje: Ovdje su ulazne veličine u algoritam $\boldsymbol{n, x_0, x_1, x_2, ..., x_n}$, a izlazna veličina $\boldsymbol{y}$. Razvijanjem, suma (1.1.) se može zapisati u obliku

$$\boldsymbol{y = x_0 - x_1 + x_2 - ...}$$

Ovo izračunavanje sume je riješeno u algoritmu sa slike 1.10. Za ovaj algoritam se može reći da se radi o petlji sa izlazom na vrhu. U predstavljenom algoritmu se



Slika 1.10. Promjenljiva struktura

operacija dodjele znaka **Z** mijenja naizmjenično (1 ili -1).
Prvo je to operacija oduzimanja (**Z**= -1), a zatim operacija sabiranja (**Z** = 1), pa opet oduzimanja itd.

Prema tome, u ovom algoritamskom koraku zakon obrade se mijenja u toku izvršavanja algoritma, naizmjenično sabiranje i oduzimanje. Pored toga, u istom algoritamskom koraku dolazi do promjena vrijednosti promjenljivih, koje učestvuju u obradi, a to su redom promjenljive $\boldsymbol{x_0, x_1, x_2, ... , x_n}$.

Ciklusi u kojima postoje algoritamski koraci sa ovakvim vrstama promjena, u toku izvršavanja algoritma, su promjenljive ciklične strukture.

Algoritan prikazan na slici 1.10. se smatra ne struktuiranim algoritmom, jer imamo dvije petlje, koje se međusobno preklapaju, a nije jedna unutar druge. Zbog toga treba izbjegavati pravljenje algoritma, koji nije struktuiran, jer je za njega teško napisati program.

1.3.3. Pisanje i unošenje programa u računar

Algoritam zapisan u obliku blok-šeme ne može biti prihvaćen i izvršen od strane računara. Da bi algoritam bio prihvaćen i izvršen od strane računara mora biti zapisan na način, koji obezbjeđuje određen nivo detaljizacije algoritma, što ne mora biti u blok-šemi algoritma.

Algoritam zapisan tako da je prihvatljiv od strane računara zove se program, a proces pisanja programa zove se kodiranje (programiranje u užem smislu).

Da bi program mogao da se izvrši na računaru mora biti zapisan na mašinskom jeziku. Međutim, pisanje programa na mašinskom jeziku predstavlja takvu detaljizaciju algoritma, da je to vrlo težak posao za čovjeka. Pored toga, pisanje programa na mašinskom jeziku zahtijeva poznavanje konstruktivnih osobina računara, što takođe onemogućuje da se veći broj ljudi obuči za ovaj posao.

Da bi se omogućilo efikasnije pisanje programa počeli su se stvarati posebni jezici za komunikaciju između čovjeka i računara. Tako su razvijeni simbolički i programski jezici. Programi zapisani na ovim jezicima mogu se pomoću posebnih programa za prevođenje tj. prevodioca, prevesti na mašinski jezik i zatim izvršiti na računaru.

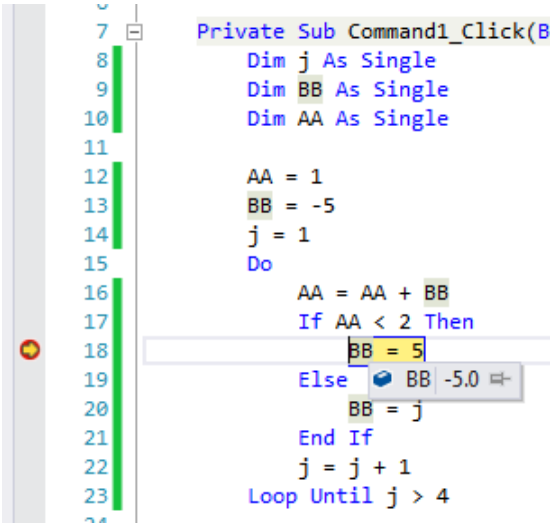
Viši programski jezici (3 generacije i nadalje) su vještački jezici konstruisani za pisanje progama, pri čemu, po pravilu nije potrebno poznavanje konstruktivnih osobina računara. Prema tome, da bi jedan zadatak riješili na računaru moramo prije svega sastaviti algoritam, koji opisuje rješenje zadatog problema.

1.3.4. Testiranje programa i ispravka greške

Kada se program napiše i prevede potrebno je uraditi njegovo testiranje. Dokaziti da program obavlja zadatak ili rješava problem, koji je postavljen nije nimalo lak zadatak. Zato se u toku samog pisanja programa i nakon završetka programa pristupa testiranju programa. Izuzev za veoma jednostavne programe, testiranjem se ne može dokazati 100% ispravnost programa. Testiranjem se pokušavaju pronaći greške u programu. Bez obzira koliko se grešaka pronađe, uvijek postoji mogućnost da ih ima još. I pored toga što testiranje ne garantuje odsustvo grešaka, testirani programi su bolji nego ne testirani. Cilj testiranja je da se pronađu okolnosti pod kojima program daje pogrešne rezultate ili zašto se uopšte ne dobija nikakav rezultat, kada se program pokrene. Ako ne možemo da pronađemo takve okolnosti, tada možemo imati razumno vjerovanje da program ispravno funkcioniše ili da smo loše uradili testiranje. Pravljenje dobrog test plana, kojim se potvrđuje ispravnost programa, je veoma težak zadatak. Kada programi imaju hiljade linija koda, teško je testirati svaku moguću putanju pri njegovom izvršavanju, kako bi se provjerilo da program ispravno radi u svim mogućim slučajevima.

Greške u programu mogu da budu sintaksne i logičke. Sintaksne greške najčešće nastaju prilikom pisanja naredbi u programu. U programskom jeziku *Visual Basic* 2013, to su naredbe koje se pišu u kodu programa. Najčešće greške su izostavljanje nekog slova ili da se neko slovo otkuca dva puta. Kada program ima sintaksnu grešku, on ne može biti preveden u izvršnu verziju. Prilikom prevođenja se obavezno pojavljuje poruka sa greškom u kodnoj liniji u kojoj postoji ova

greška. Ove greške se puno lakše otkrivaju od logičkih grešaka. Logičke greške najčešće nastaju zbog pogrešno napravljenog algoritma. Razlog je obično pogrešan izbor promjenjivih, tipa podataka, grananja, petlji, funkcija i objekata u programu. Kada u programu postoji logička greška tada program ne može da se završi do kraja ili se na kraju dobije rezultat, koji nije dobar. Posebno su opasne ligičke greške, koje dovode do pojave beskonačnih petlji, pri korišćenju *WHILE* i *UNTIL* petlji. Kada program uđe u beskonačnu petlju, on se ne može završiti, već se može samo nasilno prekinuti njegovo izvršenje, CTRL+ALT+DELETE pritiskom kombinacije tipki. Prilikom nasilnog zaustavljanja izvršenja programa može doći do trajnog gubitka napisanog programa, ako on prethodno prije pokretanja nije spašen. Da bi se otklonile logičke greške, prvo treba dobro razmotriti napisani algoritam. A upravo jedna od velikih grešaka programera je da su počeli da pišu program, a da nisu prije toga napravili algoritam. Drugi način otklanjanja logičkih grešaka je da se u programu postave kontrolne tačke. Primjer koda programa napisanog u *Visual Besic*-u 2013, u kome je postavljena kontrolna tačka na liniju 18, prikazan je na slici 1.11.



Slika 1.11. Kontrolne tačke

Kontolne tačke se postavljaju u kodu programa, tako što se mišem klikne lijevo do radne površine za kucanje koda. Nakon toga se lijevo od kodne linije pojavi crveni krug, a pozadina izabrane kodne linije postaje crvena. U jednom programu može se postaviti kontrolna tačka na svaku kodnu liniju.

Kada se program pokrene na dugme **Start**, on se izvršava samo do kontrolne tačke. Dalje izvšenje programa je moguće klikom na dugme **Continue** (nastalo od dugmeta **Start**).

Kada se program zaustavi na kontrolnoj tački, mogu se provjeravati trenutne vrijednosti svih promjenljivih, koje se nalaze u kodu programa. Ova jednostavna provjera se vrši postavljanjem kursora miša na željenu promjenljivu u kodu programa, nakon čega se odmah prikaže trenutna vrijednost te promjenljive (u primjeru na slici 1.11. prikazana je vrijednost promjenljive BB). Na ovaj način se najlakše dolazi do otkrivanja grešaka, a zatim i ispravljanja grešaka u kodu programa.

Postoji nekoliko test strategija, koje će povećati šansu za pronalaženje grešaka (bagova) u programima, a najbitnije su:

- testiranje "bijele kutije",
- testiranje "crne kutije".

Testiranje "bijele kutije"[5] može da obavlja ili programer, koji je napisao program ili neko drugi, kome je specijalnost testiranje softvera. Testiranje se naziva "bijela kutija", zato što je onom ko testira poznat algoritam ili programski kod. Pošto osoba koja testira program može da vidi kod programa, onda je u stanju da provjeri svaki dio programa, tako što će testirati program za razne vrijednosti ulaznih podatka. Za testiranje programa pored uobičajenih biraju se i takozvane granične vrijednosoti. Granične vrijednosti su one vrijednosti podataka kod kojih se mogu očekivati pogrešni rezultati (nula, negativni brojevi, decimalni brojevi, ekstremne vrednosti itd.)

Testiranje "crne kutije" se tako naziva jer je programski kod ili algoritam nevidljiv, osobi koja vrši tetsiranje. Test se sprovodi tako što se programu (crnoj kutiji) daju različiti podaci i posmatra ponašanje programa. Da bi se postigao efekat testiranja, kod ove vrste testova se test podaci pripremaju prije izrade programa, imajući na umu samo programske zahtjeve date u specifikaciji programa. Za ovu vrstu testa se obično planiraju tri vrste testnih podataka:

- ispravni podaci,
- neispravni podaci (recimo ako umesto cifara damo slova),
- granične vrijednosti podataka.

Softver može biti testiran bilo kojom od predhodne dvije metode. Uobičajeno je da se nakon testiranja, progrem prije puštanja u prodaju, da na upotrebu ograničenom broju korisnika (obično programera koji se zovu testeri) u istoj kompaniji. Ova metoda testiranja se naziva Alfa testiranje. Oni simulirajući krajnjeg korisnika, koristeći program pronalaze eventualne greške, koje su zaostale posle testiranja, a takođe mogu da daju i sugestije za unapređenje programa.

Nakon alfa testiranja uobičajena je procedura da se softver da na upotrebu ograničenom broju spoljašnjih korisnika i da se od njih dobije odgovor. Odgovor može da sadrži pronađene greske, ali takođe sugestije za funkcionalno, estetsko ili neko drugo unapređenje softvera. Ova metoda testiranja se naziva Beta testiranje.

Pravljene izvršne verzije programa

Kada se program u potpunosti istestira, potrebno je napraviti izvršnu verziju programa. U verziji programa *Visual Basic* 6.0 izvršna verzija programa se pravila posebno preko opcije ***Make imeprograma.exe*** u glavnom meniju *File*. Na ovaj način se od razvojne verzije programa, koja ima ekstenziju "*.VBP"*, pravi izvršna verzija programa, koja ima ekstenziju "*.EXE"*. U programu *Visual Basic* 2013

[5] Dr Milan Popović, *Osnove programiranja*, BSP, 2007, str.75 do 77.

izvršna verzija program se pravi automatski, kada se program pokrene preko ikonice **Start**. Tada program sam izgeneriše direktorijum **bin**, unutar njega direktorijum **Debug**, a unutar njega se izgeneriše fajl, koji nosi ime projekta, koje smo dali na početku pravljena programa i ima ektenziju "*.EXE"*. Izvršna verzija programa se sada pokreće samo klikom na ovaj fajl i korisnik tada ne vidi kod programa. Korisnik bi trebalo da radi samo sa izvršnim verzijama programa. Ako korisnik radi sa razvojnom verzijom programa tada vrlo lako može da pokvari program pomjeranjem ili brisanjem samo jednog slova u kodu programa. Sa druge strane pisac programa štiti svoj autorski rad isporučivanjem korisniku samo izvršne verzije programa.

1.3.5. Implementacija programa i obuka korisnika

Tek kada se program u potpunosti istestira, on se daje korisniku na korišćenje. Prije nego što se program da korisniku na korišćenje, svaki programer bi trebao da napravi detaljno uputstvo za korišćenje programa. Uputstvo treba da sadrži slike svih ekrana, koji se mogu pojaviti u toku izvršenja programa, a za svaki objekat na slici treba opisati njegove funkcije, tip i raspon ulaznih (izlaznih) podataka. Uputstvo može biti napravljeno u papirnoj ili elektronskoj formi, ali najbolje i u jednoj i drugoj formi. Pored uputstva programer mora da korisniku pruži obuku u korićenju isporučenog programa.

Razvijanje programa[6] za tržište je djelatnost slična proizvodnji svih drugih vrsta proizvoda. Kupac obično očekuje slične uslove kupoprodaje, kao u slučajevima kupovine uobičajenih tehničkih proizvoda. Međutim, program je veoma specifičan proizvod i njegova zaštita nije nimalo jednostavna. Pravni aspekti proizvodnje i korišćenja programa predstavljaju u opštem slučaju delikatnu, prilikom sporova komplikovanu i ponekad kontroverznu oblast. Klasična prodaja programa je dosta rijetka pojava. Pod "klasičnom prodajom" programa podrazumijevamo u stvari kupovinu programa i prava na njegovu preprodaju. Drugim riječima kupovina programa je "klasična kupovina" onda kada je slična kupovini nekretnina ili automobila. Tada kupac ima potpuno pravo raspolaganja svojim vlasništvom i može zatim da ga po volji proda, pokloni, ili uništi. Kupovinu programa praktikuju državne organizacije i/ili kompanije (vlada, vojska, policija, elektrane, banke, itd.) gdje program može bitno da utiče na sigurnost rada ili poslovanja, a za njih je jeftinije da kupe programe nego da ih sami razvijaju.

U pravnim državama piratsko kopiranje programa je jednako težak prekršaj kao i povreda patentnih prava ili piratsko kopiranje i distribucija knjiga, muzičkih i video diskova. Ako bi neko neovlašćeno lice uzelo na primjer neku knjigu i bez

[6] Dr Jozo J. Dujmović, *Programski jezici i metode programiranja*, Akademska misao, Beograd, 2003, str 3.98 do 3.104.

odobrenja izdavača i autora počelo da tu knjigu samostalno kopira i kopije stavlja u prodaju, onda bi to bio prekršaj i pravno i moralno. Treba razumjeti da zakon, koji štiti izdavača i autora štiti u stvari najviši društveni interes. Ako bi proizvođaču knjige, ili programa, njegov proizvod bio nekažnjeno piratski ukraden, onda bi kao direktna posljedica toga opao interes za proizvodnju tako riskantnog proizvoda. Društvo bi neminovno bilo suočeno sa usporenim razvojem ili nazadovanjem. To je razlog što neovlašćeno korišćenje softvera nije samo pravno nedopustiv akt, već predstavlja i ozbiljno kršenje profesionalnog morala jer potkopava temelje zdravih odnosa u okviru programerske profesije.

1.3.6. Održavanje i nadogradnja programa

Šaljiva izreka kaže da „svi programi sadrže greške dok se ne dokaže suprotno, što je inače nemoguće“. Imajući u vidu da „u svakoj šali ima i po malo istine i šale“, dolazimo do toga da je veoma teško pri razvoju programa pružiti bilo kakvu garanciju. Tačnije proizvođači programa umjesto garancije za svoje proizvode najčešće daju izjavu da ne prihvataju bilo kakvu odgovornost za primjenu i posljedice primjene svog softvera.

Čak i za softvere koji služe za rješavanje nekih matematičkih problema, koji imaju izuzetnu vrijednost i mogu se koristiti sa visokim stepenom pouzdanosti programeri ne daju ovakve izjave. Uvijek postoji mogućnost da se i pored vrlo male vjerovatnoće, desi slučaj da se pri nekim ulaznim parametrima pojavi greška u nekom programu. I ako bi se taj program koristio u proračunu ili u radu nekog veoma osjetljivog uređaja (tipičan primjer su letjelice i nuklearne elektrane), moglo bi doći do fatalnih posljedica, za koje proizvođač softvera ne može prihvatiti odgovornost. Stoga se i daje izjava o ne prihvatanju odgovornosti.

Izjave o ne prihvatanju odgovornosti postale su uobičajena pravna forma i primjenjuje se čak i onda kada to, na prvi pogled, ne izgleda neophodno.

Teško je zamisliti kakve bi to fatalne posljedice mogle da nastupe kao rezultat greške u programu za obradu teksta, a da pri tome autor teksta ili njegov izdavač ne budu krivi, nego da svu krivicu snosi programer, koji je pisao program ili kompanija, koja program distribuira. Naravno, može se dogoditi da zbog greške u programu neki podatak u tekstu neželjeno promjeni vrijednost i da to ima neke neprijatne posljedice, ali normalno bi bilo je da je kriv „onaj ko je koristio nož na naopak način, a ne proizvođač noža“. Poenta je međutim u tome da program za obradu teksta ne može da se izvršava u „apsolutnoj izolaciji“. On upisuje podatke po disku i može da greškom dovede do direktnog ili poslijedičnog gubitka nekih korisničkih programa ili podataka, pa se proizvođač softvera sa razlogom odlučio na sličnu formulaciju kao i autori numeričkih programa.

Imajući u vidu da većina programa radi u okruženju drugih programa i podataka i da se šteta drugim podacima i/ili programima ipak može dogoditi i pored najveće pažnje u programiranju, slijedi da je u ovim slučajevima poželjno koristiti izjavu o

nepriznavanju garancije. Tim prije što su korisnici softvera navikli da programski proizvodi u određenom malom procentu sadrže greške, tako da potpunu garanciju i ne očekuju. Garancija se stoga u većini slučajeva odnosi samo na besplatnu zamjenu medijuma sa programima, koji nisu čitki zbog grešaka na medijumu.

Druga forma garancije koju proizvođači unikatnog ili maloserijskog softvera ponekad nude kupcima, je garancija za interventno održavanje u ograničenom periodu. Ova garancija ima za cilj da obezbjedi da proizvođač softvera u ograničenom periodu (najčešće godina dana) bude u obavezi da po reklamaciji kupca izvrši sitnije popravke i dorade isporučenog programskog proizvoda.

Sa druge strane, imajući u vidu izuzetnu lakoću i brzinu kojom se i veoma veliki programski proizvodi mogu neovlašteno iskopirati, proizvođači programa redovno pokušavaju da svoj proizvod zaštite od toga. Pa pored pravne zaštite putem "*copiryght*" koriste i razne tehničke norme zaštite od neovlašćenog kopiranja njihovih proizvoda. Postoji veliki broj metoda kako se to može postići, a neke od njih se zasnivaju i na specijalnom hardveru. Jedan od oubičajenih načina je da se obezbjedi da određeni program može da radi isključivo na određenom računaru. Ovaj način se sastoji u tome, da se serijski broj procesora (koji se programski može očitati) dostavi proizvođaču softvera, koji zatim taj broj upisuje (najčešće kriptografisano) u program, koji treba da radi na tom računaru. U toku rada programa, program povremeno testira da li je serijski broj procesora isti kao i očekivani serijski broj, pa ako to nije slučaj prestaje sa ispravnim radom. Naravno, u tom slučaju pravni aspekt korišćenja je drugačije koncipiran od uobičajene "*copyright*" zaštite. Sada princip licence za korišćenje softvera nije više „jedna kopija jedan korisnik" (na bilo kom mjestu i na bilo kom računaru), već „jedna kopija, jedan računar". Pri tome broj simultanih korisnika takvog softvera može, ali ne mora, da bude ograničen. Osnovna ideja proizvođača softvera je obično da njegova zarada treba da bude proporcionalna koristi, koju korišćenjem softvera ostvaruje njegov korisnik.

Proizvođači softvera (naročito ako se radi o unikatnom softveru) često korisnicima nude ugovorno održavanje softvera. Pod održavanjem se obično podrazumijevaju popravke i dorade koje se rade na zahtjev korisnika, koji je otkrio grešku ili nedostatak u softveru koji koristi, a za koje ima ugovorno održavanje. Ugovorom o održavanju se moraju precizno specificirati uslovi održavanja. To najčešće može biti:

1) vremenski rok u kome proizvođač softvera treba da otkloni otkrivene greške i nedostatke, ili
2) obaveza da se na zahtjev kupca angažuje određeni stručnjak proizvođača, koji će određeno vrijeme provesti radeći u pravcu poboljšanja datog softvera, ali obično bez obaveze da se ispune neki specifični zahtjevi. Budući da je proizvođač softvera spreman na određene intervencije u softveru na zahtjev kupca, to se ponekad može smatrati kao određena forma garancije.

Primjeri navedene forme održavanja softvera obuhvataju obično intervencije u dva osnovna pravca. Korisnik u datom periodu (npr. u toku godine dana) može da detaljno ispita programski sistem sa mnogo širim spektrom kombinacija ulaznih podataka, od onoga što je mogao da uradi proizvođač prilikom razvoja programskog proizvoda. Tom prilikom postoji mogućnost da korisnik identifikuje takve kombinacije ulaznih parametara, za koje programi daju neispravne rezultate. Obaveza proizvođača se svodi na to da modifikuje program tako da se navedene neispravnosti otklone. Drugi pravac se sastoji u tome, da se modifikuje korisnička forma, odnosno onih dijelova koji definišu tip i formu unosa ulaznih podataka od strane korisnika. Korisnik koji rutinski koristi program duže vremena redovno otkrije sitne nepraktičnosti u načinu komuniciranja, podatke koji se ponavljaju, nepotrebna ograničenja, testove i slično. Tako da ima razloge da pokrene zahtjev za manjim izmjenama programa. Treba naglasiti da pod ovaj vid modifikacije ne spadaju suštinske modifikacije algoritamske prirode, jer bi to iziskivalo pravljenje nove verzije programa.

Druga vrsta ugovora o održavanju softvera, koja se primjenjuje u domenu sistemskog softvera, ne obuhvata interventno održavanje, već periodičnu isporuku novih verzija instaliranog softvera. U takvim slučajevima proizvođač ima ekipu programera, koji permanentno dorađuju i usavršavaju postojeći softver. Kada se broj intervencija u postojećem softveru nakupi, tako da se nova verzije softvera po svojim osobinama u dovoljnoj mjeri razlikuje od predhodne, onda se ta nova verzija automatski distribuira korisnicima, koji imaju ugovor o održavanju softvera.

Gotovo svaki program je doživio neke modifikacije, a razvoj programa ima obično svoju nomenklaturu u vidu oznake v.m koja označava:

- „verziju“ (v) i
- „modifikaciju“ (m) datog programa.

Kada programer razvija prototip programa onda je to „nulta verzija“, koja može imati veliki broj raznih modifikacija dok se ne dobije kompletan proizvod, koji prođe završno testiranje i spreman je za tržište. Takav proizvod se najčešće označava kao verzija 1.0 (skraćeno V1.0).

Veoma je važno da se u toku razvoja softvera vodi uredna i precizna evidencija o modifikacijama, koje su u toku. Vremenom se nagomila veliki broj listinga programa, od kojih mnogi mogu biti zastarjeli, ali se to ne može otkriti ako na njima ne stoji odgovarajuća oznaka. Prema tome, u toku razvoja i testiranja programa potrebno je poslije svake modifikacije programa unijeti na programima izmjenjenu oznaku. Počinje se sa V0.1, pa zatim svaka modifikacija ima naredni broj (V0.2, V0.3, itd.) dok se ne dođe do izlaska na tržište sa verzijom V1.0. Zatim se tokom razvoja softvera distribuiraju verzije V1.1, V1.2, itd. Po pravilu poslije manje od 10 modifikacija prelazi se na verziju V2.0. Podrazumijeva se, da se ovaj proces odvija saglasno sa potrebama tržišta, pa V2.0 treba da obuhvati pored popravki grešaka i proširenja, koja čine značajnu razliku u odnosu na verziju V1.0.

Zatim se isporučuju modifikacije V2.1, V2.2 itd. Neki popularni programi doživjeli su veći broj verzija, ali rijetko više od 10.

Lako je uvidjeti da ne postoji nikakva čvrsta definicija i ujednačeni pogled na to nakon koliko modifikacija, treba da se pojavi nova verzija programa. Tako da označavanje brojeva modifikacija i verzija u priličnoj mjeri zavisi od mjere, ukusa i tradicije, koja je svojstvena nekom proizvođaču programa. Ipak, relativno visoki broj verzija i modifikacija programa predstavlja neku formu priznanja da posmatrani programski proizvod uspješno živi na tržištu i da po svojoj prilici ima razvijenu korisničku bazu.

1.4. RJEŠENI PRIMJERI PRAVLJENJA ALGORITMA

U ovom poglavlju su dati zadaci, za koje je potrebno napraviti grafički algoritam. Za većinu zataka su predloženi grafički algoritmi, koji predstavljaju moguća rješenja.

Zadatak 1.1.

Sastaviti algoritam koji izračunava zbir (Z) dva realna broja A i B tj. Z = A + B.

Rješenje:

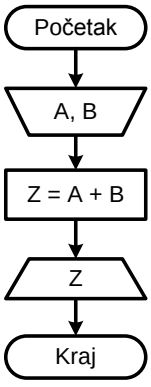
Rješenje ovog algoritma dato je na slici 1.12.

Zadatak 1.2.

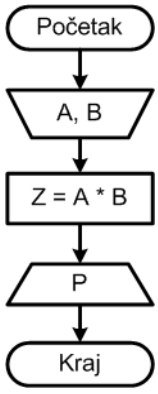
Sastaviti algoritam, koji izračunava proizvod (P) dva realna broja A i B tj. P=A*B.

Rješenje:

Rješenje ovog algoritma dato je na slici 1.13.



Slika 1.12. Zadatak 1.1.



Slika 1.13. Zadatak 1.2.

Zadatak 1.3.
Sastaviti algoritam, koji izračunava vrijednost izraza

$$y=[(x_1 + x_2) * x_3 - x_4] * x_5$$

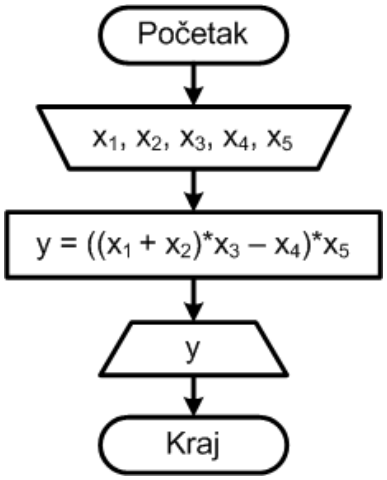
za zadate vrijednosti: x_1, x_2, x_3, x_4 *i* x_5

Rješenje:
Rješenje ovog algoritma dato je na slici 1.14.

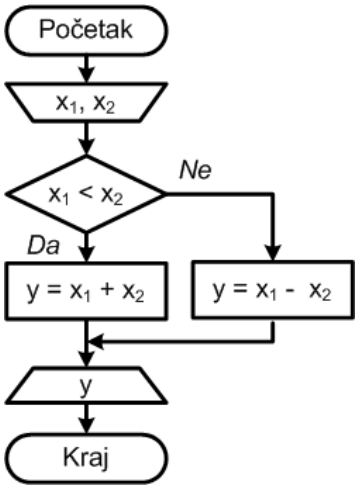
Zadatak 1.4.
Sastaviti algoritam za izračunavanje vrijednosti ***y*** po formuli

$y = x_1 + x_2$ *ako je* $x_1 < x_2$
$y = x_1 - x_2$ *ako je* $x_1 \geq x_2$.

Rješenje:
Rješenje ovog algoritma dato je na slici 1.15.



Slika 1.14. Zadatak 1.3.



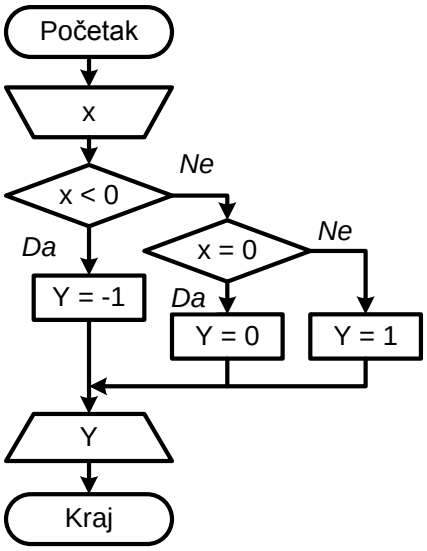
Slika 1.15. Zadatak 1.4.

Zadatak 1.5.
Sastaviti algoritam za određivanje znaka broja ***x*** (funkcija Sign), gdje se vrši izračunavanje vrijednosti funkcije ***y*** po formuli

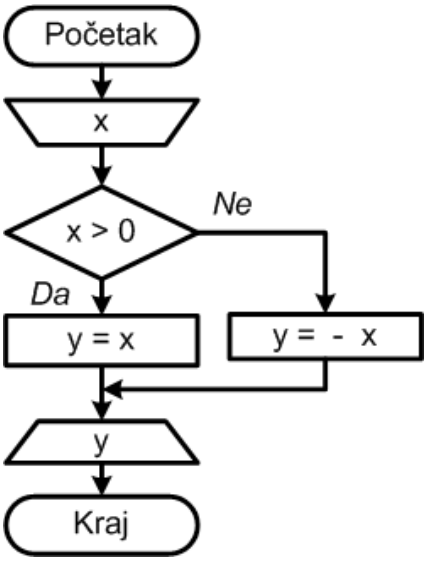
$y = -1$ ako je $x < 0$
$y = 0$ ako je $x = 0$
$y = 1$ ako je $x > 0$.

Rješenje:
Rješenje ovog algoritma dato je na slici 1.16.

Zadatak 1.6.
Sastaviti algoritam za izračunavanje funkcije $y = |x|$.
Rješenje:
Rješenje ovog algoritma dato je na slici 1.17.



Slika 1.16. Zadatak 1.5.



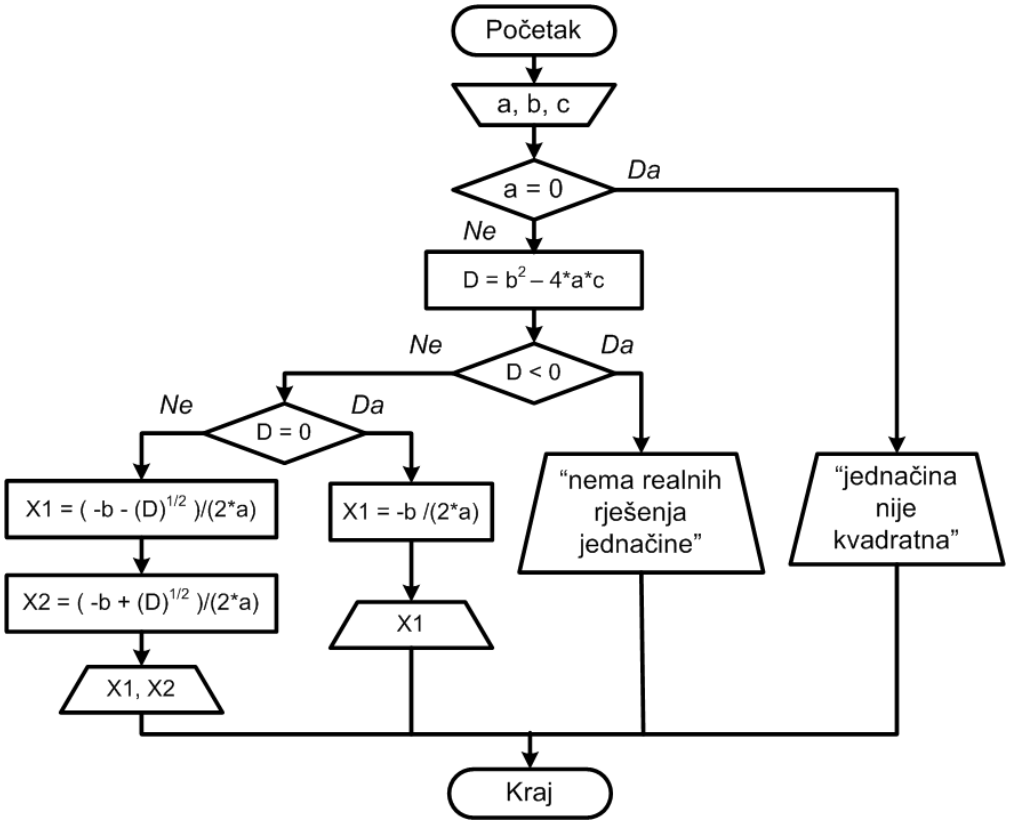
Slika 1.17. Zadatak 1.6.

Zadatak 1.7.
Sastaviti algoritam za izračunavanje funkcije $y = |x| - 1$.
Rješenje je analogno prethodnom zadatku.

Zadatak 1.8.
Sastaviti algoritam za izračunavanje rješenja kvadratne jednačine

$$Y = a * X^2 + b * X + c$$

u skupu realnih brojeva ***(R)***.
Rješenje:
Rješenje ovog algoritma dato je na slici 1.18. Prvo se vrši ispitivanje koeficijenta *a*. Ako ovaj koeficijent ima vrijednost nula ($a = 0$), onda unesena jednačina nije kvadratna već linearna, pa nema ni smisla tražiti njeno rješenje na ovaj način. Ako je jednačina kvadratna, onda se ispituje diskriminanta kvadratne jednačine ($D = b^2 - 4 * a * c$). Ako je diskriminanta manja od nule, onda kvadratna jednačina nema realnih rješenja. Ako je diskriminanta jednaka nuli, onda kvadratna jednačina ima jedno realno rješenje X1. Ako je diskriminanta veća od nule, onda kvadratna jednačina ima dva realna rješenja X1 i X2.



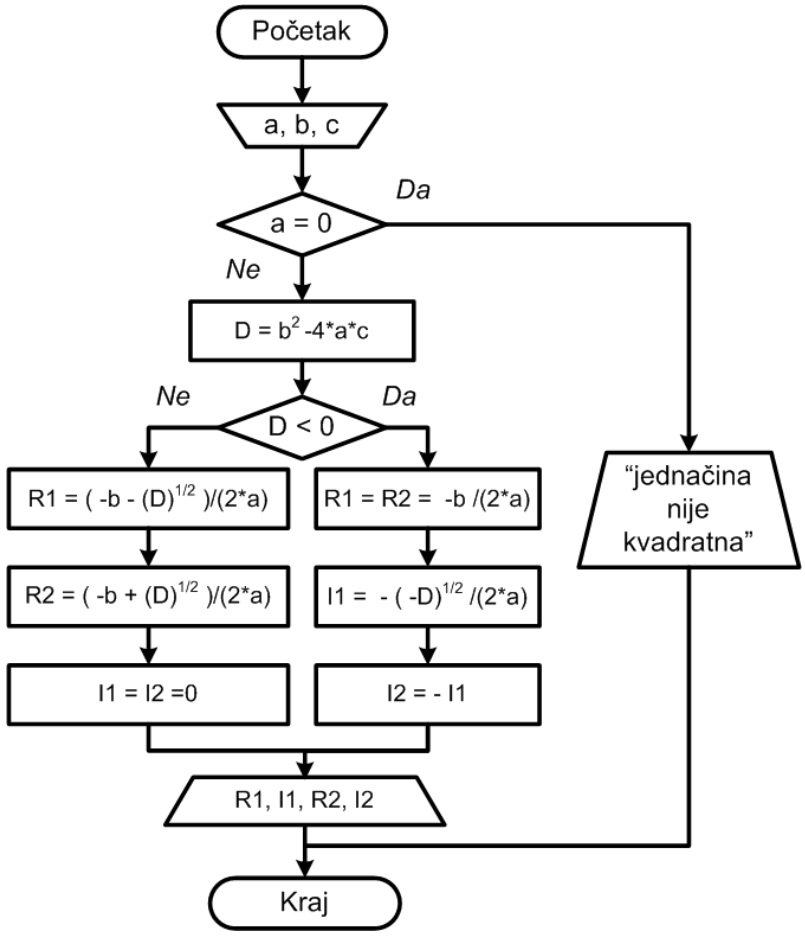
Slika 1.18. Zadatak 1.8.

Zadatak 1.9.
Sastaviti algoritam za izračunavanje rješenja kvadratne jednačine

$Y = a*X^2 + b*X + c$

u skupu realnih brojeva ***(R)*** i u skupu kompleksnih brojeva ***(Z)***.

Rješenje:
Rješenje ovog algoritma dato je na slici 1.19. Na ovoj slici *R1* i *R2* predstavljaju dva realna rješenja kvadratne jednačine iz skupa realnih brojeva ***(R)***, a *I1* i *I2* predstavljaju dva imaginarna rješenja kvadratne jednačine iz skupa kompleksnih brojeva (**Z**). Imaginarna rješenja postoje samo ako je diskriminanta kvadratne jednačine ($D = b^2 - 4 * a * c$) manja od nule. Ako je diskriminanta jednaka nuli, onda kvadratna jednačina ima jedno realno rješenje R1, kao u predhodnom algoritmu, ali zbog preglednosti u ovom algoritmu je izostavljeno to ispitivanje.



Slika 1.19. Zadatak 1.9.

Zadatak 1.10.

Sastaviti algoritam za izračunavanje kvadratnog korena prirodnih brojeva $y = \sqrt{x}$ po *Newton-Raphson* iterativnoj formuli $x_{i+1} = (x_i + x / x_i)/2$,

$i = 0, 1, 2, ...$ gde je $x_0 = (x + 1)/2$. Proces računanja, kroz niz ponavljanja, prekinuti kada se dostigne zadata tačnost e, tako da je $x_i - x_{i+1} < e$.

Rješenje:

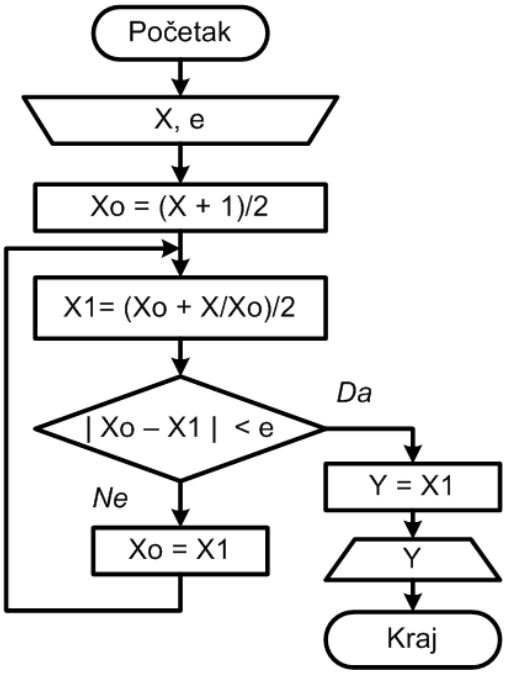
Rješenje ovog algoritma dato je na slici 1.20.

Zadatak 1.11.

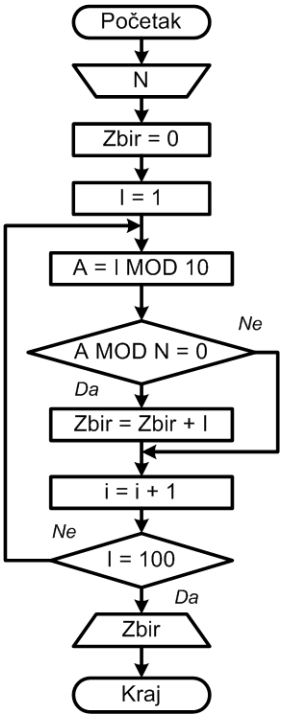
Sastaviti algoritam za unos prirodnog broja N, koji može biti samo od 0 do 9. Zatim izračunati zbir prirodnih brojeva od 1 do 100 čija je zadnja cifra N.

Rješenje:

Rješenje ovog algoritma dato je na slici 1.21.



Slika 1.20. Zadatak 1.10.



Slika 1.21. Zadatak 1.11.

Zadatak 1.12.

Sastaviti algoritam za unos niza brojeva A_i, koji ima N elemenata, i = 1...N, pri čemu N mora biti prirodan broj veći od jedan (N = 2, 3, 4, ...). Zatim pronaći najveću vrijednost (Max) unesenog niza A_i.

Rješenje:

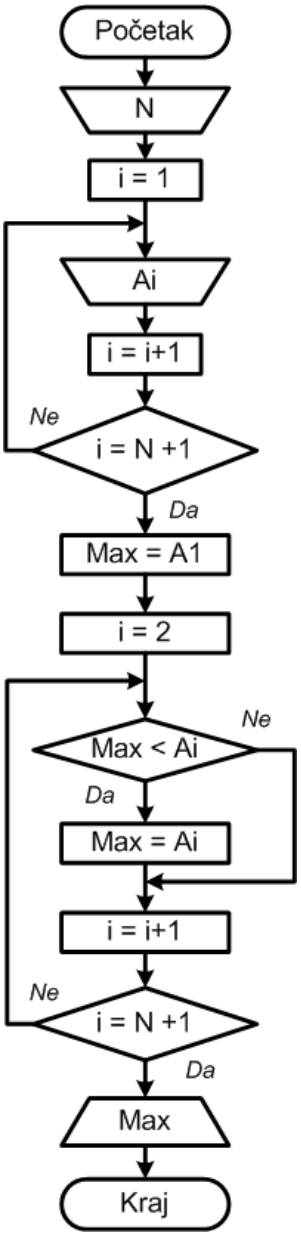
Rješenje algoritma dato je na slici 1.22. Prvo se preko petlje unese N članova niza. Zatim se pretpostavi da prvi član niza ima najveću vrijednost (Max = A1). Poslije se uzima drugi član niza i poredi sa pretpostavljenom najvećom vrijednošću. Ako drugi član niza ima veću vrijednost od maksimalne, onda se pretpostavlja da on ima maksimalnu vrijednost (Max = A2). Ako nije, onda ostaje prethodna vrednost za Max (Max=A1). Ovo je već delimično dokazana činjenica. Ovaj postupak poređenja se ponavlja u petlji, dok se ne dođe do kraja niza. Kada se dođe do kraja niza, najveća vrednost će biti prekopirana u Max i više neće biti pretpostavljena već dokazana činjenica.

Zadatak 1.13.

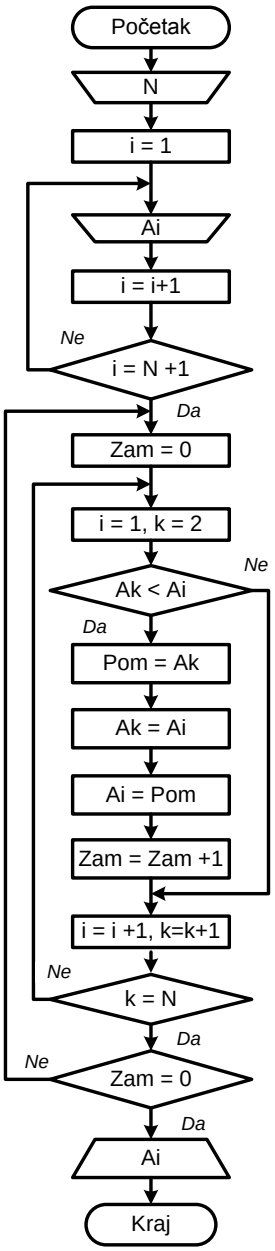
Sastaviti algoritam za unos niza A_i, koji ima N elemenata, i = 1...N, pri čemu N mora biti prirodan broj veći od jedan (N = 2, 3, 4, ...). Zatim sastaviti algoritam za ispisivanje niza u rastućem (neopadajućem) poretku.

Rješenje:

Rješenje ovog algoritma dato je na slici 1.23. Prvo se preko petlje unese *N* članova niza. U algoritmu zatim slijede dvije petlje spoljna i unutrašnja.



Slika 1.22. Zadatak 1.12.



Slika 1.23. Zadatak 1.13.

Prvo se ulazi u spoljnu petlju. Promjenljivoj (*Zam*), koja služi za brojanje zamjena mjesta u nizu, se na početku dodjeli vrijednost nula (*Zam* = 0), nakon svakog početka ponavljanja petlje. Na ovaj način se stvara mogućnost izlaska iz petlje, odnosno sprečava nastajanje beskonačne petlje. Zatim se ulazi u unutrašnju petlju. U njoj se prvo uzima prvi i drugi član niza i vrši se njihovo poređenje. Ako je prvi element veći od narednog, radi se zamjena mjesta ta dva člana niza i uvećava se vrijednost promjenljive *Zam* (*Zam* = *Zam* + 1). Ovaj postupak se ponavlja u unutrašnjoj petlji, dok se ne dođe do kraja niza. Kada se dođe do kraja niza izlazi se iz unutrašnje petlje. Onda se provjerava da li je tokom poređenja susjednih članova niza došlo do zamjene mjesta. Ako promjenljiva *Zam* ima vrijednost veću od nule, to znači da se u unutrašnjoj petlji desila bar jedna zamjena mjesta. Prva petlja vrši povratak na početak algoritma da se promjenljivoj *Zam* ponovo dodjeli vrijednost nula. Tada se postupak poređena susjednih članova niza ponavlja ponovo od prvog člana niza u unutrašnjoj petlji. Ako promjenljiva *Zam* ima vrijednost nula (*Zam* = 0), tada je završeno sortiranje niza prema rastućem poredku i izlazi se iz spoljne petlje. Na kraju se prikazuje sortiran niz prema rasturem redoslijedu.

Zadatak 1.14.

Sastaviti algoritam za unos niza A_i koji ima N elemenata, i = 1...N, pri čemu N mora biti prirodan broj veći od jedan (N = 2, 3, 4, ...). Zatim sastaviti algoritam za ispisivanje niza u nerastućem (opadajućem) poretku.

Rješenje ovog zadatka je analogno prethodnom zadatku, samo se promjeni uslov za zamjenu, *Ak>Ai*.

Zadatak 1.15.

Sastaviti algoritam za unos niza A_i koji ima N elemenata, *i = 1...N*, pri čemu N mora biti prirodan broj veći od jedan (N=2,3,4,...). Pretražiti uneseni niz i ispisati broja članova niza djeljivih sa pet (5). Ako nijedan član niza nije djeljiv sa brojem pet (5), onda ova vrijednost treba da bude nula (0).

Rješenje:

Rješenje ovog algoritma dato je na slici 1.24.

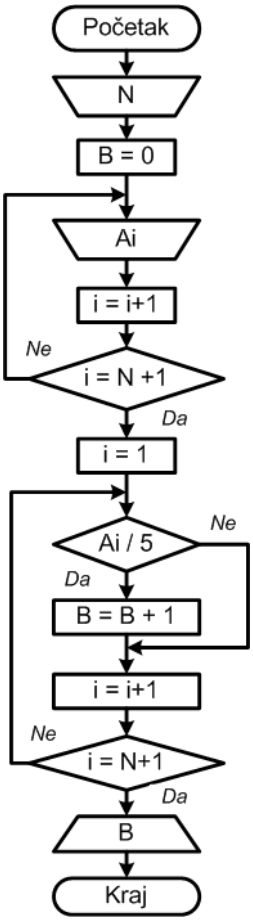
Zadatak 1.16.

Sastaviti algoritam za izračunavanje faktorijel funkcije (F = N**!**), za uneseni prirodan broj N, koji je veći od nule (N = 1, 2, 3, ...). Faktorjel funkcija se računa prema sljedećem izrazu

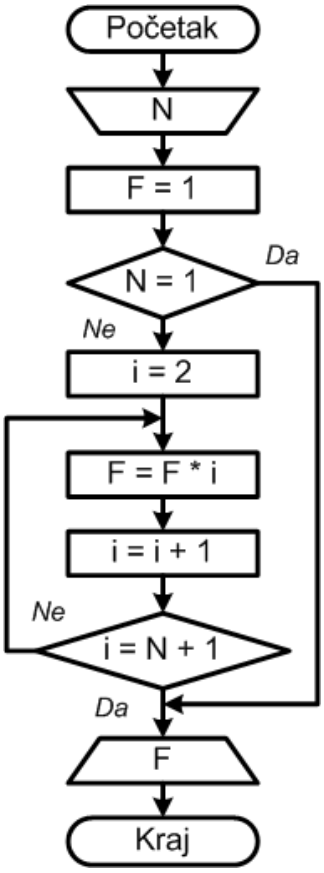
$$F = 1*2*3* \ldots *N$$

Rješenje:

Rješenje ovog algoritma dato je na slici 1.25.



Slika 1.24. Zadatak 1.15.



Slika 1.25. Zadatak 1.16.

Zadatak 1.17.

Ako je logički izraz L datog algoritma prikazanom na slici 1.26. tačan (L=TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

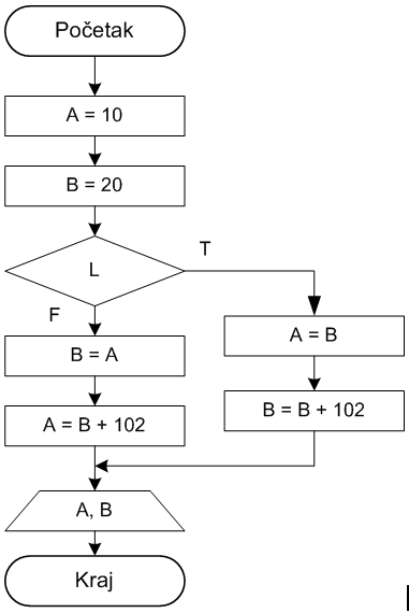
1) Ciklus A = 10, B = 20, Uslov zadovoljen (T), A = 20, B = 122
   Na kraju: A = 20, B = 122

Zadatak 1.18.

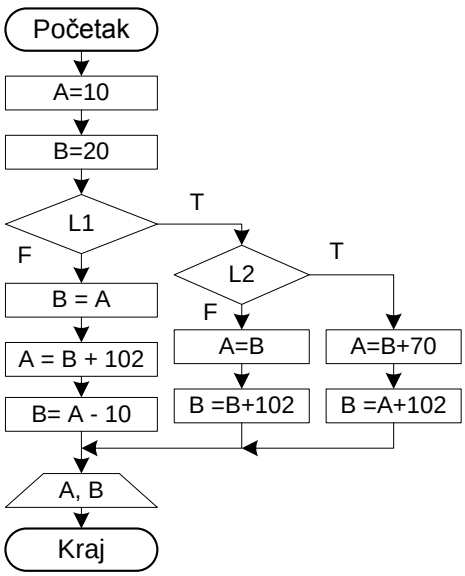
Ako je logički izraz L datog algoritma prikazanom na slici 1.26. netačan (L=FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, Uslov nije zadovoljen (F), B = 10, A = 112
   Na kraju: A = 112, B = 10



Slika 1.26. Zadatak 1.17. i 1.18.



Slika 1.27. Zadatak 1.19., 1.20 i 1.21.

Zadatak 1.19.

Ako je logički izraz L1 datog algoritma prikazanom na slici 1.27. tačan (L1=TRUE) i logički izraz L2 datog algoritma nije tačan (L2=FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 zadovoljen (T), uslov L2 nije zadovoljen (F), A = 20, B = 122
   Na kraju: A = 20, B= 122

Zadatak 1.20.

Ako je logički izraz L1 datog algoritma prikazanog na slici 1.27. tačan (L1=TRUE) i logički izraz L2 datog algoritma je tačan (L2= TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 zadovoljen (T), uslov L2 zadovoljen (T), A = 90, B = 192
   Na kraju: A = 90, B = 192

Zadatak 1.21.

Ako logički izraz L1 algoritma prikazanog na slici 1.27. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma nije tačan (L2=FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 nije zadovoljen (F), uslov L2 nije više uopšte bitan, B = 10, A = 112, B = 102
   Na kraju: A = 112, B = 102

Zadatak 1.22.

Ako logički izraz L1 algoritma prikazanog na slici 1.28. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma je tačan (L2= TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

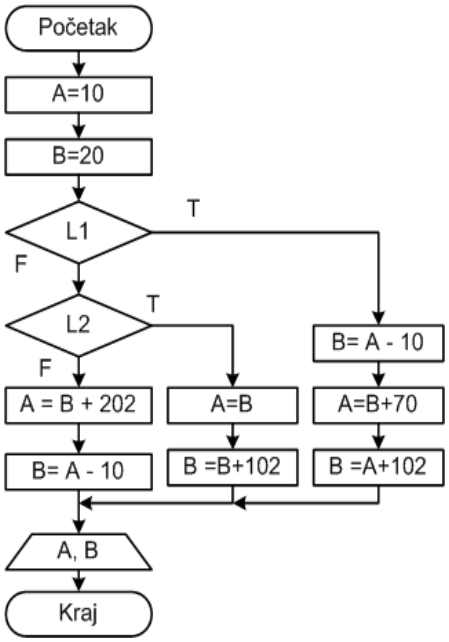
1) Ciklus A= 10, B= 20, uslov L1 nije zadovoljen (F), uslov L2 zadovoljen (T), A = 20, B = 122
   Na kraju: A = 20, B = 122

Zadatak 1.23.

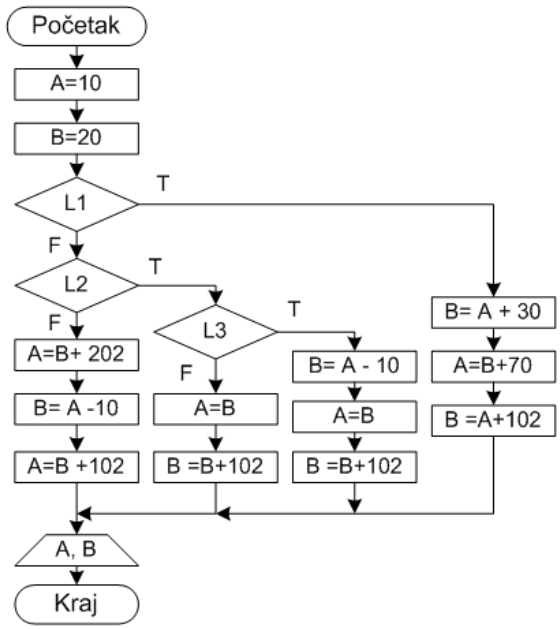
Ako logički izraz L1 algoritma prikazanog na slici 1.28. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma nije tačan (L2= FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 nije zadovoljen (F), uslov L2 nije zadovoljen (F), A = 222, B = 212
   Na kraju: A = 222, B = 212



Slika 1.28. Zadatak 1.22., 1.23. i 1.24.



Slika 1.29. Zadatak 1.25., 1.26., 1.27 i 1.28.

Zadatak 1.24.

Ako je logički izraz L1 algoritma prikazanog na slici 1.28. tačan (L1= TRUE) i logički izraz L2 datog algoritma nije tačan (L2=FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 je zadovoljen (F), uslov L2 nije više uopšte bitan, B = 0, A = 70, B = 172
   Na kraju: A = 70, B = 172

Zadatak 1.25.

Ako logički izraz L1 algoritma prikazanog na slici 1.29. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma je tačan (L2= TRUE) i logički izraz L3 datog algoritma je tačan (L3= TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 nije zadovoljen (F), uslov L2 je zadovoljen (T), uslov L3 je zadovoljen (T), B = 0, A = 0, B = 102
   Na kraju: A = 0, B = 102

Zadatak 1.26.

Ako logički izraz L1 algoritma prikazanog na slici 1.29. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma nije tačan (L2= FALSE) i logički izraz L3 datog algoritma je tačan (L3= TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 nije zadovoljen (F), uslov L2 nije zadovoljen (F), uslov L3 nije više uopšte bitan, A = 222, B = 212, A = 314
   Na kraju: A = 314, B = 212

Zadatak 1.27.

Ako logički izraz L1 algoritma prikazanog na slici 1.29. je tačan (L1= TRUE) i logički izraz L2 datog algoritma nije tačan (L2= FALSE) i logički izraz L3 datog algoritma je tačan (L3= TRUE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 je zadovoljen (T), uslov L2 i L3 nisu više uopšte bitni, B = 40, A = 110, B = 212
   Na kraju: A = 110, B = 212

Zadatak 1.28.

Ako logički izraz L1 algoritma prikazanog na slici 1.29. nije tačan (L1= FALSE) i logički izraz L2 datog algoritma je tačan (L2= TRUE) i logički izraz L3 datog algoritma je tačan (L3= FALSE), koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 10, B = 20, uslov L1 nije zadovoljen (F), uslov L2 je zadovoljen (T), uslov L3 nije zadovoljen (F), A = 20, B = 122

Na kraju: A = 20, B = 122

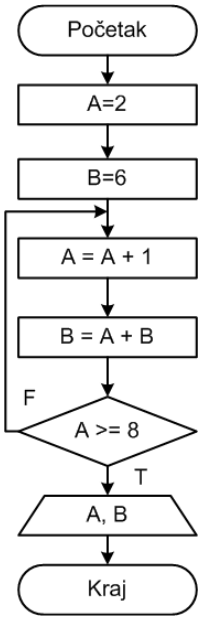
Zadatak 1.29.

Koliko ciklusa će se u algoritmu prikazanom na slici 1.30. izvršiti blok (B=A+B) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

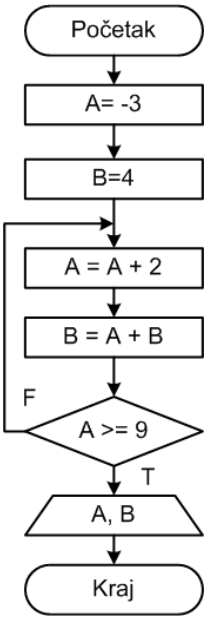
Rješenje:

1) Ciklus A = 3, B = 9, Uslov nije zadovoljen (F)
2) Ciklus A = 4, B = 13, Uslov nije zadovoljen (F)
3) Ciklus A = 5, B = 18, Uslov nije zadovoljen (F)
4) Ciklus A = 6, B = 24, Uslov nije zadovoljen (F)
5) Ciklus A = 7, B = 31, Uslov nije zadovoljen (F)
6) Ciklus A = 8, B = 39, Uslov zadovoljen (T)

Na kraju: Broj ciklusa 6, A = 8 , B = 39



Slika 1.30. Zadatak 1.29.



Slika 1.31. Zadatak 1.30.

Zadatak 1.30.

Koliko ciklusa (ponavljanja) će se u algoritmu prikazanom na slici 1.31. izvršiti blok (B = A + B) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = -1, B = 3, Uslov nije zadovoljen (F)
2) Ciklus A = 1, B = 4, Uslov nije zadovoljen (F)
3) Ciklus A = 3, B = 7, Uslov nije zadovoljen (F)
4) Ciklus A = 5, B = 12, Uslov nije zadovoljen (F)
5) Ciklus A = 7, B = 19, Uslov nije zadovoljen (F)
6) Ciklus A = 9, B = 28, Uslov zadovoljen (T)

Na kraju: Broj ciklusa 6, A = 9 , B = 28

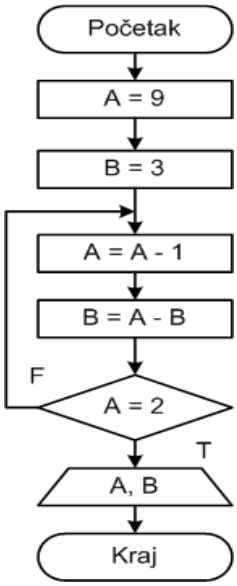
Zadatak 1.31.

Koliko ciklusa (ponavljanja) će se u algoritmu prikazanom na slici 1.32. izvršiti blok (B = A - B) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

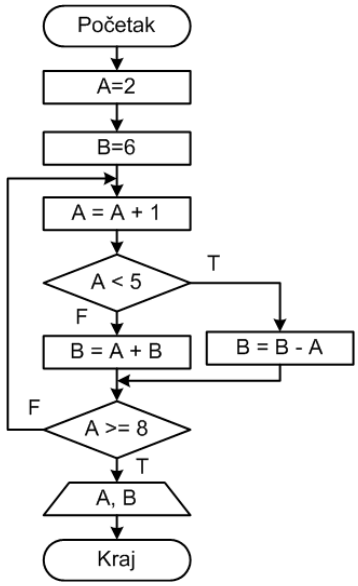
Rješenje:

1) Ciklus A = 8, B = 5, Uslov nije zadovoljen (F)
2) Ciklus A = 7, B = 2, Uslov nije zadovoljen (F)
3) Ciklus A = 6, B = 4, Uslov nije zadovoljen (F)
4) Ciklus A = 5, B = 1, Uslov nije zadovoljen (F)
5) Ciklus A = 4, B = 3, Uslov nije zadovoljen (F)
6) Ciklus A = 3, B = 0, Uslov nije zadovoljen (F)
7) Ciklus A = 2, B = 2, Uslov zadovoljen (T)

Na kraju: Broj ciklusa 7, A = 2 , B= 2



Slika 1.32. Zadatak 1.31.



Slika 1.33. Zadatak 1.32.

Zadatak 1.32.

Koliko ciklusa će se u algoritmu prikazanom na slici 1.33. izvršiti blok (A=A+1) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 3, Uslov1 je zadovoljen (T), B = 3, Uslov2 nije zadovoljen (F)
2) Ciklus A = 4, Uslov1 je zadovoljen (T), B = -1, Uslov2 nije zadovoljen (F)
3) Ciklus A = 5, Uslov1 nije zadovoljen (F), B = 4, Uslov2 nije zadovoljen (F)
4) Ciklus A = 6, Uslov1 nije zadovoljen (F), B= 10, Uslov2 nije zadovoljen (F)
5) Ciklus A = 7, Uslov1 nije zadovoljen (F), B= 17, Uslov2 nije zadovoljen (F)
6) Ciklus A = 8, Uslov1 nije zadovoljen (F), B = 25, Uslov2 zadovoljen (T)

Na kraju: Broj ciklusa 6, A = 8 , B = 25

Zadatak 1.33.

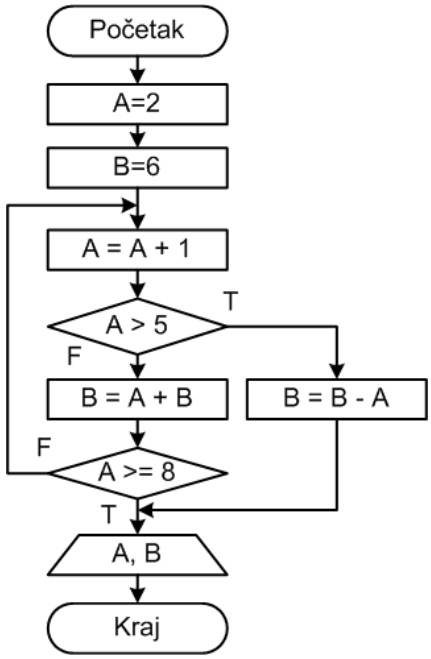
Koliko ciklusa će se u algoritmu prikazanom na slici 1.34. izvršiti blok (A=A+1) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?

Rješenje:

1) Ciklus A = 3, Uslov1 nije zadovoljen (F), B= 9, Uslov2 nije zadovoljen (F)
2) Ciklus A = 4, Uslov1 nije zadovoljen (F), B= 13, Uslov2 nije zadovoljen (F)
3) Ciklus A = 5, Uslov1 nije zadovoljen (F), B= 18, Uslov2 nije zadovoljen (F)
4) Ciklus A = 6, Uslov1 je zadovoljen (T), B= 12

Uslov2 se više neće ni jednom ispitivati i izlazi se iz petlje za ponavljanje.

Na kraju: Broj ciklusa 4, A = 6 , B = 12



Slika 1.34. Zadatak 1.33.

Probajte u programu *Visuel Studio* 2013 da realizujete zadnji algoritam. Vidjećete da vam to neće poći za rukom pomoću jedne *IF THEN ELSE* strukture i jedne *DO WHILE* petlje. To govori da zadnji algoritam nije struktuiran.

Za prikazane ulazne podatke Uslov2 (A >= 8) se neće nikada ostvariti. Za vježbu modifikovati prvi uslov (A > 5) u postojećem zadatku, kako bi postigli da Uslov2 nekada bude zadovoljen.

2. UVOD U *VISUAL STUDIO* 2013

Visual Studio 2013 je objektno orijentisano programski integrisano razvojno okruženje, koje u sebi ima više programski jezika: *Visual Basic, Visual C#, Visual C++, Visual F#.* U ovoj knjizi će najveća pažna biti posvećena programskom jeziku *Visual Basic*. *Visual Studio* 2013 je jedan od najpopularnijih objektno orijentisanih programskih jezika. Napravila ga je kompanija *Microsoft*. Upravo zbog toga veliki broj korisnika *Windows* operativnog sistema, koristi *Visual Basic* za pisanje programa i pravljenje aplikacija. Pogotovo je olakšana dvosmjerna komunikacija sa svim *Microsoft* programskim paketima.

Visual Basic je nastao od programskog jezika *Basic* (*Beginner's Allpurpose Symbolic Introduction Code*). Kod programskog jezika *Basic* proces programiranja se prvenstveno svodio na pisanje koda. Svaki objekat koji se pojavljivao na ekranu, morao se posebno programirati u kodu. Komandno dugme kao jedan od najjednostavnijih objekata se morao svaki put posebno programirati crtanjem svake linije na dugmetu posebno. To je predstavljao veliki napor za programere, jer su programi imali veliki broj kodnih linija, koji su se ponavljale.

Zato su kreatori objektno orijentisanih programskih jezika odlučili da olakšaju posao programerima. Napravljen je određen broj objekata, koji su se do tada najviše koristili u programiranju i koje su sada programeri samo postavljali na radnu površinu za pravljenje programa. Na taj način je proces programiranja značajno ubrzan i olakšan. Takođe se promjenila i sama filozofija programiranja, jer je sada objekat postao centralno mjesto u programiranju. Kod je postao samo onaj dio programa, koji dodatno opisuje objekte i međusobno ih povezuje. Cjelokupni program je podjeljen na mnoštvo djelova, koji se izvršavaju kada korisnik izvrši neku akciju. Na primjer, akcija bi mogla biti klik miša (*Click*) na komandno dugme. U tom slučaju komandno dugme je objekt. Dugme prepoznaje događaj, kada korisnik klikne na njega. Za taj događaj (*Click*) objekta komandno dugme (*CommandButton*) pišemo kod. Ovaj kod će se startovati i izvršiti samo kada korisnik klikne mišem na komadno dugme. Programiranje vođeno događajima je u stvari manji ili veći broj segmenata programa, koje korisnik aktivira svojim akcijama. Svaki objekat ima svoj set osobina. Njih podešavamo u *Properties* prozoru, koji dobijamo kada selektujemo željeni objekat. Takođe, svaki objekat prepoznaje neke događaje. Duplim klikom miša na bilo koji objekat postavljen na radnu površinu (*Form*), otvaramo prozor za pisanje koda. U ovom prozoru pišemo kod programa, koji želimo da se izvrši nad postavljenim objektima. Na slici 2.12. prikazan je prozor za pisanje koda, iznad koga se nalaze dvije liste (*ComboBox*). Lijeva lista daje spisak imena svih objekat, koji postoje na toj radnoj površini. U desnoj listi se nude svi mogući događaji, koji se mogu programirati nad selektovanim objektom. Prvo se bira željeni objekat, a zatim i događaj koji se želi programirati nad njim. Izborom odgovarajućeg događaja, automatski se pojavljuje

procedura događaja u prostoru za pisanje koda. Ime procedure se sastoji od imena događaja i imena objekta.

Activex kontrole su jedna od osobina programskih jezika, koji su integrisani u *Visual Studio* 2013. *Activex* kontrole omogućuju direktno povezivanje programa sa drugim programima iz *Windows* okruženja, ali i mnogim drugim programima, koje nije proizveo *Microsoft*. Ono što je danas posebno aktuelno, je činjenica da se pomoću *Activex* kontrola programi mogu postaviti na internet.

Visual Studio 2013 omogućuje pristup raznim bazama podataka, na više različitih načina. Na taj način se mogu napraviti profesionalne aplikacije, pogotovo one koje se zasnivaju na relacionim bazama podataka.

2.1. PREDNOSTI PROGRAMIRANJA U *VISUAL BASIC*-U

Visual Basic kao objektno orijentisani programski jezik ima više prednosti, koje ga izdvajaju od ostalih programskih jezika. Osnovne prednosti upotrebe programskog jezika *Visual Basic* su:

1) Jednostavnost korišćenja je jedna od osobina, koja ga izdvaja od ostalih programskih jezika.
2) Lako učenje preko velikog broja urađenih primjera programa, koji se nalaze u *Help*-u ovog programa.
3) Veliki broj gotovih i raznolikih objekata za upravljanje.
4) Funkcionalan i raznolik grafički interfejs za unos i prikaz podataka.
5) Sve što se vidi kao standard u *Windows* okruženju, može se programirati u *Visual Basic*-u.
6) Podržava komunikaciju sa svim programima iz *Windows* okruženja, tako da se ne gubi vrijeme na programiranje onoga što *Windows* već sadrži.
7) Programiranje u *Visual Basic*-u pruža mogućnost, da se isti problem riješi na više različitih načina, uz upotrebu različitih objekata. Pri programiranju do izražaja dolazi kreativnost programera.
8) Masovno korišćenje ovog programa u svijetu, dovelo je do toga da se na internetu može naći veliki broj gotovih programa sa kodom, koji se slobodno mogu preuzeti. Na taj način programer samo preuzima kod programa, problema koji je neko prije toga riješio. Na taj način je *Visual Basic*, kao programski jezik svakim danom sve bogatiji.

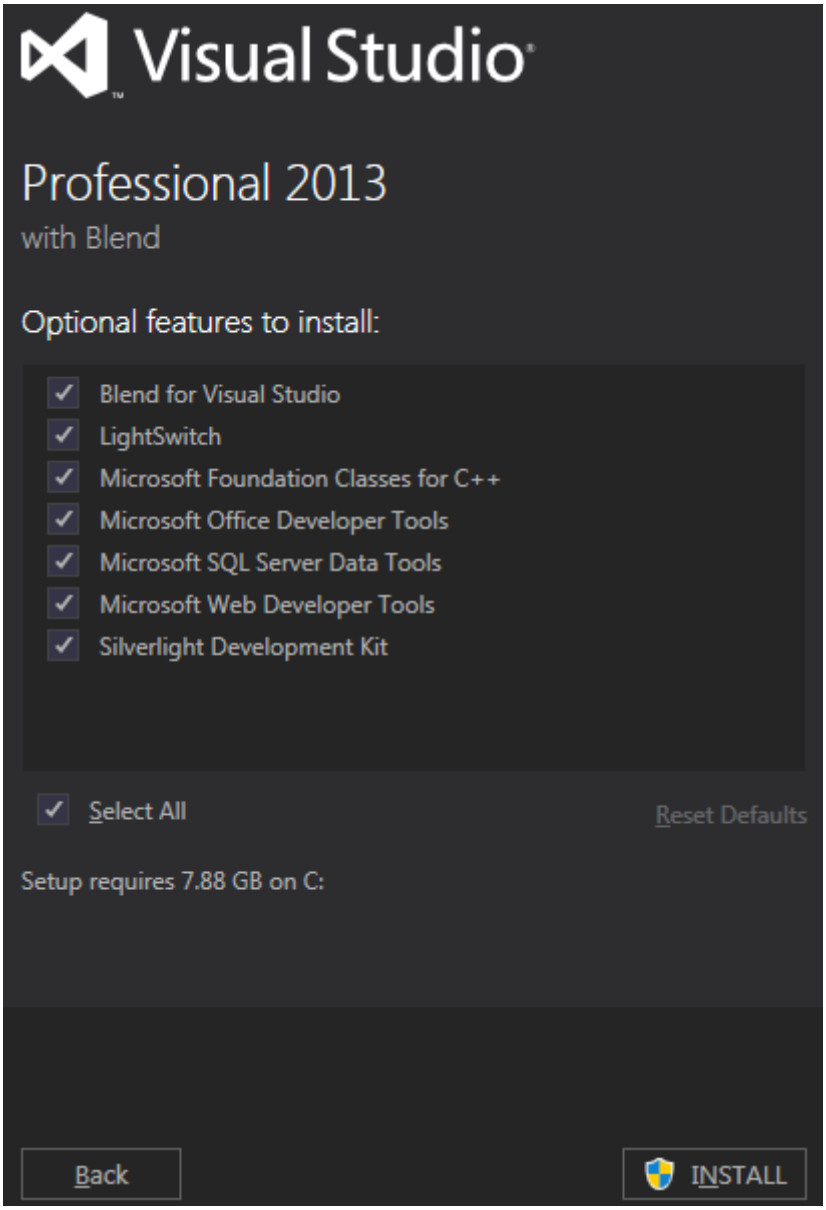
2.2. INSTALIRANJE *VISUAL STUDIO* 2013

Za instaliranje integrisanog razvojnog okruženja *Visual Studio* 2013 postoji instalacioni disk. *Visual Studio* 2013 se može instalirati na računar automatski, kada se ubaci glavni instalacioni disk. Prvo se pojavi ekran prikazan na slici 2.1., na kome se može saznati minimalno potreban memorijski prostor za instaliranje

ovog programa. Na ovom prozoru se bira direktorijum na kome će program biti instaliran. Inicijalno se pojavljuje putanja "**C:\Program Files(x86)\Microsoft Visual Studio 12.0**". Ova putanja se može promjeniti klikom na dugme **„ ... “**. Potrebno je čekirati opciju ***I agre with License Terms and Privacy Policy*** da bi nastavili postupak instaliranja programa. Čekiranjem ove opcije korisnik prihvata sve uslove pod kojima se ovaj program može koristiti, a koje je dao proizvođač. U ovim uslovima se proizvođač između ostalog ograđuje od svake eventualne štete, koja nastane na računaru prilikom rada sa ovim programom.

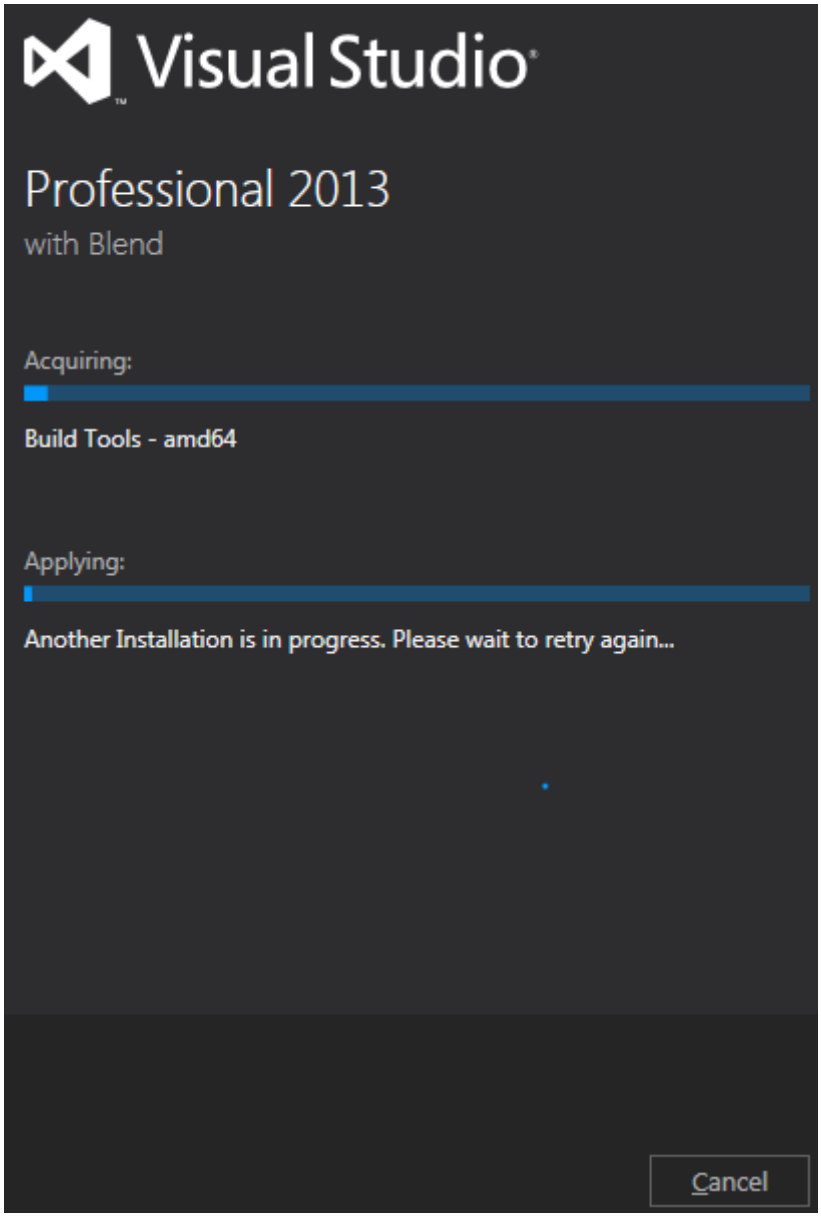
Slika 2.1. Instaliranje Visual Studio 2013 programa

Klikom na dugme ***Next*** prelazi se na naredni prozor ***Optional features to install*** prikazan na slici 2.2. Na ovom prozoru korisnik bira pojedinačno opcije programa, koje želi da instalira ili čekira opciju ***Select All***, da bi instalirali sve opcije koje programski jezik nudi, a što je i najpoželjnije da se uradi.



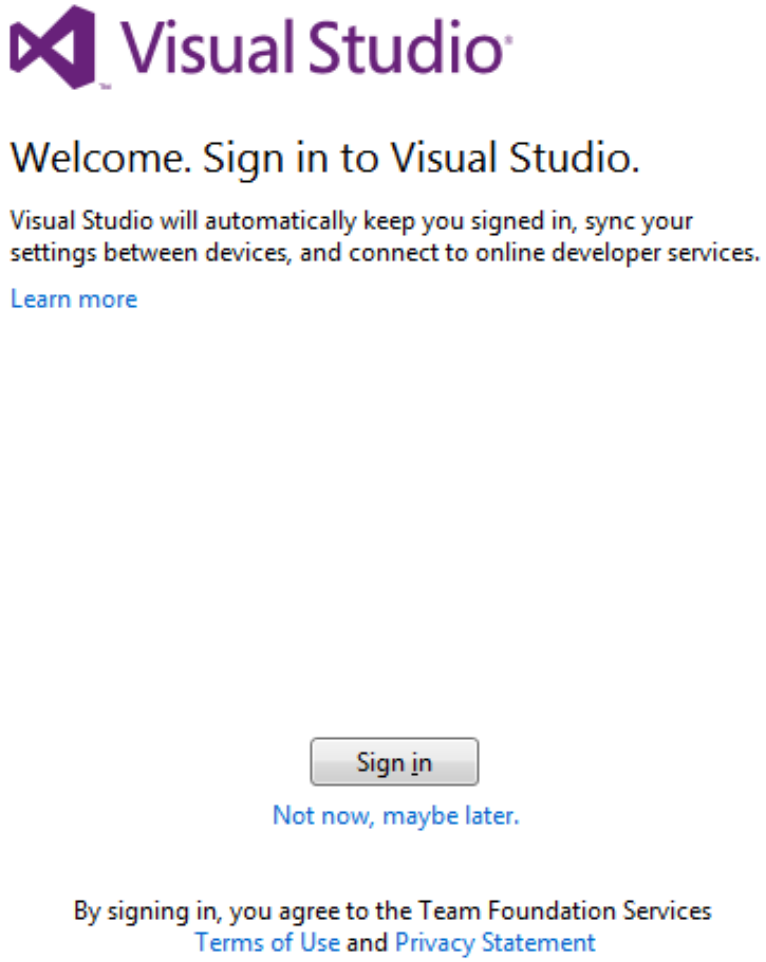
Slika 2.2. Izbor opcija koje se instaliraju

Klikom na dugme ***INSTALL*** prelazi se na prozor prikazan na slici 2.3., za početak instaliranja programa na hard disk računara.



Slika 2.3. Instaliranje glavnog programa

Kada se završi postupak instaliranja svih fajlova glavnog programa, pojavljuje se prozor ***Setup Completed***, a klikom na opciju ***LAUNCH*** prelazi se na prozor prikazan na slici 2.4. Na ovom prozoru se može OnLine prijaviviti kao novi korisnik kod proizvođača progrma klikom na dugme ***Sign in*** ili odložiti prijava klikom na tekst ***Not now, maybe later***. Korisnici koji nisu kupili originalni instalacioni disk ovog programskog jezika preporučuje se da izaberu opciju ***Not now, maybe later***.



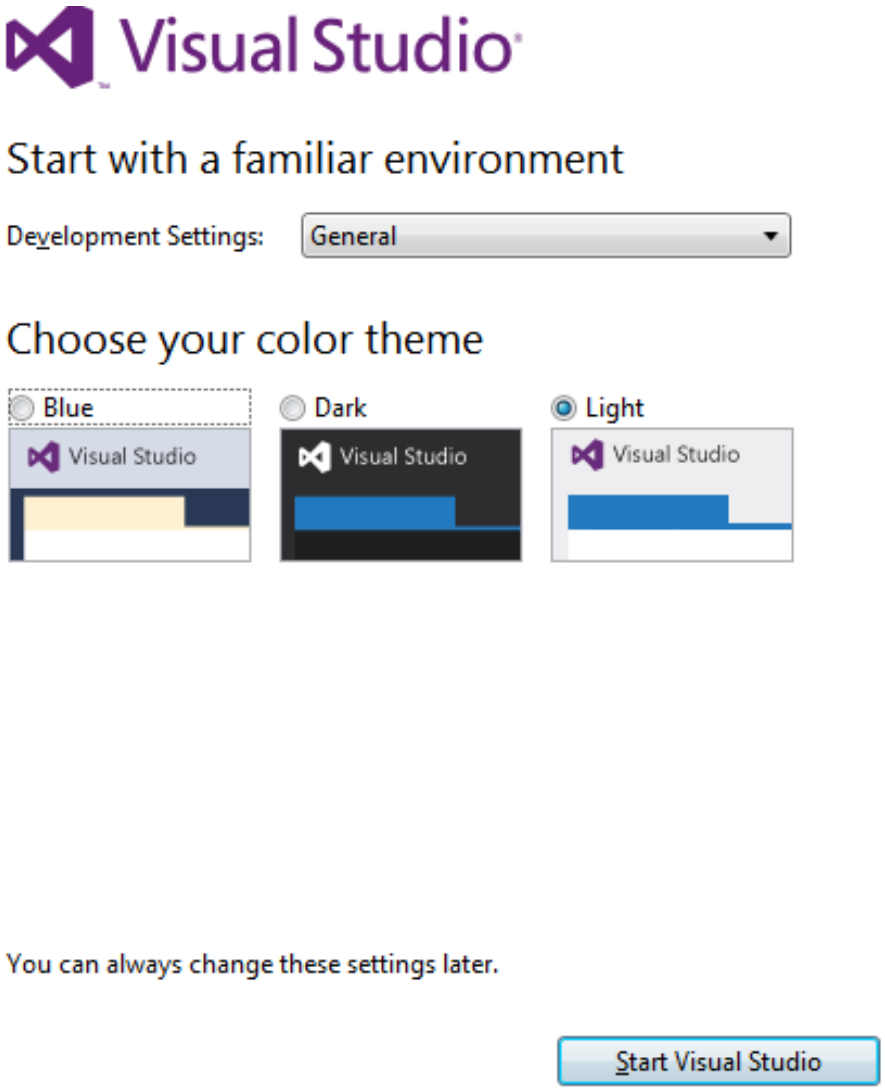
Slika 2.4. Prijava kod proizvođača programa

Nakon toga se prelazi na prozor prikazan na slici 2.5., na kome se bira boja pozadine radnog okruženja. Inicijalno je postavljena opsija ***Dark*** (crna boja pozadine za pisanje koda), ali mi ne preporučujemo ovu opciju. Mi preporučujemo

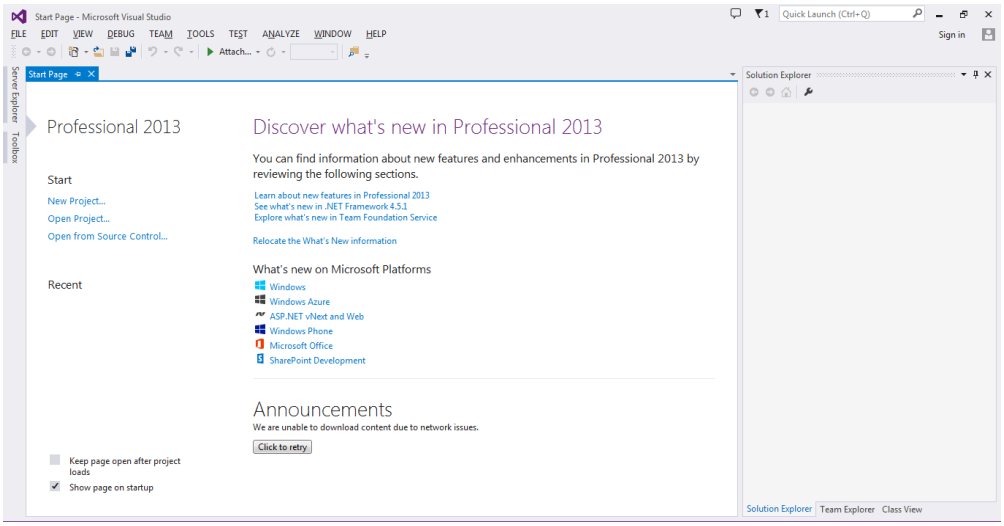
da se odabere treća opcija ***Light*** (svijetlo siva boja), jer je najsličnija ostalim Windows aplikacijama.

Klikom na komandno dugme ***Start Visual Studio*** prelazi se na prozor prikazan na slici 2.6., za početak rada sa ovim programskim jezikom. U meniju ***Start*** bira se najčešće između dvije opcije:

- ***New Project*** - ako se želi pisati novi program ili,
- ***Open Project*** - ako se želi otvoriti postojeći program.



Slika 2.5. Izbor boje radnog okruženja



Slika 2.6. Početni ekran kada se program pokrene

2.3. POKRETANJE *VISUAL BASIC* PROGRAMA

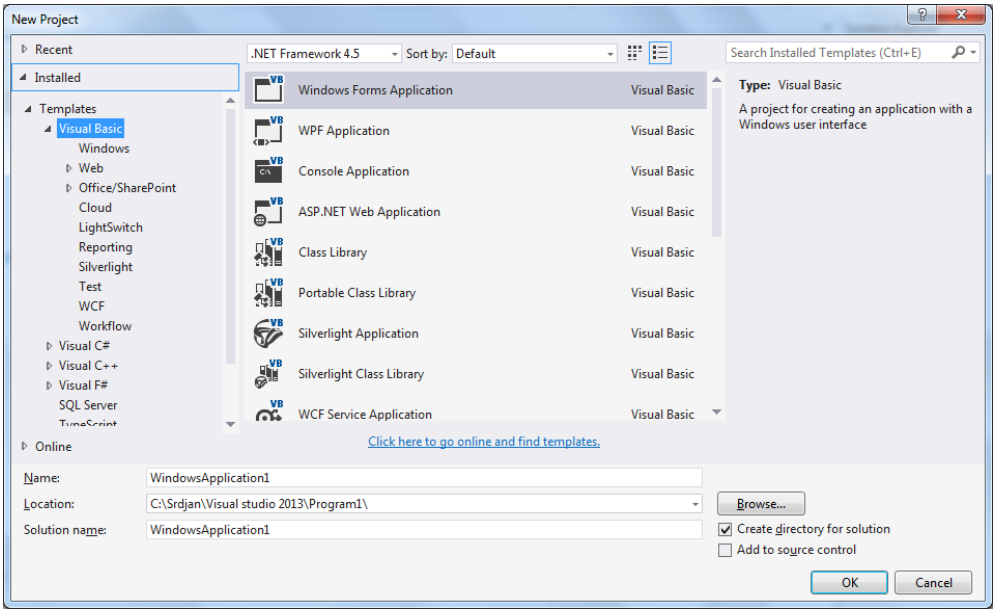
Aktiviranje programa *Visual Studio* 2013 u *Windows* operativnim sistemima vrši se najlakše kroz Start meni (*Start→Programs→Microsoft Visual Studio* 2013). Naravno, da ovo nije jedini način da se pokrene program *Visual Studio* 2013. Kao i za bilo koji drugi program u *Windows* operativnim sistemima postoji više načina za njihovo pokretanje npr. može se napraviti ikona-prečica (*Shortcut*) pa preko nje pokretati program.

Nakon pokretanja programa pojavljuje se prozor prikazan na slici 2.6. Na njemu se može izabrati jedna od sljedećih opcija:

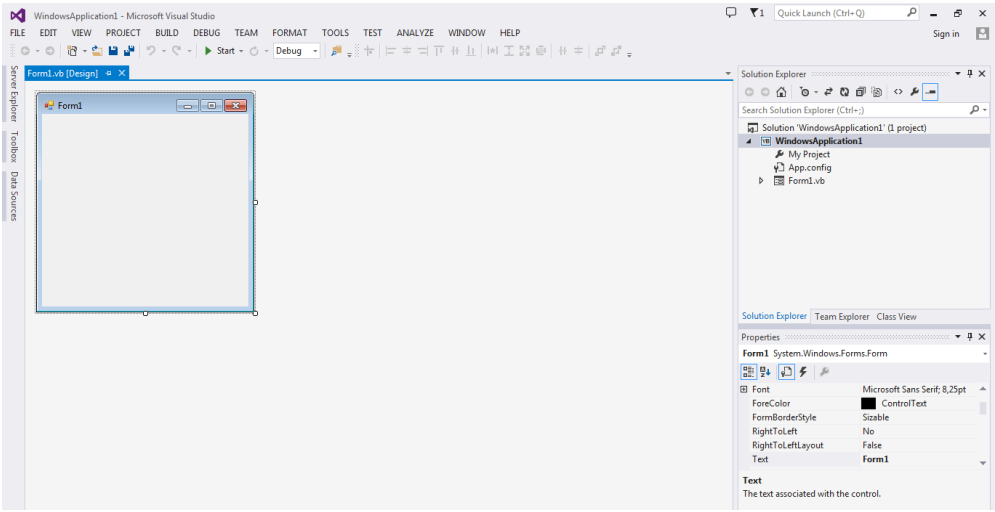
- ***New Project ...*** - za pravljenje potpuno novog projekta. Ponuđeno je više opcija za početak pravljenja programa.
- ***Open Project ...*** - ako se želi izabrati i otvoriti ranije napravljeni program.
- ***Recent*** - ako se želi izabrati i otvoriti neki zadnje korišćeni program. U listi se programi sortirani prema datumu zadnje modifikacije. Ovo je naručito pogodno, kada se zaboravi ime programa i mjesto (direktorijum) gdje je program snimljen.

Izborom opcije ***New Project*** prelazi se na prozor prikazan na slici 2.7. na kome je ponuđeno više opcija za početak pravljenja programa. U gornjem lijevom uglu ovog prozora potrebno je izabrati meni ***Installed*** pa pod meni ***Templates*** pa ***Visual Basic***. Pored programskog jezika ***Visual Basic*** na raspolaganju stoje i jezici: ***Visual C#***, ***Visual C++***, ***Visual F#***, ali oni nisu predmet ovog kursa. Kada se izabere opcija ***Visual Basic***, automatski se u sredini prozora pojavi lista unaprijed

pripremljenim šablonima formi za pokretanje programa. Za početnike je najbolje da izaberu opciju ***Windows Forms Application***. Iskusniji programeri mogu da izaberu neko već ranije pripremljeno radno okruženje. Na taj način se ubrzava proces programiranja.

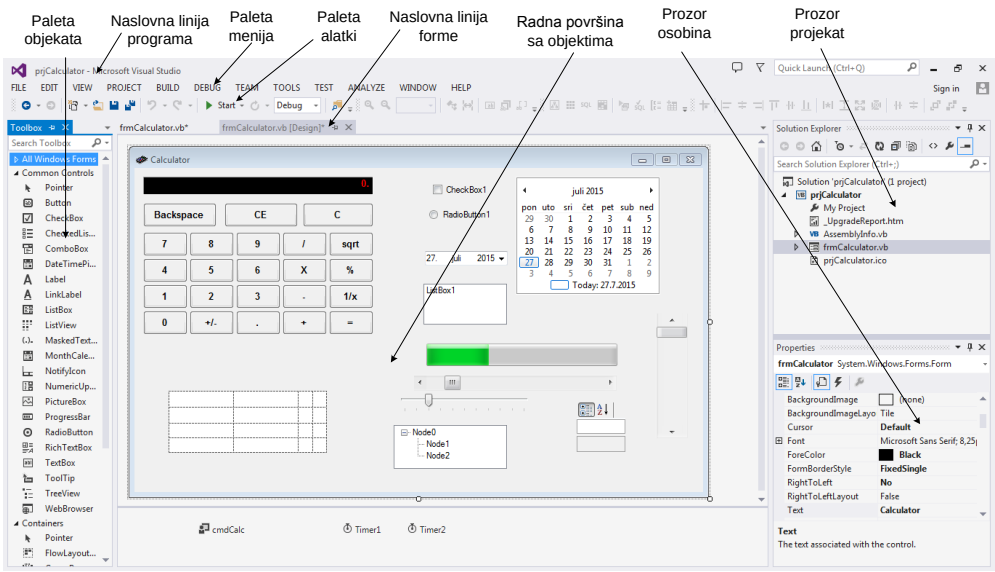


Slika 2.7. Izbor početnog radnog okruženja za pisanje programa



Slika 2.8. Početni ekran radnog okruženja za novi program

Izborom opcije ***Windows Forms Application*** automatski se popunjavaju polja: *Name*, *Location* i *Solution Name*. U polje *Name* i *Solution Name* poželjno je da se unese novo ime projekta, a ne ono koje predloži *Visual Basic*. U polje *Location* unosi se lokacija (direktorijum) na kome će se program *Visual Basic*, sa svim njegovim sastavnim dijelovima koje sam izgeneriše, nalaziti. Nije preporučljivo da se ova lokacija za smještaj programa upisuje ručno, već da se izabere preko dugmeta ***Browse...*** Ako programer sam ne unese ovu lokaciju, onda će program da je dodjeli automatski. To može biti problem, jer programer može lako da zaboravi tu lokaciju i neće moći da drugi put nađe program, koji je već napravio. Kada se upiše novo ime projekta i izabere lokacija gdje će se projekat nalaziti, potrebno je kliknuti na dugme ***OK***, nakon čega se pojavljuje prozor sa slike 2.8. Automatski se pojavljuje forma sa imenom Form1.vb, na koju se postavljaju gotovi objekti. U desnom dijelu ekrana pojavljuje se prozor ***Solution Explorer*** na kome se vidi struktura sa imenima svih elemenata, koji čine program i prozora ***Properties*** na kome se vide osobine selektovanog objekta. Pokretanjem programa *Visual Basic*, odmah se uočava da prozor programa pripada grupi standardnih prozora u *Windows* okruženju. Samim tim njegovo radno okruženje čini naslovna linija sa dugmadima za automatsko upravljanje veličinom prozora, linija menija, palete sa alatima, radni prostor, klizači za vertikalno i horizontalno listanje sadržaja dokumenta i statusna linija. Primjer jedne radne površine za pisanje programa prikazan je na slici 2.9.



Slika 2.9. Radna površina za pisanje programa

U slučaju da se želi otvoriti već postojeći program dok se već nalazimo u razvojnom okruženju programa, potrebno je otići na paletu menija na ***File*** pa ***Open*** i iz već poznatog okruženja (kao i kod ostalih programa *Windows* okruženja) iz eksplorer menija izabrati program, koji želimo da pokrenemo.

U naslovnoj liniji uvijek stoji ime programa, sa kojim se trenutno radi. Projekt prozor (***Solution Explorer***) se pojavljuje sa desne strane ekrana i u njemu se prikazuju: forme, moduli i klase koje postoje u aktivnom programu. Ispod ovog prozora se obično nalazi prozor ***Properties***, u kome se prikazuju sve osobine trenutno izabranog objekta na formi. Paleta objekata (***Toolbox***) se obično nalazi na lijevoj strani ekrana. Prikazuje standardne objekte, koji programeru stoje na raspolaganju. Ako se greškom zatvori ova paleta, ponovo se postavlja na radni ekran preko glavnog menija ***View*** i opcije ***Toolbox*** iz padajućeg menija. Ako se greškom zatvore sve palete ili ako se poremeti rasored paleta na ekranu, vraćanje na inicijalno (početno) podešavanje rasporeda svih paleta se vrši preko glavnog menija ***Window*** i opcije ***Reset Window Layout*** iz padajućeg menija.

2.4. SIMBOLI U PROGRAMU *VISUAL BASIC*

Visual Basic, kao i svaki drugi programski jezik, ima svoju gramatiku. Osnovne dvije grane gramatike programskog jezika su sintaksa i semantika. Pod sintaksom se podrazumijeva skup pravila, pomoću kojih se formiraju složeniji jezički oblici od prostijih: riječi od simbola, rečenice od riječi i viši (složeniji) oblici od rečenica. Semantika proučava značenje onoga što je iskazano u nekoj jezičkoj formi, odnosno ona se bavi smisaonim stranama jezičkih konstrukcija. Ako neki jezički oblik nema svoga smisla i značenja, onda ga nema potrebe ni pisati.

Simboli pomoću kojih se grade složeniji jezički oblici u programskom jeziku *Visual Basic* su: slova, cifre dekadskog brojnog sistema, interpunkcijski znaci, operatori i posebni znaci. U okviru ovog dijela knjige biće razmotrena specifična, nestandardna značenja pojedinih simbola.

Tačka (.) se upotrebljava za:
1) razdvajanje cijelog od decimalnog dijela broja (npr. 55.2345),
2) povezivanje objekata sa njihovim osobinama (npr. Dugme.Enabled = False) i
3) povezivanje imena fajla sa njegovim tipom (npr. Strana1.vb).

Znak za podvlačenje (_) se upotrebljava ako sadržaj komande ne može da stane u jedan programski red. Tada se na kraju toga reda upisuje znak "_", pa se nastavlja unos sadržaja komande u sljedeći programski red. Primjer:

```
Set baza = DBEngine.Workspace(0).OpenDatabase _
(App.Path & "\PodaciGorivo.mdb")
```

Ovaj simbol se koristi u procedurama događaja za pravljenje imena procedure, tako što se spaja ime objekta i ime događaja nad objektom. Primjer:

```
Private Sub Command1_Click()
```

Jednostruki navodnik (') (apostrof) u programskom jeziku *Visual Basic* služi za navođenje komentara. Tekst komentara je u kodu obojen zelenom bojom. Poželjno je da u programu postoji što više komentara. Komentari su korisni za samog programera, jer mu omogućuju lakše snalaženje u programu, a posebno kod modifikacije programa poslije određenog vremenskog perioda. Primjer:

```
Sub Command4_Click()  ' dugme povratak
```

Komentar u programu može takođe da se napiše i sa komandom ***REM***. Primjer:

```
REM Komentar za imena promjenjljivih
```

Dvostruki navodnici (") imaju ulogu definisanja tekstualnog podatka, koji se nalazi između njih. Primjer:

```
Text1.Text = "Unos teksta"
```

Simbol (&) služi za spajanje više stringova u jedan string i za definisanje slovne tipke, za aktiviranje potrebne opcije u meniju. Primjer:

```
Text2.Text = "Datum"&"2000" rezultat je
"Datum2000"
Text3.Text = "Datum"&"  " &"2000"
rezultat je  "Datum  2000"
```

Zapeta (,) koristi se za razdvajanje argumenata kod funkcija i procedura. Primjer:

```
Odgovor1 = MsgBox(Poruka, Stil, Naslov)
```

Ovaj simbol se ne smije koristiti za pisanje realnih brojeva, koji se isključivo pišu korištenjem simbola tačka (.).

Znak jednakosti (=) koristi se, pored standardnog relacijskog značenja poređenja dva izraza, u svojstvu operatora dodjele konkretne vrijednosti nekoj osobini objekta ili promjenljivoj. Primjer:

```
If  A=B Then              - poređenje dvije promjenljive
HG = HG + 1               - dodjela vrijednosti promjenljivoj
Command1.Enabled = True   - dodjela vrijednosti osobini objekta
```

Male zagrade ( ) služe za grupisanje više izraza u jednu cjelinu i za postavljanje argumenata funkcija i procedura. U jednom matematičkom izrazu

može biti više zagrada jedna unutar druge. Svaka otvorena zagrada se mora zatvoriti. Primjer:

```
BETA1 = 2 * Atn(Sqr(1 - b1 * b1) / b1)
Odgovor2 = MsgBox(poruka, Stil, Naslov)
```

Tri tačke (...), u okviru neke osobine objekta, najavljuju pojavu menija sa više ponuđenih opcija.

Matematički operator za oduzimanje (-) ima i ulogu logičkog negatora.

Znak plus (+) pored sabiranja brojeva služi i za spajanje stringova. Primjer:

```
A= 2 + 3                daje broj 5, dok
B="2" + "3"             daje string "23", a
C="Stefan"+" "+"Covic" daje string "Stefan Covic".
```

Simbol zvijezdica (*) posjeduje značenje operatora za množenje i znaka za popunu skrivenog teksta u *TextBox*-u (*PasswordChar*).

Operator stepenovanja (^) služi za stepenovanje nekog broja. Primjer:

```
x=a^2     ili   x=a*a
y=d^5     ili   y=d*d*d*d*d
```

Kose crte (/ i \ ) koriste se kao operator za dijeljenje. Kada se kao rezultat dijeljenja dobija realni broj koristi se operator (/). Za cjelobrojno dijeljenje kada se kao rezultat dobija cijeli broj koristi se operator (\). Primjer:

```
x=5/2     - daje rezultat 2.5
y=5\2     - daje rezultat 2
```

Relacioni operatori: manje (<), manje ili jednako (=<) , različito (< >), veće (>), veće ili jednako (>=) imaju funkciju poređenja dva izraza i redovno se koriste u naredbama selekcije i iteracije.

Riječi u programskom jeziku *Visual Basic*, kao i u drugim programskim jezicima, predstavljaju skup simbola bez logičkih i relacijskih operatora. One su dvojakog karaktera. Neke su propisane od strane autora dotičnog jezika (tzv. obavezne ili ključne riječi, koje se u kodu programa pojavljuju pisane plavom bojom). Druga vrsta su izvedene riječi, koje definiše programer programa. Propisane riječi su riječi engleskog jezika ili njihove skraćenice i upotrebljavaju se doslovno. Pomoću njih su definisane ključne riječi u pojedinim naredbama i funkcijama. Izvedenim riječima opisuju se imena pojedinih objekata, procedura i funkcija, struktura podataka i njihovih atributa, promjenljivih, konstanti, polja u kojima su smješteni podaci itd.

2.5. OBJEKTI I NJIHOVE OSOBINE U *VISUAL BASIC*-U

Visual Basic posjeduje veliki broj gotovih objekata, koji stoje programeru na raspolaganju. U standardnoj paleti alatki (**Toolbox**) programskog jezika *Visual Basic* se nalaze objekti, koji se najčešće koriste u programiranju, a to su:

- **Form** radna površina za postavljanje ostalih objekata.
- **Label** za ispis teksta.
- **TextBox** za prikaz i unos tekstualnih i numeričkih vrijednosti.
- **Button** komandno dugme.
- **CheckBox** i **RadioButton** za izbor opcija.
- **Panel** pravljenje posebnog okvira sa objektima na formi.
- **ListBox** i **ComboBox** za prikaz vrijednosti u padajućim listama.
- Klizne trake **HScrollBar** i **VScrollBar** za izbor brojne vrijednosti preko kliznih traka.
- **Timer** za korišćenje sistemskog datuma i vremena, kao i za vremenska podešavanja i kretanje objekata po formi.
- **Picturebox** za prikaz slika raznih formata.
- **MenuStrip** za pravljenje vlastitog padajućeg menija.
- **DataGridView** za tabelarni prikaz podataka.
- **ProgressBar** za prikazivanje horizontalnog Bargrafa.
- **CommonDialogs** za pravljenje raznih dijalog prozora (OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog, FolderBrowserDialog, PrintDialog, PrintPrewiewDialog, PageSetupDialog).

Pored ovih *Visual Basic* ima još veliki broj gotovih objekata. Objekti se postavljaju na radnu površinu (formu), tako što se prvo klikne mišem na odgovarajuću ikonicu objekta u paleti objekata. Potom, uz zadržavanje pritisnutog lijevog tastera miša, u odgovarajućoj veličini i na željenoj poziciji na formi razvije končanica oblika paravougaonika, koja sadrži traženi objekat. Nakon otpuštanja lijevog tastera miša, ostaje objekat okružen linijom pravougaonika. Dovođenjem kursora miša na jedan od tih pravougaonika pojavljuje se dvosmjerna strelica, pomoću koje se objekat može proširivati ili sužavati u željenom pravcu. Postavljanjem kursora na objekat i zadržavanjem lijevog tastera objekat se može, povlačenjem miša, premjestiti na drugo mjesto na formi. Objekat se može izbrisati, tako što se prvo klikom miša izvrši njegov izbor (pojave se pravougaonici na središtima stranica i na tjemenima) i pritisne tipka *Delete (Del).* Forma, takođe, sama po sebi, predstavlja jedan objekat. On je prozor u korisničkom interfejsu, čiji se oblik i veličina podešavaju, slično ostalim objektima, tako što se pokazivač postavi na jednu od ivica ili na tjeme i zatim povuče u potrebnom smjeru.

Svaki objekat u programu *Visual Basic* ima više osobina, koje se podešavaju. Osobine objekata se mogu dodjeljivati preko prozora osobina (*Propereties*), ili programski preko koda programa. Kada se klikne na posmatrani objekat, tada se u

prozoru osobina (*Propereties*) pojavljuju postojeće osobine selektovanog objekta ili grupe selektovanih objekata. U prozoru *Propereties* na lijevoj strani tabele se nalaze imena osobina. Na desnoj strani tabele treba preko tastature unijeti odgovarajuću vrijednost osobine ili osobinu izabrati od onih vrijednosti, koje su unaprijed ponuđene. Izbor se vrši tako što se klikne na osobinu i tada se pojave tri tačke (...) ili lista ▼ (sa padajućim menijem). Klikom mišem na tri tačke aktivira se okvir za dijalog, pomoću koga se biraju vrijednosti osobine. Aktiviranjem neke osobine za boje (na primjer *BackColor*), pojavljuju se opcije: *Custom, Web* i *System.* One omogućavaju da se sa palete boja izvrši željeni izbor boje. Kada se izabere željena boja prikazuje se njena vrijednost u ARGB kodu, koji se sastoji od 3 broja. ARGB kod boje je za većinu programera nerazumljiv broj i koristi se uglavnom kod dodjeljivanja boja preko koda na sljedeći način:

```
Button1.BackColor = Color.FromArgb(192, 0, 192)
```

Ovo je primjer koda gdje je boja pozadine komandnog dugmeta pod imenom Button1 promjenjena u ljubičastu boju.

Često je prozor za osobine tako mali da se u njemu ne mogu vidjeti sve osobine odgovarajućeg objekta. U tom slučaju pored desne ivice, postoje vertikalni klizači za pomjeranje osobina. Za veliki broj osobina postoje predložene standardne vrijednosti, mada korisnik može mijenjati i postavljati nove prema svome ukusu.

U nastavku objasnićemo samo neke osobine, koje se često koriste i odnose se, uglavnom, na više objekata.

Osobina ***Alignment*** služi za poravnanje teksta u objektima. Klikom na osobinu pojavljuje se *ComboBox*, čijim se aktiviranjem otkrivaju mogućnosti poravnanja:

- *Left* (poravnanje u lijevo),
- *Right* (poravnanje u desno),
- *Center* (centriranje teksta).

Klikom na osobinu ***BackColor***, na desnoj strani ove osobine, pojavljuju se strelica na dole (▼). Klikom na nju otvara se okvir za dijalog sa mogućnošću izbora odgovarajuće boje iz palete boja ili potvrde ponuđenog sistema boja. Odabrana vrijednost boje odnosi se na pozadinu posmatranog objekta.

BorderStyle je osobina koja ima ulogu da označi ivicu dotičnog objekta. Klikom na ▼ aktivira se lista mogućih vrijednosti:

- *None* (bez okvira),
- *FixedSingle* (objekat uokviren jednom linijom),
- *Fixed3D* (okvir koji simulira treću dimenziju).

Enabled je osobina koja omogućuje ili zabranjuje rad sa objektom. Može da ima dvije vrijednosti: *False* i *True.* Dodjelom osobini vrijednosti *True* objekat postaje aktivan, odnosno dostupan tako da nad njim mogu da se izvršavaju određene akcije. Dodjelom osobini vrijednosti *False* objekat je i dalje vidljiv na

formi, ali nije više aktivan, odnosno nemože mu se pristupiti ni sa jednom programskom akcijom.

Klikom na osobinu ***Font***, koju posjeduju svi objekti tekstualnog tipa, otvara se standardni *Word*-ov okvir za dijalog na kojem se može odabrati naziv, veličina, stil i druge karakteristike *Fonta.*

Osobina ***ForeColor*** služi za bojenje teksta. Postupak dodjele boja isti je kao i kod osobine *BackColor*.

Osobinama ***Height*** i ***Width*** unosom brojnih vrijednosti određuje se visina i širina objekta u pikselima (*pixels)*.

Pomoću osobine ***Location*** određuje se udaljenost objekta u pikselima (*pixels)* od lijeve ivice ekrana (osobina **X**) i gornje ivice ekrana (osobina **Y**).

Osobina ***Name*** određuje ime objekta. Pomoću ove osobine se objekti pozivaju u kodu programa.

Pomoću osobine ***ReadOnly***, koja može da ima vrijednosti *True* i *False*, definiše se da li se dati podaci mogu samo pregledati ili se mogu i mijenjati.

Osobinom ***ScroolBars*** definišu se klizači za pomjeranje teksta odnosno drugog sadržaja na objektu. Na listi postoje sljedeće mogućnosti izbora:

- *None* (bez klizača),
- *Horizontal*,
- *Vertical* i
- *Both* (obostrani).

Osobina ***Size*** ima pod osobinu ***With*** kojom se podešava širina objekta u pikselima i ***Hight*** kojom se podešava visina objekta u pikselima.

Bitna je opcija ***StretchImage*** čijim se izborom u objektu *PictureBox* vrši prilagođavanje originalne dimenzije importovane slike, prema dimenzijama objekta na formi za prikaz slike.

U osobinu ***Text*** upisuju se proizvoljan tekst, koji će pisati na dotičnom objektu. Ovu osobinu treba razlikovati od osobine imena objekta (Name).

Pomoću osobine ***Visible***, koja se često primjenjuje za više objekata, reguliše se vidljivost objekta. Ako je izabrana vrijednost ove osobine *True* objekat će biti vidljiv, a ako je izabrana vrijednost *False* objekat neće biti vidljiv, kada se program pokrene. Ova osobina objekata se često podešava u kombinaciji sa osobinom *Enable*.

Osobina forme ***StartUpPosition*** služi za određivanje početnog položaja forme na ekranu, kada se program pokrene. Može imati opcije:

- Manual (ručno postavljanje),
- CenterScreen (centriranje prema čitavom ekranu),
- WindowsDefoultLocation (pozicioniranje preko čitavog ekrana).

2.6. SKRAĆENICE U PROGRAMU *VISUAL STUDIO* 2013

Programiranje u programu *Visual Basic* vrši se postavljanjem gotovih objekata na formu. Objekti se međusobno povezuju preko koda. U kodu se pišu naredbe, koje omogućavaju da program izvršava zamišljene funkcije. Da bi se olakšalo i značajno ubrzalo pisanje programa korisno je poznavati skraćenice na tastaturi. Osnovne skraćenice koje se koriste u programskom jeziku *Visual Basic* prikazane su u tabeli 2.1.

Tabela 2.1. Skraćenice u programu Visual Basic

Skraćenica	Opis radnje
	Kontrola dokumenata
CTRL+N	Kreiranje novog projekta. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *File => New*. Da se ne bi izgubio predhodno aktivan program, pojaviće se dijalog prozor za spašavanje otvorenog programa.
CTRL+SHIFT+N	Kreiranje novog fajla.
CTRL+O	Otvaranje već postojećeg *Visual Basic* programa. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *File => Open.*
CTRL+SHIFT+O	Otvaranje već postojećeg fajla, istog tipa kao onaj na kome se trenutno nalazimo.
CTRL+S	Spašavanje trenutno aktivnog *Visual Basic* programa. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *File => Save.*
CTRL+SHIFT+S	Spašavanje svih trenutno aktivnih fajlova, projekata i dokumenata *Visual Basic* programa.
Alt+Print Screen	Kopiranje trenutno vidljivog programa u druge programe. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *File => Print Scrin.*
CTRL+D	Prikaz prozora *Add Item dialog box* da bi dodali neki već postojeći element u vaš projekat.
Alt+...	Pristup glavnom meniju programa (*F-File, E-Edit, V-View, P-Project, F-Format, D-Debug, ... , W-Windows, H-Help*).
CTRL+A	Selektovanje svih objekata u trenutno aktivnoj formi.
Alt+F4	Zatvaranje aktivne aplikacije.
CTRL+P	Štampanje svih djelova dokumenta.
	Izmjena u kodu nekog objekta
Enter	Prelazak u novi red za pisanje koda.
Delete	Brisanje karaktera u kodu, koji se nalazi desno od pokazivača.
CTRL+Delete	Brisanje riječi u kodu, koja se nalazi desno od pokazivača.
Backspace	Brisanje karaktera u kodu, koji se nalazi lijevo od pokazivača.
CTRL+Backspace	Brisanje riječi u kodu, koja se nalazi lijevo od pokazivača.

Skraćenica	Opis radnje
CTRL+C	Kopiranje objekta ili koda.
CTRL+X	Isijecanje objekta ili koda.
CTRL+V	*Past* objekta ili koda (postavljanje iskopiranog dijela).
CTRL+Z	*Undo* (korak unazad) od zadnje transakcije.
CTRL+SHIFT+Z	*Redo* (korak unaprijed) od zadnje transakcije.
CTRL+Up	Pomjeranje trenutne transakcije naviše.
CTRL+Down	Pomjeranje trenutne transakcije nadole.
CTRL+F	Otvaranje prozora *Find* za pronalaženje željenog objekta.
CTRL+H	Prikaz prozora *Replace* za pretragu i zamjenu teksta u kodu.
CTRL+U	Obilježeni tekst pretvara u mala slova.
CTRL+SHIFT+U	Obilježeni tekst pretvara u velika slova.
	Navigacija kroz program
End	Skok na kraj reda koda u kome se trenutno nalazi pokazivač.
CTRL +End	Skok na prvo slovo zadnjeg reda koda.
Home	Skok na početak reda koda u kome se nalazi pokazivač.
CTRL +Home	Skok na prvo slovo prvog reda koda.
Shift+End	Selekcija teksta u kodu od trenutnog položaja pokazivača do kraja kodne linije.
CTRL + Shift + End	Selekcija teksta u kodu od trenutnog položaja pokazivača do kraja koda.
Shift+Home	Selekcija teksta u kodu od početka kodne linije do trenutnog položaja pokazivača.
CTRL+Shift+ Home	Selekcija teksta u kodu od početka stranice do trenutnog položaja pokazivača.
SHIFT+Strelica ←	Selekcija teksta u kodu do početka riječi u lijevu stranu od pokazivača.
SHIFT+Strelica →	Selekcija teksta u kodu do kraja riječi u desnu stranu.
Page Down/Up	Kretanje kroz radnu površinu programa za veličinu jednog ekrana dole/gore.
Shift+strelica ←↑↓→	Kretanje kroz kod programa lijevo/gore/dole/desno.
CTRL+Page Down	Kretanje kroz radnu površinu programa za veličinu jednog ekrana lijevo.
CTRL+Page Up	Kretanje kroz radnu površinu programa za veličinu jednog ekrana desno.
Tab	Pomjeranje teksta koda za par karaktera u desnu stranu. Na formi se koristi za prelazak sa jednog objekta na drugi.
Shift +Tab	Pomjeranje teksta koda za par karaktera u lijevu stranu. Na formi se koristi za prelazak sa jednog objekta na drugi.
	Testiranje (*Debaging*) programa
F5	Pokretanje programa. Potpuno isti efekat se dobije ako se u glavnom meniju izabrala opcija *Run => Start.*
Shift + F5	Zaustavljanje izvršenja programa. Potpuno isti efekat se dobije ako se u glavnom meniju izabrala opcija *Run => End.*

Skraćenica	Opis radnje
CTRL +F5	Pokretanje programa ali bez pronalaska grešaka.
F11	Izvršenje programa korak po korak, od jednog do drugog objekta. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *Debug => Step InTo.*
F10	Izvršenje programa korak po korak, od jednog do drugog objekta. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *Debug => Step Over.*
Shift +F11	Izvršenje programa korak po korak, od jednog do drugog objekta. Dobije se potpuno isti efekat, kao da se u glavnom meniju izabrala opcija *Debug => Step Out.*
CTRL+BREAK	Zaustavljanje izvršenja programa.
F1	Pomoć u programu. Kada je neki objekat u programu selektovan i zatim se pritisne tipka F1 otvoriće se prozor sa tekstom, u kome je opisana pomoći za taj objekat.
F9	Dodavanje odnosno uklanjanje kontrolne tačke za tekuću liniju koda u kojoj se nalazi kursor.
CTRL+SHIFT+F9	Uklanjanje svih predhodno postavljenih kontrolnih tačaka u programu.
CTRL+F9	Onemogućavanje rada kontrolnoj tački.

2.7. PRAVLJENJE PRVOG PROGRAMA

U ovom poglavlju opisani su osnovni koraci za kreiranje novog programa, kroz izradu jednostavnog primjera. Programski jezik *Visual Basic* se najlakše uči kroz rješavanje praktičnih problema. Kroz ovaj primjer početnici u radu sa ovim programskim jezikom mogu da uoče veliku jednostavnost i lakoću pravljenja programa. Zadatak opisanog primjera je da se korisniku programa omogući sabiranje, oduzimanje, množenje i dijeljenje dva realna broja, koji se unose preko tastature. Potrebno je postaviti i komandno dugme pomoću koga se program završava.

Osnovni koraci kroz koje treba proći prilikom kreiranja programa u programskom jeziku *Visual Basic* su:

1) Izbor i postavljanje na radnu površinu potrebnih objekata za rješavanje postavljenog zadatka.
2) Definisanje osobina svakog postavljenog objekta.
3) Međusobno povezivanje objekata.
4) Testiranje programa.

Ako se preskoči bilo koji od ovih koraka, prilikom pravljenja programa, onda će programer vjerovatno imati probleme kod pisanja programa, kao i krajnji korisnik prilikom korišćenja programa.

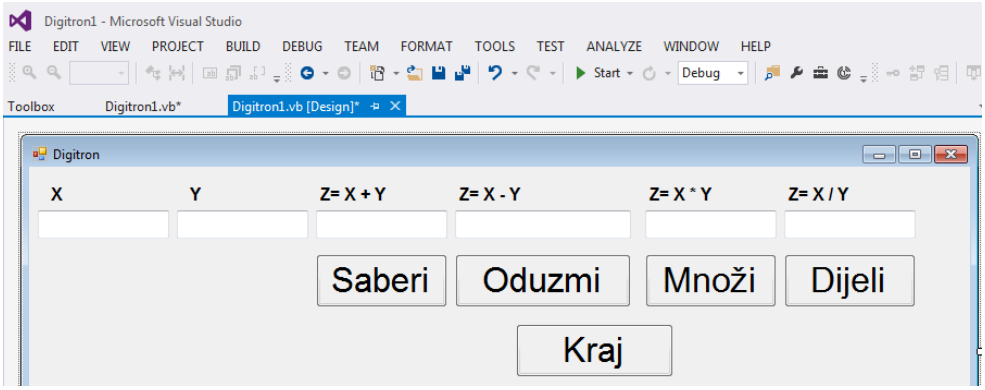
2.7.1. Izbor potrebnih objekata za rješavanje postavljenog zadatka

Prva i najbitnija faza u procesu pisanja programa je definisanje problema. Preskakanje ove faze znači sigurno velike probleme u svim narednim fazama, koje slijede. U ovoj fazi treba da se uoči problem, određuje se način rješavanja, vrši se analiza postojećeg stanja, analiziraju se iskustva u radu sa ovakvim i sličnim zadacima. Kada je izvršena postavka zadatka, može se pristupiti pravljenju algoritma. Zatim je potrebno izvršiti izbor objekata, koji su nam potrebni za rješavanje postavljenog zadatka.

Za pisanje programa u programskom jeziku *Visual Basic*, sa kojim želimo da riješimo postavljeni problem, potrebni su nam sljedeći objekti:

- 2 objekata za unos brojeva (*Text Box*),
- 4 objekta za prikaz izračunate vrijednosti (*Text Box*),
- 6 objekata koji opisuju objekte za unos i prikaz vrijednosti (*Label*),
- 5 komandnih dugmadi za početak i kraj programa (*Button*).

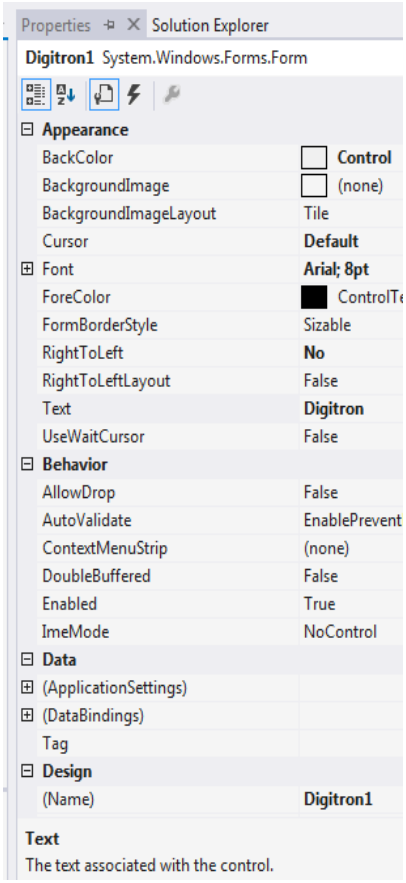
Svi ovi objekti se biraju iz palete *ToolBox* i postavljaju na radnu površinu za pravljenje programa (koja ima svije ime (inicijalno *Form1*). Na radnoj površini se ovaj prozor prepoznaje po obliku *ImeForme.vb* [Design]**). Objekti za ovaj program su prikazani na slici 2.10.



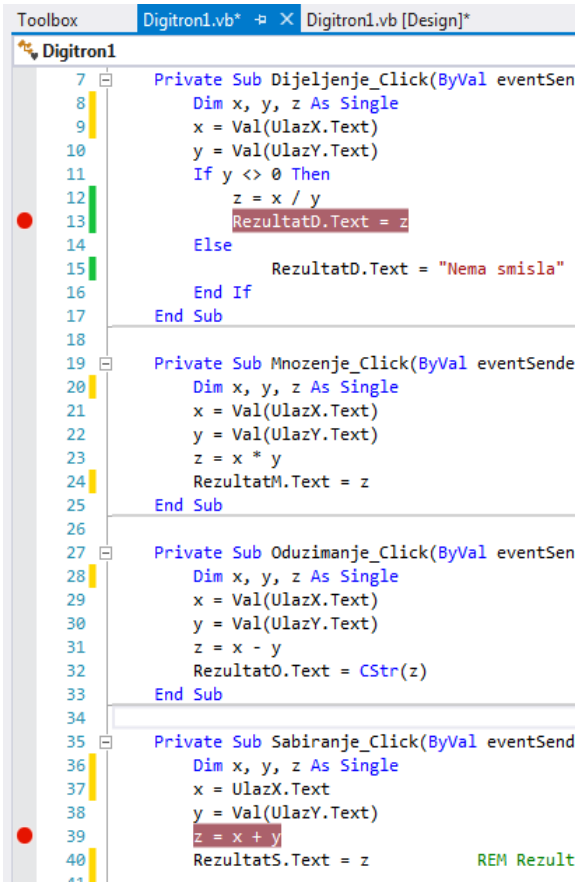
Slika 2.10. Postavljanje potrebnih objekata na radnu površinu

2.7.2. Definisanje osobina objekata

Kada se objekti postave na radnu površinu za pravljenje programa potrebno je izvršiti podešavanje osobina svakog objekta. Na taj način se objekti prilagođavaju njihovoj namjeni, ali se postiže i da program estetski lijepo izgleda. Podešavanje osobina objekata vrši se u prozoru *Propertes*. Do osobina svakog objekta se dolazi klikom miša na objekat. Pojavljuje se prozor prikazan na slici 2.11.



Slika 2.11. Osobine objekta



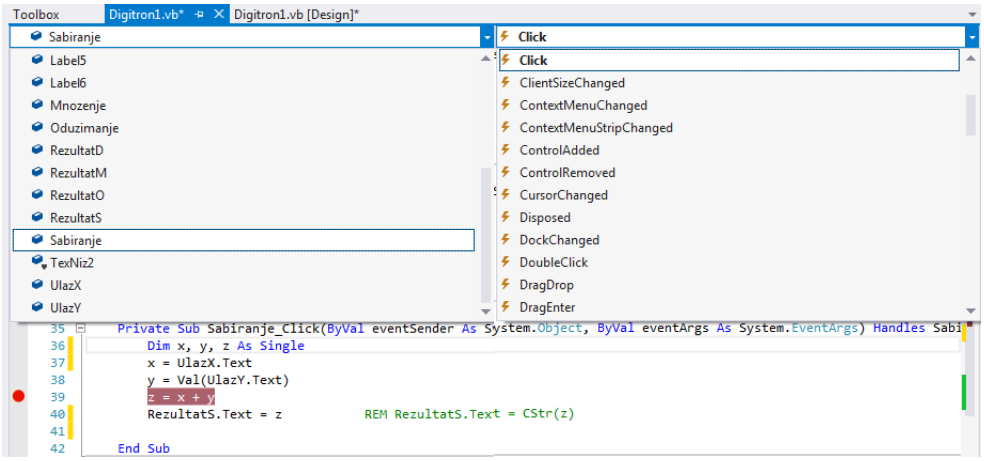
Slika 2.12. Kod programa

Svaki objekat u programskom jeziku *Visual Basic* ima različite osobine, pa samim tim i različite palete za podešavanje. Preko ovih paleta se najčešće podešavaju neke od sljedećih osobina:

- ime objekta,
- natpis na objektu,
- oblik objekta,
- font, boja i veličina slova,
- boje pojedinih dijelova objekta,
- dimenzije objekta,
- veze sa drugim objektima,
- vidljivost ili nevidljivost objekta,
- dostupnost objekta.

2.7.3. Međusobno povezivanje objekata

Kada su objekti postavljeni na radnu površinu i kada su im postavljene željene osobine, potrebno je izvršiti povezivanje objekata. Objekti se međusobno povezuju pisanjem koda programa. Do prozora za pisanje koda programa dolazi se klikom miša na bilo koji objekat, koji je postavljen na formu, ili desni klik miša bilo gdje na formi i izbor opcije *View Code*. Izgled prozora za pisanje koda programa, koji omogućuje računanje osnovnih matematičkih operacija, prikazan je na slici 2.12.



Slika 2.13. Lista postojećih objekata i događaja nad njima

Na slici 2.13. prikazan je isti prozor za pisanje koda, iznad koga su sada posebno istaknute dvije liste (*ComboBox*). Lijeva lista daje spisak imena svih objekat, koji postoje na toj radnoj površini. U desnoj listi se nude svi mogući događaji, koji se mogu programirati nad predhodno selektovanim objektom u lijevoj listi. Pri pisanju koda prvo se bira željeni objekat, a zatim i događaj koji se želi programirati nad njim. Izborom odgovarajućeg događaja klikom miša u desnoj listi, automatski se pojavljuje procedura događaja u prostoru za pisanje koda sa imenom događaja i objekta. Procedura događaja klik miša na komandno dugme (***Private Sub Sabiranje_Click()***) se automatski pojavljuje u kodu i kada se na radnoj površini uradi klik miša na željeno komandno dugme. ***Sub*** je ključna riječ, koja znači da se radi o proceduri. Dok riječ ***Private*** označava da se radi o lokalnoj proceduri, koja vrijedi samo na trenutno aktivnoj formi.

Za operacije sabiranja, oduzimanja i množenja kod programa je gotovo identičan. Prvo se vrši deklarisanje tri promjenljive realnog tipa. Zatim slijede dvije kodne linije u kojima se iz objekata *TextBox* preuzimaju unesene vrijednosti u promjenljive. Zatim slijedi kodna linija u kojoj se izvršava željena matematička operacija. Zadnja kodna linija u svakoj proceduri je dodjela izračunate vrijednosti u *TextBox*, koji služi za prikaz rezultata.

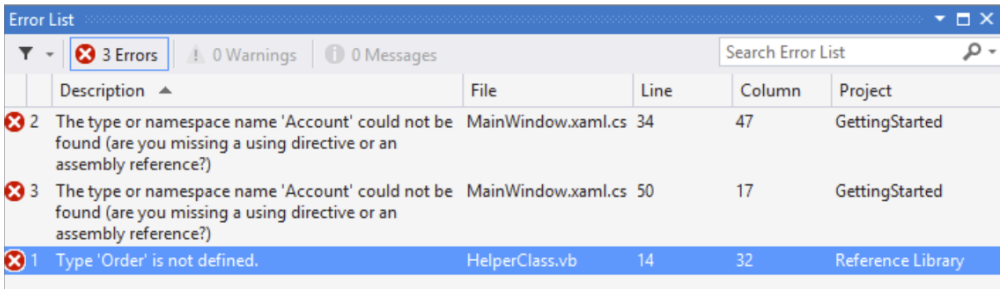
Operaciju dijeljenja je malo složenije isprogramirati u odnosu na predhodne tri operacije. Dijeljenje sa nulom nije dozvoljena operacija u programskim jezicima. Zato se prije operacije dijeljenja mora ispitati da li je broj sa kojim se vrši dijeljenje različit od nule (y < > 0). Ovo ispitivanje se vrši preko *If ...Then* strukture grananja. Ako je uslov ispunjen, vrši se matematička operacija dijeljenja. Ako uslov nije ispunjen, mora se korisnik programa obavijestiti da se pojavila greška pri unosu podataka i da se ne smije izvršiti operacija dijeljenja.

Tekst koda se može povećavati i smanjivati (*Code Zooming*) držanjem tipke Ctrl+ pomjeranjem *Scrol* točkića na mišu.

2.7.4. Testiranje programa i pravljenje izvršne verzije programa

Kada se program napiše, potrebno je prvo uraditi njegovo testiranje. Testiranje programa se pokreće klikom na ikonicu ▶ ***Start*** (ili tipka F5), a prekida izvršavanje programa klikom na ikonicu ■ ***Stop Debugging*** (ili tipka Shift + F5).

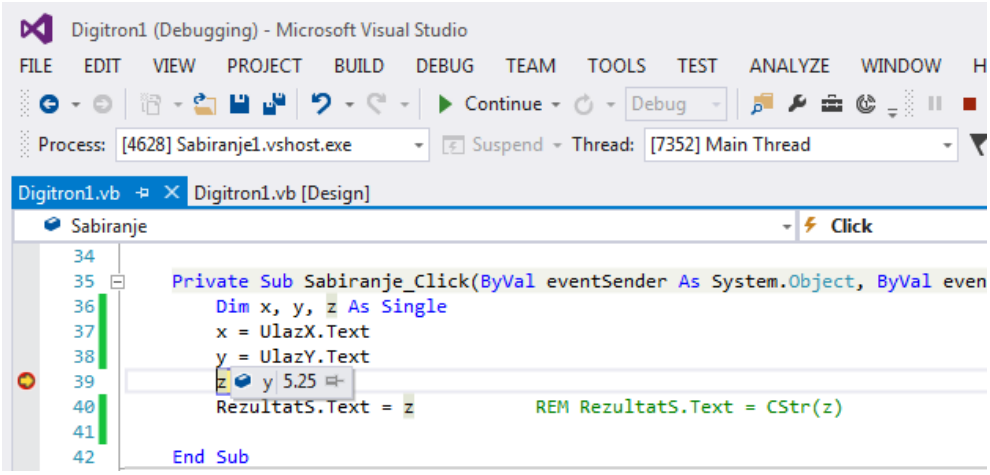
Prilikom testiranja programa otkrivaju se greške u pisanju programa. Greške mogu da budu sintaktičke i logičke. Sintaktičke greške nastaju prilikom kucanja naredbi u kodu programa. Najčešće greške su izostavljanje nekog slova ili se neko slovo otkuca dva puta. Ove greške se puno lakše otkrivaju od logičkih grešaka. Logičke greške nastaju zbog pogrešnog rasporeda objekata u programu, tako da program ne može da se završi do kraja ili se na kraju dobije rezultat, koji nije dobar. Program sam prepoznaje neke greške u kodu, a označava ih sa crvenom tačkom i sa plavom krivudavom linijom ispod teksta koda. Prilikom testiranja programa svaka otkrivena greška se posebno numeriše i pojavljuje u prozoru ***Error List***, sa detaljnim opisom uzroka greške, kao na slici 2.14. Za svaku grešku u koloni ***Description*** opisan je tip i uzrok nastanka greške. U koloni ***Line*** stoji broj linije koda, u kojoj je greška nastala. U koloni ***Column*** stoji broj, koji pokazuje u kom slovu te linije je nastala greška. Duplim klikom na grešku u ovom prozoru, odlazimo na kodnu liniju, u kojoj je nastala ova greška. Sve dok postoji neka greška u kodu programa, nije moguće da se program izvrši.



Slika 2.14. Prozor za prikaz grešaka u kodu programa

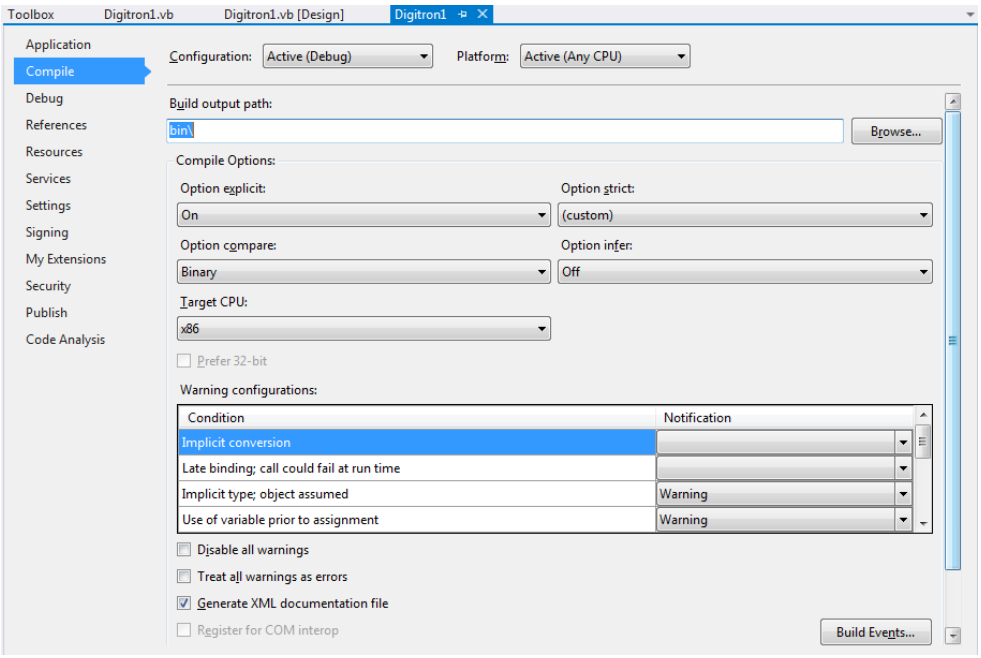
Jedan od načina otklanjanja logičkih grešaka je da se u programu postave kontrolne tačke, kao na slici 2.15., gdje je kontrolna tačka postavljena na 39. liniju koda. Kontrolne tačke se postavljaju klikom miša lijevo od linije, koja sa lijeve strane ograničava prostor za kucanje koda programa. Kontrolne take se označavaju kao crveni krugovi, a kontrolne kodne linije biće obojene crvenom bojom. Ponovnim klikom miša na kontrolnu tačku, ona se uklanja. Kontrolna tačka se može postaviti na svaku kodnu liniju, osim na kodnu liniju koja služi za pisanje komentara i koja počinje sa ključnom riječi *Rem*. Klikom na ikonicu ▶ ***Start*** program se izvršava do prve kontrolne tačke. Nastavlja se izvršavanje programa klikom na ikonicu ▶ ***Continue*** (koja je ranije imala ime ***Start***) postepeno od jedne do druge kontrolne tačke. Kada program ima više kontrolnih tački, sa žutom strelicom se označava kontrolna tačka do koje se stiglo sa izvršenjem programa. Pri tome se provjeravaju vrijednosti promjenljivih dolaskom miša na bilo koju promjenljivu ili funkciju. U primjeru sa slike 2.15. to je promjenljiva **Y** koja ima vrijednost 5.25 (y=5.25). Ako je prikazana vrijednost neke promjenljive navedena između dvostrukih navodnika (“5.25“), to odmah znači da se radi o tekstu, a ne o broju. Na ovaj način se i najlakše dolazi do otkrivanja grešaka, a zatim i ispravljanja grešaka u programu.

Za lakše praćenje koda programa, poželjno je da se u editoru za pisanje koda programa automatski označavaju redni brojevi kodnih linija, kao na slici 2.15. Da bi u editor dodali brojeve kodnih lija potrebno je proći kroz sljedeće korake. U glavnom meniju ***Tools***, izabrati ***Options***. U listi izabrati direktorijum ***Text Editor***, a u poddirektorijumu ***All Languages*** izabrati opciju ***General***. U desnom prozoru pod nazivom ***Settings*** čekirati opciju ***Line numbers***. Prihvatiti podešavanje sa dugmetom ***OK***.



Slika 2.15. Kontrolne tačke u kodu programa

Kada se program u potpunosti istestira, kupcu programa se na korišćenje daje samo izvršna verzija programa. Ako bi kupcu programa dali i kod programa, onda bi ga on mogao program lako prepraviti i dalje prodavati kao njegov program, a da nije uložio pamet i trud za razvijanje programa. Izvršnu verziju programa *Visual Basic* pravi automatski, kada se program pokrene preko dugmeta ***Start*** i ako u kodu programa nije pronađena nijedna greška. Na ovaj način se od razvojne verzije programa, koja ima ekstenziju "*.sln*", pravi izvršna verzija programa, koja ima ekstenziju "*.exe*". Programa sam izgeneriše direktorijum **bin**, unutar njega direktorijum **Debug**. U direktorijumu **Debug** se izgeneriše fajl, koji ima ektenziju "*.exe*". Ovaj fajl ima isto ime kao i ime projekta, koje smo dali na početku pisanja programa. Naknadno se može promjeniti ime projekta, direktorijum za pravljenje izvršne verzije programa, kao i mnogi drugi parametri. Da bi ovo uradili potrebno je prvo uraditi klik miša na ime projekta u prozoru *Solution Explorer*, nakon čega se otvara prozor prikazan na slici 2.16.

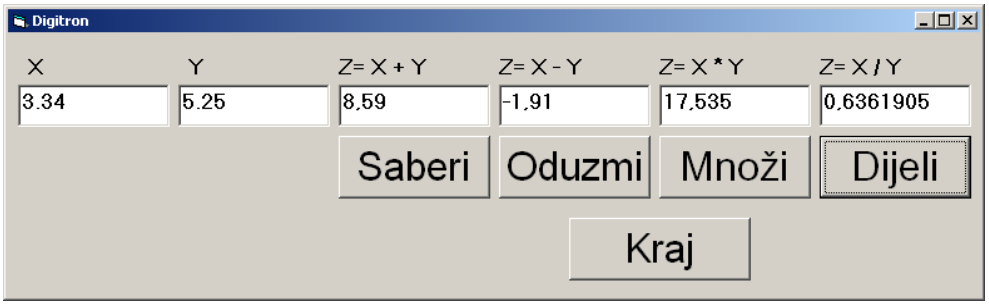


Slika 2.16. Prozor za podešavanje parametara programa

U meniju ***Application***, koji se pojavljuje uz lijevu ivicu ovog prozora, se podešava ime projekta preko parametra ***Assemblly name***, početna forma nakon pokretanja programa preko parametra ***Startup object*** i druge parametre.

U meniju ***Compile*** se podešava lokacija za smještaj izvršne verzije programa preko parametra ***Build output path***. U ovom prozoru postoji još niz parametara, koji se mogu podešavati.

Pisac programa štiti svoj autorski rad davanjem korisniku samo izvršne verzije programa. Izvršna verzija programa se pokreće samo klikom na "*.exe*" fajl i korisnik tada ne vidi kod programa. Na slici 2.17. prikazan je izgled izvršne verzije programa za računanje osnovne četiri matematičke operacije, čiji razvoj i kod smo opisali u ovom poglavlju.



Slika 2.17. Izgled izvršne verzije programa

3. TIPOVI PODATAKA

Sve komande u računaru se izvršavaju u mašinskom jeziku. Mašinski jezik radi samo sa podacima predstavljenim u obliku nizova binarnog alfabeta, odnosno sa nulama i jedinicama. Međutim, za običnog čovjeka binarni zapis je teško razumljiv i težak za izvođenje čak i osnovnih matematičkih operacija. Ljudi su naučeni da rade sa slovima i decimalnim brojevima, sa kojima većina lako izvode razne matematičke operacije. Zato su i nastali viši programski jezici, gdje se naredbe za komunikaciju sa računarem pišu preko slova i brojeva.

Skoro svaki program sadrži podatke, a podaci se obično smještaju u promjenljive (varijable). Tako promjenljivu možemo shvatiti kao kontejner za vrijednosti, koje mogu da se mijenaju tokom izvršavanja programa. Promjenljive mogu biti tekst, broj, datum, podešene osobine i druge vrijednosti. Promjenljive poprimaju vrijednosti raznih proračuna, informacija vezane za tok izvršenja programa i vrijednosti, koje se žele prikazati kao izlaz u nekom objektu ili se šalju preko nekog interfejsa. Uvijek treba da se trudite da promjenljivim u programu date smislena imena, što će program učiniti čitljivijim i razumljivijim. To je posebno važno kod velikih i kompleksnih programa.

Pravila za davanje imena promjenljivim u programu *Visual Basic* su:

- Ime promjenljive mora činiti jedna riječ.
- Ime promjenljive mora početi slovom (A - Z, velika i/ili mala). Iza prvog slova može da slijedi niz slova (A-Z, velika i/ili mala), cifre (0 - 9) i znak "donja povlaka" (_).
- Specijalna slova kao što su ? - + * / \ ! @ # $ % ^ ( ) [ ] š ć , ; : . nije preporučljivo da se koristiti u imenu promjenljive.
- Ime promjenljive ne smije da bude neka od službenih riječi programa *Visual Basic*, kao što su riječi ***If***, ***Then, For, Public,*** itd.

Sljedeća konvencija nije obavezujuća, već predstavlja preporuku kako treba davati imena promjenljivim:

- Ime promjenljive treba da bude sa značenjem, koje opisuje promjenljivu, a najbolje je da bude neka imenica.
- Kod imenovanja promjenljive uobičajeno je označavanje, gde je prvo slovo u imenu veliko a sva ostala mala.
- Ime promjenljive poželjno je da ima više od jednog slova.

Da bi se olakšalo programiranje i kontrolisala upotreba memorije računara uvedeni su tipovi podataka. Tip podataka je skup vrijednosti, koje imaju određene zajedničke karakteristike prilikom izvršavanja programa. Najznačajnija od ovih karakteristika je skup operacija, koje su definisane nad vrijednostima tog tipa. Za svaki tip podataka su definisane određene operacije, ali i gotove funkcije u svakom programskom jeziku. Kapacitet RAM memorije računara je uvijek bio jedan od

glavnih ograničavajućih faktora, za brzinu rada računara. Različiti tipovi podataka zauzimaju i različit kapacitet memorije, pa zato pravilan izbor tipova podataka u programu može značajno da ubrza ili uspori rad računara. Ovo posebno dolazi do izražaja kod pisanja programa, koji imaju veliki broj promjenljivih.

Osnovni tipovi i strukture podataka se mogu se podijeliti na:

- statički skalarni tipovi (elementarni podaci koji su skalari),
- statički struktuirani tipovi,
- dinamički tipovi sa promjenljivom veličinom,
- dinamički tipovi sa promjenljivom strukturom.

Tabela 3.1. Tipovi podataka

Red. br.	Naziv tipa	Memo-rija	Vrsta podatka	Vrijednosti
1.	**Boolean**	1 bit	Logička vrijednost	True ili False
2.	**Byte**	1 byte	1 byte	0 do 255
3.	**Char**	2 byte	Karakteri	1 do 65400 znakova
4.	**String**		Karakteri	0 do približno 2 milijarde karaktera
5.	**Date**	8 byte	Datum i vrijeme	od 0:00:00 (ponoć) 1. Januar 0001. god. do 11:59:59 31. December 9999.
6.	**Integer**	4 byte	Cijeli brojevi	-2,147,483,648 do 2,147,483,647
7.	**Long**	8 byte	Cijeli brojevi	-9,223,372,036,854,775,808 do 9,223,372,036,854,775,807 (9.2...E+18)
8.	**SByte**	1 byte	Cijeli brojevi	-128 do 127
9.	**Short**	2 byte	Cijeli brojevi	-32,768 do 32,767
10.	**UInteger**	4 byte	Cijeli brojevi	0 do 4,294,967,295
11.	**ULong**	8 byte	Cijeli brojevi	0 do 18,446,744,073,709,551,615 (1.8...E+19)
12.	**UShort**	2 byte	Cijeli brojevi	0 do 65,535
13.	**Single**	4 byte	Realni brojevi sa pokretnom tačkom	-3.4028235E+38 do -1.401298E-45 za negativne brojeve; 1.401298E-45 do 3.4028235E+38 za pozitivne brojeve
14.	**Double**	8 byte	Realni brojevi sa pokretnom tačkom	-1.79769313486231570E+308 do -4.94065645841246544E-324 za negativne brojeve; 4.94065645841246544E-324 do 1.79769313486231570E+308 za pozitivne brojeve
15.	**Decimal**	16 byte	Realni brojevi	+/- 79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) bez decimalne tačke; +/-7.92281625142643375935439503 35 sa 28 mjesta iza decimalne tačke; Najmanja ne nulta vrijednost (+/-1E-28) +/-0.0000000000000000000000000001
16.	**Object**	8 byte	Svi tipovi	Proizvoljna

Pod statičkim tipovima podataka[7] (ili tipovima sa statičkom strukturom) podrazumijevamo tipove, kod kojih je unaprijed fiksno definisana unutrašnja struktura svakog podatka. Veličina podataka (koliko memorije zauzima) se fiksno definiše prije (ili u vrijeme) izvršenja programa, koji koristi podatke statičkog tipa. Statički tipovi podataka obuhvataju skalarne i struktuirane podatke.

Pod skalarnim tipovima podrazumijevamo najprostije tipove podataka čije su vrijednosti skalari, odnosno takve veličine koje se tretiraju kao elementarne cjeline i za koje nema potrebe da se dalje razlažu na prostije komponente. Osnovni skalarni tipovi podataka koji postoje u programu *Visual Basic* dati su u tabeli 3.1.

Pod struktuiranim tipovima podataka podrazumijevamo sve složene tipove podataka, koji se realizuju povezivanjem nekih elementarnih podataka u precizno definisanu strukturu. U ulozi elementarnih podataka obično se pojavljuju skalari. Primjer struktuiranog tipa podataka je zapis (eng. *Record*). Na primjer zapis **student** može da se sastoji od polja: **broj indeksa** (tip *Integer*), **ime i prezime** (tip *String*), **datum rođenja** (tip *Date*), **godina studija** (tip *Integer*) i **smjer** (tip *String*). Skalarni tipovi podataka mogu biti linearno uređeni ili linearno neuređeni. Linearno uređeni tipovi podataka su tipovi kod kojih se vrijednosti osnovnog skupa preslikavaju na jedan interval iz niza cijelih brojeva. Tada se za svaki podatak zna redni broj podatka. Stoga svaki podatak izuzev početnog ima svog predhodnika u nizu, odnosno svaki podatak ima svog slijedbenika izuzev krajnjeg.

Pod dinamičkim tipovima podataka podrazumijevamo tipove podataka, kod kojih se veličina i/ili struktura podataka slobodno mijenja u toku obrade. Kod dinamičkih tipova sa promjenljivom veličinom podrazumijevamo da je struktura podataka fiksna, ali se njihova veličina dinamički mijenja tokom obrade. Saglasno tome se dinamički mijenjaju i memorijski zahtjevi. Primjer dinamičkog tipa podataka je niz (eng. *Array*), koji može da ima promjenljiv broj elemenata, ali svi njegovi elementi moraju biti istog tipa.

Kod dinamičkih tipova sa promjenljivom strukturom unaprijed je fiksno definisan jedino princip po kome se formira struktura podataka, dok se sama konkretna struktura i količina podataka u memoriji slobodno dinamički mijenjaju.

Sve promjenljive koje će se koristiti u programu, prije njihove upotrebe moraju se eksplicitno definisati i specificirati tip podataka. Definisanje tipa promjenljive ima sljedeću sintaksu

Public [Private ili **Dim] ImePromjenljive As TipPodatka**

Obavezna riječ *Public* upotrebljava se kada se promjenljiva definiše u okviru standardnog modula i dostupna je svim procedurama u programu (globalna promjenljiva). Ključna riječ *Private* ili *Dim* piše se onda kada se promjenljiva primjenjuje samo u procedurama u kojima je i definisana (lokalna promjenljiva).

[7] Dr Jozo J. Dujmović, *Programski jezici i metode programiranja*, Akademska misao, Beograd, 2003, str 2.3.

Primjer deklarisanja lokalne (A1) i globalne (A2) promjenljive:

```
Private A1 As Integer     ili Dim A1 As Integer
Public A2 As Double
```

U program se mogu definisati konstante, odnosno vrijednosti koje se nikada neće mijenjati u programu. Konstante su korisne, jer povećavaju preglednost koda programa, smanjuje se broj mogućih grešaka u kodu i lakša promjena neke vrijednosti u čitavom programu, promjenom te vrijednosti samo na jednom mjestu. Sa konstantama se u programu radi na isti način kao sa promjenljivim, ali se konstantama ne može nikada mijenjati vrijednost u toku izvršenja programa. Konstante mogu biti deklarisane lokalno ili globalno u programu, ali je uobičajeno da se konstante deklarišu na globalnom nivou u modulu. Primjer deklarisanja dvije konstante sa imenima **Pi** i **Ocjena5**:

```
Const Pi As Double = 3.14159265
Const Ocjena5 As Integer = 5
```

3.1. LOGIČKI TIP (*BOOLEAN*)

Logičke promjenljive (eng. *Boolean*) predstavljaju najjednostavnije promjenljive. One zauzimaju i najmanje memorijskog prostora prilikom pokretanja programa. Standarni identifikatori *True* i *False* označavaju dvije moguće logičke vrijednosti: istina (1) i laž (0). Odnosno u elektrotehnici to su dva stanja uključeno (*On*) ili isključeno (*Off*). Primjenom relacionih operatora =, >, <, >=, <=, <> dobijaju se veličine logičkog tipa. Osnovne logičke operacije nad logičkim tipom podataka su:

1) *NOT* - negacija
2) *AND* - konjukcija (i)
3) *OR* - disjunkcija (ili)

Negacijom se dobija suprotna logička vrijednost i za nju vrijedi

not(False) = True
not(True) = False.

Konjukcijom dva logička izraza se dobije tačna vrijednos, samo ako obadva (ili svi izrazi ako ih ima više) izraza za rezultat imaju tačnu vrijednost i za nju vrijedi

False And False = False
False And True = False
True And False = False
True And True = True

Disjunkcijom dva logička izraza se dobije tačna vrijednos, ako bilo koji od dva (ili samo jedan izrazi ako ih ima više) izraza za rezultat imaju tačnu vrijednost

False Or False = False
False Or True = True
True Or False = True
True Or True = True

Izrazi u kojima se primjenjuju logičke promjenljive nazivaju se logički izrazi. Primjer nekih ispravno napisanih logičkih izraza u strukturi grananja *If/Then/Else*

```
(A=B And A>5) Or B<1000
(A>B Or A>5) And B<1000
(A=B Or A>5) And (A<100 Or B<1000).
```

Kada se upotrebljavaju logički operatori u složenijim izrazima, evo šeme prioriteta među njima:

- NOT ima najveći prioritet među operatorima.
- AND ima srednji nivo prioriteta među operatorima.
- OR ima najniži prioritet među operatorima.

Slijedi pregled prioriteta svih do sada spomenutih operatora:

1) NOT ima najveći prioritet među operatorima.
2) /, *, \, mod, AND
3) + , -, OR
4) =, <, >, <=, >=, <> imaju najniži prioritet među operatorima.

3.2. CIJELOBROJNI TIP (*INTEGER*)

Cjelobrojni tip podataka (eng. *Integer*) predstavlja najjednostavniji brojni tip podataka. On predstavlja podskup skupa cijelih brojeva. Definicija sadrži broj, koji govori sa koliko bita se predstavlja taj decimalni broj u binarnom obliku, što automatski određuje minimalni (početni) i maksimalni (krajnji) broj, koji ovaj tip podatka može da uzme. U programskom jeziku *Visual Basic* 2013 postoje sljedeće definicije cjelobrojnog tipa podataka:

1) ***Integer*** je 32-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-2147483648 do 2147483647).
2) ***Long*** (*long integer*) je 64-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-9223372036854775808 do 9223372036854775807 (9.2...E+18)).
3) ***SByte*** je 8-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-128 do 127).
4) ***Short*** je 16-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-32768 do 32767).
5) ***UShort*** je 16-bitno predstavljanje, a ovaj tip podataka sadrži samo pozitivne cjelobrojne brojeve u rasponu (0 do 65535).
6) ***UInteger*** je 32-bitno predstavljanje, a ovaj tip podataka sadrži samo pozitivne cjelobrojne brojeve u rasponu (0 do 4294967295).
7) ***ULong*** je 64-bitno predstavljanje, a ovaj tip podataka sadrži samo pozitivne cjelobrojne brojeve u rasponu (0 do 18446744073709551615).

Na primjer, cjelobrojne vrijednosti su: 1202; -2801; +802. Znak + je ispred brojne vrijednosti proizvoljan. Ako ne postoji nikakav predznak pretpostavlja se da

je konstanta pozitivna. Kada se piše program za svaki objekat i promjenljivu, a na osnovu maksimalno očekivanog broja, koji se može pojaviti, treba izabrati odgovarajući tip cijelog broja. Tip se bira na osovu predstavljenih raspona brojeva, kako se memorija ne bi nepotrebno trošila i time usporavalo izvršenje programa.

Kao i na svakom drugom standardnom tipu, tako je i na tipu *Integer* definisan tačno određeni skup operacija. To su u ovom slučaju uobičajene cjelobrojne aritmetričke operacije: sabiranje (+), oduzimanje (-), množenje (*), cjelobrojno dijeljenje (\ ) i ostatak cjelobrojnog dijeljenja (MOD). Primjer:

Cjelobrojno djeljenje	Ostatak cjelobrojnog djeljenja	Realno dijeljenje
7 \ 2 = 3	7 MOD 2 = 1	7 / 2 = 3.5
7 \ 3 = 2	7 MOD 3 = 1	7 / 3 = 2.333
15 \ 4 = 3	15 MOD 4 = 3	15 / 4 = 3.75

U programskom jeziku *Visual Basic* postoji dosta gotovih funkcija, koje rade sa ovim tipom podataka. Jednostavni izrazi dodjele vrijednosti se pišu:

X = 55 (dodjela brojne vrijednosti)
Y = X + 3 (sabiranje vrijednosti promjenljive i broja)
Z = X + Y (sabiranje vrijednosti dvije promjenljive)
X = X + 1 (uvećanje trenutne vrijednosti promjenljive za brojnu vrijednost)

Upotrebljavajući jednostavne izraze mogu se računati i složeniji izrazi, ali u više koraka sa više naredbi. Na primjer, ako želimo sabrati vrijednosti X, Y, Z te rezultat pridružiti cjelobrojnoj promenljivoj SUM, možemo napisati:

SUM = X + Y
SUM = SUM + Z

ali logičnije je da se to napiše u obliku:

SUM = X + Y + Z

i tako dobije isti rezultat koristeći samo jednu naredbu umjesto dvije. Redosljed promenljivih u predhodnom izrazu ne utiče na rezultat. Ako imamo izraz u obliku:

R = X + Y * Z

moramo biti puno pažljiviji! Pretpostavimo da su promenljivama X, Y, Z pridruženi podaci 2, 3, 5. Rezultat može biti različit. Ako bi se prvo uradilo sabiranje rezultat bi bio 25, a ako se prvo uradi množenje rezultat bi bio 30. Znači potrebno je istaći pravila o prioritetu operatora. Prioritet aritmetičkih operatora u programu *Visual Basic* je da se prvo radie operacije: *, /, \, mod, a tek onda + , -.

Evo nekoliko primjera za vježbu:

```
a) 7*7-4*3=?                    b) 9 * 5 + 7 \ 3 =?
c) 1.5*1.1+4.5/9.0-0.1 =?       d) 3.6/1.2*3.0 = ?
```

Ukoliko je u izrazu nekoliko operatora iste grupe prioriteta, a želimo da se ipak izvrši množenje prije dijeljenja, upotrebljavamo zagrade. Primjer:

```
a) R=36/(12 * 3)              b) R=3 * (8 + 56) * 2
```

U ovim izrazima se daje prednost pri izračunavanju operatorima u zagradama.

Dozvoljena je upotreba više zagrada, pri čemu treba voditi računa da se svaka otvorena zagrada mora zatvoriti.

Primjer:

```
R=4 * (5 + 4 * (6 - 2))+ ((X+Y)/Z).
```

Provjeriti da li se dobije isti rezultat u sljedećim jednakostima.

```
a)(5+3)/2 = 5/2+3/2          b) (5+3)\2 = 5\2+3\2
```

3.3. REALNI TIP (*REAL*)

Realni tip podataka (eng. *Real*) predstavlja podskup skupa realnih brojeva. Koji je to tačno podskup, zavisi od konkretne definicije ovog tipa u programskom jeziku *Visual Basic*. Definicija sadrži broj, koji govori sa koliko bita se predstavlja taj decimalni broj u binarnom obliku, što automatski određuje minimalni (početni) i maksimalni (krajnji) broj koji ovaj tip podatka može da uzme. Za ovaj tip podataka je bitno da se uvijek navede broj značajnih cifri. Značajna cifra predstavlja broj, koji pokazuje koliko će realni broj imati prikazanih cifri iza decimalne tačke. Ovo je potrebno uraditi kako nam program ne bi automatski dodijelio 4, 8 ili 16 značajnih cifri iza decimalne tačke realnog broja. U programskom jeziku *Visual Basic* 2013 postoje sljedeće definicije realnog tipa podataka:

1) ***Single*** (*single-precision floating-point*) je 32-bitno predstavljanje, a ovaj tip podataka sadrži realne brojeve sa pokretnom decimalnom tačkom u rasponu (-3.4028235E+38 do -1.401298E-45 za negativne brojeve; 1.401298E-45 do 3.4028235E+38 za pozitivne brojeve), sa približno 8 značajnih decimalnih mjesta).
2) ***Double*** (*double-precision floating-point*) je 64-bitno predstavljanje, a ovaj tip podataka sadrži realne brojeve sa pokretnom decimalnom tačkom u rasponu (-1.79769313486231570E+308 do -4.94065645841246544E-324 za negativne brojeve; 4.94065645841246544E-324 do 1.79769313486231570E+308 za pozitivne brojeve), sa približno 16 značajnih decimalnih mjesta).
3) ***Decimal*** je 128-bitno predstavljanje, a ovaj tip podataka sadrži cijele brojeve u rasponu (+/-79 228 162 514 264 337 593 543 950 335), realne brojeve u rasponu (+/-7.9228162514264337593543950335), sa 28 značajnih decimalnih mjesta). Pri tome je najmanja vrijednost pozitivnog/negativnog broja (+/-0.0000000000000000000000000001).

Na primjer, realne vrijednosti su: 8.5; -4.73; +3.777; -7.257. Obratite posebno pažnju da se kod realnih brojeva cjelobrojna vrijednost, od decimalne vrijednosti odvaja sa tačkom ("."), a ne sa zapetom (","). Realna vrijednost ne smije počinjati ili završavati decimalnom tačkom, na Primjer: "459." i ".357". Umjesto toga treba da se napiše "459.0" i "0.357". Za ovaj tip podataka definisane su osnovne aritmetričke operacije: sabiranje (+), oduzimanje (-), množenje (*) i dijeljenje (/). U programskom jeziku *Visual Studio* postoji dosta funkcija, koje rade sa ovim tipom podataka.

Neke od funkcija koje se koriste za konverziju realnog u cijeli broj su:

1) ***Math.Round(****expression*[***,*** *numdecimalplaces*]**)** - zaokruživanje realnog broja ili izraza (*expression*) na bližu cjelobrojnu vrijednost, ako nije naveden opcioni parametar (*numdecimalplaces*). Ako je opcioni parametar naveden, onda se ulazni realni broj zaokružuje na realan broj sa brojem decimalnih mjesta navedenim u ovom parametru.
2) ***Int***(*number*) - uzimanje samo cjelobrojne vrijednosti realnog broja, pri čemu se zaokruživanje radi na manju cjelobrojnu vrijednost.
3) ***Fix***(*number*) - uzimanje samo cjelobrojne vrijednosti realnog broja, pri čemu se zaokruživanje radi na manju cjelobrojnu vrijednost, ali po apsolutnoj vrijednosti za negativne brojeve.
4) ***Math.Sqrt***(*number*) – računanje kvadratnog korjena realnog ili cjelobrojnog broja, a rezultat može da bude realni boj.

Primjer:

Round(23.0), daje 23. Round(23.1), daje 23. Round(23.9), daje 24. Round(23.5), daje 24. Round(24.5), daje 24.	Round(-23.5), daje -24. Round(29.1234,2) daje 29.12; Round(29.1256,2) daje 29.13. Round(-29.1234,2) daje -29.12; Round(-29.1256,2) daje -29.13.
Int(23.0), daje 23. Int(23.1), daje 23. Int(23.9), daje 23. Int(23.5), daje 23. Int(-23.5), daje -24. Int(-24.9), daje -25.	Fix(23.0), daje 23. Fix(23.1), daje 23. Fix(23.9), daje 23. Fix(23.5), daje 23. Fix(-23.5), daje -23. Fix(-24.9), daje -24.

3.4. ZNAKOVNI TIP (*STRING*)

Znakovni tip podataka (eng. *String*) predstavlja skup: malih i velikih slova engleskog alfabeta, cifri od 0 do 9, specijalnih karaktera i kontrolnih znakova. U programu *Visual Basic* 2013 promjenljive ovog tipa podatka mogu imati od 0 do približno 2 milijarde znakova. Ovaj tip podataka je značajan, jer kada korisnik komunicira sa računarom preko ulaznih i izlaznih uređaja, vrlo je bitno da se podaci pojavljuju u formi, koja je čitljiva za čovjeka. Da bi se tekst koji se želi dodjeliti nekoj promjenljivoj razlikovao od naziva promjenljivih, naziva objekata i raznih komandi potrebno je da taj tekst stoji naveden između dvostrukih navodnika (apostrofa) (primjer: "tekst koji se navodi").

ASCII kod (*Amjerican Standard Code for Information Interchange*[8]) je jedan od najznačanijih načina predstavljanja znakovnog tipa podataka. *ASCII* kod je

[8] http://frank.harvard.edu/aoe/images/t10r3.pdf

poznat u 7-bitnoj verziji sa 128 znakova i 8-bitnoj verziji sa 256 znakova. *ASCII* kod sa 128 znakova prikazan je u tabeli 3.2.

Tabela 3.2. ASCII kod

non-printing					printing			printing			printing		
Name	Control char	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
null	ctrl-@	NUL	00	00	SP	20	32	@	40	64	‘	60	96
start of heading	ctrl-A	SOH	01	01	!	21	33	A	41	65	a	61	97
start of text	ctrl-B	STX	02	02	"	22	34	B	42	66	b	62	98
end of text	ctrl-C	ETX	03	03	#	23	35	C	43	67	c	63	99
end of xmit	ctrl-D	EOT	04	04	$	24	36	D	44	68	d	64	100
enquiry	ctrl-E	ENQ	05	05	%	25	37	E	45	69	e	65	101
acknowledge	ctrl-F	ACK	06	06	&	26	38	F	46	70	f	66	102
bell	ctrl-G	BEL	07	07	’	27	39	G	47	71	g	67	103
backspace	ctrl-H	BS	08	08	(	28	40	H	48	72	h	68	104
horizontal tab	ctrl-I	HT	09	09	)	29	41	I	49	73	i	69	105
line feed	ctrl-J	LF	0A	10	*	2A	42	J	4A	74	j	6A	106
vertical tab	ctrl-K	VT	0B	11	+	2B	43	K	4B	75	k	6B	107
form feed	ctrl-L	FF	0C	12	,	2C	44	L	4C	76	l	6C	108
carriage return	ctrl-M	CR	0D	13	-	2D	45	M	4D	77	m	6D	109
shift out	ctrl-N	SO	0E	14	.	2E	46	N	4E	78	n	6E	110
shift in	ctrl-O	SI	0F	15	/	2F	47	O	4F	79	o	6F	111
data line escape	ctrl-P	DLE	10	16	0	30	48	P	50	80	p	70	112
device control 1	ctrl-Q	DC1	11	17	1	31	49	Q	51	81	q	71	113
device control 2	ctrl-R	DC2	12	18	2	32	50	R	52	82	r	72	114
device control 3	ctrl-S	DC3	13	19	3	33	51	S	53	83	s	73	115
device control 4	ctrl-T	DC4	14	20	4	34	52	T	54	84	t	74	116
neg acknowledge	ctrl-U	NAK	15	21	5	35	53	U	55	85	u	75	117
synchronous idle	ctrl-V	SYN	16	22	6	36	54	V	56	86	v	76	118
end of xmit block	ctrl-W	ETB	17	23	7	37	55	W	57	87	w	77	119
cancel	ctrl-X	CAN	18	24	8	38	56	X	58	88	x	78	120
end of medium	ctrl-Y	EM	19	25	9	39	57	Y	59	89	y	79	121
substitute	ctrl-Z	SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
escape	ctrl-[	ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
file separator	ctrl-\	FS	1C	28	<	3C	60	\	5C	92	|	7C	124
group separator	ctrl-]	GS	1D	29	=	3D	61	]	5D	93	}	7D	125
record separator	ctrl-^	RS	1E	30	>	3E	62	^	5E	94	~	7E	126
unit separator	ctrl-_	US	1F	31	?	3F	63	_	5F	95	DEL	7F	127

U programskom jeziku *Visual Basic* 2013 postoji veliki broj gotovih funkcija koje rade sa ovim tipom podataka:

1) ***LCase (UlazniString)*** - vrši konverziju svih velikih u mala slova.
[LCase("Mira%+#123"), daje "mira%+#123"]

2) ***UCase(UlazniString)*** - vrši konverziju svih malih u velika slova.
[UCase("Mira%+#123"), daje "MIRA%+#123"]

3) ***Left(UlazniString, BrojSlova)*** - od zadatog teksta izdvaja pod tekst određene dužine (*BrojSlova*) sa lijeve strane teksta od prvog znaka.
[Left("Teodora je mala",6), daje "Teodor"]

4) ***Right(UlazniString, BrojSlova)*** - od zadatog teksta izdvaja pod tekst određene dužine (*BrojSlova*) sa desne strane teksta od krajnjeg znaka.
[Right("Teodora je mala",4), daje "mala"]

5) ***Mid(UIazniString,Pozicija[,Dužina])*** - od zadatog teksta izdvaja pod tekst zadanog broja karaktera (*Dužina*), od startnog znaka (*Pozicija*).
[Mid("Teodora je mala",0,6), daje "Teodor"]
[Mid("Teodora je mala",9,2), daje "je"]

6) ***Len(str)*** - daje broj znakova u zadatom tekstu.
[Len("Mihailo"), daje 7] i [Len("123456"), daje 6]

7) ***InStr([start,] string1, string2 [, compare])*** - Vraća poziciju prvo pronađenog teksta (*string2*) u zadatom tekstu (*string1*). Poređenje počinje od karaktera, koji je zadat preko opcionog parametra *start*.
[InStr("Mirko je već", "je"), daje 7] i InStr(8, "Mirko je već", "e"), daje 10]

8) ***Trim(string)*** - pravi tekst bez praznih karaktera na početku i kraju teksta.
[Trim (" Teodora je mala "), daje "Teodora je mala"]

9) ***LTrim(string)*** - pravi tekst bez praznih karaktera na početku teksta.
[LTrim (" Teodora je mala "), daje "Teodora je mala "]

10) ***RTrim(string)*** - pravi tekst bez praznih karaktera na kraju teksta.
[RTrim (" Teodora je mala "), daje " Teodora je mala"]

11) ***StrRev(str)*** - daje tekst sa obrnutim redoslijedom znakova.
[strRev("Mihailo") daje "oliahiM")]

12) ***Chr*(broj)** - vrši konverziju broja u tekst, prema ASCII standardu.
[Chr(65), daje slovo A] [Chr(97), daje slovo a]
[Chr(62), daje simbol >] [Chr(37), daje simbol %]

13) ***Asc*(slovo)** - vrši konverziju prvog slova teksta u broj, prema ASCII standardu.
[Asc(A), daje broj 65] [Asc (a), daje broj 97]
[Asc (aprl), daje broj 97] [Asc (Mirko), daje broj 77]

14) ***Riplace*(***expression***, *find*, *replacewith*[, *start*[, *count*]])** - pravi zamjenu određenih znakova (*find*) sa novim znakovima (*replacewith*) u zadatom tekstu (*expression*). Zamjena se u zadatom tekstu vrši od početnog karaktera (*start*), ali samo onoliko puta koliko je zadato preko parametra (*count*). Ako

parametar *count* ima vrijednost -1 onda se rade sve moguće zamjene u zadatom tekstu.
[Riplace ("Mihailo je veliki.", "je", "nije"), daje "Mihailo nije veliki."]
[Riplace ("Mihailo nije veliki.", "i", "$",-1), daje "M$ha$lo n$je vel$k$."]
[Riplace ("Mihailo nije veliki.", "i", "$", 4, 3), daje "Miha$lo n$je vel$ki."]

Visual Basic 2013 ima mogućnost prikaza izlaznog stringa u željenoj formi uz upotrebu funkcije *Format*, koja ima sljedeću sintaksu

Format(*expression*[, *format*[, *firstdayofweek*[, *firstweekofyear*]]])

gdje je:

- *expression* tekst koji se želi formatirati
- *format* način na koji se uneseni tekst želi formatirati.
- *firstdayofweek* predstavlja dan u sedmici i mogu biti brojevi od 1 do 7, kod formatiranja teksta koji predstavlja datum.
- *firstweekofyear* predstavlja označavanje prve sedmici, kod formatiranja teksta koji predstavlja datum.

Primjeri korišćenja funkcije Format:

- < prikazuje sve malim slovima

AA = Format ("POSAO", "<@@@@@") ; daje "posao".

- > prikazuje sve velikim slovima

AA = Format ("Ilija Rosic", ">@@@@@@@@@@@") ; daje "ILIJA ROSIC".

- prikaz broja sa određenim brojem decimalnih mjesta

AA = Format(5459.4, "##,##0.00") ; daje "5,459.40".
AA = Format(334.9, "###0.00") ; daje "334.90".

- prikaz broja kao procentualna vrijednost (množenje brojne vrijednosti sa 100 i dodavanje znaka **%** odmah iz dobijenog broja)

AA = Format(5.12, "0.00%") ; daje "512.00%".
AA = Format(0.085, "0.00%") ; daje "8.50%".

- prikaz broja sa novčanom jedinicom

AA = Format(6.268, "0.00 KM") ; daje "6.27 KM".
AA = Format(12468.268, "$#,##0.00") ; daje "$12,468.27".

- prikaz vrijednosti vremena i datuma

AA = Format(Time, "Long Time") ; daje trenutno sistemsko vrijeme
AA = Format(#17:04:23#, "h:m:s") ; daje "17:4:23".
AA = Format(#17:04:23#,, "hh:mm:ss AMPM") ; daje "05:04:23 PM".
AA = Format(#January 28, 2003#, "dddd, mmm d yyyy") ;
daje "Wednesday, ' Jan 28 2003".

3.5. DATUMSKI TIP (*DATE*)

Datumski tip podataka (eng. *Date*) se koristi za prikaz datuma i vremena u raznim formatima. Ovaj tip podataka u suštini predstavlja realni broj, koji se predstavlja sa 64 bita. Ovaj realni broj predstavlja broj proteklih sekundi od 1. januara 0001. godine poslije Hrista, prema *UTC* (*Universal Coordinated Time*) vremenu. *UTC* vrijeme predstavlja korekciju lokalne vremenske zone, koja je podešena na računaru (*Control Panel => Date and Time => Time Zone => (GMT +01:00) Belgrade, Ljubljana*), u odnosu na *GMT* (*Greenwich Mean Time*) vrijeme. U našoj zemlji *UTC* vrijeme iznosi +1 sat, jer naša vremenska zona prednjači u odnosu na *Greenwich* kod Londona za jedan sat. Manje vremenske jedinice od dana (sati, minute i sekunde) predstavljaju decimalni dio realnog broja.

Na primjer,
- 1 sat iznosi 1/24 dijela dana, odnosno = 0,04166666667.
- 3 sata iznosi 3/24 dijela dana, odnosno = 0,125.
- 1 minuta iznosi 1/(24*60) dijela dana, odnosno = 0,00069444444.
- 1 sekunda iznosi 1/(24*60*60) dijela dana, odnosno = 0,000015741.

Programski jezik *Visual Basic* 2013 ima gotove funkcije, koje omogućuju razne operacije sa datumom i vremenom. Neke od tih funkcija su:

1) ***Now()*** - uzima sistemski datum i vrijeme sa računara, a na svom izlazu daje realni broj u formatu *Real64*. Dobijeni realni broj predstavlja broj sekundi od 1. januara 0001. godine, do trenutka aktiviranja ove funkcije u programu.
2) ***TimeOfDay()*** – uzima sistemsko vrijeme sa računara.
3) ***Today()*** – uzima sistemski datum sa računara.
4) ***DateValue(string)*** - koja uzima string i od njega pravi datum. [DateValue("7/21/02 5: 25:35 PM"), daje #7/21/02#]
5) ***TimeValue(string)*** - koja uzima string i od njega pravi vrijeme. [DateValue("7/21/02 5: 25:35 PM"), daje #5:25:35 PM#]
6) ***Day(***Datum***)*** - daje broj dana u trenutnom mjesecu, kada je ulazna vrijednost u ovu funkciju datum. Opseg vrijednosti izlaza ove funkcije je od 1 do 31.
7) ***Month(***Datum***)*** - daje broj mjeseca u trenutnoj godini. Opseg vrijednosti izlaza ove funkcije je od 1 do 12.
8) ***Year(***Datum***)*** - daje broj godina u trenutnom datumu.
9) ***Hour(***Datum***)*** - izdvaja dio datuma koji sadrži sat.
10) ***Minute(***Datum***)*** - izdvaja dio datuma koji sadrži minute.
11) ***Second(***Datum***)*** - izdvaja dio datuma koji sadrži sekunde.
12) ***DateSerial(y,m,d)*** - formira datum od 3 ulazna parametra. Parametar ***d*** je dan u mjesecu. Parametar ***m*** je mjesec u godini. Parametar ***y*** je godina.

13) ***TimeSerial(h,m,s)*** - formira vrijeme od 3 ulazna parametra. Parametar ***h*** je broj sati. Parametar ***m*** je broj minuta. Parametar ***s*** je broj sekundi.
14) ***DateDiff ("Tip jedinice", Datum1, Datum2)*** - daje broj vremenskih jedinica između dva datuma. Gdje je "Tip jedinice" predstavljen u obliku stringa, koji simbolički prikazuje vremensku jedinicu (godinu, mjesec, dan, sat, minut, sekund). Vremenske jedinice baziraju se na razlici između dva datuma Datum2 - Datum1.

Primjer:
tekst1 = DateSerial(2000, 2, 12)
tekst2 = DateSerial(2003, 1, 28)

Text1.Text = DateDiff("d", tekst1, tekst2), daje rezultat 1081 dana.
Text2.Text = DateDiff("h", tekst1, tekst2), daje rezultat 25944 sati.
Text3.Text = DateDiff("m", tekst1, tekst2), daje rezultat 35 mjeseci.

U sljedećoj tabeli su date skraćenice za vremenske tipove jedinica, koje se koriste u gotovim funkcijama, koje rade sa ovim tipom podataka.

Tabela 3.3. Skraćene oznake za vremenske tipove jedinica

Skraćenica	Tip jedinice	Skraćenica	Tip jedinice
"s"	sekunda	"w"	sedmica
"n"	minuta	"m"	mjesec
"h"	sat	"q"	kvartal 3 mjeseca
"d"	dan	"y"	godina

3.6. NEODREĐENI TIP (*OBJECT*)

Object je neodređeni tip podataka, koji postoji u programskom jeziku *Visual Basic* 2013. Kada se neka promjenljiva deklariše kao *Object*, to znači da ona može da poprimi, bilo koji do sada opisani tip podatka. Promjenjva deklarisana kao ovaj tip podatka, se u toku izvršenja programa prilagođava onom tipu podatka, koji prvi dođe na njen ulaz. Međutim, ne treba pretjerivati sa korišćenjem ove promjenljive, pogotovo, kada smo sigurni da je promjenljiva tačno poznatog tipa podataka.

Ovaj tip podatka se posebno koristi kod deklarisanja promjenljivih, koje se koriste za povezivanje sa drugim programima instalisanim na računaru.

3.7. STATIČKI NIZOVNI TIP (*ARRAY*)

Statički nizovi služe za realizaciju jednoindeksnih i višeindeksnih nizova. Skup ***S*** iz grupe elemenata istog tipa podataka i koji imaju isti smisao, naziva se niz. Elementi niza se u programu obično obrađuju na sličan ili isti način. Elementi niza mogu biti i objekti, koji pripadaju jednoj formi i kojima se indeksi dodjeljuju redoslijedom njihovog postavljanja na formu. Svaki element niza ima svoj indeks iz skupa prirodnih brojeva N_0 (0,1,2, ...). Dakle, niz obrazuju elementi $S = f(n)$, gdje se brojevi *n* nazivaju indeksi niza.

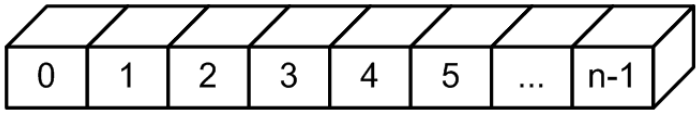
Ako se skalarna promjenljiva zamisli kao jedna fioka, koja ima svoje jedinstveno ime i služi za smještaj određenog tipa podataka. Niz se može zamisliti kao jedna fioka sa više pregrada, pri čemu svaka pregrada ima svoju adresu (kao na slici 3.1.), ali fioka ima samo jedno ime i služi za smještaj samo jednog tipa podataka.

Nizovi su nastali radi smanjenja broja deklarisanih promjenljivih, povećanja opštosti koda i lakšeg pravljenja izmjena u programu. Nizovi nastoje da grupno opišu istorodne podatke, tj. da im se zajedničke osobine istaknu samo jednom. Na taj način se izbjegava posebno opisivanje svakog podatka. U tom slučaju cijeloj grupi podataka (nizu) daje se zajedničko ime. Da bi se mogla izvršiti obrada i pristup podacima iz grupe, potrebno je prethodno izvršiti deklaraciju niza, odnosno navesti ukupan broj članova niza i identifikatore (indekse) pojedinih podataka. Prema tome, deklaracija niza promjenljivih sastoji se od: imena, broja elemenata i tipa podatka i ima sljedeću formu.

```
Dim ImeNiza (Indeks1, [Indeks2],...) As TipPodatka
```

Obavezna riječ *Public* upotrebljava se kada se niz definiše kao globalna promjenljiva u okviru standardnog modula i dostupan je svim procedurama i formama u programu. Ključna riječ *Dim* piše se samo onda, kada se niz primjenjuje u procedurama u kojima je i definisan. Za ime niza obično se uzima riječ ili skup riječi (pišu se sastavljeno), koje asociraju na vrijednosti koje niz može da ima. Broj promjenljivih (Indeks) ili dimenzija niza je, u stvari, indeks posljednjeg člana niza. Budući da je indeks prvog člana niza 0, osim ako se drugačije ne definiše, to znači da je ukupan broj članova niza za jedan veći od indeksa posljednjeg člana slika 3.1.

```
Dim Niz1(14) As Integer      ' 15 elemenata.
Dim Niz2(21) As Double       ' 22 elemenata.
```



Slika 3.1. Jednoindeksni niz

Međutim, u brojnim praktičnim primjerima postoji potreba da prvi indeks niza bude broj 1, pa je tada ukupan broj promjenljivih identičan indeksu posljednjeg člana niza.

```
Dim Niz3(1 to 20) As Single  ' 20 elemenata.
```

Tip niza određuje se pomoću simbola ili pomoću propisane riječi, onako kako je to opisano u tabeli 3.1., a zavisno od vrste podataka, koji će biti locirani u memorijske promjenljive. U slučaju da ne postoji saznanje o tipu podatka ili da promjenljive mogu da poprimaju podatke različitih tipova, tada se za format tipa podatka uzima opcija *As Object*. Radi bržeg rada i efikasnijeg korišćenja memorijskog prostora, svrsishodnije je da se dodijeli stvarni tip promjenljivim iz niza, naravno, ako je to moguće.

Nizovi mogu da budu statički (konstantni ili fiksni), ako je broj njihovih elemenata unaprijed poznat i dinamički (varijabilni), ako se njihova dimenzije naknadno određuje. Međutim, prilikom određivanja statističkih funkcija, kao na primjer, određivanje srednje vrijednosti ili standardne devijacije za bilo koji uzorak slučajnih promjenljivih (primjer - statističke funkcije), broj promjenljivih nije unaprijed poznat, nego tek onda kada se pristupi izračunavanju konkretnog uzorka.

Tada se u standardnom modulu niz definiše na sljedeći način

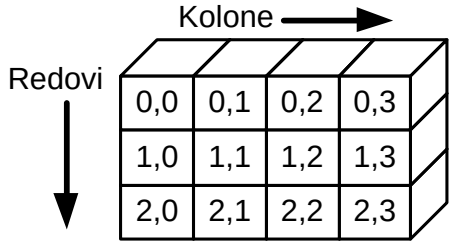
```
Public Promjenljiva1 () As Singl
```

Zagrade, bez navedenog sadržaja, imaju ulogu da označe da je niz dinamičke prirode i da će njegova dužina, odnosno broj elemenata, biti naknadno određeni. Podaci koji pripadaju promjenljivim, u okviru jednog niza, memorišu se u susjedne lokacije memorijskog prostora. Za vrijeme izvršavanja programa elementi niza nalaze se u sistemskoj memoriji *(RAM*-u*)*. Zbog toga, prilikom rada sa nizovima, ne treba uzimati nizove sa ekstremno velikim brojem elemenata, jer može doći do prezasićenja sistemske memorije i samim tim do usporavanja rada računara.

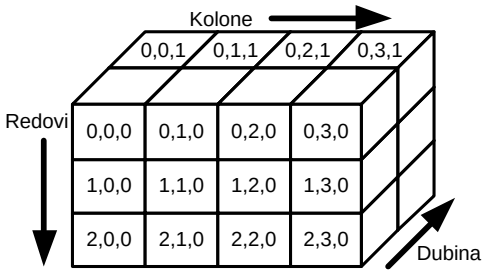
Osnovna prednost u radu sa nizovima sastoji se u efikasnosti izvršavanja programa i u preglednosti izrade programskog koda. Promjenljive deklarisane kao niz mogu se obrađivati grupno, obično uz upotrebu petlji: *For... Next* i *Do While (Until)... Loop*. Dakle, umjesto da se naredbe navode pojedinačno za obradu svake memorijske promjenljive, one se pisu samo jednom uz varijaciju indeksa i odnose se na sve promjenljive. Način rada sa nizovima može se sagledati na primjeru unosa i prikaza niza slučajnih brojeva upotrebom petlje *For... Next.*

```
Dim i As Integer ' Promjenljiva za brojanje.
Dim Niz1(8) As Single
For i = 0 To 8
  Niz1(i) = 10*Rnd
Next i
```

U programiranju se pored jednodimenzionih koriste i višedimenzioni nizovi. Najčešće korišćeni višedimenzioni nizovi u praksi su dvodimenzioni nizovi, čija je struktura prikazana na slici 3.2.



Slika 3.2. Dvodimenzioni niz



Slika 3.3. Trodimenzioni niz

Dvodimenzioni niz je najlakše shvatiti, kao dvodimenzionu matricu (tabelu) *T2(i, j)*, koja se sastoji se od *i* - redova i *j* - kolona. Dvodimenzionini niz se obično popunjava pomoću dvije *For* petlje. Jedna *For* petla (spoljna) se obično koristi za promjenu redova tabele (*i*), a druga *For* petlja (unutrašnja) za promjenu kolona tabele (*j*). Ukupan broj ciklusa izvršenja ove dvije *For* petlje se dobije, kao umnožak broja ciklusa spoljne i broja ciklusa unutrašnje petlje. Deklaracija dvodimenzionog niza se može vršiti na sljedeći način

```
Dim Niz1(4,6) As Integer             '35=5*7 elemenata.
Dim Niz2(1 To 3, 4) As Double        '15=3*5 elemenata.
Dim Niz3(2, 1 To 4) As Double        '12=3*4 elemenata.
Dim Niz4(1 To 3, 1 To 5) As Double '15=3*5 elemenata.
```

Prvi način deklaracije niza (Niz1) se koristi kada tabela ima nulti red i nultu kolonu.

Način popunjavanja dvodimenzionog niza upotrebom dvije petlje *For... Next*, dat je u sljedećem primjeru.

```
Dim i, j As Integer ' Promjenjljive za brojanje.
Dim Niz2(2,3) As Integer
For i = 0 To 2      ' Spoljna petlja za redove
  For j = 0 To 3    'Unutrasnja petlja za kolone
    Niz2(i, j) = i * 10 + j
  Next J
Next I
```

Struktura trodimenzinog niza prikazana je na slici 3.3. Trodimenzioni niz je najlakše shvatiti, kao trodimenzionu matricu (koja ima obik kocke) *T(i, j, k)*, koja se sastoji se od *i*- redova, *j* - kolona i *k*- dubina matrice. Trodimenzioni niz se obično popunjava pomoću tri *For* petlje. Jedna *For* petla (spoljna) se koristi za promjenu redova tabele (*i*), druga *For* petlja (unutrašnja) za promjenu kolona tabele (*j*) i treća *For* petlja (unutrašnja) za promjenu dubine kocke (*k*). Ukupan broj ciklusa izvršenja ove tri *For* petlje se dobije, kao umnožak broja ciklusa sve tri petlje. Deklaracija trodimenzionog niza se može vršiti na sljedeći način

```
Dim Niz1(4,5,2) As Integer      ' 90=5*6*3 elemenata.
Dim Niz2(1 To 3, 4,5) As Double ' 90=3*5*6 elemenata.
Dim Niz3(4, 1 To 4,2) As Double ' 60=5*4*3 elemenata.
Dim Niz4(1 To 3, 1 To 4,3) As Double
                                 '48=3*4*4 elemenata.
```

Način popunjavanja troindeksnog niza upotrebom tri petlje *For... Next*, dat je u sljedećem primjeru.

```
Dim i, j, k As Integer ' Promjenjljive za brojanje.
Dim Niz3(2,3,1) As Integer
For i = 0 To 2        ' Spoljna petlja za redove
  For j = 0 To 3      ' Unutrasnja petlja za kolone
    For k = 0 To 1    ' Unutrasnja petlja za dubinu
      Niz3(i, j, k) = i * 10 + j + k
    Next k
  Next j
Next i
```

Važno je napomenuti da prilikom rada sa nizovima nije neophodno da se popune svi elementi niza. Zbog toga je prilikom deklaracije nizova poželjno da se ostavi nekoliko elemenata u rezervi, kako bi se niz naknadno mogao proširivati.

Postoji komanda ***ReDim,*** koja se koristi za deklaraciju ili predefinisanje broja elemenata ranije deklarisanog niza. Pokretanjem izvršenja ove komande brišu se sve vrijednosti, koje je do tada u toku izvršenja programa niz imao. Sintaksa ove komande je:

```
ReDim ImeNiza (Indeks1, Indeks2,...)
```

4. NAREDBE SELEKCIJE I ITERACIJE

Skup naredbi čije se izvršavanje odvija u onom redoslijedu u kome su i navedene naziva se sekvencija. U programima se pojavljuju i tačke grananja, odnosno mjesta u kojima se vrši izbor odgovarajuće odluke, poslije kojih izvršavanje programa je moguće nastaviti u raznim smjerovima, zavisno od postavljenih uslova. Postupak izbora jednog od mogućih smjerova naziva se selekcija. U programu sekvencija čini jednu logičku cjelinu. U praksi se često javlja potreba da se jedan sekvencija naredbi izvrši vise puta. Višestruko izvršavanje jedne sekvencije naziva se iteracija (ponavljanje).

4.1. NAREDBE SELEKCIJE

Programski jezik *Visual Basic* 2013 ima dvije osnovne strukture selekcije. To su strukture:

- *If Then* i
- *Case*.

Bilo koji problem grananja u programu se može riješiti preko ove dvije strukture. Koja će se struktura izabrati zavisi prvenstveno od broja grananja po jednom uslovu u konkretnom problemu. Kada postoji jedno do tri grananja najčešće se koristi struktura *If Then Else*. Međutim, kada postoji više od tri grananja najčešće se koristi struktura *Case*.

4.1.1. *If Then* struktura

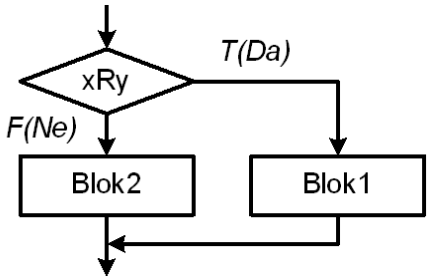
If ... ***Then*** predstavlja osnovnu struktura grananja, koja se koristi u skoro svim programskim jezicima. Standardni dijagram ove strukte prikazan je na slici 4.1., a struktuirani na slici 4.2. Osnovu ovog grananja čini logički uslov. Ako je ovaj logički uslov zadovoljen (Da - *True*) izvršava se jedna naredba (ili blok naredbi). Ako ovaj logički uslov nije zadovoljen (Ne - *False*) izvršava se druga naredba (ili blok naredbi). Postoji više verzija ove strukture grananja. Prva verzija ove strukture je samao *If* ... *Then* struktura jednostrukog izbora, a njena sintaksa je

If uslov1 **Then** naredba1

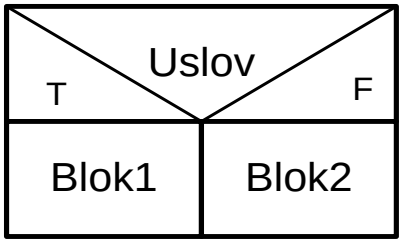
pri čemu je:

- **uslov1** - uslov iskazan preko logičkih operatora =, >, <, < >, >=, <=.
- **naredba1** - mora da bude samo jedna nardba.

Uslov1 predstavlja logički izraz, koji može biti tačan (*True*) ili netačan (*False*). U slučaju da je ligički izraz (**uslov1**), tačan izvršava se samo jedna naredba (**naredba1**) navedena iza komande ***Then***.



Slika 4.1. Standardni dijagrama toka



Slika 4.2. Struktuirani dijagrama toka

Primjer 4.1.

Izračunavanje vrijednosti nekog izraza u obliku razlomka u svim programskim jezicima moguće je samo onda, ako je imenilac razlomka različit od nule. Slijedi kod programa, koji pokazuje, kako se ovaj problem nedefinisanog razlomka rješava u programu *Visual Studio* 2013 pomoću strukture grananja *If ... Then*.

```
If Imenilac < >0 Then Rezultat = Brojilac/Imenilac
```

U slučaju da logički uslov nije zadovoljen (Imenilac = 0) navedena naredba dijeljenja biće ignorisana (preskočena) i počeće izvršavanje sljedećeg programskog reda, koji prvi slijedi iza ove kodne linije.

Druga verzija ove strukture je ***If ... Then ... End If*** struktura, a njena sintaksa je

```
If  uslov1  Then
     Blok1
End If
```

pri čemu je:

- **uslov1** - uslov iskazan preko logičkih operatora =, >, <, < >, >=, <=.
- **Blok1** - može da bude samo jedna nardba, ali i više naredbi (kodnih linija).
- **End If** - predstavlja naredbu koja označava kraj strukture grananja.

Sada se kod programa iz primjera 4.1. može napisati kao:

```
If Imenilac < >0 Then
  Rezultat = Brojilac/Imenilac
End If
```

Kada se koristi ova struktura, **Blok1** obično ima više od jedne naredbe. Slijedi kod programa za primjer 4.1. gdje se unos podataka vrši preko dva tekst boksa (Text1 i Text2). Rezultat se prikazuje u trećem tekst boksu (Text3). U ovom primjeru programa **Blok1** ima dvije kodne linije. U ovom slučaju se ne bi mogla upotrijebiti *If ... Then* struktura, već se mora upotrijebiti *If ... Then ... EndIf* struktura.

```
Brojilac = Text1.Text
Imenilac = Text2.Text
If Imenilac < >0 Then
  Rezultat = Brojilac/Imenilac
  Text3.Text = Rezultat
End If
```

U jednom programu može postojati više struktura grananja, pa je tada puno praktičnija i preglednija ova druga varijanta grananja (*If ... Then ... End If*), u odnosu na prvu varijantu (*If ... Then*), jer se lako uočava kraj svake strukture grananja *End If*. Slijedi primjer koda programa, koji ima više struktura grananja, ali gdje je teško uočiti gdje počinjinju, a gdje završavaju strukture grananja.

Ne pregledan kod	Pregledan kod

`If A > 0 Then B = 5`
`If A = 0 Then`
`B = A`
`C = B + 1`
`End If`
`If A < 0 Then C = 2`
`If C = 4 Then`
`B = A - 1`
`A = C + 3`
`End If`
`End If`
	`If A > 0 Then B = 5`
	`  If A = 0 Then`
	`  B = A`
	`  C = B + 1`
	`  End If`
	`  If A < 0 Then C = 2`
	`  If C = 4 Then`
	`  B = A - 1`
	`  A = C + 3`
	`  End If`
	`End If`

Treća verzija ove strukture je ***If ... Then ... Else ... End If*** struktura. Ovo je struktura dvojnog izbora, gdje postoji mogućnost izvršavanja samo jedne od dvije naredbe (bloka naredbi), zavisno od logičkog uslova koji je postavljen. Sintaksa ove strukture je

If uslov1 **Then**
    Blok1
Else
    Blok2
End If

pri čemu se **Blok1** naredbi izvršava samo ako je **uslov1** ispunjen, a ako **uslov1** nije ispunjen tada se izvršava samo **Blok2** naredbi.

Primjer 4.2.

Izračunavanje modula nekog broja X. Kod ovog programa bi izgledao:

```
If X >= 0 Then
  Rezultat = X
Else
  Rezultat = -X
End If
```

Četvrta verzija ove strukture je ***If ... Then ... ElseIf ... Else ... End If*** struktura, a njena sintaksa je

```
If  uslov1  Then
        Blok1
ElseIf  uslov2
        Blok2
ElseIf  uslov3
        Blok3
  .
  .
  .
Else
        Blok4    ' izvršava se ako nije ispunjen nijedan prethodni uslov
End If
```

Pri čemu se **Blok1** naredbi izvršava, samo ako je **uslov1** ispunjen. Tada se ostali blokovi naredbi (**Blok2**, **Blok3** i **Blok4**) neće izvršiti. Ako **uslov1** nije ispunjen, tada se provjerava da li je ispunjen sljedeći logički uslov **uslov2** i ako je on uspunjen izvršava se **Blok2** naredbi. Tada se ostali blokovi naredbi (**Blok1**, **Blok3** i **Blok4**) neće izvršiti. Ako ni **uslov2** nije ispunjen, tada se provjerava da li je ispunjen sljedeći logički uslov **uslov3** i ako je on uspunjen izvršava se **Blok3** naredbi. Tada se ostali blokovi naredbi (**Blok1**, **Blok2** i **Blok4**) neće izvršiti. Međutim, ako nije ispunjen ni jedan od postavljenih uslova u *ElseIf* strukturama, tada se izvršava samo **Blok4** naredbi, koji se nalazi u strukturi *Else*. Broj *ElseIf* blokova (odnosno logičkih uslova) u jednoj *IF ... Then* strukturi nije ograničen i zavisi samo od broja uslova, koje je potrebno postaviti u programu da bi se riješio postavljeni problem.

Primjer 4.3.

Neka se uz pomoć programa vrši ocjenjivanje učenika na testu, gdje je učenik mogao osvojiti najviše 100 bodova. Ocjene se daju po sljedećem principu: do 45 bodova ocjena je jedan, od 46 do 55 bodova ocjena je dva, od 56 do 70 bodova ocjena je tri, od 71 do 85 bodova ocjena je četiri, a za 86 i više bodova dobija se ocjena pet. Kod ovog programa za ocjenjivanje učenika bi izgledao:

```
If Bod <= 40 Then
  Ocjena = 1
Elself Bod <= 55 Then
  Ocjena = 2
Else If Bod <= 70 Then
  Ocjena = 3
Else If Bod <= 85 Then
  Ocjena = 4
Else
  Ocjena = 5
End If
```

U ovoj strukturi grananja treba posebno voditi računa, da se logički uslovni izrazi moraju postaviti u rastućem redoslijedu promjenjljive, od koje se prave uslovi grananja. Dakle, prethodni uslov ne smije u sebi da sadrži bilo koji naredni uslov. U primjeru 4.4. prikazan je kod programa, koji će za isti tekst zadatka kao u primjeru 4.3. vršiti pogrešno izračunavanje ocjena, za bodove koje se nalaze u opsegu od 41 do 55 bodova, jer drugi uslov (Bod <= 70) u sebi već sadrži naredni uslov (Bod <= 55).

Primjer 4.4.
Pogrešno izračunavanje ocjena učenika na testu.

```
If Bod <= 40 Then
  Ocjena = 1
Elself Bod <= 70 Then
  Ocjena = 3
Else If Bod <= 55 Then
  Ocjena = 2
Else If Bod <= 85 Then
  Ocjena = 4
Else
  Ocjena = 5
End If
```

Primjer 4.5.
Pregledno pisanje višestrukog grananja pomoću strukture *If ... Then... End If* .

```
If MP > (HHMAX / 2 - 16) Then    Rem prvi If
    If K < 4 Then                Rem drugi If
           SR = MP + 8
         MP = Int(SR)
         K = K + 1
    Else
         MP = SR + SS * KORAK
         MP = Int(MP)
           If SS = 10 Then       Rem treći If
              SS = 0
           Else
            SS = SS + 1
         End If                  Rem kraj trećeg If
    End If                       Rem kraj drugog If
End If                           Rem kraj prvog If
```

Rezervisane riječi *If* i *End If* uvijek se pišu u paru, da bi se znao početak i kraj strukture. Prilikom obavljanja nekoliko uzastopnih logičkih testiranja različite prirode, često je potrebno umetnuti jednu strukturu grananja unutar druge. Pri tome je poželjno da se prilikom pisanja koda programa, svaka unutrašnja struktura uvuče za nekoliko karaktera od početka reda. Pri tome sve naredbe istog nivoa treba da budu u istoj vertikalnoj liniji. Ovo pravilo neće ni ubrzati, a ni usporiti izvršenje izvršne verzije (.exe) programa. Ali će primjena ovog pravila značajno pomoći programeru prilikom pisanja koda programa, a posebno će olakšati postupak pronalaženja i otklanjanja grešaka u programu. Kod programa u primjeru 4.5. je potpuno identičan kodu u primjeru 4.6., a vi sami procjenite, koji kod programa je za vas pregledniji i lakši za praćenje.

Primjer 4.6.
Nepregledno pisanje višestrukog grananja pomoću strukture *If ... Then... End If* .

```
If MP > (HHMAX / 2 - 16) Then
If K < 4 Then
SR = MP + 8
MP = Int(SR)
K = K + 1
Else
MP = SR + SS * KORAK
MP = Int(MP)
If SS = 10 Then
SS = 0
Else
SS = SS + 1
End If
End If
End If
```

Da bi se postigla bolja preglednost i kontrola koda programa, poželjno je da se prilikom pisanja programa u kodu pišu komentari. Komentari se u kodu pišu pomoću ključne riječi ***REM***, ili korišćenjem simbola jednostrukog navodnika ('). Komentari se u kodu prepoznaju po tome što su napisani zelenom bojom. Što postoji više komentara u kodu, to je program razumljiviji. Ovo posebno dolazi do izražaja kada jedna osoba piše program, a druga osoba treba da program analizira ili dorađuje. Komentari su posebno bitni svakom programeru, kada vrši doradu programa poslije određenog vremenskog perioda. Ako u programu ne postoje komentari, programer će potrošiti puno vremena, da otkrije za šta se koriste određene promjenljive i kako funkcionišu grananja i petlje.

Jedan logički uslov za grananje može biti i složena kombinacija više logičkih uslova, koji se vezuju logičkim operatorima ***Or*** (ili), ***And*** (i), ***NOT*** (negacija).

Primjer 4.7.
Logički uslov grananja pomoću operatora *Or* i *And* sa tri promjenljive.

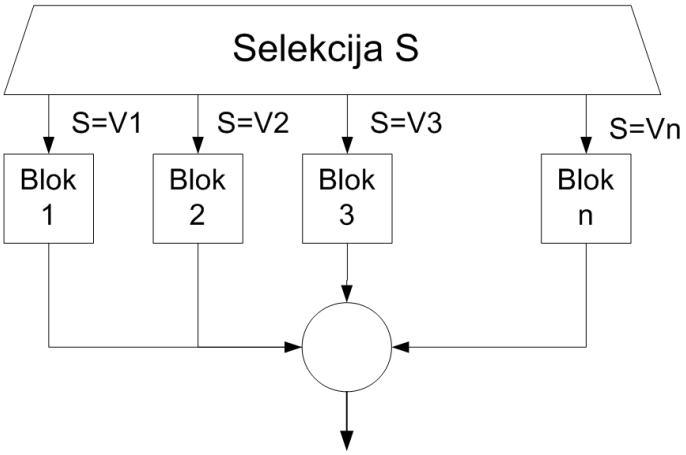
```
If X<5 Or Y=7 Or Z>10 Then
   ' kod kada je X manje od 5 ili Y jednako 7 ili Z
    veće od 10, odnosno kada je zadovoljen bar jedan
   od tri uslova
Else
   ' kod kada nije zadovoljen nijedan od tri uslova
End If

If X<5 And Y=7 And Z>10 Then
   ' kod kada je X manje od 5 i Y jednako 7 i Z veće
    od 10, odnosno kada su zadovoljena sva tri uslova
Else
   ' kod kada nije zadovoljen bar jedan od tri uslova
End If

If (X<5 Or Y=7) And Z>10 Then
   ' kod kada je X manje od 5 ili Y jednako 7, a Z
    veće od 10, odnosno kada je zadovoljen jedan od
    prva dva uslova i zadovoljen treći uslov
Else
   ' kod kada nisu zadovoljena oba prva dva uslova
    ili nije zadovoljen treći uslov
End If

If X<5 Or Y=7 And Z>10 Then
   ' kod kada je X manje od 5, ili Y jednako 7 i Z
    veće od 10, odnosno kada je zadovoljen prvi uslov
    ili istovremeno zadovoljen drugi i treći uslov.
    Pošto nema zagrada operator And ima veći prioritet
    (prvo se izvršava)od operatora Or.
Else
   ' kod kada nije zadovoljen prvi uslov ili nisu
    istovremeno zadovoljeni drugi i treći uslov
End If
```

4.1.2. *Case* struktura

Select Case predstavlja struktura grananja, koja se koristi u skoro svim programskim jezicima. Predstavlja složeniju varijantu ***If ... Then*** strukture. Standardni dijagram ove strukte prikazan je na slici 4.3. Struktura ***Select Case***

upotrebljava se, obično, kada se na određenom mjestu u programu donosi odluka o više različitih smjerova izvršavanja programa na osnovu vrijednosti jedne promjenljive. Takvi slučajevi mogu se rješavati i upotrebom ***If ... Then*** strukture, ali je, često, struktura ***Select Case*** jasnija i efikasnija. Ova struktura grananja je posebno pogodna za programe, koji imaju menije sa više opcija (prilikom korišćenja objekata *OptionButton*). Takođe je korisna kod upotrebe padjućih menija, odnosno prilikom preuzimanja podataka iz liste ili prilikom dodavanja novih stavki u listu primjenenom metode *Addltem.*

Ova struktura se obavezno završava propisanim riječima ***End Select,*** kako bi program ali i programer prepoznao kraj ove strukture.



Slika 4.3. Standardni dijagrama toka Case strukture

Postoji više verzija ove strukture grananja. Prva verzija ove strukture je samao ***Select Case*** struktura, a njena sintaksa je:

```
Select Case  promjenljiva
Case  vrijednost1
    Blok1            Rem naredbe ako je promjenljiva = vrijednost1
Case  vrijednost2
    Blok2            Rem naredbe ako je promjenljiva = vrijednost2
Case  vrijednost3
    Blok3            Rem naredbe ako je promjenljiva = vrijednost3

:
.
Case Else          Rem Ovo može biti opcioni uslov
     Blok4         Rem naredbe ako promjenljiva nema zadane vrijednosti
End Select
```

Pri čemu se **Blok1** naredbi izvršava, samo ako promjenljiva po kojoj se ispituje *Select Case* struktura poprimila tačno **vrijednost1**. **Blok2** naredbi se izvršava, samo ako promjenljiva po kojoj se ispituje *Select Case* struktura poprimila **vrijednost2**. ***Case Else*** je opcioni dio *Select Case* struktura, koji će se izvršiti, samo ako promjenljiva po kojoj se radi ***Select***, nije poprimila nijednu predhodno navedenu vrijednost. Broj ***Case*** blokova (odnosno uslova) u jednoj *Select Case* strukturi nije ograničen i zavisi samo od broja uslova, koje je potrebno postaviti da bi uspješno riješili postavljeni problem. Za ovaj vid strukture nije bitan redoslijed postavljanja *Case* vrijednosti, kao što je bilo bitno, kod postavljanja uslova prema rastućem redoslijedu u ***If ... Then ... ElseIf ... Else ... End If*** strukturi.

Primjer 4.8.

Napisati program koji ispisuje poruku sa ocjenom koju je dobio učenik, na osnovu ocjene koja se upisuje preko *Textbox*-a pod imenom TexUnos.

```
Ocjena = TexUnos.Text
Select Case Ocjena
Case 2
  poruka = "Ucenik je dobio ocjenu 2"
Case 3
  poruka = "Ucenik je dobio ocjenu 3"
Case 4
  poruka = "Ucenik je dobio ocjenu 4"
Case 5
  poruka = "Ucenik je dobio ocjenu 5"
Case Else
  poruka = "Ucenik je dobio ocjenu 1"
End Select
MsgBox(poruka, vbYesNo, "Ocjena")
```

Druga verzija ove strukture je ***Select ... Case Is*** struktura. ***Case Is*** naredba se koristi u kombinaciji sa relacionim operatorima >, <, <=, >=, a njena sintaksa je

Select Case promjenljiva
Case Is relacioni operator vrijednost1
  Blok1
Case Is relacioni operator vrijednost2, relacioni operator vrijednost3**, ...**
  Blok2
:
.
Case Else Rem Ovo može biti opcioni uslov
  Blok4 Rem naredbe ako nijedan predhodni uslov nije ispunjen
End Select

Pri čemu se **Blok1** naredbi izvršava samo ako promjennjiva po kojoj se ispituje *Select Case* struktura poprimila vrijednost, koja zadovoljava logički uslov zadat pomoću relacionog operatora. **Blok2** naredbi se izvršava, samo ako promjenjljiva po kojoj se ispituje *Select Case* struktura poprimila vrijednost, koja zadovoljava jedan od dva navedena logička uslova.

U ovoj strukturi grananja treba posebno voditi računa, da se uslovni izrazi moraju pretpostaviti u rastućem redoslijedu promjenljive po kojoj se radi grananje, za razliku od predhodne verzije ove strukture. Dakle, prethodni uslov ne smije u sebi da sadrži nijedan naredni uslov.

Primjer 4.9.

Neka se uz pomoć programa vrši ocjenjivanje učenika na testu, gdje je učenik mogao osvojiti najviše 100 bodova. Ocjene se daju po sljedećem principu: do 45 bodova ocjena je jedan, od 46 do 55 bodova ocjena je dva, od 56 do 70 bodova ocjena je tri, od 71 do 85 bodova ocjena je četiri, a za 86 i više bodova dobija se ocjena pet. Kod ovog programa za obračun poreza bi izgledao:

```
Bod = Text1
Select Case Bod
Case Is >= 86
    Ocjena = 5
Case Is >= 71
    Ocjena = 4
Case Is >= 56
    Ocjena = 3
Case Is >= 46
    Ocjena = 2
Case Else
    Ocjena = 1
End Select
```

Treća verzija ove strukture je ***Select ... Case To*** struktura. Ova struktura se koristi, kada uslov grananja želimo zadati kao interval brojeva. Uslov je zadovoljen i kada promjenljiva poprimi vrijednost brojeva preko kojih je interval određen, a njena sintaksa je

Select Case promjenljiva
Case Broj1 **To** Broj2
  Blok1
Case Broj3 **To** Broj4
  Blok2
:
.
Case Else Rem Ovo može biti opcioni uslov
  Blok4 Rem naredbe ako nijedan predhodni uslov nije ispunjen
End Select

Pri čemu se **Blok1** naredbi izvršava samo ako promjenljiva po kojoj se ispituje *Select Case* struktura poprimila vrijednost, koja se nalazi u intervalu između **Broja1** i **Broja2**. Za ovaj vid strukture bitan je redoslijed postavljanja ***Case*** logičkih uslova, jer predhodni u sebi ne smije sadržati naredni logički uslov. Jer će se od svih napisanih izvršiti samo jedan uslov i to onaj koji se prvi ispuni.

Primjer 4.10.

Jedna trgovačka kuća prodaje robu uz različite iznose popusta. Za iznos robe koji je između 100 $ i 499 $ uz 8% popusta. Za iznos robe koji je između 500 $ i 2000 $ uz 12% popusta. Za iznos robe koji je veći od 2000 $ uz 15% popusta. Treba izračunati iznos popusta za svaki navedeni slučaj.

```
Select Case Cijena
Case Is < 100
  Popust = 0
Case 100 To 499
  Popust = Cijena * 0.08
Case 500 To 2000
  Popust = Cijena * 0.12
Case Is > 2000
  Popust = Cijena * 0.15
End Select
```

Četvrta verzija ove strukture je kombinacija ***Case To*** i ***Case Is*** struktura. Ova struktura se koristi kada želimo na napravimo složen logički uslov grananja. Logički uslov je zadovoljen i kada promjenljiva poprimi vrijednost ili iz intervala brojeva (*Case To*) ili ako je zadovoljen logički uslov (*Case Is*), a njena sintaksa je

Select Case promjenljiva
Case Broj1 **To** Broj2**, Is** relacioni operator vrijednost1
  Blok1
Case Broj3 **To** Broj4**, Is** relacioni operator vrijednost2
  Blok2
:
.
Case Else Rem Ovo može biti opcioni uslov
  Blok4 Rem naredbe ako nijedan predhodni uslov nije ispunjen
End Select

Primjer 4.11.

```
Case 100 To 499,  Is = 777
Case 500 To 2000, Is > 5000
```

Case struktura može biti nepregledna, ako svaki blok unutar *Case* strukture ima proizvoljnu dužinu. To bi dovelo do toga da program postaje nepregledan, ako bi

imali više *Case* grananja. Zbog toga je poželjno da blokovi unutar *Case* strukture ne budu previše dugi. Ako blok treba da bude dug onda ga treba napisati kao proceduru. Tada se u *Case* strukturi piše samo poziv procedura, koje se pozivaju, a ne čitav kod tih procedure. Na ovaj način *Case* stuktura postaje puno preglednija za programera.

4.2. NAREDBE ITERACIJE

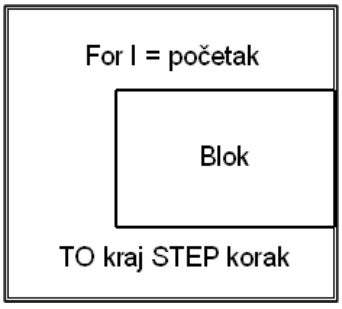
Programski jezik *Visual Basic* 2013 ima nekoliko struktura iteracije (ponavljanja). Ovdje će biti predstavljene samo neke strukture:

- ***For ... Next,***
- ***Do While ... Loop***
- ***Do ... While Loop*** i
- ***Do ... Until Loop.***

Bilo koji problem višestrukog ponavljanja u programu se može riješiti preko ovih struktura. Na primjer računanje prosječne ocjena za 30 predmeta svih studenata na fakultetu je bolje isprogramirati kroz petlju, nego da se za svakog studenta pojedinačno ponavlja pisanje istog koda. Koja će se struktura ponavljanja izabrati zavisi prvenstveno od toga da li se neka struktura ponavlja tačno određeni broj puta, promjenljiv broj puta, sa izlazom na početku ili na kraju petlje.

4.2.1. *For ... Next* struktura

For ... Next predstavlja strukturu ponavljanja, koja se koristi u skoro svim programskim jezicima. Struktuirani dijagram ove strukte prikazan je na slici 4.4.



Slika 4.4. Dijagram For Next

For ... Next petlja omogućava da se određena naredba (ili skup naredbi) bezuslovno ponavlja, tačno predviđeni broj puta, sa mogućnosti da se mijenja korak ponavljanja. Sintaksa ove naredbe glasi:

For promjenljiva = početak **To** kraj [**Step** korak]
Iterativne naredbe
[**Exit For**]
Next promjenljiva

Vrijednost **promjenljive** kojom počinje izvršavanje strukture ponavljanja je **početak**. Ponavljanje traje sve dok **promjenljiva** ne dostigne vrijednost **kraj**. **Promjenljiva** se uvećava za jedan (promjenljiva = promjenljiva + 1) u svakom sljedećem ponovnom izvršavanju strukture ponavljanja, ukoliko nije navedena

opcija ***Step*** (korak). Promjenljiva je, u stvari, brojač čija vrijednost zavisi od veličine koraka. Ako je korak jedan (1) onda onda se on podrazumjeva i nije ga potrebno posebno pisati u *For ... Next* strukturi. Međutim, ako želimo da korak bude različit od jedan, onda se on mora navesti. Korak ponavljanja može biti cijeli, realan, ali i negativan broj, a može se napisati kao:

- For x = 1 To 10 Step 0.5 (sa korakom 0,5, ponavljanje 19 ciklusa)
- For x = 1 To 10 Step 2 (sa korakom 5, ponavljanje 5 ciklusa)
- For x = 10 To 1 Step -1 (sa korakom -1, ponavljanje 10 ciklusa)

U primjeru 4.12. opisano je kako funkcioniše jedna *For* petlja, koja ponavlja dvije naredbe u 4 ciklusa.

Primjer 4.12.

```
AA = 6
BB = 0
For j = 1 To 4
  AA = AA - BB - j
  BB = BB + j
Next j
```

Rješenje

1) Ciklus j = 1, AA = 5, BB = 1
2) Ciklus j = 2, AA = 2, BB = 3
3) Ciklus j = 3, AA = -4, BB = 6
4) Ciklus j = 4, AA = -14, BB = 10
Na kraju AA = -14, BB = 10

Unutar jedne *For* petlje može postojati jedna, ali i više *If ... Then* struktura grananja. U primjeru 4.13. opisano je kako funkcioniše jedna *For* petlja, koja ima 4 ciklusa. Ponavljaju se dvije naredbe i jedno *If ... Then* grananje (sa 5 kodnih linija).

Primjer 4.13.

```
AA = 2
BB = 3
For J = 1 To 4
  BB = AA + BB - J
  AA = AA + J
  If BB = 6 Then
     AA = -6
  Else
     AA = AA + J
  End If
Next J
```

Rješenje

1) Ciklus j = 1, BB = 4, AA = 3, Else, AA = 4
2) Ciklus j = 2, BB = 6, AA = 6, Then, AA = -6
3) Ciklus j = 3, BB = -3, AA = -3, Else, AA = 0
4) Ciklus j = 4, BB = -7, AA = 4, Else, AA = 8
Na kraju AA = 8, BB = -7

Unutar jedne *For* petlje može postojati jedna, ali i više *Select Case* struktura grananja. Unutar jedne *For* petlje može se kombinovati više grananja od kojih su neka *If ... Then*, a neka *Select ... Case*. Bez obzira kako se kombinuju ova grananja, uvijek se izvršavaju jedna po jedna kodna linija. U primjeru 4.14. opisano je kako funkcioniše jedna *For* petlja koja ima 4 ciklusa ponavljanja jedne naredbe i jednog *Case* grananja (sa 10 kodnih linija).

Primjer 4.14.

```
AA = 3
BB = -1
For J = 2 To 5
  BB = BB + 2 * J
  Select Case BB
  Case 3
    AA = AA - J
  Case 4 To 12
    BB = AA - BB + J
  Case Is > 12
    BB = AA - J
  Case Else
    BB = AA + J
  End Select
Next J
```

Rješenje

1) Ciklus j = 2, BB = 3, Case 3, AA = 1
2) Ciklus j = 3, BB = 9, Case 4 To 9, BB = -5
3) Ciklus j = 4, BB = 3, Case 3, AA = -3
4) Ciklus j = 5, BB = 13, Case Is>12, BB = -8

Na kraju AA= -3, BB = -8

Unutar jedne *For* petlje može postojati druga *For* petlja. Ovo se uglavnom koristi kod popunjavanja dvodimenzionih matrica. Ukupan broj ciklusa se određuje kao umnožak broja ciklusa vanjske i broja ciklusa unutrašnje petlje. U primjeru 4.15. opisano je kako funkcioniše program u kome vanjska *For* petlja (**I**), koja ima 2 ciklusa (određuje redni broj reda tabele), a unutrašnja *For* petlja (**J**) ima 4 ciklusa (određuje redni broj kolone tabele). U ovom programu ima ukupno 8 ciklusa, u kojima se upisuju podaci u *DataGridView* dvodimenzionu tabelu.

Primjer 4.15.

```
AA = 0
BB = 3
For I = 1 To 2
   If AA > 0 Then
     BB = BB - AA
   Else
     AA = I + AA
   End If
   For J = 1 To 4
      If BB > 1 Then
         AA = AA - I + J
         BB = BB - AA + I
      Else
         AA = AA - BB + I
         BB = BB + AA
      End If
      If AA > 5 Then
```

Rješenje

1) Ciklus I = 1, If 1 - Else, AA = 1, J = 1,
   If2 - Then, AA = 1, BB = 3,
   If3 - Else, Ćelija Tabele(1,1) = BB = 3
2) Ciklus I = 1, J = 2,
   If2 - Then, AA = 2, BB = 2,
   If3 - Else, Ćelija Tabele(1,2) = BB = 2
3) Ciklus I = 1, J = 3,
   If2 - Then, AA = 4, BB = -1,
   If3 - Else, Ćelija Tabele(1,3) = BB = -1
4) Ciklus I = 1, J = 4,
   If2 - Else, AA = 6, BB = 5,
   If3 - Then, Ćelija Tabele(1,4) = AA = 6
5) Ciklus I = 2, If 1- Then, BB = -1, J = 1,
   If2 - Else, AA = 9, BB = 8,
   If3 - Then, Ćelija Tabele(2,1) = AA = 9
6) Ciklus I = 2, J = 2,
   If2 - Then, AA = 9, BB = 1,

```
     DataGridView(I, J) = AA
       Else
     DataGridView(I, J) = BB
       End If
    Next J
Next I
End With
```

If3- Then, Ćelija Tabele(2,2) = AA = 9
7) Ciklus I = 2, J = 3,
If2 - Else, AA = 10, BB = 11,
If3 - Then, Ćelija Tabele(2,3) = AA = 10
8) Ciklus I = 1, If1- Else, AA = 1, J = 1,
If2 - Then, AA = 12, BB = 1,
If3 - Then, Ćelija Tabele(2,4) = AA = 12
Na kraju popunjena tabela će izgledati

	3	2	-1	6
	9	9	10	12

Primjer 4.16.

```
AA = 0
BB = 3
For K = 0 To 1
  For I = 1 To 2
    For J = 1 To 4
      If BB > 1 Then
         AA = AA - I + J - K
         BB = BB - AA + I - K
      Else
         AA = AA - BB + I + K
         BB = BB + AA + K
      End If
      If BB > 5 Then
   DataGridView(K,I,J)= AA
      Else
  DataGridView(K,I,J) = BB
      End If
    Next J
  Next I
Next K
End With
```

Rješenje

1) Ciklus K = 0, I = 1, J = 1, If1 - Then, AA = 0,
BB = 4, If2 - Else, Tabele(0,1,1) = BB = 4
2) Ciklus K = 0, I = 1, J = 2, If1 - Then, AA = 1,
BB = 4, If2 - Else, Tabele(0,1,2) = BB = 4
3) Ciklus K = 0, I = 1, J = 3, If1 - Then, AA = 3,
BB = 2, If2 - Else, Tabele(0,1,3) = BB = 3
4) Ciklus K = 0, I = 1, J = 4, If1 - Then, AA = 6,
BB = -3, If2 - Else, Tabele(0,1,4) = BB = -3
5) Ciklus K = 0, I = 2, J = 1, If1 - Else, AA = 11,
BB = 8, If2 - Then, Tabele(0,2,1) = AA = 11
6) Ciklus K = 0, I = 2, J = 2, If1 - Then, AA = 11,
BB = -1, If2 - Else, Tabele(0,2,2) = BB = -1
7) Ciklus K = 0, I = 2, J = 3, If1 - Else, AA = 14,
BB = 13, If2 - Then, Tabele(0,2,3) = AA = 14
8) Ciklus K = 0, I = 2, J = 4, If1 - Then, AA = 16,
BB = -1, If2 - Else, Tabele(0,2,4) = BB = -1
9) Ciklus K = 1, I = 1, J = 1, If1 - Else, AA = 19,
BB = 19, If2 - Then, Tabele(1,1,1) = AA = 19
10) Ciklus K = 1, I = 1, J = 2, If1 - Then, AA =19,
BB = 0, If2 - Else, Tabele(1,1,2) = BB = 0
11) Ciklus K = 1, I = 1, J = 3, If1 - Else, AA = 21,
BB = 22, If2 - Then, Tabele(1,1,3) = AA = 21
12) Ciklus K = 1, I = 1, J = 4, If1 - Then, AA =23,
BB = -1, If2 - Else, Tabele(1,1,4) = BB = -1
13) Ciklus K = 1, I = 2, J = 1, If1 - Else, AA = 27,
BB = 27, If2 - Then, Tabele(1,2,1) = AA = 27
14) Ciklus K = 1, I = 2, J = 2, If1 - Then, AA =26,
BB = 2, If2 - Else, Tabele(1,2,2) = BB = 2
15) Ciklus K = 1, I = 2, J = 3, If1 - Then, AA =26,
BB = -23, If2 - Else, Tabele(1,2,3) = BB = -23
16) Ciklus K = 1, I = 2, J = 4, If1 - Else, AA = 52,
BB = 30, If2 - Then, Tabele(0,1,1) = AA = 52

Unutar jedne *For* petlje može postojati i više od jedne unutrašnje *For* petlja, ali praktično se najčešće koriste samo tri nivoa ugnježdenja *For* petlje. Program koji ima tri nivoa *For* petlje obično se koristi za kod popunjavanja trodimenzionih matrica. Ukupan broj ciklusa se određuje kao umnožak broja ciklusa vanjske, prve unutrašnje petlje i druge unutrašnje petlje. U primjeru 4.16. opisano je kako funkcioniše program u kome vanjska *For* petlja (**K**), koja ima 2 ciklusa (određuje dubinu tabele), prva unutrašnja *For* petlja (**I**) koja ima 2 ciklusa (određuje redni broj reda tabele), a druga unutrašnja petlja (**J**) ima 4 ciklusa (određuje redni broj kolone tabele). U ovom programu ima ukupno 16 ciklusa u kojima se upisuju podaci u trodimenzionu tabelu. Na kraju popunjena trodimenziona matrica će izgledati

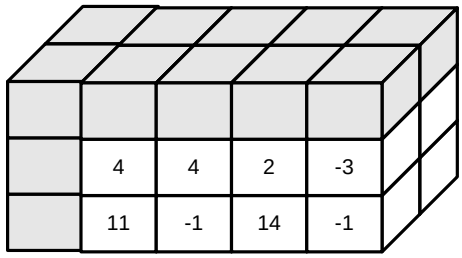
Tabela u prvom redu (K = 0)

	4	4	2	-3
	11	-1	14	-1

Tabela u drugom redu (K = 1)

	19	0	21	-1
	27	2	-23	52

ili trodimenziono kada se vidi samo prvi dio



Sljedi primjer kako se *For* petlja može koristiti za unos podataka u niz, kao i za naknadnu obradu unesenih podataka. Program radi sa dva niza koja mogu imati maksimalno 100 elemenata, a elementi mogu biti samo cijeli brojevi. Prvi niz pod imenom Niz1 predviđeno je da se unosi preko *InputBox*-a, dok se drugi niz pod imenom Niz2 unosi preko generatora slučajnih brojeva. Drugi niz može imati brojeve u rasponu od 0 do 10.

Da bi jedan niz mogao da se koristi u više procedura potrebno ga je deklarisati kao globalnu promjenljivu u modulu (**Module1.vb**).

Primjer 4.17.

```
Module Module1
    Public Niz1(100) As Integer
    Public Niz2(100) As Integer
    Public BrojCN As Integer
End Module
```

Procedura za unos niza.

```
Private Sub CBUnos_Click() Handles CBUnos.Click
  Dim aa, bb As Integer
  Dim Poruka1, ii As String
  Dim SNiz1, SNiz2 As String
  Dim TekstNiza1 As String
  Dim i As Integer

  BrojCN = TxtBrojCN.Text
  If BrojCN < 101 Then
     CijeliNiz.Text = ""
     TekstNiza1 = ""
     For i = 1 To BrojCN
         ii = i
         Poruka1 = "Unesi " + ii + "-ti Clan Niza"
         aa = InputBox(Poruka1, "Unos clanova niza")
         Niz1(i) = aa
         SNiz1 = aa
          If i = 1 Then
             CijeliNiz.Text = SNiz1
          Else
             CijeliNiz.Text=CijeliNiz.Text+"; "+SNiz1
         End If
         bb = Math.Round(10 * Rnd())
         Niz2(i) = bb
         SNiz2 = bb
         If i = 1 Then
            TexNiz2.Text = SNiz2
         Else
            TexNiz2.Text = TexNiz2.Text + "; " + SNiz2
         End If
      Next i
  Else
    MsgBox("Vas niz moze imati maksimalno 100 clanova")
  End If
End Sub
```

Za Niz2 pronalaženje najnećeg člana niza.

```
Private Sub ButMaxNiz2_Click() Handles ButMaxNiz2.Click
  Dim MaxNiz2 As Integer
  Dim i As Integer

  MaxNiz2 = Niz2(1)
```

```
  For i = 2 To BrojCN
     If Niz2(i) > MaxNiz2 Then
        MaxNiz2 = Niz2(i)
     End If
  Next i
  TexMaxNiz2.Text = MaxNiz2
End Sub
```

Za Niz2 pronalaženje zbira svih članova niza.

```
Private Sub ButZbir2Svi_Click()
  Dim Zbir, i As Integer
  Dim PomocZ As Integer
  Zbir = 0
  For i = 0 To BrojCN
      PomocZ = Niz2(i)
      Zbir = Zbir + PomocZ
      TexZbir2Svi.Text = Zbir
  Next i
End Sub
```

Za Niz2 pronalaženje zbira svih neparnih članova niza.

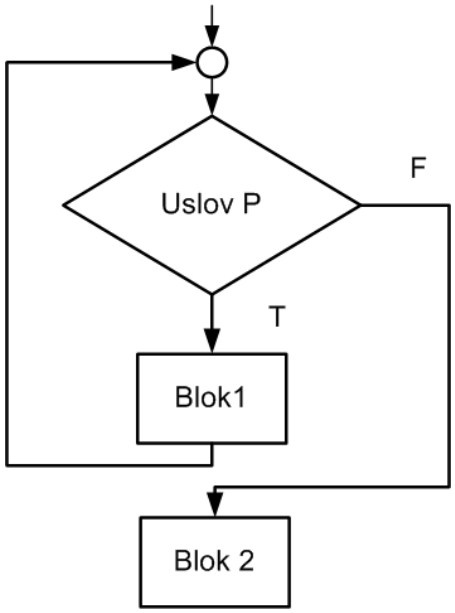
```
Private Sub ButZbir2Neparni_Click()
  Dim Zbir, i As Integer
  Dim PomocZ As Integer
  Zbir = 0
  For i = 1 To BrojCN Step 2
      PomocZ = Niz2(i)
      Zbir = Zbir + PomocZ
      TexZbir2Neparni.Text = Zbir
  Next
End Sub
```

Za Niz2 pronalaženje proizvoda svih članova niza.

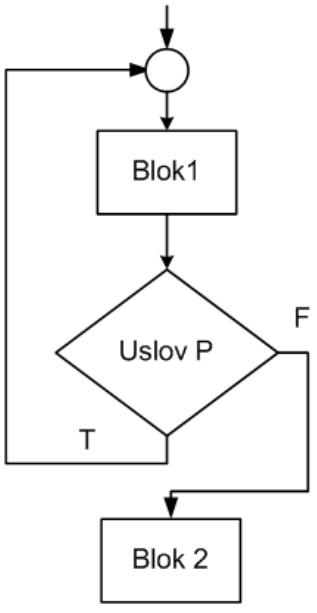
```
Private Sub ButProizvodNiz2_Click()
  Dim Proizvod, i As Long
  Dim PomocP As Integer
  Proizvod = 1
  For i = 1 To BrojCN
      PomocP = Niz2(i)
      Proizvod = Proizvod * PomocP
      TexProizvodNiz2.Text = Proizvod
  Next i
End Sub
```

4.2.2. *Do While ... Loop* struktura

Do While ... Loop predstavlja strukturu ponavljanja, koja se koristi u skoro svim programskim jezicima. Blok dijagram ove strukte prikazan je na slici 4.5. Za korišćenje naredbe *FOR* moramo znati tačan broj ponavljanja. On se mora naznačiti prije nego što se i jedno ponavljanje izvrši. Ali ponekad želimo ponavljati naredbe sve dok je ispunjen neki uslov. U tom slučaju ne znamo unaprijed kada se program pokrene koliko će biti ponavljanja. Petlja *Do While ... Loop* omogućava da se određena naredba (ili skup naredbi) ponavlja sve dok postavljeni logički uslov ne bude ispunjen. Prema ovom blok dijagramu blok naredbi **(Blok1**) će se izvršavati sve dok je logički uslov (**Uslov P)** ispunjen. U ciklusu ponavljanja kada logički uslov (**Uslov P)** ne bude ispunjen program izlazi iz petlje. Tada počinje da se izvršava blok naredbi (**Blok2**), koji se nalazi izvan ove petlje. Kod ove strukture treba dobro voditi računa da ne dođe do pojave, koja se zove beskonačna petlja. Beskonačna petlja je pojava u kojoj se petlja neprestano ponavlja, jer je uslov na ulazu u petlju uvijek ispunjen. Ako se desi da program uđe u beskonačnu petlju, tada je izlaz moguć samo ako se izvrši reset (Ctrl+Alt+Delete) kompletnog programa *Visual Studio* 2013. Ako prije toga nismo spasili izmjene u programu, one će onda biti trajno izgubljene. Za ovu petlju je karakteristično, da se blok koji se ponavlja (**Blok1**) ne mora nikada biti izvršavan, ako logički uslov (**Uslov P**) za ponavljanje u prvom prolazu kroz petlju nije ispunjen. Tada će se izvršavati samo naredbe **Blok2**, a naredbe koje se nalaze u **Blok1** se neće izvršiti ni jedan put.



Slika 4.5. Dijagram Do While



Slika 4.6. Dijagram Do Loop While

Sintaksa ove naredbe glasi:

DO WHILE logički uslov
Iterativne naredbe
[EXIT DO]
LOOP

U ovoj strukturi **logički uslov** (**Uslov P**) predstavlja izraz sastavljen od jedne ili više promjenljivih povezanih logičkim operatorima (=, <,>, < >, >= ili <=). Ovaj uslov može biti ispunjen (T) ili ne ispunjen (F). **Iterativne naredbe** (**Blok1**) predstavljaju jednu ili više naredbi koje se ponavljaju u ovoj petlji. ***LOOP*** predstavlja ključnu riječ, koja označava da se od ove naredbe program vraća na početak petlje. Naredba ***EXIT DO*** predstavlja opcionu komandu u ovoj strukturi. ***EXIT DO*** predstavlja komandu koja omogućuje trenutni izlazak iz ove petlje, bez obzira da li je početni **logički uslov** (**Uslov P**) ispunjen.

U primjeru 4.18. opisano je kako funkcioniše jedna *Do While ... Loop* petlja koja ponavlja tri naredbe u 4 ciklusa.

Unutar jedne *Do While ... Loop* petlje može postojati još jedna ali i više drugih *Do While ... Loop* petlji. Unutar *Do While ... Loop* petlje može postojati jedna ali i više *If ... Then* ili *Case* struktura grananja. U primjeru 4.19. opisano je kako funkcioniše jedna *Do While ... Loop* petlja, koja ima 4 ciklusa ponavljanja dvije naredbe i jedno *If ... Then* grananje (sa 5 kodnih linija).

U predhodna dva primjera se odmah iz koda lako može uočiti broj ponavljanja u petlji. Međutim, u većini praktičnih primjera za ove petlje se iz samog koda nemože odmah lako utvrditi broj ponavljanja petlje, već to zavisi od vrijednosti ulaznih promjenljivih i uslova grananja u petlji. U primjeru 4.20. opisano je kako funkcioniše *Do While ... Loop* petlja, koja ima 5 ciklusa ponavljanja, ali se to nemože odmah uočiti iz koda. Za vježbu uraditi ovaj zadatak kada ulazne promjenljive imaju početne vrijednosti AA=1, a BB=3. I za ovu kombinaciju ulaznih promjenljivih *Do While* pelja će se ponoviti 5 puta, a promjenljive će na kraju imati vrijednosti AA = 11, a BB = 2.

Primjer 4.18.

```
AA = 3
BB = 2
J = 1
Do While J < 5
  AA= AA + BB - J
  BB = BB + J
  J = J + 1
Loop
```

Rješenje

1) Ciklus, uslov ispunjen, AA = 4, BB = 3, j = 2
2) Ciklus, uslov ispunjen, AA = 5, BB = 5, j = 3
3) Ciklus, uslov ispunjen, AA = 7, BB = 8, j = 4
4) Ciklus, uslov ispunjen, AA = 11, BB = 12, j = 5
5) Ciklus se neće izvršiti jer uslov nije ispunjen i nakon 4 ciklusa petlja se završava.
Na kraju AA= 11, BB = 12

Primjer 4.19.

```
AA = 1
BB = 2
J = 0
Do While J < 4
  BB = AA + BB - J
  AA = AA + J
  If BB = 4 Then
    AA = -1
  Else
    AA = AA + J
  End If
  J = J + 1
Loop
```

Rješenje

1) Ciklus, uslov ispunjen, BB = 3, AA = 1, If - Else, AA = 1, j = 1
2) Ciklus, uslov ispunjen, BB = 3, AA = 2, If - Else, AA = 3, j = 2
3) Ciklus, uslov ispunjen, BB = 4, AA = 5, If - Then, AA = -1, j = 3
4) Ciklus, uslov ispunjen, BB = 0, AA = 2, If - Else, AA = 5, j = 4
5) Ciklus se neće izvršiti jer uslov nije ispunjen i nakon 4 ciklusa petlja se završava.
Na kraju AA= 5, BB = 0

Primjer 4.20.

```
AA = 1
BB = 2
J = 0
Do While J < 4
  BB = AA + BB - J
  AA = AA + J
  If BB = 4 Then
    AA = -1
    J = J - 1
  Else
    AA = AA + J
  End If
  J = J + 1
Loop
```

Rješenje

1) Ciklus, uslov ispunjen, BB = 3, AA = 1, If - Else, AA = 1, j = 1
2) Ciklus, uslov ispunjen, BB = 3, AA = 2, If - Else, AA = 3, j = 2
3) Ciklus, uslov ispunjen, BB = 4, AA = 5, If - Then, (AA = -1, j = j-1=1), j = j+1=2
4) Ciklus, uslov ispunjen, BB = 1, AA = 1, If - Else, AA = 3, j = 3
5) Ciklus, uslov ispunjen, BB = 1, AA = 6, If - Else, AA = 9, j = 4
6) Ciklus se neće izvršiti jer While uslov nije ispunjen i nakon 5 ciklusa petlja se završava.
Na kraju AA= 9, BB = 1

4.2.3. *Do ... Loop While* struktura

Do ... Loop While struktura predstavlja strukturu ponavljanja, koja je vrlo slična strukturi *Do While ... Loop*. Glavna razlika između ove dvije strukture leži u činjenici, da se blok naredbi koji se želi ponavljati (**Blok1**) u strukturi *Do ... Loop While* mora izvršiti bar jednom. Dok se u predhodnoj strukturi ovaj blok naredbi nije morao izvršiti ni jedan put. Blok dijagram ove strukte prikazan je na slici 4.6.

Sintaksa ove naredbe glasi:

DO
Iterativne naredbe
LOOP WHILE logički uslov

U ovoj strukturi **logički uslov** (**Uslov P**) predstavlja izraz sastavljen od jedne ili više promjenjljivih povezanih logičkim operatorima (=, <,>, < >, >= ili <=). Ovaj uslov može biti ispunjen ili ne ispunjen. **Iterativne naredbe** (**Blok1**) predstavljaju jednu ili više naredbi koje se ponavljaju u ovoj petlji. ***LOOP WHILE*** predstavlja ključnu riječ, koja označava da se od ove naredbe program vraća na početak petlje (do ključne riječi DO), ali samo ako je logički uslov (**Uslov P**) ispunjen. Iz petlje se izlazi čim logički uslov (**Uslov P**) nije prvi put ispunjen. Kod ove strukture treba također voditi računa, da ne dođe do pojave koja se zove beskonačna petlja.

U primjeru 4.21. opisano je kako funkcioniše jedna *Do ... Loop While* petlja, koja ima 4 ciklusa ponavljanja osam kodnih linija.

Primjer 4.21.

```
AA = 3
BB = 2
J = 1
Do
  AA = AA + BB - J
  BB = BB + J
  If AA = 6 Then
    BB = -7
  Else
    BB = BB + J
  End If
  J = J + 1
Loop While J < 5
```

Rješenje

1) Ciklus, uslov ispunjen, AA = 4, BB = 3, If - Else, BB = 4, j = 2
2) Ciklus, uslov ispunjen, AA = 6, BB = 6, If - Then, BB = -7, j = 3
3) Ciklus, uslov ispunjen, AA = -4, BB = -4, If - Else, BB = -1, j = 4
4) Ciklus, uslov ispunjen, AA = -9, BB = 3, If - Else, BB = 7, j = 5
5) Ciklus se neće izvršiti jer uslov nije ispunjen i nakon 4 ciklusa petlja se završava.
Na kraju AA= -9, BB = 7

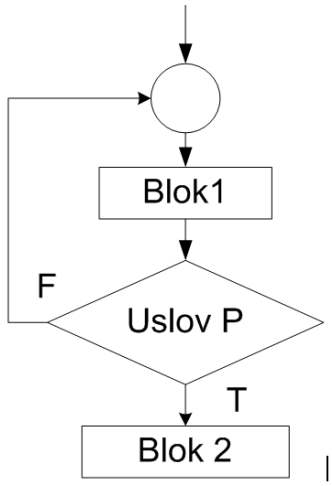
4.2.4. *Do ... Loop Until* struktura

Do ... Loop Until struktura predstavlja strukturu ponavljanja, koja je vrlo slična strukturi *Do ... Loop While*. Glavna razlika između ove dvije strukture leži u činjenici da se u strukturi *Do ... Loop Until*, blok naredbi koji se želi ponavljati (**Blok1**) izvršava sve dok logički uslov (**Uslov P**) nije ispunjen, za razliku od predhodne strukture *Do ... Loop While*, kod koje se ovaj blok naredbi izvršava dok je logički uslov (**Uslov P**) ispunjen. I kod ove strukture se blok naredbi za ponavljanje (**Blok1**) mora izvršiti bar jednom. Blok dijagram ove strukte prikazan je na slici 4.7.

Sintaksa ove naredbe glasi:

DO
Iterativne naredbe
LOOP UNTIL logički uslov

U ovoj strukturi **logički uslov** (**Uslov P**) predstavlja izraz sastavljen od jedne ili više promjenljivih koje su povezane logičkim operatorima (=, <,>, < >, >= ili <=).



Slika 4.7. Dijagram Do Loop Until

Ovaj **logički uslov** (**Uslov P**) može biti ispunjen ili ne ispunjen. **Iterativne naredbe** (**Blok1**) predstavljaju jednu ili više naredbi, koje se ponavljaju u ovoj petlji. Kod ove strukture će se naredne (**Blok1**) izvršiti bar jednom, bez obzira na postavljeni logički uslov.

LOOP UNTIL predstavlja ključnu riječ, koja označava da se od ove naredbe, program vraća na početak petlje, ali samo ako logički uslov nije ispunjen. Kada logički uslov postane ispunjen tada se izlazi iz petlje.

Kod ove strukture treba također voditi računa, da ne dođe do pojave koja se zove beskonačna petlja.

U primjeru 4.22. opisano je kako funkcioniše jedna *Do ... Loop Until* petlja, koja ima 4 ciklusa ponavljanja tri naredbe.

Primjer 4.22.

```
AA = 0
BB = 1
j = 0
Do
  AA= AA - BB + 2
  If AA < 2 Then
     BB = 5
  Else
     BB = AA + 1
  End If
  j = j + 1
Loop Until j > 3
```

Rješenje

1) Ciklus, uslov ispunjen, AA = 1, If - Then,
   BB = 5, j = 1
2) Ciklus, uslov ispunjen, AA = -2, If - Then,
   BB = 5, j = 2
3) Ciklus, uslov ispunjen, AA = -5, If - Then,
   BB = 5, j = 3
4) Ciklus, uslov ispunjen, AA = -8, If - Then,
   BB = 5, j = 4
5) Ciklus se neće izvršiti jer uslov nije ispunjen i nakon 4 ciklusa petlja se završava.
Na kraju AA= -8, BB = 5

U primjeru 4.23. i 4.24. predstavljena je *Do ... Loop Until* petlja slična predhodno urađenom primjeru 4.20. Potrebno je izračunati koliko ima ciklusa ponavljanja i koje vrijednosti će imati promjenjljive AA i BB na kraju petlje

.

Primjer 4.23.

```
AA = 0
BB = 1
j = 0
Do
  AA= AA - BB + 2
  If AA < 2 Then
     BB = 5
     j = j - 1
  Else
     BB = AA + 1
  End If
  j = j + 1
Loop Until j > 3
```

Primjer 4.24.

```
AA = 0
BB = 1
j = 0
Do
  AA= AA - BB + 2
  If AA < 2 Then
     BB = 5
     j = j + 1
  Else
     BB = AA + 1
  End If
  j = j + 1
Loop Until j > 3
```

5. FUNKCIJE I PROCEDURE

Kad se gradi jedan složeni sklop, kao što je na primjer automobil, konstruktori neće početi od najosnovnijih sirovinskih materijala, kao što je ruda gvožđa, ili nafta od koje se prave plastični materijali. Umjesto toga, automobil će se sklapati od (jednom) prethodno napravljenih dijelova, kao što su autogume, akumulatori, motori, stakla itd. Svi ti dijelovi napravljeni su u različitim fabrikama.

Funkcije i procedure su strukturni blokovi, koji omogućuju da program bude konstruisan od ranije napravljenih dijelova. Korišćenje takvih strukturnih blokova ima tri prednosti:

1) Kada radimo na bilo kom, ali jednom, strukturnom bloku, možemo usmjeriti pažnju na samo jedan dio programa.

2) Različiti ljudi mogu raditi na različitim strukturnim blokovima u isto vrijeme.

3) Ako je isti strukturni blok potreban na više mjesta u programu, možemo ga jedanput napisati i koristiti više puta.

Procedure i funkcije su sastavni dio *Visual Basic* 2013, tj. nalaze se u nekoj biblioteci jezika ili ih definišu korisnici. Pomoću njih obično se obavljaju složeniji zadaci u odnosu na pojedine naredbe. Ako se pojedine procedure i funkcije koriste u više različitih formi, odnosno imaju opštu namjenu, onda se one definišu u posebnom, standardnom modulu zajedno sa globalnim (javnim) promjenljivim. Iz modula, kao i iz drugih izvora pozivaju se po svome imenu. Standardni modul prepoznaje se po svome imenu, koje je dao korisnik i po ekstenziji ***.vb***, koja je od imena odvojena tačkom, npr. "Ime_Modula.vb".

Da bi vrijednosti proslijeđene procedurama i funkcijama ostale neizmijenjene dodaju im se ključne riječi ***ByVal***. Inače bi, prilikom određenih izračunavanja, te procedure i funkcije mogle da poprime drugačiju vrijednost, koja nemože da se predvidi. Upotreba propisanih riječi ***ByVal*** ilustrovana je na primjerima iz geometrije, koji su navedeni u sljedećem poglavlju.

5.1. FUNKCIJE

Neke standardne funkcije, kao što su kvadratni korjen (*Sqrt*), zaokruživanje brojeva (*Round*), su već ugrađene u *Visual Basic* 2013. Sada ćemo pokazati kako se definišu vlastite funkcije, koje su nam potrebne da bi rješili postavljeni zadatak, a nisu ugrađene u *Visual Basic* 2013. Funkcija se definiše koristeći funkcijsku deklaraciju. Korisnici definišu funkciju pomoću sljedećeg formata:

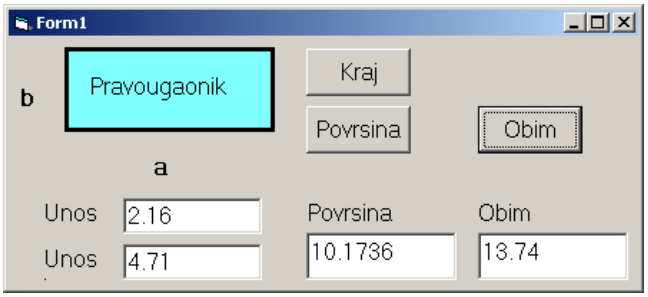
[Public ili **Private] Function** ImeFunkcije **([**argumenti**] [As Tip]) [As Tip]**
    Sekvencija naredbi
    ImeFunkcije = izraz ili vrijednost
End Function

Funkcija koja se programira počinje ključnom riječi ***Function.*** Prije ove ključne riječi opciono (sve navedeno u uglastim zagradama [ ] su opcioni parametri pri programiranju) mogu da stoje riječi ***Public*** ili ***Private. Public*** znači da je funkcija definisana na globalnom nivou i da se može koristiti na svim formama, koje postoje u programu. ***Private*** znači da je funkcija definisana na lokalnom nivou i da se može koristiti samo na jednoj formi, u kojoj je definisana. Iza kjučne riječi ***Function*** slijedi ime funkcije, koje može da bude samo jedna riječ. Ime funkcije poželjno je da nas asocira na problem, koji rješavamo. Iza imena funkcije u malim zagradama se navode argumenti i njihov tip. Argumenti su u stvari promjenljive preko kojih ćemo u funkciju da unesemo neke vrijednosti iz glavnog programa. Ako imamo više argumenata funkcije, onda se oni međusobno razdvajaju sa zapetom. Ako funkcija ima više argumenata, onda oni mogu biti istog, ali i različitog tipa. Iza male zagrade navodi se tip funkcije, odnosno kojeg tipa će biti vrijednost koju funkcija može da poprimi tokom njenog izvršavanja. Tip funkcije i tip argumenta može biti bilo koji tip podataka, koji podržava *Visual Basic* 2013.

Prva kodna linija u funkciji je obično deklaracija lokalnih promjenljivih, koje se koriste samo u toj funkciji. Zatim se piše radni kod funkcije. Prije ključne riječi ***End Function***, koja označava kraj postojeće funkcije, obavezno treba da stoji ime funkcije kome se pridružuje neka vrijednost. Ovo je posebno bitno, jer funkcija vraća rezultat kroz svoje ime.

Pravljenje korisničkih funkcija najlakše je objasniti na primjerima izračunavanja površina i obima nekih geometrijskih figura. Slijedi primjer koda programa u *Visual Basic*-u u kome se računa površina i obim pravougaonika, a na slici 5.1. prikazana je izvršna verzija ovog programa.

```
  Private Function PovrsinaP(ByVal a As Single, ByVal b As
Single) As Single
      Povrsina = a * b
  End Function
  Private Function ObimP(ByVal a As Single, ByVal b As
Single) As Single
      Obim = 2 * a + 2 * b
  End Function
  Private Sub Povrsina1_Click()
     Dim a, b As Single
     a = Ulaza.Text
     b = Ulazb.Text
     TxtPovrsina.Text = PovrsinaP(ByVal a, ByVal b)
  End Sub
  Private Sub Obim1_Click()
     Dim a, b As Single
     a = Ulaza.Text
     b = Ulazb.Text
     TxtObim.Text = ObimP(ByVal a, ByVal b)
  End Sub
```



Slika 5.1. Primjena funkcija za izračunavanje površine i obima pravougaonika

U navedenom primjeru definisana je funkcija **PovrsinaP**, koja daje površinu pravougaonika, ako su date njegove stranice a i b. Rezervisana riječ ***Function*** predstavlja funkcijski naslov. Zatim dolazi ime funkcije, **PovrsinaP**, slijede u zagradama argumenti funkcije. Argumenti funkcije moraju se podudarati sa stvarnim parametrima, koji će se pojaviti kad se funkcija bude koristila. Argumenti funkcije su deklarisani kao

```
ByVal a as Singl, ByVal b as Singl
```

i kaže, da će funkcija uzeti dva parametra, koji moraju biti realni broj. Konačno, „*as Singl*" na kraju naslova funkcije specificira, da će ova funkcija kao rezultat imati realnu vrijednost.

Naredba dodjele vrijednosti funkciji ima samo jednu kodnu liniju

```
PovrsinaP = a*b
```

koja računa vrijednost funkcije i označava se imenom funkcije. Vrijednost, koja se dobija računanjem, uvijek se obeležava imenom funkcije. Bez obzira koliko postoji naredbi u funkciji koja se programira, konačno jedna od njih mora označavati vrednost koja će se izračunati, a ta je uvijek označena imenom funkcije.

Kako se funkcija koristi u programu? Neka se, na primjer, u programu pojavila naredba koja koristi deklarisanu funkciju **PovrsinaP**, a njen rezultat se upisuje u promjenljivu pod imenom **Zid**.

```
Zid = PovrsinaP (2.16, 4.71)
```

Kada se pri izvođenju programa u računaru, dođe do imena funkcije **PovrsinaP** organizuje se područje memorije, koje se dodjeljuje funkciji. To se područje sastoji od tri lokacije, **a, b** i **PovrsinaP**. Zatim se dodjeljuje vrednost stvarnog parametra formalnim parametrima **a** i **b**:

```
a = 2.16  i   b = 4.71
```

Sada će naredba biti izvršena

```
PovrsinaP = a*b
```

i brojna vrednost 10.1736 je smeštena u memorijsku lokaciju nazvanu **PovrsinaP**. Kad je izvršena ova dodjela vrijednosti, sadržaj memorijske lokacije nazvane **PovrsinaP** biće upotrebljen u programu na mjestu, gdje se poziva funkcija sa **PovrsinaP**.

Može se reći, da je

```
Zid = PovrsinaP(2.16, 4.71)
```

ekvivalentno sa

```
Zid = 10.1736
```

U naredbi funkcijske deklaracije ime funkcije nalazi se na lijevoj strani dodjeljene naredbe i ponaša se kao i svaka druga promjenljiva. Međutim, ako se ime funkcije koristi u nekom izrazu ima drugačije značenje! Već je bilo spomenuto da parametar funkcije može biti i izraz.

Na primjer:

```
y = 2.5
soba = 2 + PovrsinaP (2.0 * y + 1.5, 4.5 - y)
```

ekvivalentno je sa

```
y = 2.5
a = 2.0 * y + 1.5
b = 4.5 - y
Soba = 2 + PovrsinaP (a,b)
```

U ovom primjeru vrednost 7.5 dodeljena je promjenljivoj **a**, a vrednost 2.0 dodeljena je promjenljivoj **b**. Funkcija **PovrsinaP** je dobila vrijednost 15. Izraz **Soba** je na kraju poprimio vrijednost 17.

5.2. PROCEDURE

Procedure su takve strukture kojima se u programu izvršava neki postupak i to jednom ili više puta prema potrebi. To je vrsta potprograma u *Visual Basic*-u. Za razliku od funkcija procedure ne vraćaju izračunatu vrijednost u svome imenu, nego preko jedne ili više promjenljivih.

U zavisnosti od svrhe kojoj služe procedure se dijele na:

1) procedure događaja,
2) procedure osobina i
3) potprogramske procedure.

Pomoću procedura događaja obično se vrši pokretanje nekih operacija u programu. Događaji se programiraju obično nad objektima, koji se postavljaju na radnu površinu. Najčešće korišćeni događaj je jednostruki i dvostruki klik miša. Pored njega koriste se i događaji: prelaz miša preko objekta, klik na tastaturu, promjena vrijednosti u objektu itd. U proceduri događaja, vezanoj za neki objekat, ime procedure se formira od imena objekta kome se iza znaka "_" pridodaje događaj nad tim objektom. Slijedi primjer procedure klik miša na dugme pod imenom **Start**.

```
Private Sub Start_Click()

End Sub
```

Definisanje i uvođenje novih osobina, koja se ne nalaze u prozoru osobina *(Properties),* za neki objekat ili za grupu objekata obavlja se pomoću procedure osobina. Slijedi primjer koda za programsku dodjelu osobine ***Text*** i ***Visible*** objektu *TextBox*, koji nosi ime **Text1**:

```
Text1.Text = "Unos teksta u objekat"
Text1.Visible = False
```

Pomoću potprogramske procedure se vrši rješavanje nekog praktičnog problema, koji će se ponavljati više puta u programu. Procedure se, uglavnom, koriste za obradu ulaznih podataka, prikazivanje rezultata i obradu više osobina vezanih za neki uslov. Osnovna sintaksa potprogramske procedureje je:

[Public ili **Private] Sub** ImeProcedure **([argumenti] [As Tip])**
Sekvencija naredbi
End Sub

Evo jednog jednostavnog programa u kome postoji procedura, koja se koristi za izračunavanje površine i obima pravouganika. Za razliku od funkcije koja je davala uvijek samo jedan rezultat, ova procedura će davati dva rezultata. Ova dva rezultata se u glavni program prenose preko dvije promjenljive **PovrsinaP** i **ObimP,** koje su deklarisane u posebnom modulu.

```
Public PovrsinaP, ObimP As Single
Rem deklaracija 2 globalne promjenljive mora se
uraditi u modulu

Private Sub PovObim(ByVal a As Single, ByVal b As
Single)
    PovrsinaP = a * b
    ObimP = 2 * a + 2 * b
End Sub

Private Sub CBProcedura_Click()
Dim a, b As Single
a = Ulaza.Text
b = Ulazb.Text
Call PovObim(ByVal a, ByVal b)    Rem poziv
procedure
TxtPovrsinaP.Text = PovrsinaP
TxtObimP.Text = ObimP
End Sub
```

Procedura koja se programira počinje ključnom riječi ***Sub.*** Prije ove ključne riječi opciono mogu da stoje riječi ***Public*** ili ***Private. Public*** znači da je procedura definisana na globalnom nivou i da se može koristiti na svim formama, koje

postoje u programu. ***Private*** znači da je procedura definisana na lokalnom nivou i da se može koristiti samo na jednoj formi. Iza kjučne riječi ***Sub*** slijedi ime procedure, koje može da bude samo jedna riječ. Ime procedure poželjno je da nas asocira na problem, koji rješavamo. Iza imena procedure u malim zagradama se navode argumenti i njihov tip. Argumenti su u stvari promjenljive preko kojih ćemo u proceduru da unesemo neke vrijednosti iz glavnog programa. Ako imamo više argumenata u proceduri, onda se oni međusobno razdvajaju sa zapetom. Iza male zagrade se ne navodi tip procedure, jer procedura ne vraća vrijednost preko svog imena, već preko globalnih promjenljivih. Prva kodna linija u proceduri je obično deklaracija lokalnih promjenljivih, koje se koriste samo u toj proceduri. Zatim se piše radni kod procedure, u kome globalne promjenljive moraju da poprime neku vrijednost. Preko globalnih promjenljivih koje se deklarišu u modulima, vrši se prenos vrijednosti iz procedure u glavni program. Procedura se obavezno mora završiti ključnom riječi ***End Sub***. Procedura se iz glavnog programa poziva pomoću ključne riječi ***Call***, iza koje se navodi ime procedure, a zatim u malim zagradama vrijednosti argumenta prema sljedećoj sintaksi

Call Ime procedure (argumenti)

U navedenom primjeru poziv procedure se vršio preko sljedeće kodne linije.

```
Call PovObim(ByVal a, ByVal b)
```

Argumenti u pozivu procedure moraju da odgovaraju argumentima u opisu potprogramske procedure.

Iz prethodnih izlaganja mogu se sumirati zaključci da se procedure i funkcije, pogotovo one koje imaju opštu namjenu, najčešće primjenjuju.

1) Kada treba na više mjesta u programu, po istim obrascima, vršiti izračunavanja za različite vrijednosti argumenata.

2) Kada se ponavljaju pojedina izračunavanja za različite vrijednosti argumenata u skladu sa postavljenim uslovima.

3) Prilikom izdavanja vrijednosti rezultata dobijenih na razne načine i u različitim dijelovima programa.

4) Kada se istovjetna grupa naredbi višestruko pojavljuje u različitim dijelovima programa.

5) Kada se želi postići bolja preglednost programa i brže otkrivanje i otklanjanje grešaka u kodu.

Višestruko ponavljanje pojedinih dijelova programa značajno proširuje i usložnjava programski kod. Cjelishodnije je dio programa, koji treba pisati na više mjesta, izdvojiti u posebnu cjelinu i pisati ga samo na jednom mjestu.

Prednosti upotrebe procedura, uglavnom, su sljedeće.

1) Procedura se jednom definiše, a može da se koristi više puta,

2) Program koji se sastoji od više procedura lakše se razumije od programa sastavljenog iz jedne cjeline,

3) Lakše se otklanjaju greške,
4) Programi se mogu timski razvijati,
5) Pojedine procedure iz standardnih modula mogu da se uklope u različite projekte i
6) Uvođenjem novih procedura obogaćuje se *Visual Studio* jezik.

Sljedeći primjer pokazaće postupak, kojim će se moći uređivati parovi realnih brojeva i to tako, da je na prvom mestu manji broj, a na drugom veći. Ovo uređivanje redoslijeda brojeva, se drugim riječima može nazvati sortiranje brojeva prema rastućem redoslijedu. Procedura u *Visual Basic*-u 2013, kako je rečeno, počinje rezervisanom reči ***Sub*** iza čega dolazi ime procedure, te u zagradi argumenti. Ime se bira tako da asocira na postupak, koji procedura izvršava. Imena procedura moraju biti različita i u programu se nemogu pojaviti dvije procedure sa istim imenom. Imena procedura ne smiju biti ista kao imena bilo koje konstante, promenljive ili objekta, koji je korisnik definisao u programu. Argumenti procedure deklarišu se nabrajajući ih sa njihovim tipovima, isto kao kod funkcija. U datom primjeru deklarišu se dva parametra *a* i *b* oba realnog tipa (*Single*). Neka se sada iskoristi data procedura ZAMJENA za uređenje 4 para brojeva i to:

12, 8
5, 3
29, 22
28, 12

Kod programa će izgledati ovako:

```
  Public Prvi, Drugi As Single
  Private Sub Zamjena(ByVal a As Single, ByVal b As
Single)
     Prvi = b
     Drugi = a
  End Sub
  Rem glavni program klik na dugme
  Private Sub CBZamjena2_Click()
  Dim Niz1(5) As Single
  Dim Niz2(5) As Single
  Niz1(0) = TxtP1.Text Rem upis u niz ulaza
  Niz1(1) = TxtP2.Text
  Niz1(2) = TxtP3.Text
  Niz1(3) = TxtP4.Text
  Niz2(0) = TxtD1.Text
  Niz2(1) = TxtD2.Text
  Niz2(2) = TxtD3.Text
  Niz2(3) = TxtD4.Text
```

```
For i = 0 To 3
   If Niz1(i) > Niz2(i) Then
      a = Niz1(i)
      b = Niz2(i)
      Call Zamjena(ByVal a, ByVal b)
      Niz1(i) = Prvi
      Niz2(i) = Drugi
   End If
Next i
For j = 0 To 3
Rem upis sortiranog niza u MSFlexGrid tabelu
    Me.DataGridView (j + 1, 1) = Niz1(j)
    Me.DataGridView (j + 1, 2) = Niz2(j)
Next j
End Sub
```

Nakon izvršenja program će dati ispis uređenih parova u *DataGridView* tabeli, kao na izvršnoj verziji programa datoj na na slici 5.2.

Slika 5.2. Zamjena mjesta dva parametra

Ista procedura se može iskoristiti za sortiranje, na Primjer, grupe od 3 broja. U tom slučaju procedura **Zamjena** će se u glavnom programu pozivati tri puta pa, će program biti sljedeći, a izgled izvršne verzije programa prikazan je na slici 5.3.

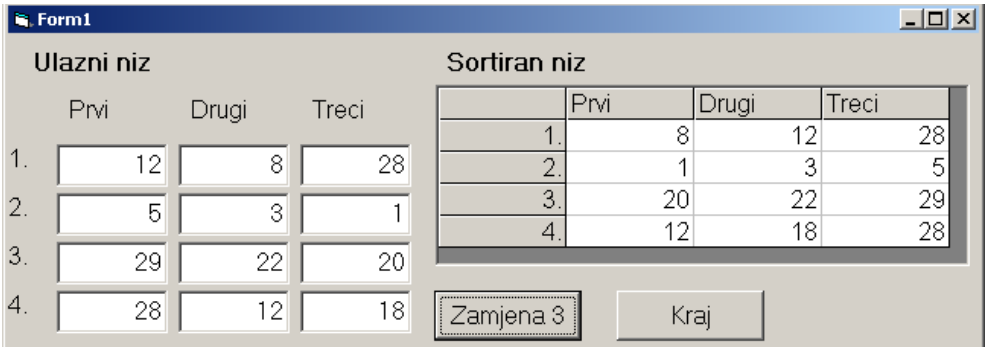
```
Public Prvi, Drugi As Single
Rem deklarisanje dvije globalne promjenljive

Private Sub Zamjena(ByVal a As Single, b As Single)
   Prvi = b
   Drugi = a
End Sub
```

```
Rem glavni program klik na dugme
Private Sub CBZamjena2_Click()
Dim Niz1(5) As Single
Dim Niz2(5) As Single
Dim Niz3(5) As Single
Niz1(0) = TxtP1.Text
Niz1(1) = TxtP2.Text
Niz1(2) = TxtP3.Text
Niz1(3) = TxtP4.Text
Niz2(0) = TxtD1.Text
Niz2(1) = TxtD2.Text
Niz2(2) = TxtD3.Text
Niz2(3) = TxtD4.Text
Niz3(0) = TxtT1.Text
Niz3(1) = TxtT2.Text
Niz3(2) = TxtT3.Text
Niz3(3) = TxtT4.Text
For i = 0 To 3
   If Niz1(i) > Niz2(i) Then Rem zamjena 1 i 2
      a = Niz1(i)
      b = Niz2(i)
      Call Zamjena(ByVal a, ByVal b)
      Niz1(i) = Prvi
      Niz2(i) = Drugi
   End If

   If Niz2(i) > Niz3(i) Then Rem zamjena 2 i 3
      a = Niz2(i)
      b = Niz3(i)
      Call Zamjena(ByVal a, ByVal b)
      Niz2(i) = Prvi
      Niz3(i) = Drugi
   End If

   If Niz1(i)>Niz2(i) Then Rem zamjena 1 i 2 ponovo
      a = Niz1(i)
      b = Niz2(i)
      Call Zamjena(ByVal a, ByVal b)
      Niz1(i) = Prvi
      Niz2(i) = Drugi
   End If
Next i
```

```
For j = 0 To 3
    Me.DataGridView (j + 1, 1) = Niz1(j)
    Me.DataGridView (j + 1, 2) = Niz2(j)
    Me.DataGridView (j + 1, 3) = Niz3(j)
Next j
End Sub
```

Pretpostavimo da su data sljedeća četiri niza od po tri broja:
12, 8, 28
5, 3, 1
29, 22, 20
28, 12, 18

Nakon izvršenja program će dati ispis uređenih parova u *DataGridView* tabeli, kao na izvršnoj verziji programa datoj na na slici 5.3.



Slika 5.3. Zamjena mjesta tri parametra

Prilikom korišćena procedura u programiranju, u glavnom programu se pozivamo na promjenljive deklarisane u programu kao globalne. Promjenljive deklarisane u svakoj proceduri ili funkciji, kažemo da su *lokalne* za tu funkciju ili proceduru. Vrijednost lokalne promjenljive se čuva u memoriji računara samo dok se ta procedura ili funkcija izvršava. Čim se procedura ili funkcija završi iz memorije se brišu vrijednosti svih lokalnih promjenljivih, a u memoriji ostaju samo vrijednosti globalnih promjenljivih. Naredbe u proceduri ili funkciji mogu koristiti globalnu promjenljivu uz uslov da lokalna promjenljiva nema isto ime. One mogu koristiti vrijednost globalnih promjenljivih i dodjeljuju im nove vrijednosti. Ovo osigurava drugi način u kom podatak može prelaziti u ili iz procedure ili funkcije. Vrijednosti, koje su prešle u funkciju ili proceduru mogu biti dodjeljene globalnim promjenljivama, prije nego što je funkcija ili procedura pozvana. Procedura ili funkcija može dodjeliti vrijednosti globalnim promjenljivama i one mogu biti dostupne glavnom programu nakon povratka iz procedure ili funkcije. Zbog toga koristimo argumente češće nego globalne promjenljive.

Ali postoje dvije situacije u kojima često upotrebljavamo globalne promenljive:

1) Želimo sačuvati neke vrijednosti od jednog poziva funkcije ili procedure do sljedećeg. To ne možemo učiniti koristeći lokalne promenljive. Svaki put kad je funkcija ili procedura pozvana njene lokalne promjenljive zauzimaju prostor u memoriji samo dok se funkcija ili procedura ne izvrši. Znači da nakon izvršenja ne postoje lokalne promjenljive, pa je slobodan prostor u memoriji za druge promjenljive. Ako se u programu funkcija ili procedura ponovo poziva, lokalnim promjenljivama se dodeljuje novi prostor u memoriji.

2) Ponekad, mnoge od funkcija i procedura u programu manipulišu nekom opštom strukturom podataka, recimo „površina". U tom slučaju „površina" može biti deklarisana kao globalna promjenljiva, i sve funkcije i procedure mogu koristiti te globalne promjenljive.

5.3. MATEMATIČKE FUNKCIJE

Programski jezik *Visual Basic* 2013 ima veliki broj gotovih matematičkih funkcija, koje stoje programerima na raspolaganju. U tabeli 5.1. date su najčešće primjenjivane funkcije u *Visual Basic*-u 2013. Svaka od navedenih funkcija ima jedan ulazni argument, označen kao *x*. Ovaj argument može biti broj određenog tipa ili bilo koji numerički izraz, koji kao rezultat daje broj. Kod trigonometrijskih i inverznih trigonometrijskih funkcija brojne vrijednosti se izražavaju u radijanima, a ne u stepenima (1 radijan = 180/π stepeni = 57,2958 stepeni).

Tabela 5.1. Matematičke funkcije

Sintaksa funkcije	**Opis funkcije**
Math.Sqrt (x)	Izračunava kvadratni korijen broja ≥ 0.
Math.Exp (x)	Izračunava vrijednost stepena e^x .
Math.Log (x)	Izračunava prirodni logoritam ln x , gdje je x >0.
Math.Log10 (x)	Izračunava logoritam po bazi 10, gdje je x >0.
Int (x)	Izdvaja cijeli dio decimalnog broja. 99=Int(99.2); 99=Int(99.8); -100=Int(-99.2); -100=Int(-99.8)
Fix (x)	Izdvaja cijeli dio decimalnog broja. 99=Fix(9.2); 99=Fix(99.8); -99=Fix(-99.2); -99=Fix(-99.8)
Math.Celing (x)	Daje prvi veći cijeli broj od decimalnog broja. 8=Celing(7.02); 8=Celing (7.87); -7=Celing(-7.02);
Math.Floor (x)	Daje prvi manji cijeli broj od decimalnog broja. 7= Floor (7.02); 7= Floor (7.87); -8= Floor (-7.02);
Math.Sgn (x)	Daje znak broja ili izraza.
Math.Abs (x)	Daje apsolutnu vrijednost broja.
x mod y	Daje ostatak dijeljenja broja x sa brojem y. 1 = 10 Mod 3; 2 = 12 Mod 4.7; 2 = 12 Mod 5; 0 = 12 Mod 4.2; 0= 12 Mod 4
Rnd [(x)]	Daje slučajan broj veći ili jednak 0 i manji od 1.
Math.Round(x[, y])	Zaokružuje broj x na y broj decimalnih mjesta. 3.25=Round(3.251,2)

Sintaksa funkcije	Opis funkcije
Math.Max (x,y)	Daje veću vrijednost od x ili y.
Math.Min (x,y)	Daje manju vrijednost od x ili y.
Math.Sin (x)	Izračunava sinus ugla.
Math.Cos (x)	Izračunava kosinus ugla.
Math.Tan (x)	Izračunava tangens ugla.
Math.Arcsin (x)	Izračunava ugao čiji je sinus jednak broju x.
Math.Arccos (x)	Izračunava ugao čiji je kosinus jednak broju x.
Math.Atan (x)	Izračunava ugao čiji je tangens jednak broju x.

5.4. FINANSIJSKE FUNKCIJE

Programski jezik *Visual Basic* 2013 ima veliki broj gotovih finansijskih funkcija. One koje se najviše koriste date su u tabeli 5.2. Možemo ih podjeliti u tri osnovne grupe: funkcije amortizacije, funkcije anuiteta i funkcije novčanih tokova.

Tabela 5.2. Finansijske funkcije

Sintaksa funkcije	Opis funkcije
DDB(cost, salvage, life, period[, factor])	Amortizacija sredstava u datom vremenskom periodu obračunatu metodom uravnotežene amortizacije.
FV(rate, nper, pmt[, pv[, type]])	Buduća vrijednost nekog ulaganja na osnovu fiksnih periodičnih uplata i stopa.
IPmt(rate, per, nper, pv[, fv[, type]])	Iznos kamate na neko ulaganje za dati vremenski period.
IRR(values()[, guess])	Interna stopa povraćaja za serije novčanih tokova.
MIRR(values(), finance_rate, reinvest_rate)	Daje vrijednost modifikovane interne stope u iskazu povraćaja za serije periodičnih novčanih tokova.
NPer(rate, pmt, pv[, fv[, type]])	Izračunavanje broja otplatnih perioda.
NPV(rate, values())	Izračunava neto sadašnju vrijednost ulaganja na osnovu serije periodičnih novčanih tokova i eskontne stope.
Pmt(rate, nper, pv[, fv[, type]])	Periodični obračun anuiteta.
PPmt(rate, per, nper, pv[, fv[, type]])	Obračun otplate glavnice za određeni period.
PV(rate, nper, pmt[, fv[, type]])	Sadašnja vrijednost budućih fiksnih periodičnih uplata.
Rate(nper, pmt, pv[, fv[, type[, guess]]])	Predstavlja kamatnu stopu na anuitet.
SLN(cost, salvage, life)	Linearna amortizacija sredstava u nekom periodu
SYD(cost, salvage, life, period)	Amortizaciju sredstava za dati upotrebni vijek sredstva izračunat za zadati period.

Funkcija amortizacije sredstava koristi se za izračunavanje novčane vrijednosti, koju sredstva gube za dati vremenski period. U pojedinim zemljama često se daju posebna uputstva za obračun amortizacije, jer iznos poreza i drugih dažbina vlasnika sredstava, u značajnoj mjeri, zavisi od izvršene amortizacije.

Pod amortizacijom kredita podrazumijeva se serija plaćanja koja, obično, predstavlja vraćanje nekog uloga ili kredita u određenim iznosima (anuiteti).

Funkcije novčasnih tokova vrše finansijska izračunavanja na osnovu serija periodičnih rashoda i prihoda. Pri tome se podrazumijeva da negativni brojevi predstavljaju rashode, a pozitivni prihode.

Primjer 5.1.

Uzimate kredit od 10000 $. Kredit trebate vratiti za godinu dana, sa godišnjom kamatnom stopom od 10%, plaćajući jednake mjesečne rate na kraju mjeseca. Pomoću finansijske funkcije PMT izračunati mjesečnu ratu za uzeti kredit pomoću programa napravljenog u *Visual Basic*-u 2013.

Rješenje:

Iznos pojedinačnih otplata duga (kredita) računa se u *Visual Basic*-u 2013 pomoću finansijske funkcije PMT. Ova funkcija ima sljedeću sintaksu:

PMT(rate, nper, pv[, fv[, type]])

Pri čemu je:

Rate interesna stopa po otplatnom periodu (u našem zadatku to je mjesečna kamata na uzeti kredit izražena u procentima). U ovom zadatku ovaj parametar se dobije kada se godišnja kamata na kredit podijeli sa 12, odnosno sa brojem mjeseci u jednoj godini. Za razliku od *Excel*-a u *Visual Basic*-u 2013 interesnu stopu je potrebno iz procenta prebaciti u realni broj. To se vrši djeljenjem sa 100 brojne vrijednosti interesne stope prikazane u procentima (10% = 10/100 = 0.1).

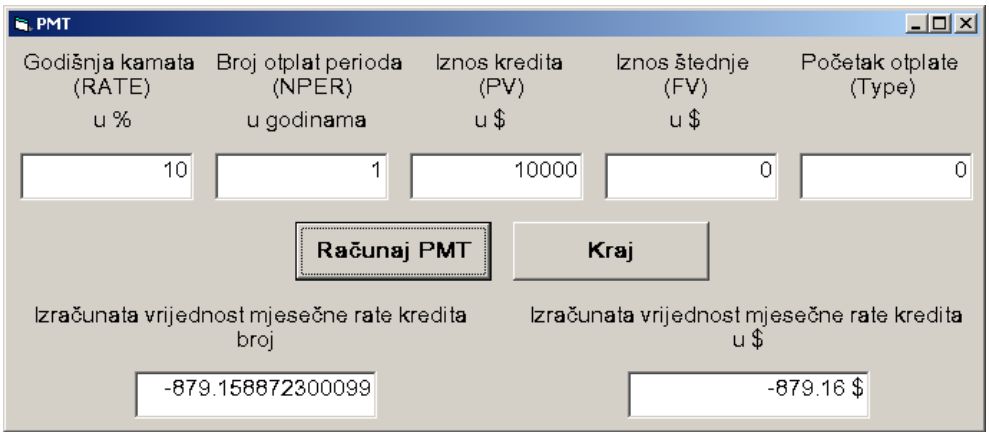
Nper predstavlja preostali broj otplatnih perioda (u našem slučaju to je 12 broj mjeseci koliko kredit treba da se vraća).

Pv označava sadašnju vrijednost koja se otplaćuje (u našem slučaju 10000 predstavlja iznos kredita, koji treba da se otplaćuje).

Fv je opcioni parametar. Predstavlja buduću vrijednost investicije, odnosno vrijednost koju želite da uštedite nakon zadnje uplate. Ako se FV izostavi, smatra se da tada ovaj parametar ima vrijednost 0 (nula), odnosno vrijednost koja će se uštediti je 0. U ovom zadatku ovaj parametar ima vrijednost 0.

Type je opcioni parametar. Parametar koji pokazuje kada dospijeva rata i može imati vrijednost 0 (nula) ili 1 (jedan). Ovaj parametar ima vrijednost 0 (nula), ako rata dospijeva za otplatu na kraju otplatnog perioda, a vrijednost 1 (jedan) ako rata dospijeva na početku otplatnog perioda. Ako se ovaj parametar izostavi u funkciji PMT, onda se podrazumijeva da ima vrijednost 0 i da rata dospijeva za otplatu na kraju otplatnog perioda. U ovom zadatku ovaj parametar ima vrijednost 0, što znači da rata za kredit dospijeva na kraju mjeseca.

Na slici 5.4. prikazan je izgled izvršne verzije programa, koji služi za rješavanje postavljenog zadatka. Zatim je prikazan kod, koji se koristi u ovom programu. Mjesečna rata koja se dobija kao rezultat pomoću funkcije PMT je sa predznakom minus.



Slika 5.4. Finansijska funkcija PMT

```
Private Sub CBPmt_Click()
Rem pretvaranje procenta u broj
KamataGod = URATE.Text / 100
Rem pretvaranje godine u mjesece
PeriodMj = UNPER.Text * 12
Kredit = UPV.Text
Stednja = UFV.Text
Otplata = UTYPE.Text
   Rem poziv finansijske funkcije PMT
MRataBroj=Pmt(KamataGod/12,PeriodMj, Kredit, , Otplata)
RBroj.Text = MRataBroj
 Rem formatiranje zaokruzivanjem na 2 decimalna mjesta
MRataDolar = Format(MRataBroj, "0.00 $")
RPMT.Text = MRataDolar
End Sub
```

5.5. *INPUTBOX* FUNKCIJA

InputBox je sistemska funkcija, koju obezbeđuje *Windows* i koja omogućava korisniku da unese odgovarajući tekst preko tastature i potom potvrdi (*OK* dugme) ili odustane (*Cancel* dugme) od unosa. *InputBox* je funkcija koja vraća tekst, koji korisnik unese u polje za unos. U slučaju da korisnik nije unio ništa ili je odustao

od dijaloga biće vraćen prazan tekst. Uneseni podaci mogu se memorisati u promjenljivoj definisanoj za tu svrhu ili prikazati na ekranu. Sintaksa ove funkcije je:

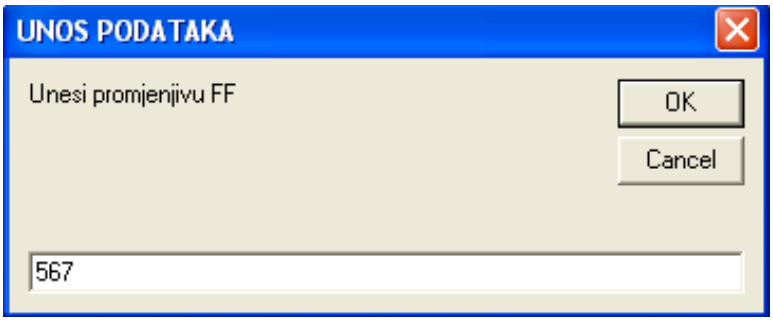
Promjenljiva = InputBox(*prompt*[, *title*][, *default*][, *xpos*][, *ypos*])

gdje:

- **Promjenljiva** podrazumijeva ime promjenljive, kojoj se dodjeljuje ulazni podatak unesen preko ove funkcije.
- ***Prompt*** je obavezan tekst poruke, koji daje informaciju koriniku šta se traži od njega da se unese preko ove funkcije. Porukom se korisnik bliže obavještava, koju akciju treba da preduzme i poželjno je da bude što detaljnija.
- ***Title*** je tekst, koji se pojavljuje na naslovnoj liniji dijaloškog prozora ove funkcije. Ovaj kao i sljedeći parametri ove funkcije su opcioni.
- ***Default*** je inicijalni tekst, predstavlja inicijalnu (*default*) vrijednost, koja se nudi korisniku, kada se ovaj dijaloški prozor pojavi. Može se izostaviti, pa je tada inicijalna vrijednost prazan tekst.
- ***Xpos*** je numerička vrijednost, koja predstavlja broj piksela, koliko će lijeva ivica ovog dijaloškog prozora biti pomjerena od lijeve ivice forme. Ako se ovaj parametar izostavi, onda će dijaloški prozor biti centriran.
- ***Ypos*** je numerička vrijednost, koja predstavlja broj piksela, koliko će gornja ivica ovog dijaloškog prozora biti pomjerena od gornje ivice forme. Ako se ovaj parametar izostavi, onda će dijaloški prozor biti centriran.

Za bolje razumijevanje ove funkcije izvršena je njena prezentacija na primjeru:

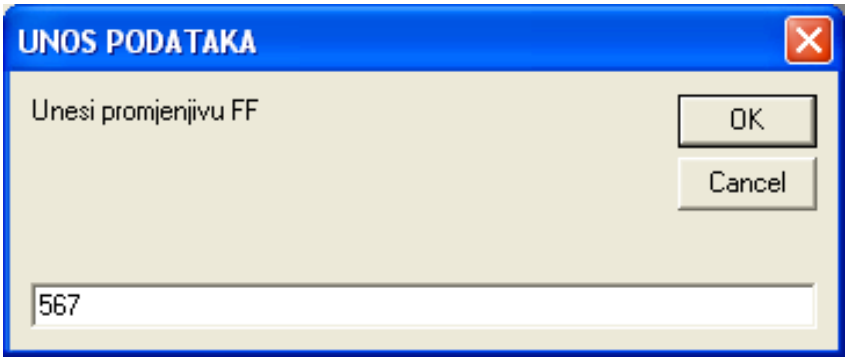
```
FF = InputBox("Unesi promjenljivu FF", "UNOS PODATAKA")
Text1.Text = FF
```



Slika 5.5. Izgled dijaloškog prozora funkcije InputBox

Pozivanjem funkcije *InputBox* pojavljuje se okvir za dijalog, slika 5.5. Na slici se vidi uloga *Prompta* (Poruke) "Unesi promjenljivu FF" i *Naslova* "UNOS PODATAKA". Porukom se zahtijeva unos podataka u obliku stringa. Pri dnu dijaloškog okvira nalazi se okvir sa prostorom za unos podataka. U navedenom

primjeru, pomoću tastature, unesen je tekst "567". Klikom na dugme *OK* uneseni tekst biće lociran u memorijskoj promjenljivoj pod imenom *FF*, dok se klikom na komandu *Cancel* odustaje od memorisanja unesenog podatka. U našem primjeru vrijednost promjenljive *FF* zatim je prenesena u *TextBox* pod imenom *Text1*.



Slika 5.5. Izgled dijaloškog prozora funkcije InputBox

Ako je uneseni podatak numeričke prirode i nad njim treba da se dalje izvode određene matematičke operacije, onda se prilikom njegovog prihvatanja iz *InputBox* funkcije u memorijsku promjenljivu vrši transformacija stringa u broj pomoću funkcije *Val*. Primjer unosa cijene nekog proizvoda:

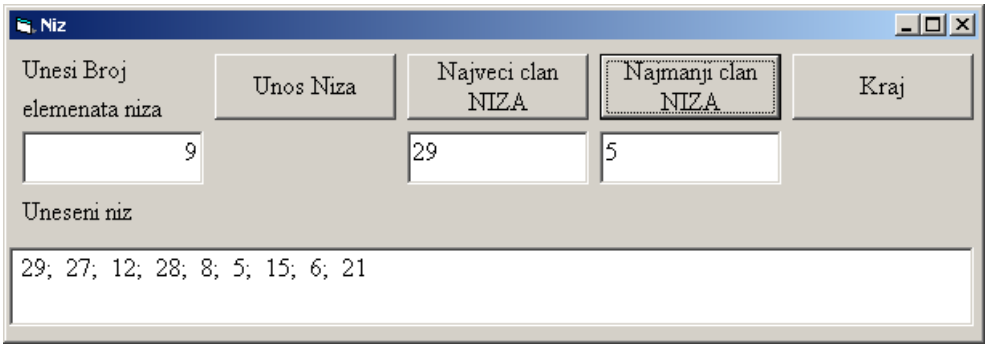
```
Cijena=Val(InputBox("Unesi cijenu proizvoda", "Kasa")).
```

Parametri *Promt* i *Title InputBox* funkcije mogu da se unose i preko tekstualnih promjenljivih, koje su definisane van ove funkcije. Tako da kod predhodnog primjera možemo napisati i na sljedeći način:

```
Naslov1= "Kasa"
Poruka1= "Unesi cijenu proizvoda"
Cijena=Val(InputBox(Poruka1, Naslov1)).
```

Na osnovu ovoga se može lako zaključiti da tekst poruke ne mora da bude fiksan, već da se mjenja u zavisnosti od toka izvršavanja programa. To se najlakše vidi kroz sljedeći primjer u kome je data funkcija, koja učitava N cjelobrojnih vrijednosti i vraća maksimalnu. Na sličan način se nalazi minimalna vrijednost niza brojeva. U navedenom primjeru se uočavaju promjenljive, koje su deklarisane kao globalne, da bi se omogućio prenos vrijednosti iz procedure u funkciju. U ovom primjeru se vidi i način korišćenja funkcije *MsgBox* za ispis poruka, a koja će detaljnije biti opisana u narednom poglavlju. Izgled izvršne verzije ovog programa prikazan je na slici 5.6.

```
Public Niz1(100) As Integer
Public BrojCN As Integer

Private Sub CBUnos_Click()
Dim Poruka1, ii, SNiz1 As String
Dim i As Integer
BrojCN = Val(TxtBrojCN)

If BrojCN < 101 Then
    CijeliNiz.Text = ""
    For i = 1 To BrojCN
       ii = Str(i)
       Poruka1 = "Unesi " + ii + "-ti Clan Niza"
       aa = InputBox(Poruka1, "Unos clanova niza")
       Niz1(i) = aa
       SNiz1 = Str(aa)
       If i = 1 Then
          CijeliNiz.Text = SNiz1
       Else
         CijeliNiz.Text = CijeliNiz.Text + "; " + SNiz1
       End If
    Next i
Else
    MsgBox ("Vas niz je definisan da se moze unijeti
            maksimalno 100 clanova niza")
End If
End Sub

Private Function Maksimum(ByVal N As Integer) As Integer
Dim Max, i As Integer

Max1 = Niz1(1)
For i = 2 To N
    If Niz1(i) > Max1 Then
        Max1 = Niz1(i)
    End If
Next i
Maksimum = Max1
End Function
```



Slika 5.6. Nalaženje maksimalnog elementa niza

5.6. *MESAGEBOX* FUNKCIJA

MsgBox je sistemska funkcija, koju obezbeđuje *Windows*. Ova funkcija omogućava korisniku da na ekranu vidi određenu tekstualnu poruku, u vidu dijaloškog prozora. Na ovom dijaloškom prozoru se pojavljuje jedno ili više komandnih dugmadi, pomoću kojih se bira dalje kretanje kroz program. Sintaksa ove funkcije je:

Promjenljiva = MsgBox(*prompt*[, *buttons*][, *title*])

gdje:

- **Promjenljiva** podrazumijeva ime promjenljive, kojoj se dodjeljuje vrijednost izabranog dugmeta na dijaloškom prozoru.
- ***Prompt*** je obavezan tekst poruke, koji daje željenu informaciju koriniku. Ovaj tekst se navodi pod navodnicima ili preko neke tekstualne promjenljive.
- ***Buttons*** je opcioni parametar, koji definiše dugmad, koja će se pojaviti na ovom dijaloškom prozoru. Ovaj parametar može poprimiti jednu vrijednost navedenu u tabeli 5.3. Ako se ovaj parametar izostavi, smatra se da je izabrana vrijednost 0 i na dijaloškom prozoru će se pojaviti samo komandno dugme ***OK***.
- ***Title*** je tekst, koji se pojavljuje na naslovnoj liniji dijaloškog prozora ove funkcije. Ovaj tekst se navodi pod navodnicima ili preko neke tekstualne promjenljive.

Funkcija *MsgBox* u praksi najčešće ima jedan od sljedeća tri oblika:

1) Promjenljiva = *MsgBox* (Poruka, Opciona dugmad, Naslov)
2) *MsgBox* (Poruka, , Naslov) , ili samo
3) *MsgBox* (Poruka).

Tabela 5.3. Dugmad koja se mogu pojaviti u funkciji MsgBox

Naziv	Vrijednost	Opis
vbOKOnly	0	Prikaz samo **OK** dugmeta (*Difolt*).
vbOKCancel	1	Prikaz **OK** i **Cancel** dugmadi.
vbAbortRetryIgnore	2	Prikaz **Abort**, **Retry** i **Ignore** dugmadi.
vbYesNoCancel	3	Prikaz **Yes**, **No** i **Cancel** dugmadi.
vbYesNo	4	Prikaz **Yes** i **No** dugmadi.
vbRetryCancel	5	Prikaz **Retry** i **Cancel** dugmadi.
vbCritical	16	Prikaz **Critical Message** ikonice.
vbQuestion	32	Prikaz **Warning Query** ikonice.
Exclamation	48	Prikaz **Warning Message** ikonice.
Information	64	Prikaz **Information Message** ikonice.

Iz navedenih formata funkcije *MsgBox* se vidi, da funkcija može da primi nekoliko argumenata i da se rezultat njenog djelovanja dodjeljuje promjenljivoj, ali samo u slučaju navedenom pod 1), kada postoji mogućnost izbora dugmadi. U ostalim slučajevima, pod 2) i 3) funkcija *MsgBox* primjenjuje se bez deklaracije promjenljive i znaka za dodjelu (=).

Slijedi primjer koda programa, u kome se vrijednost izabranog dugmeta u dijaloškom prozoru funkcije *MsgBox*, dodjeljuje promjenljivoj ***Dugme3***. Ova promjenljiva može poprimiti samo dvije vrijednosti ***vbYes*** ili ***vbNo***. Ova vrijednost se poredi u strukturi grananja *If ... Then*. Ako je izabrano dugme ***Yes*** blokira se pristup komandnom dugmetu ***Command1*** i ***Command3*** na trenutno aktivnoj formi, a to su obično dugmad kojim se zatvara baza podataka ili aktivna forma. Na ovaj način se korisnik programa upozorava da nije spasio neke podatke u bazu podataka i da ako želimo te podatke da spasimo, onda je potrebno da se blokira pristup dugmadima, koja zatvaraju bazu ili aktivnu formu. Ako je izabrano dugme ***No*** neće biti blokiran pristup komandnom dugmetu ***Command1*** i ***Command3*** na trenutno aktivnoj formi. Što znači, da je korisnik potvrdio da ne želi da spasi naznačene podatke.

```
  Dugme3 = MsgBox("Da li želiš da spasiš ove podatke o
rezervoaru u bazu podataka?", vbYesNo,"Poruka")
  If Dugme3 = vbYes Then  'ako zelis da spasis
      Command1.Enabled = False
      Command3.Enabled = False
  End If
```

Dobije se potpuno isti dijaloški prozor i kada se primjeni sljedeći kod, u kome se umjesto naziva dugmeta napiše njegova brojna vrijednost iz tabele 5.3.:

```
Dugme3 = MsgBox("Da li želiš da spasiš ove podatke o
rezervoaru u bazu podataka?", 4,"Poruka")

If Dugme3 = vbYes Then  'ako zelis da spasis
    Command1.Enabled = False
    Command3.Enabled = False
End If
```

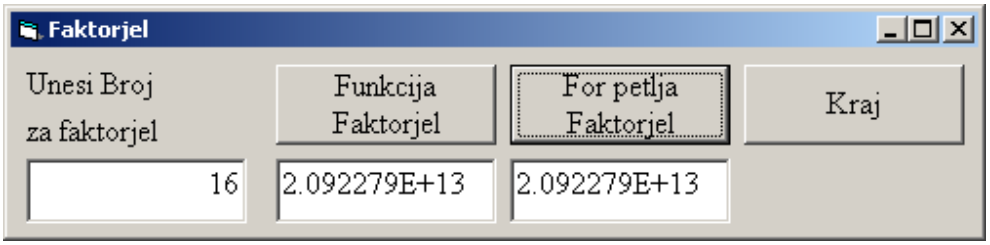
5.7. REKURZIVNE FUNKCIJE I PROCEDURE

Funkcije i procedure mogu tokom svog izvršavanja da pozivaju razne funkcije i procedure, ali mogu da pozivaju i same sebe. Pojam kada funkcija li procedura poziva samu sebe nazivamo rekurzija. Rekurzijom treba rješavati samo one probleme, koji su definisani na rekurzivan način i koji se ne mogu programirati na jednostavniji način. Za programere je nekada primjena rekurzije jednostavnija, ali sa druge strane ona je neefikasnija jer se za izračunavanje rekurzivnih programa troši više vremena. Najpoznatija rekurzivna funkcija je faktorjel, koja je definisana na sljedeći način

$f(0) = 1$
$f(n) = n * f(n-1)$, za $n > 1$.

Suština ove definicije leži u tome da se prvo navedu vrijednosti za početnu vrijednost argumenta. Zatim se vrijednost funkcije izračunava u zavisnosti od predhodno izračunatih vrijednosti funkcije, koje odgovaraju manjim vrijednostima argumenta. Polazeći od navedene definicije faktorjela može se napraviti program sa sljedećim kodom, a izgled rješenja ovog proglema za faktorjel broja 16 prikazan je na slici 5.7. U ovom primjeru promjenljiva **N** je deklarisana na globalnom nivou kao cjelobrojna vrijednost. Programirana funkcija **Faktorjel** takođe će tokom svog izvršenja poprimati samo cjelobrojne vrijednosti (*Integer*). Međutim, vidimo u kodu da je ova funkcija deklarisana da može da poprimi realne brojeve (*Single*). Ovo je urađeno da bi se mogao izračunati faktorjel od većih cjelobrojnih vrijednosti, jer tip *Integer* može da poprimi samo brojne vrijednosti u rasponu od -32768 do 32767, a *Long* u rasponu od -2147483648 do 2147483647. Sa druge strane tip *Single* može da poprimi brojeve u rasponu ±3.40282347E±38, pa je ova osobina u ovom programu zato iskorišćena za definisanje pomenute funkcije faktorjel.

```
  Private  Function  Faktorjel(ByVal  N  As  Integer)As
Single
  If N < 2 Then
      Faktorjel = 1
  Else
      Faktorjel = N * Faktorjel(ByVal N - 1)
    Rem rekurzivni poziv funkcije Faktorjel
  End If
  End Function

  Private Sub CBFaktorjelFunk_Click()
  N = Val(TxtN)
  TxtFaktorjelFunk = Faktorjel(ByVal N)
  End Sub
```



Slika 5.7. Funkcija faktorjel

Rekurzivne procedure mogu da se realizuju na analogan način, kao što je riješena predhodna rekurzivna funkcija. Rekurzivne funkcije često se praktično mogu isprogramirati korišćenjem iterativnog (ponavljajućeg) postupka. Predhodno riješena funkcija za izračunavanje faktorjela prirodnih brojeva može se riješiti nerekurzivno, korišćenjem *For* petlje na sljedeći način.

```
Private Function FaktorFor(ByVal N As Single) As Single
  Rem iterativno rjesenje
  Dim i As Integer
  FaktorFor = 1
  For i = 2 To N
      FaktorFor = i * FaktorFor
  Next i
  End Function

  Private Sub CBFaktorjelFor_Click()
  N = Val(TxtN)
  TxtFaktorjelFor = FaktorFor(ByVal N)
End Sub
```

6. OBJEKTNO ORIJENTISANO PROGRAMIRANJE

Objektno-orijentisano programiranje[9] *(OOP - Object Oriented Programming)* predstavlja pristup realizaciji programa kao modela realnog svijeta. To je poseban način razmišljanja pri projektovanju programa, koji sadrži nekoliko ključnih elemenata, a posebno:

- **objekte (O)** sa njihovim osobinama (atributima) i stanjima kao konkretnim vrijednostima osobina,
- **relacije (R)** između objekata, tzv. korisničke interfejse, i
- **procedure (P)** događaja pomoću kojih se, na osnovu naredbi nekog objektno-orijentisanog programskog jezika (npr. *Visual Basic* 2013), vrši promjena stanja objekata, što i jeste cilj ovakvog oblika programiranja.
- **podatke (S)**.

Prema tome, objektno-orijentisano programiranje može se shvatiti kao uređena četvorka: objekata, relacija, procedura i podataka, tj. ***OOP = (O, R, P, S).***

Program ima cilj da modelira realni svijet, a objektno-orijentisano programiranje da programski model što više približi realnom svijetu. Ova metodologija je nastala, kao odgovor na tzv. softversku krizu - fenomen, koji je nastao kao posljedica ogromnog povećanja složenosti problema modeliranih softverom, upravljanja razvojem softvera i zahtjeva za fleksibilnijim softverom. Pokazalo se da organizacija velikih softverskih projekata nije više dovoljno dobra, da bi bila u stanju da izađe na kraj sa velikim zahtjevima korisnika. Projektovanje, izrada i održavanje softvera postali su preskupi i previše glomazni poslovi, koji su davali sve manje rezultata u odnosu na rad i sredstva koji su u njih ulagani.

6.1. OBJEKTNO ORIJENTISAN NAČIN MIŠLJENJA

Razvoj pravljenja softvera u svijetu traje godinama. Samo programiranje softvera prošlo je kroz nekoliko faza. Programiranje koje je napravilo revoluciju u proizvodnji softvera svodi se na niz programskih naredbi, koje se sekvencijalno izvršavaju jedna za drugom. Ovo i jeste jedna od prvih definicija programiranja, koja danas nije univerzalno primjenjiva. Sekvencijalno programiranje se izvodilo korišćenjem jasno definisanih programskih komandi – sekvenci, koje su se kasnije grupisale u veće module, koji su nazvani procedurama. Samim tim, ovakvo programiranje dobija naziv **proceduralno**. Veliki broj današnjih programskih

[9] Dr Lazar Miličević, Mr Lazar Radovanović, *Programiranje (Visual Basic)*, Ekonomski fakultet, Brčko, 2005, str.24.

jezika je proceduralan i visoko upotrebljiv. Prilikom proceduralnog programiranja, podaci koje program koristi su odvojeni od samog koda, tj. programske logike, što je velika mana, s obzirom na umanjene mogućnosti kontrole pristupa podacima ili njenog potpunog odsustva. Ovo je veliki sigurnosni rizik u programiranju. U filozofiji proceduralnog programiranja, program se posmatra kao crna kutija (*black box*), koja, poput procesa, prima ulazne podatke, obrađuje ih i šalje izlazne podatke. Sami podaci se najčešće čuvaju u nekoj globalnoj bazi podataka ili na udaljenom serveru. Tekst koda programa, prilikom proceduralnog programiranja, smješten je unutar funkcija i procedura, koje obrađuju ulaze i generišu tražene izlaze.

Objektno orijentisano programiranje uvodi jedan poseban termin – objekat. Objektno orijentisano programiranje je, kao što ime kaže, orijentisano ka objektu, koji je osnova ovakvog programiranja. Za početak, za objekat je najbolje reći da je jedna od osnovnih gradivnih jedinica objektno orijentisanog programiranja. Ako program posmatramo kao čovjeka, bio bi sastavljen iz modula različitih funkcija, poput organa. Ovi funkcionalni moduli sačinjeni su iz sopstvenih dijelova, koji su, u stvari, objekti, koji međusobno komuniciraju. Definisanjem jednog ovakvog entiteta, u objektno orijentisanoj programerskoj filozofiji su sjedinjeni podaci i ponašanje u jednom potpunom paketu. Obezbjeđena je jednistvena kontrola pristupa podacima, pošto objekat nije prost tip podatka, poput cijelog broja ili niza znakova, već skup primitivnijih vrsta podataka, kojima je, po potrebi, dodato određeno ponašanje.

6.2. OBJEKAT

U programiranju termin objekat ima veoma jasno definisano značenje. Ali oslonimo se trenutno na definiciju objekta u logičnom smislu. Objekat je, dakle, svaki entitet koji je potrebno predstaviti pomoću određenih standardom definisanih metoda opisivanja prirodnih ili vještačkih entiteta, pojava, događaja. Dakle, ideja je jednostavna, sve se predstavlja objektom, pomoću objekta ili pomoću više različitih objekata. Objekat je jednistvena cjelina, koja sadrži **podatke** i **ponašanje**. Sam po sebi jedan objekat, može sadržati druge podatke, cijele brojeve, nizove karaktera, brojeva i druge objekte. Podaci iskazuju stanje objekta, i zovemo ih **atributi** (osobine objekta). Dakle, objekat ima stanje i ponašanje. Uzmimo kao Primjer objekta jedan obični radni sto kao na slici 6.1.

Da bi bio objekat, potrebno ga je predstaviti određenim stanjem i ponašanjem. Za početak, odgovorimo na pitanje šta je to stanje jednog radnog stola? To su njegovi atributi poput dimenzija, boje, materijal, broj fioka i tako dalje. Na stolu mogu da se nalaze knjige, sveske, olovke i razni drugi predmeti. I oni su objekti, samo drugačijeg tipa nego što je sto. Svaki atribut stola ima neku svoju vrijednost. Ovo su vrijednosti koje jedan objekat razlikuju od drugog. Sve ove vrijednosti su

određenog tipa. Možemo pretpostaviti da su dimenzije ovog stola date u centimetrima. U programu, ovaj podatak bio bi zapisan kao nenegativna realna vrijednost, dok bi ime materijala izrade predstavljao neki niz karaktera. Broj fioka bi bio neki cijeli broj. Ono što na slici nije navedeno, kao dio specifikacije jednog radnog stola jeste njegova mogućnost da se na njega postavljaju drugi objekti poput knjiga, olovaka i drugih predmeta. U programu, ovo bi moralo biti implementirano, tj. kao podatak, atribut, pokazivač stanja. Postojao bi neki niz objekata, koji predstavljaju predmete postavljene na sto.

Objekat sto:
- visina: 110 cm,
- dužina: 140 cm,
- širina: 60 cm,
- boja: bijela
- materijal: iverica

Objekat knjige:
- broj knjiga: 6,
- materijal: papir.

Slika 6.1. Objekat radni sto sa atributima

Sada je potrebno odgovoriti na pitanje, šta je to ponašanje jednog radnog stola? Zdrav razum nam govori da se jedan radni sto ne "ponaša", osim ako u njega nisu ugrađeni neki motori, koji ga pokreću. Mi kao programeri možemo ovo pitanje sagledati iz jednog drugačijeg ugla. Samo ponašanje ne mora značiti da sam sto nešto radi, već i ono što se radi sa stolom može predstavljati ponašanje. Tako, proces slaganja knjiga možemo nazvati **stavik()**, a proces uklanjanja svih knjiga sa stola **uklonik()**. Zagrade nam omogućuju da razlikujemo atribute od metoda, jer su ovo zaista metode ponašanja stola, tj. metode kontrole nad njim.

Upravo smo naveli primjer objekta. Sada je potrebno objasniti još neke termine objektno orijentisanog koncepta, od kojih je svakako najbitniji pojam **klasa**. U prirodi se može sresti veliki broj sličnih entiteta. Neki su identični, neki imaju jako male razlike, a neki su potpuno različiti. Međutim, čovjek je, zahvaljujući svojoj inteligienciji, izvršio neke osnovne klasifikacije stvari, bića i pojava, koje je sretao tokom života. Tako je, na primjer, nastala klasifikacija živih bića na ljude, životinje i biljke. Prije svega je logično da neke pojmove zovemo zajedničkim imenima, iako se oni u nekim detaljima razlikuju. Dakle, klasa u prirodnom poretku definiše neke zajedničke osobine i ponašanja entiteta, koje se ubrajaju pod njenu

klasifikaciju, a koji se razlikuju u detaljima vezanim za određene osobine. U programiranju, klasa se tretira kao šablon objekta, koji se koristi prilikom kreiranja svakog novog objekata. Klasa definiše koje će atribute i ponašanja posjedovati svaki objekat te klase, sve početne vrijednosti i zahtjevana ponašanja realizovana metodama. Dakle, svaki objekat je jedna stepenica svoje klase. Uzmimo kao primjer naš radni sto. Za početak, postoji više različitih radnih stolova. Svaki od njih imaće neke svoje dimenzije, biće određene boje i imaće određeni broj fioka. Svi oni su stolovi, tako da bi klasa "Sto" definisala sve objekte tog tipa, obavezujući svaki od njih da posjeduju navedene atribute da iskažu njihovo stanje.

Sama po sebi, klasa je tip podatka višeg nivoa (cjelobrojne vrijednosti, karakteri i ostali su prosti – primitivni tipovi podataka). Ukoliko bi, na primjer, pomenuti sto mogao da prima i druge predmete pored knjiga, tada bi za predmete koji se postavljaju na sto trebalo dodati atribut tipa "StoPredmet". U objektno orijentisanom konceptu, pojam kada jedan objekat sadrži druge objekte (kuća sadrži zid, prozore, vrata ...) naziva se **kompozicija**. Relacija kompozicije između dva objekta naziva se relacija ***Ima***. Dakle, kuća ***Ima*** prozor.

6.3. METODE I KOMUNIKACIJA MEĐU OBJEKTIMA

Kao što je već rečeno, metode realizuju zahtjevano ponašanje klase. Svi objekti iste klase imaju iste metode, tj. metode koje vrše istu funkciju ili niz funkcija. One realizuju aktivnosti objekata i obezbeđuju ponašanje. Neke metode omogućavaju izmjenu vrijednosti atributa. Ovo je veoma uobičajena praksa u objektno orijentisanom programiranju. U većini slučajeva, pristup podacima unutar nekog objekta bi trebalo da kontroliše sam objekat. Ni jedan objekat ne bi trebalo da može da mijenja vrijednosti atributa drugog objekta, ukoliko to nije eksplicitno dozvoljeno ili ne postoji metoda koja je zadužena za to. Svaka metoda se mora definisati, tj. za svaku metodu moraju biti poznati sljedeći parametri:

- ime metode (jedinstveno opisuje određeno ponašanje),
- argumenti (vrijednosti koje joj se prosljeđuju),
- povratna vrijednost (tip povratne vrijednosti).

Metode služe i za komunikaciju među objektima. Komunikacioni mehanizam između objekata su poruke. Da bi neki objekat pozvao neku metodu drugog objekta, on mu šalje poruku, a odgovor drugog objekta defnisan je povratnom vrijednošću, koja je vraćena nakon inicijalne poruke. Na primjer, ukoliko bi objekat "Mirko" želeo da sazna ime objekta "Mira", on bi pozvao metodu objekta "Mira", pod imenom "getIme()", koja je zadužena da kao svoju povratnu vrijednost vrati niz karaktera, koji predstavljaju ime.

6.4. ENKAPSULACIJA I INTERFEJSI

U toku programiranja od velikog je značaja vjerno kontrolisati pristup podacima nekog objekta. Veliki je propust, ukoliko programer u svakom momentu nije svjestan, ko sve može nehotice promjeniti interne podatke nekog objekta. Ovo se najviše izražava prilikom testiranja softvera, bez čega nijedan softver neće izaći na tržište.

Enkapsulacija je sposobnost objekta da sakrije svoje podatke ili da ih učini selektivno dostupnim drugim entitetima. Određene detalje, koji nisu važni za korišćenje objekta, potrebno je sakriti od ostalih objekata, kako oni direktnim pristupom ne bi mijenjali bitne podatke. Na primjer, ukoliko neki objekat ima mogućnost računanja neke složene matematičke operacije. Korisniku je potrebno pružiti interfejs, koji će primiti parametre, koji su potrebni za računanje, a vratiti samo rezultat. Nije potrebno da korisnik poznaje algoritme, koji se koriste za samo izračunavanje, kao ni međupodatke ili konstante. Enkapsulacija se obezbjeđuje **interfejsima**, tj. implementacijom samog objekta.

Interfejs je osnovno sredstvo komunikacije među objektima. Bilo koje ponašanje objekta mora biti pozvano porukom preko interfejsa, koji mora potpuno da opiše kako korisnik komunicira sa objektom klase. U većini slučajeva, interfejsi su javni, što je logično, s obzirom da su posrednici u komunikaciji među objektima i sa korisnikom (s tim što je, u programu, sam korisnik jedan objekat, predstavljen u kodu).

Suština objektno-orijentisanog programiranja može se, ukratko, rezimirati pomoću sljedećih stavova:

- OOP uvodi u programiranje drugačiji način razmišljanja.
- Troši se manje vremena za implementaciju (programiranje, kodiranje), a mnogo više za samo projektovanje.
- Više se razmišlja o problemu koji se rješava, a manje direktno o programskom rješenju.
- Pažnja se posvećuje dijelovima sistema (objektima) koji nešto rade, a ne algoritmima, tj. načinu rada.
- Prebacuje se pažnja sa realizacije dijelova programa na međusobne veze između tih dijelova. Težnja je da se broj i intenzitet tih veza što više smanji i da se one strogo kontrolišu. Cilj objektno-orijentisanog programiranja je da smanji interakcije između softverskih dijelova.

7. OBJEKTI U *VISUAL BASIC*-U 2013

Visual Basic 2013 posjeduje veliki broj gotovih objekata, koji stoje programeru na raspolaganju u paleti ***Tolbox***. U ovom poglavlju biće predstavljeni samo neki objekti, koji se najčešće koriste u *Visual Basic*-u 2013, a koji su dati u meniju ***Common Controls***. Svi oni koji žele da pronađu dodatne objekte, mogu ih lako pronaći u meniju ***All Windows Forms***. Za većinu objekata su dati primjeri programa, kako se navedeni objekti mogu koristiti prilikom pisanja programa. Da bi se objekti mogli koristiti u programiranju, potrebno ih je prvo dodati na radnu površinu (Formu).

7.1. OBJEKAT *FORM*

U pogramskom jeziku *Visual Basic* 2013 objekat *Form* predstavlja elementarni nivo komunikacije sa korisnicima. Ovaj objekat se kod nas naziva još i obrazac ili radna površina. Na ovaj objekat se postavljaju drugi potrebni objekti. On je tipa kontejner i u relaciji je sa drugim objektima, koji se na njega postavljaju, tako što ih sadrži. Prilikom otvaranja novog projekta, inicijalno se kreira početna forma pod nazivom *Form1*. U jednom programu može postojati više formi, pri čemu treba voditi računa da se omogući prelazak sa jedne forme na drugu. Forma posjeduje više osobina, pomoću kojih podešavamo pojavu i ponašanje forme u programu. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta. Ovo je ključna osobina svakog objekta u *Visual Basic*-u 2013. U jednom projektu svaka forma mora imati svoje jedinstveno ime, preko koga se u kodu programa pozivamo na nju. U jednom projektu ne mogu postojati dva objekta sa istim imenom. Ime objekta mora da počne sa slovom i ime ne može da se sastoji od dvije riječi.
- ***BackColor*** za podešavanje boje pozadine forme.
- ***Text*** za upis teksta koji će biti ispisan u naslovu forme.
- ***Icon*** za definisanje ikone koja će predstavljati formu (vidi se u gornjem lijevom uglu). Ako je u pitanju glavna forma programa, ta ikona najčešće predstavlja i sam program (kada se napravi izvršna EXE verzija).
- ***MaximizeBox*** prikaz dugmeta za maksimizaciju forme (*True* ili *False*).
- ***MinimizeBox*** prikaz dugmeta za minimizaciju forme (*True* ili *False*).

Prilikom rada sa formama mogu se primjenjivati sljedeće metode:

- ***Show*** služi za prikaz određene forme. Komanda u kodu programa ***FrmPrva.Show*** će prikazati formu čije je ime (*name* osobina) ***FrmPrva***.

- ***Hide*** služi za uklanjanje forme sa ekrana, ali je ostavlja u memoriji. Sljedeći ***Show*** metod je prikazuje znatno brže jer je forma već u memoriji i nema potrebe da se učitava sa diska.
- ***Load*** učitava formu, ali je ne prikazuje na ekranu.
- ***Me.Visible*** trenutno aktivnu formu čini vidljivom ili ne vidljivom na ekranu.
- ***Me.Close*** uklanja trenutno aktivnu formu i sa ekrana i iz memorije.

7.1.1. Program sa više formi

Rijetki su danas programi, koji imaju samo jednu formu (radnu površinu). Jedna forma se pojavljuje uglavnom prilikom učenja programiranja ili kod testiranja manjih programa. Prilikom pokretanja *Visual Basic*-a 2013 inicijalno se uvijek pravi prazna forma pod imenom **Form1.vb**. Prva stvar koju bi trebalo uraditi, prije postavljanja objekata na formu i pisanja koda programa, je davanje novog imena formi. To se može uraditi preko prozora *Properties*, kao i za bilo koji drugi objekat. Drugi način za promjenu imena je komanda ***Save Form1.vb as***. Ova komanda će nam omogućiti ne samo da damo novo ime formi, već i da odredimo lokaciju (direktorijum) na kome će ta forma biti spašena. Poželjno je da forma bude spašena na istom direktorijumu, na kome je spašen i projekat (ekstenzija ***.sln***) u kome tu formu koristimo. Ovo posebno ističemo, jer postoji mogućnost da jednu formu može koristiti više programa. Izmjena forme iz jednog programa, tada automatski povlači i izmjenu te forme i kada se koristi iz drugih programa. To često može da napravi veliki problem programerima, jer se može pokvariti program, koji je ranije napravljen i bio je ispravan.

U mnogim slučajevima rješavanje problema mora se odvijati na više formi. Zbog toga se javlja potreba da se u jednom projektu formira više dodatnih formi *(*Form1, Form2, Form3 itd.). Svaka nova forma ima svoje jedinstveno ime, vlastite objekte, osobine i procedure događaja. *Visual Basic* 2013 ne dozvoljava programeru, da u jednom programu budu dvije forme sa istim imenom. Na jednoj formi ne mogu postojati dva objekta sa istim imenom. U jednom programu ali na dvije različite forme se mogu postavljati objekti, koji imaju isto ime, ali ni to nije preporučljivo da se radi. Radi lakšeg praćenja programa i otkrivanja eventualnih grešaka, poželjno je da u jednom programu svi objekti imaju različito ime.

Na prvoj formi *(Form1)* obično se prikazuju: naziv i verzija programa, ulazne poruke, slike, podaci o autorskim pravima, pravima i načinu pristupa programu i informacije o strukturi i korišćenju programa.

Nova forma (*Form2*) se dodaje u postojeći projekat klikom na komandu *Add Windows Forms* u meniju *Project*, sa linije menija.

Pomoću makronaredbe ***Show*** prikazuju se učitane forme. Potpuni format naredbe za prikazivanje forme glasi

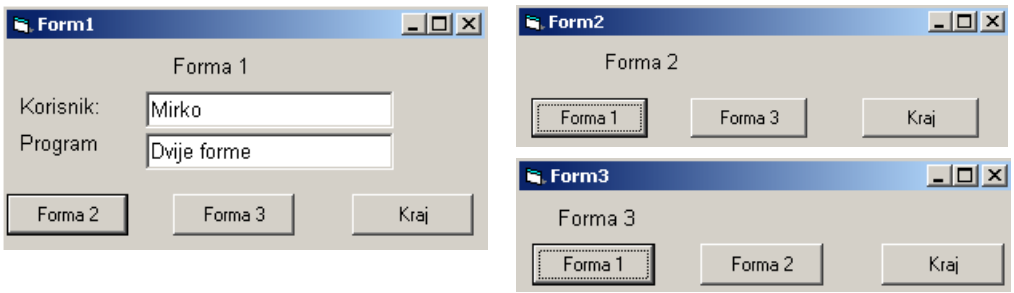
ImeForme.Show()

Brisanje forme postiže se primjenom naredbe ***Me.Close.*** Uklanjanjem forme iz memorije biće odstranjeni i svi njemu pripadajući objekti, promjenljive i osobine, koje su vezane za tu formu. Komanda ***Me.Visible*** se koriti, ako želimo da trenutno vidljiva forma postane ne vidljiva (ili da ne vidljiva forma postane vidljiva), ali da je ona i dalje aktivna kao i sve promjenljive koje se nalaze u njoj.

Prilikom zatvaranja i otvaranja formi potrebno je voditi računa o redoslijedu pisanja komandi.

1) Prvo se piše komanda da učinimo nevidljivom formu, koja je trenutno vidljiva i do koje je stiglo izvršavanje programa.
2) Zatim se piše komanda za otvaranje nove forme.

```
Me.Visible = False
NovaForma2.Show()
```



Slika 7.1. Izgled 3 forme

Na slici 7.1. prikazan je primjer programa koji ima 3 različite forme, od kojih svaka ima po 3 komandna dugmeta. Dva komandna dugmeta su namjenjena za prelazak sa jedne forme na druge dvije. Treće komandno dugme **Kraj** je na svakoj formi, namjenjeno za izlazak iz kompletnog programa. Slijedi kod, koji opisuje rad komandnih dugmadi na prvoj formi.

```
Private Sub ComForma2_Click()
Me.Visible = False     Rem Zatvaranje forme
Form2.Show()           Rem prelazak na formu
End Sub

Private Sub ComForma3_Click()
Me.Visible = False     Rem Zatvaranje forme
Form3.Show()           Rem prelazak na formu
End Sub

Private Sub ComKraj_Click()
End
End Sub
```

Sljedi kod, koji opisuje rad komandnih dugmadi na drugoj formi.

```
Private Sub ComForma1_Click()
Me.Visible = False     Rem Zatvaranje forme
Form1.Show()           Rem prelazak na formu
End Sub

Private Sub ComForma3_Click()
Me.Visible = False     Rem Zatvaranje forme
Form3.Show()           Rem prelazak na formu
End Sub

Private Sub ComKraj_Click()
End
End Sub
```

7.1.2. Određivanje početne forme i prikaz uvodne slike

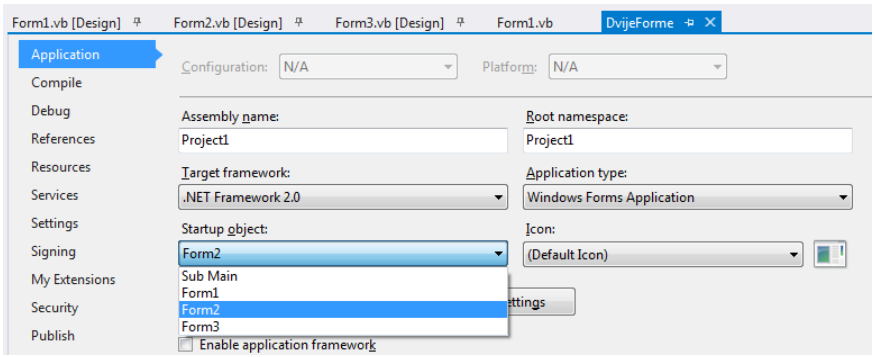
U pravilu, prva napravljena forma vašeg programa određena je kao *početna forma*. Kad vaš program počne sa izvršavanjem, prvo će se prikazati upravo ta forma (pa je i prvi programski kod ,koji će se izvesti kod, koji se nalazi u događaju **Form_Initialize** ili **Form_Load** te forme). Ako želite da se prikaže druga forma kada se pokrene program, morate promijeniti početnu formu.

Promjena početne forme se vrši kroz naredne korake:

1) U meniju **Project**, odaberite opciju **ImeProjekta Properties**. Nakon toga se pokazuje prozor prikazan na slici 7.2. Isti efekat se postiže i klikom miša na opciju My Project u prozoru **Solution Explorer**.

2) Zatim odaberite paletu **Application**.

3) U padajućoj listi **Startup Object**, odaberite formu koju želite kao novu početnu formu.



Slika 7.2. Podešavanje početne forme

Prikaz uvodne slike pri pokretanju programa

Ako pri pokretanju programa trebate izvršiti duži potprogram, kao što je učitavanje veće količine podataka iz baze podataka ili učitavanje nekoliko velikih slika, muzičkih ili video fajlova, možete prikazati uvodnu sliku pri pokretanju. Uvodna slika (*splash scren*) je forma koja obično prikazuje informacije poput imena programa, verzije programa, imena autora, informacija o autorskim pravima i jednostavnije slike. Slika koja se prikazuje kada pokrenete *Visual Basic* 2013 je uvodna slika.

Za prikaz uvodne slike upotrijebite potprogram *Sub Main,* kao početni objekat te uz pomoć postupka *Show* prikažite formu:

```
Private Sub Main()
frmUvod.Show()    Rem Prikaz uvodne slike.
Rem Ovdje možete dodati početne potprograme.
...
frmGlavna.Show()  Rem Prikaz glavne forme.
End Sub
```

Uvodna slika zauzima pažnju korisnika dok se izvršavaju vaše početne komande, stvarajuć i privid da se program brže učitava. Kad su početne komande završene, može se učitati vaša prva forma i izbaciti uvodna slika. Pri oblikovanju uvodne slike dobro je da ona bude što jednostavnija. Ako upotrijebite veće slike i puno objekata, sama uvodna slika mogla bi se sporo učitati.

Završetak programa

Program upravljana događajima prestaje raditi, kad su sve forme zatvorene i nema programskog koda, koji se izvodi. Ako i dalje postoji skrivena forma iako je posljednja vidljiva forma zatvorena, stvoriće se utisak kao da je program završio rad (jer nema vidljivih formi). Ali će program zapravo i dalje raditi dok ne budu zatvorene sve skrivene forme. Ovakva situacija se može pojaviti pri izvršenju programa, jer svaki pristup osobinama ili objektima forme, koja nije učitana, bezuslovno učitava tu formu bez prikazivanja.

Najbolji način izbjegavanja ovog problema, kod zatvaranja vašeg programa, je osigurati izbacivanje svih formi. Ako imate više od jedne forme, možete upotrijebiti naredbu *Me.Close*. Na primjer, vaša glavna forma može imati komandno dugme pod imenom *cmdKraj,* koje dopušta korisniku izlaz iz programa. Ako vaša aplikacija ima samo jednu formu, potprogram događaja *Click* mogao bi biti jednostavan poput ovog:

```
Private Sub cmdKraj_Click()
  Me.Close
End Sub
```

U nekim slučajevima treba završiti rad programa, bez obzira na stanje postojećih formi ili objekata. *Visual Basic* 2013 pruža naredbu *End* za takvu svrhu. Naredba ***End*** trenutno završava rad aktivnog programa. Nakon naredbe ***End*** ne

izvodi se nikakav programski kod, i ne pojavljuju se novi događaji. Tačnije, *Visual Basic* 2013 neće izvršiti potprograme događaja *QueryUnload, Unload* ili *Terminate* nijedne forme. Pokazivači objekata biće oslobođeni, *Visual Basic* 2013 neće izvesti događaje *Terminate* objekata stvorenih iz vaših klasa.

Kao dodatak naredbi *End*, može se koristiti i naredba *Stop* za zaustavljanje programa. Međutim, trebali bi koristiti naredbu *Stop* samo kod testiranja programa, kada tražite greške, jer ona ne oslobađa pokazivače prema objekatima. Tako da pri ponovnom pokretanju može doći do neplaniranih grešaka u radu programa.

7.2. OBJEKAT *LABEL*

U pogramskom jeziku *Visual Basic* 2013 objekat *Label*  se uglavnom koristi za prikaz teksta. Pojavljuje se uvijek na istom mjestu na formi. Ovaj fiksni tekst se najčešće koristi da opiše, šta je namjena nekog drugog objekta pored koga se postavlja. U tom smislu najčešće se koristi u kombinaciji sa objektima *TextBox, ListBox, ComboBox* i ostalima. Ovaj objekat ne može dobiti fokus, tj. prilikom izvršavanja programa nije moguće kurzor postaviti na njega i mijenjati mu sadržaj. Što se tiče korisnika ovaj objekat je samo za čitanje (*Read-Only*). Naravno, sadržaj teksta i ostale osobine ovog objekta možemo programski mijenjati u vrijeme izvršavanja programa, zavisno od potrebe. Tekst koji se ispisuje na ovom objektu može biti fiksan, kada se mijenja preko prozora osobina (*Properties*) ili može biti promjenljiv u toku izvršenja programa, kada se mijenja kroz kod programa. Labele se obično postavljaju lijevo ili iznad drugih objekata, kako bi tekst u njima bliže opisivao te objekte ili čitavu formu. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***TextAlig*** za izbor poravnanja ispisa teksta u objektu (lijevo, desno, centrirano ili uz obadvije strane).
- ***AutoVel*** omogućuje automatsko proširenje širine objekta na onu veličinu koja odgovara unesenom tekstu, ako je izabrana osobina *True*.
- ***Text*** za upis teksta koji će biti ispisan na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.

Nad ovim objektom nije praktično da se programiraju neki događaji.

7.3. OBJEKAT *TEXTBOX*

Objekat *TextBox*  se koristi za unos i izmjenu brojnih i tekstualnih vrijednosti, u vrijeme izvršavanja programa. Koristi se i za prikaz rezultata obrade programa. Objekat može dobiti fokus i najčešće se postavlja poslije *Label* objekta. Inicijalno, naziv prvog objekta je postavljen na *Text1*, sljedeći *Text2* itd. Preporučuje se da se odmah po postavljanju ovog objekta na formu da novo ime objektu, a da ime za svaki ovaj objekat počne sa tri slova „Tex“ (primjer: TexUlaz,

TexIzlaz), kako bi se u kodu programa odmah po imenu međusobno razlikova vrsta objekta (primjer: LabUlaz je ime za labelu, a TexUlaz je ime za TextBox, a PicUlaz je ime za sliku itd.). Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Text*** za upis teksta, koji će biti ispisan u ovom objektu.
- ***TextAlig*** za izbor poravnanja ispisa teksta u objektu (lijevo, desno, centrirano ili uz obadvije strane).
- ***Font*** za izbor tipa, stila i veličine fonta.
- ***MaxLength*** za određivanje maksimalne dužine teksta, koja se može ispisati u objektu.
- ***PasswordChar*** za izbor karaktera, koji će se koristiti za skriveno prikazivanje teksta. Kada se ovaj objekat koristi za unos lozinke, najčešće se za skrivanje teksta koristi karakter "*".
- ***ReadOnly*** može biti True ili False u zavisnosti da li se u objekat može i pisati i čitati ili se iz ovog objekta mogu samo čitati podaci.

Nad ovim objektom se najčešće koriste sljedeći događaji:

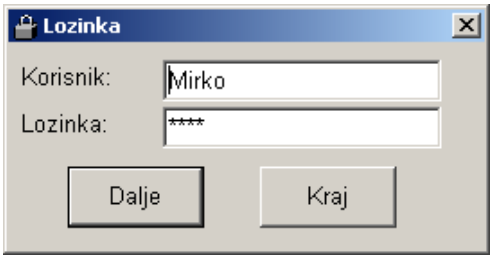
- ***Change*** se pokreće kada se promjeni osobina *Text* u objektu.
- ***LastFocus*** se pokreće kada se napusti ovaj objekat.
- ***KeyPress*** se pokreće pritiskom tastera na tastaturi, kada se kursor nalazi u ovom objektu.
- ***MouseUp, MouseDown, MouseMove*** se pokreće prelaskom miša preko objekta.

U ovaj objekat (sa imenom ***Text1***) se programski preko koda upisuje željeni tekst (mora se navesti između navodnika " ") preko sljedeće sintakse

```
Text1.Text = "Tekst koji se upisuje u objekat"
```

Iz ovog objekta se u željenu promjenjljivu (sa imenom Proba1) programski preko koda upisuje trenutni sadržaj ovog objekta (sa imenom Text1) preko sljedeće sintakse

```
Proba1 = Text1.Text
```



Slika 7.3. Forma za unos lozinke

Enabled	True
HideSelection	True
ImeMode	NoControl
MaxLength	32767
Multiline	False
PasswordChar	*

Slika 7.4. Osobina tekstboksa

Ovaj objekat se može vrlo lako iskoristiti za pravljenje forme, preko koje se omogućuje dalji pristup programu kao na slici 7.3. Za ovo su nam potrebna samo dva ova objekta, od kojih jedan koristimo za unos korisnika, a drugi za unos

lozinke. *TextBox* u koji se unosi korisnik ima standardnu postavku i omogućuje da se potpuno vidi tekst, koji se unosi u njega (odnosno da se vidi ime korisnika). Drugi *TextBox* za unos lozinke potrebno je podesiti, tako da tekst koji se upisuje u njega bude nevidljiv. To se postiže podešavanjem osobine *PasswordChar* na *. U osobinu *Text* se unosi tekst, koji želimo da posluži kao lozinka (u našem primjeru je to tekst "Mira"), kao na slici 7.4.

Slijedi kod programa, koji omogućuje provjeru lozinke.

```
Private Sub Form_Load()
Rem Brisanje sadržaja TextBoxsa kada se forma pokrene
    Txtkorisnik.Text = ""
    TxtLozinka.Text = ""
End Sub
Private Sub Dalje_Click()
Korisnik = Txtkorisnik.Text
Lozinka = TxtLozinka.Text
If Korisnik = "Mirko" Then
  If Lozinka = "Mira" Then
    MsgBox "Korisniku je odobren rad", 0, "Lozinka"
    Forma2.Show vbModal
  Else
    MsgBox "Lozinka nije dobra. Ponovi unos!", 0, "Lozinka"
  End If
Else
  MsgBox "Korisniku nije dobro unešen!", 0, "Lozinka"
End If Rem otkazivanje ulogovanja
End Sub
```

7.4. OBJEKAT *BUTTON*

Pomoću komandnog dugmeta *Button*  se najčešće vrši upravljanje u programu. Koristi se za pokretanje i zaustavljanje nekih procesa u programu. Na sebi može imati ili tekst ili sliku. Kada se ovaj objekat postavi na formu i zatim uradi klik miša na njega, u kodu programa se automatski generiše procedura za rad sa ovim objektom. Na jednoj formi obično ima više ovih objekata. Jedno dugme se obično koristi za pokretanje neke akcije u programu, drugo za prelazak na druge forme, a treće za kraj cijelog programa korišćenjem komande "*End*". Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Text*** za upis teksta koji će biti ispisan na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.

Nad ovim objektom se najčešće koristi događaj ***Click*** ili ***DoubleClick***, a to je jednostruki ili dvostruki klik miša na ovaj objekat.

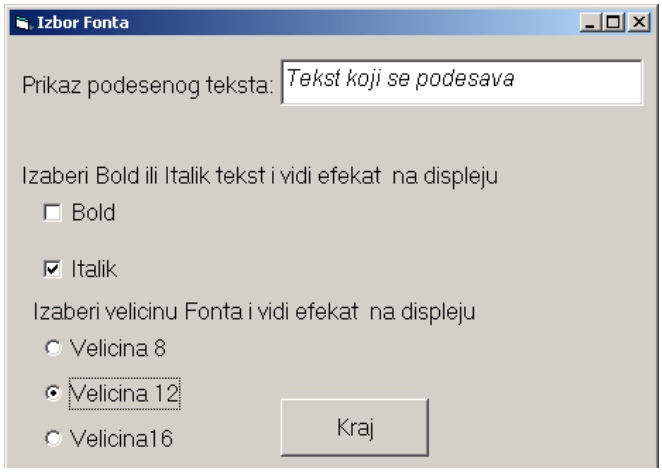
7.5. OBJEKTI *CHECKBOX* I *RADIOBUTTON*

Kontrolna kućica *CheckBox* je objekat, koji omogućuje da korisnik u programu na jednoj formi može istovremeno da izabere više ponuđenih opcija. Ovaj objekat služi kao prekidač za neku opciju u programu. Za razliku od prekidača koji ima dva stanja, ovaj objekat ih ima tri. Omogućava vezivanje za polje iz baze podataka, tipično za polje tipa Da/Ne. Ako imamo više *CheckBox* objekata na formi, oni su međusobno nezavisni, tj. ne isključuju se međusobno.

Sa druge strane dugme izbora *RadioButton* je objekat, koji se koristi kada korisnik u programu može istovremeno da izabere samo jednu od više ponuđenih opcija. Izbor jednog dugmeta izbora, klikom miša, trenutno poništava sve ostale izbore, koji se nalaze na formi ili na jednoj grupi, ako forma ima više grupa. Da bi na jednu formu postavili više grupa ovog objekta, potrebno je koristiti objekat *Panel*, koji služi za razdvajanje grupa. Najbitnije osobine ovih objekata su:

- ***Name*** za definisanje imena objekta.
- ***Text*** za upis teksta, koji će biti ispisan na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.
- ***Checked*** za izbor objekta (može biti izabran *True* ili ne izabran *False*).
- ***CheckState*** za izbor stanja objekta *CheckBox* (može biti ne izabran *Unchecked*, izabran *Checked*, ili uslovno izabran *Interminate*).

Nad ovim objektima se najčešće koristi događaj ***Click***, a to je jednostruki klik miša na ovaj objekat čime se objekti biraju ili poništavaju.



Slika 7.5. Upotreba objekta CheckBox i RadioButton

Na slici 7.5. prikazan je izgled programa u kome se pomoću dva objekta *CheckBox* biraju oblici fonta (*Bold* ili *Italic*), a pomoću tri objekta *RadioButton* bira se samo jedna veličina fonta (8 ili 12 ili 16). Događaj *Click* kontrolne kućice

pojavljuje se onog trenutka kad kliknete na taj objekat. Potprogram ovog događaja ispituje je li kontrolna kućica potvrđena (je li *Value* = *Checked*). Ako je potvrđena, tekst se pretvara u podebljano ili nakošeno pismo postavljanjem svojstava *Bold* ili *Italic* objekata u kome se nalazi tekst, koji se podešava.

Kod koji prati ovaj program je:

```
Private Sub ChkBold_Click()
If ChkBold.Checked = True Then Rem ako je potvrđen
  Tex1.Font = New Font(Tex1.Font, FontStyle.Bold)
Else      Rem ako nije potvrden
  Tex1.Font = New Font(Tex1.Font, FontStyle.Regular)
End If
  End Sub
Private Sub ChkItalik_Click()
If ChkItalik.Checked = True Then    Rem potvrđen
  Tex1.Font = New Font(Tex1.Font, FontStyle.Italic)
Else      Rem ako nije potvrden
  Tex1.Font = New Font(Tex1.Font, FontStyle.Regular)
End If
End Sub

Private Sub Opt8_Click()
Tex1.Font = New Font(Tex1.Font.Name, 8)
     Rem izabrana opcija veličine fonta 8
End Sub
Private Sub Opt12_Click()
Tex1.Font= New Font(Tex1.Font.Name,12)
     Rem izabrana opcija veličine fonta 12
End Sub
Private Sub Opt16_Click()
Tex1.Font= New Font(Tex1.Font.Name,16)
     Rem izabrana opcija veličine fonta 16
End Sub
```

Da bi se inicijalno postavile osobine ovih objekata, kada se program pokrene, potreno je u proceduri *Form_Load* inicijalno postaviti vrijednosti svakog objekta na *True* (ako želimo da objekat bude selektovan) ili na *False* (ako želimo da objekat ne bude selektovan). Kod koji prati ovaj program je:

```
  Private Sub Form_Load()
    ChkBold. Checked = False
    ChkItalik. Checked = False
    Opt8.Checked = True
    Opt12.Checked = False
    Opt16.Checked = False
  End Sub
```

Ako se na jednoj formi želi istovremeno selekovati više objekata *RadioButton*, onda ih je potrebno postaviti u različite okvire (*Panel*). Na slici 7.6 prikazan je izgled programa, u kome se bira jelo sa prilozima.

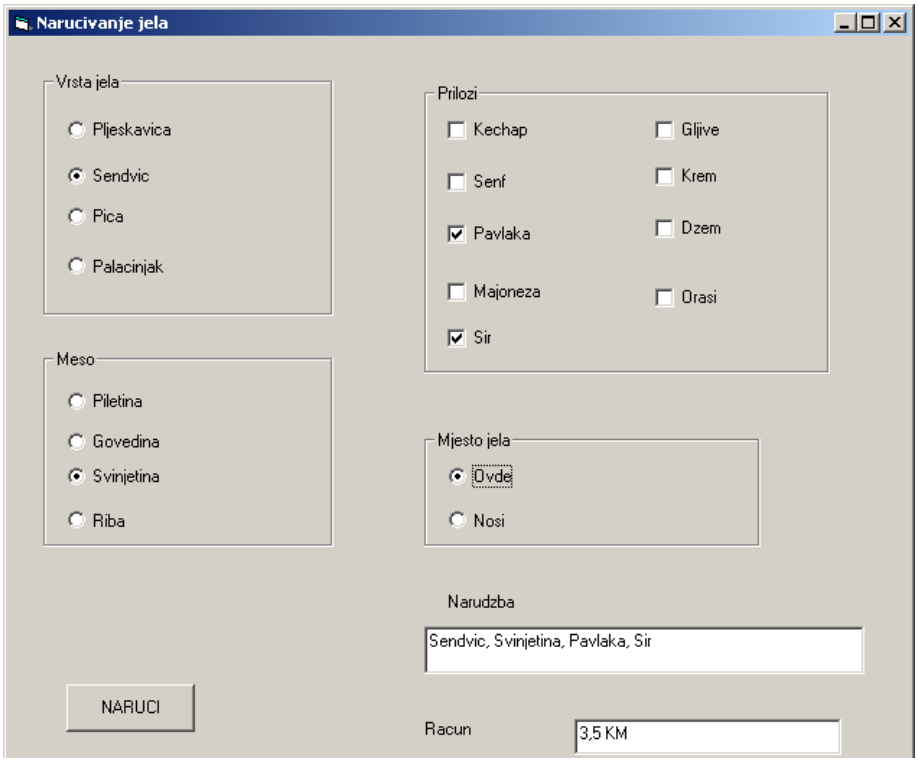
7.6. OBJEKAT *PANEL*

Objekat okvir *Panel*  se koristi za grupisanje drugih objekata najčešće istog tipa na jednoj formi. Na jednoj formi može postojati više okvira, kao u primjeru programa prikazanom na slici 7.6. Postupak grupisanja objekata *RadioButton* u okviru odvija se u sljedećim koracima:

1) Odaberite objekat okvira (*Panel*) u paleti objekata i kreirajte okvir na formi.

2) Odaberite objekat *RadioButton* u paleti objekata i kreirajte ga unutar okvira.

3) Ponavljajte 2. korak za svaki novi objekat *RadioButton*, koji želite dodati u isti okvir.



Slika 7.6. Upotreba objekta CheckBox i RadioButton u više okvira

Kreiranje okvira pa tek zatim kreiranje objekata na okviru, dopušta vam pomjeranje okvira zajedno sa objektima. Ako već postojeće objekte pomjerite na okvir, oni se neće pomjerati pri pomjeranju okvira. Ako imate već kreirane objekte

na formi, a koje želite grupisati na novom okviru, prvo odaberite sve objekte. Zatim ih isijecite sa *cut* i nalijepite sa *paste* na objekat okvir.

Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***BorderStyle*** za definisanje okvira oko ovog objekta.
- ***Font*** za izbor tipa, stila i veličine fonta.

7.7. OBJEKTI *LISTBOX* I *COMBOBOX*

Objekat lista (*ListBox* slika 7.7.) i objekat padajuća lista (*ComboBox* slika 7.8.) omogućavaju prikaz više tekstualnih stavki u vidu liste. Iz ovih listi korisnik može da bira željenu vrijednost, koja se zatim može prenositi u drugi objekat. Liste i padajuće liste su djelotvoran način predstavljanja velikog broja stavki korisniku na ograničenom prostoru. U pravilu, stavke su ispisane okomito u jednoj koloni, iako se može napraviti i lista sa više kolona. Ako je broj stavki veći od onog što liste ili padajuće liste mogu prikazati, u objektu će se automatski pojaviti klizač (*ScrollBar*). Ovi objekti imaju ugrađene metode za dodavanje, brisanje i pronalaženje vrijednosti iz njihovih listi, tokom rada aplikacije.

Slika 7.7. Objekat ListBox

Slika 7.8. Objekat ComboBox

Najbitnije osobine ovih objekata su:

- ***Name*** za definisanje imena objekta.
- ***DataSource*** za povezivanje sa postojećom bazom podataka i tabelom u toj bazi.
- ***DisplayMember*** za povezivanje sa kolonom u izabranoj tabeli baze podataka, sa kojom je lista povezana.
- ***Items*** polje u koje se unose stavke koje se nalaze u listi.
- ***Font*** za izbor tipa, stila i veličine fonta.
- ***Sorted*** za sortiranje elemenata u listi po *Ascii* kodu (ako je postavljeno *True*).
- ***Text*** za prikaz izabrane stavke u kombinovanom okviru (*ComboBox*).

Visual Basic 2013 ima nekoliko komandi za programsko upravljanje listama:
- ***Add*** - programski dodavanje novog člana liste na kraj liste.
- ***Insert*** - programski dodavanje novog člana liste u sredinu liste.
- ***Clear*** - briše sve stavke iz liste.
- ***Remov*** - briše pojedinačne objekte iz liste.
- ***RemovAt*** - briše pojedinačne stavke iz liste, koje su indeksirane kao broj.

Nad ovim objektom se najčešće koristi događaj ***Click*** (jednostruki klik miša na ovaj objekat) i događaj ***DblClick*** (dvostruki klik miša). Kod koji prati ove naredbe u programu je:

```
Private Sub Brisi_Click()
  List1.Items.RemoveAt(3)  Rem brisanje  4  člana  liste,
jer prvi član liste ima redni broj 0
End Sub
Private Sub Brisisve_Click()
  List1.Items.Clear() Rem brisanje svih članova liste
End Sub
  Private Sub Dodaj_Click()
  List1.Items.Add("Dodatak1") Rem dodavanje novog člana
End Sub
  Private Sub Umetni_Click()
    List2.Items.Insert(3, "Unesi4") Rem dodavanje novog
člana u listu između 3 i 4 člana, odnosno na 4 mjesto
End Sub
```

Podaci se u listu mogu puniti direktno preko osobina objekata (*Properties*), preko osobine *List* u koju se dodaje jedan po jedan član liste. Novi član koji se dodaje prikazuje se odmah u listi. Možete dodati stavke u listu bez ponavljajućeg korištenja postupka *AddItem.* U osobinama liste (*Properties*) da upišete stavke u osobinu *List*. Lista se može puniti i sa podacima, koji se preuzimaju iz baze podataka, a primjer jednog takvog koda je:

```
  Rezervoar.MoveFirst
  Rem Pozicioniranje na pocetak tabele Rezervoar u bazi
  Do While Rezervoar.EOF = False
     Priv = Rezervoar("Regbroj")
       Rem Preuzimanje vrijednosti u promjenjljivu Priv
  iz kolone Regbroj tabele Rezervoar
     List1.AddItem (priv)
     Combo1.AddItem (priv)
   Rezervoar.MoveNext
  Loop
```

Iz unesene liste mogu se preuzeti vrijednosti u promjenjljive ili u druge objekte (najčešće TextBox). Preuzimanje podataka iz liste se najčešće radi preko procedure klik miša na listu, a primjer jednog takvog koda je:

```
Private Sub List1_DblClick()
  Text1.Text = List1.Text  Rem Upis u TextBox
  AA = List1.Text     Rem Upis u promjenjljivu AA
End Sub
```

7.8. OBJEKTI KLIZNE TRAKE *HSCROLLBAR* I *VSCROLLBAR*

Klizne trake *HScrollBar* (vodoravna traka) i *VScrollBar* (vertikalna traka) su često vezane uz okvire sa tekstom ili prozore. Međutim ponekad ćete vidjeti i kako se koriste kao ulazne jedinice. Budući da ovi objekti prikazuju trenutni položaj na skali, oni se mogu koristiti zasebno za unos podataka u program. Na primjer, može se koristiti za kontrolu jačine zvuka ili za prilagođavanje boja na slici. Obadva objekta su identična osim po horizontalnoj ili vertikalnoj orijentaciji na ekranu. Klizne trake djeluju nezavisno od ostalih objekata i imaju vlastitu zbirku događaja, osobina i postupaka. Moguće je podesiti minimalnu i maksimalnu vrijednost, korak izmjene vrijednosti u dizajn režimu i u vrijeme izvršavanja programa. Takođe je moguće postaviti i pročitati aktuelnu vrijednost (poziciju) klizača za vrijeme pomjeranja ili po završenom pomjeranju. Prilikom izvršavanja programa vrijednost ovog objekta može se pomoću miša podešavati na tri načina:

- Pritiskom na strelice lijevo i desno, odnosno gore i dole zavisno da li je u pitanju horizontalna ili vertikalna traka za pomjeranje. Ovaj način se zove *SmallChange* (mala izmjena).
- Klikom na lijevu ili desnu, odnosno gornju ili donju stranu objekta u odnosu na aktuelnu poziciju klizača. Ovaj način se zove *LargeChange* (velika izmena).
- Klikom miša na sam klizač, držeći lijevi taster pritisnut i povlačenjem miša lijevo-desno, odnosno gore-dole. Ovaj način se zove *Scroll* (pomjeranje).

Najbitnije osobine kliznih traka su:

- ***Name*** za definisanje imena objekta.
- ***Maximum*** je maksimalna vrijednost klizača (opseg vrijednosti od -2147483648 do 2147483647).
- ***Minimum*** je minimalna vrijednost klizača (opseg vrijednosti od -2147483648 do 2147483647).
- ***LargeChange*** je inkrementalna ili dekrementalna vrijednost, koju dobijemo klikom na kliznu traku lijevo ili desno od dugmeta klizača.
- ***SmallChange*** je inkrementalna ili dekrementalna vrijednost, koju dobijemo klikom na strelice, koje se nalaze na krajevima objekta.
- ***Value*** je tekuća vrijednost i zavisi od trenutnog položaja klizača.

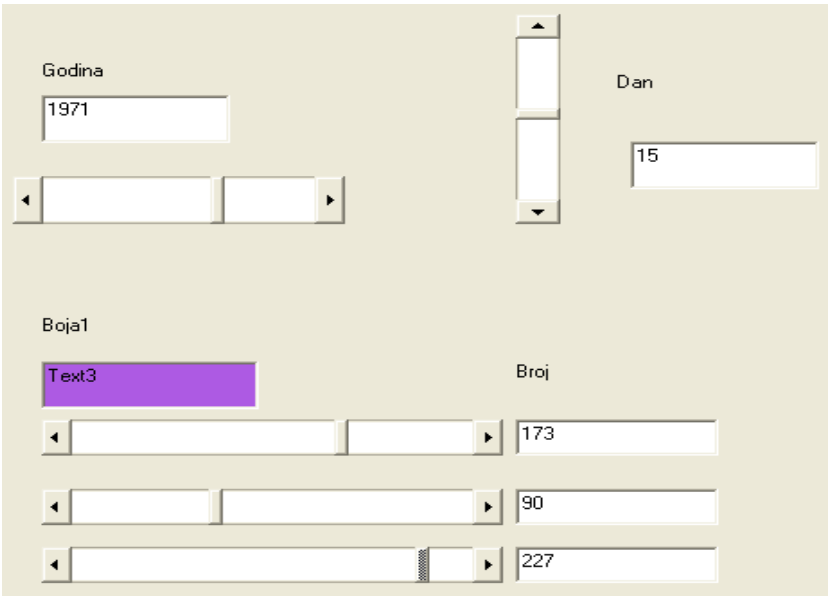
Nad ovim objektom se najčešće koriste sljedeći događaji:

- ***Change*** je događaj, kada se klikne mišem bilo gdje na kliznu traku (više se koristio u ranijim verzijama *Visual Basic*-a).

- ***Scroll*** je događaj, kada se klizač na kliznoj traci pomjera mišem, ali u ovoj verziji programa ovaj događaj će reagovati i kada se klikne mišem bilo gdje na kliznu traku ili se klikne mišem na strelice za početak i kraj klizne trake.

Na slici 7.9. prikazano je kako se klizne trake mogu koristiti kao ulazni objekti u programu. Sada slijedi dio koda, koji pokazuje kako se klizači koriste. Prvo slijedi kod, koji pokazuje kako se u *TextBox* pod imenom **Text1** upisuje trenutna vrijednost klizača. Upotrijebljene su dvije procedure događaja: *Change* i *Scroll*.

```
Private Sub HScGodina_Change() Rem promjena vrijednosti
  Text1.Text = HScGodina.Value
   Rem upis vrijednosti klizača u TextBox
End Sub
Private Sub HScGodina_Scroll() Rem pomjeranje klizaca
Text1.Text = HScGodina.Value
End Sub
```



Slika 7.9. Primjer korišćenja kliznih traka

Zatim slijedi kod programa, pomoću koga se mijenja boja pozadine drugog *TextBox*-a pod imenom **Text3**. Ovo je urađeno pomoću funkcije RGB, koja ima tri ulazna argumenta, na osnovu kojih pravi spektar boja. Ovi ulazni argumenti mogu da poprime vrijednosti od 0 do 255, a u programu se do ovih vrijednosti dolazi pomoću tri klizača, sa dvije procedure događaja: *Change* i *Scroll*.

```
Form_Load()
Dim x, y, z As Integer
x = 0     Rem na početku se klizači postavljaju na nulu
y = 0
z = 0
Text4.Text = x
Text5.Text = y
Text6.Text = z
End Sub
Private Sub HScBoja1_Change() Rem 3 klizaca za 3 boje
Text4.Text = HScBoja1.Value
x = CInt(Text4.Text)
y = CInt(Text5.Text)
z = CInt(Text6.Text)
Text3.BackColor=System.Drawing.ColorTranslator._
FromOle(RGB(x, y, z))   Rem pravljenje boje od 3 broja
End Sub

Private Sub HScBoja1_Scroll()
Text4.Text = HScBoja1.Value
x = CInt(Text4.Text)
y = CInt(Text5.Text)
z = CInt(Text6.Text)
Text3.BackColor=System.Drawing.ColorTranslator._
FromOle(RGB(x,y, z))
End Sub
```

7.9. OBJEKAT *TIMER*

Pomoću objekta *Timer*  moguće je u jednakim vremenskim intervalima izvršavati neku akciju. Ta akcija može biti trivijalna. Na primjer ispis tekućeg vremena i datuma na formi, a može se iskoristiti za mnogo složenije svrhe: pravljenje rezervne kopije, alarm u programu tipa rokovnik i raspored, vremenski kontrolisano štampanje periodičnih izvještaja, slanje faksova i slično. Koristi se i za mjerenje proteklog vremena od nekog fiksnog trenutka, ali i za izradu pauza u toku rada programa. Objekat *Timer* posjeduje osobinu korišćenja sistemskog sata računara na kome se program izvršava. U ovom objektu se vremenski interval može postavljati sa rezolucijom, koja iznosi do hiljaditog dijela sekunde. Takođe možemo programski uključivati i isključivati ovaj objekat. Objekat *Timer* prenosi se, na uobičajeni način, u istoj veličini iz palete objekata na radnu formu (ali se neće vidjeti na formi već u prostoru ispod forme). Objekat je vidljiv samo dok se

piše program, ali kada se program pokrene ovaj objekat neće biti vidljiv na formi. Važno je napomenuti da prilikom intenzivnih operacija sa diskom, računskih operacija i sličnih koje znatno usporljavaju procesor, *Timer* objekat neće tačno odbrojavati vrijeme. Zbog ovoga on nije podesan za pisanje programa nekih vremenski intenzivnih ispitivanja i analiza procesa. Najbitnije osobine objekta *Timer* su:

- ***Name*** za definisanje imena objekta.
- ***Enabled*** za aktiviranje objekta dodjeljivanjem vrijednosti *True*. Isključivanje pokrenutog objekta *Timer*, vrši se postavljanjem osobine *Enabled* na *False*.
- ***Interval*** je brojna vrijednost, koja pokazuje koliko traje dejstvo objekta u mili sekundama (hiljaditi dio sekunde).

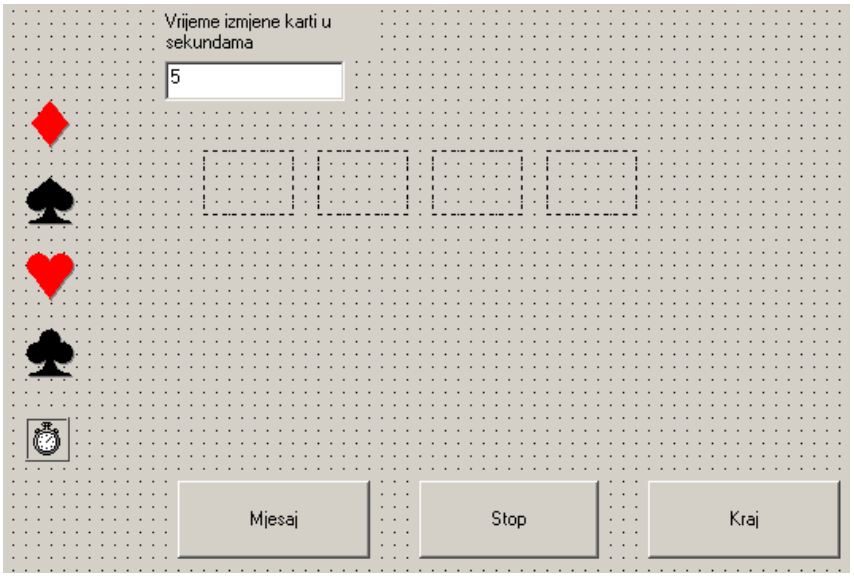
Nad ovim objektom se može koristi samo jedan događaj ***Tick.*** U njega se upisuje kod, koji želimo da se izvrši kada se objekat *Timer* starta.

Jedna od primjena objekta *Timer* je postavljanje digitalnog sata na radnoj formi u labeli namijenjenoj za te svrhe. Digitalni sat pokazuje sistemsko vrijeme u formatu: sat:minuta:sekunda (**16:07:21**). Da bi sat funkcionisao pod nadzorom objekta *Timer* i uz sistemsko vrijeme računara objektu *Timer* potrebno je zadati vrijednosti *True* za osobinu *Enabled* da bi objekat bio uključen i 1000 za osobinu *Interval* da bi sat ažurirao vrijeme svake sekunde. Da bi se dobio zvučni signal nakon svakog aktiviranja objekta *Timer* koristi se komanda *Beep.* Slijedi dio koda, koji pokazuje, kako se digitalni sat pojavljuje u labeli pod imenom **Label1**.

```
Private Sub Form_Load()
   Timer1.Interval = 1000   Rem Interval 1 sekunda.
   Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick()
    Label1.Caption = CStr(TimeOfDay)
Beep
End Sub
```

Druga značajna primjena objekta *Timer* jeste definisanje vremena čekanja. Objekat *Timer* može se uključiti, kako bi se program zaustavio na određeno vrijeme i na taj način omogućili korisniku izvršenje neke akcije. Na primjer pomoću ovog objekta se može podesiti potrebno vrijeme za unošenje lozinke. Ako se akcija uspješno ne obavi u predvidenom vremenu, objekat *Timer* onemogućava nastavak rada na aplikaciji. Slijedi dio koda, koji pokazuje kako na jednoj formi mogu koristiti dva objekta *Timer*. Objekat *Timer1* se koristi za prikaz digitalnog sata u labeli *Label1* svake 2 sekunde i prikazivanje dugmeta *Comand1*. Objekat *Timer2* se koristi za pravljenje kašnjenja od 6 sekundi nakon čega se ovaj objekat isključuje.

```
Private Sub Form_Load()
   Timer1.Interval = 2000   Rem Podesiti 2 sekunde.
   Timer2.Interval = 6000   Rem Podesiti 6 sekundi.
End Sub
Private Sub Command1_Click()
    Command1.Visible = False
    Timer1.Enabled = True
End Sub
Private Sub Command2_Click()
      Command2.Visible = False
      Timer2.Enabled = True
End Sub
Private Sub Timer1_Tick()
    Label1.Caption = Time
    Command1.Visible = True
End Sub
     Private Sub Timer2_Tick()
        Label1.Caption = "drugo dugme"
   Command2.Visible = True
   Timer2.Enabled = False
End Sub
```



Slika 7.10. Izgled razvojne forme

Slijedi primjer programa u kome se objekat *Timer* koristi za periodično mješanje 4 karte. Na slici 7.10. prikazana je razvojna forma ovog programa na kojoj se vidi objekat *Timer*, jedna labela, jedan *TextBox* (za unos vremenskog kašnjenja u sekundama), tri komandna dugmeta i 8 okvira za unos slika. Okviri za unos slike su podjeljeni u dvije cjeline, organizovane kao dva niza objekata od po četiri okvira (PicNizU i PicNizI). Da bi deklarisali niz objekata (PictureBox) koji se može koristiti u cijelom programu, potrebno je prvo deklarisati javnu klasu (Karte), a zatim i dva niza objekata prema sljedećem kodu.

```
Public Class Karte
Dim PicNizU As New ArrayList Rem za 4 ulazne slike
Dim PicNizI As New ArrayList Rem za 4 izlazne slike
Dim i, J, x As Integer
```

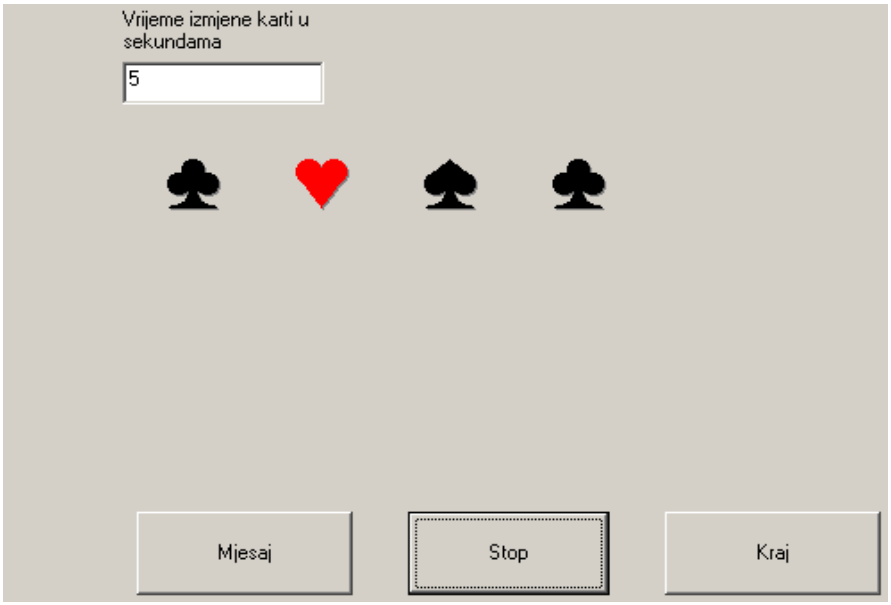
Zatim je potrebno u ova dva niza objekata unijeti objekte (PictureBox) preko njihovog imena (PicU0, PicU1, PicU2, PicU3, PicI0, PicI1, PicI2 i PicI3 su imena objekata PictureBox), koji su predhodno postavljeni na formu.

```
PicNizU.Add(PicU0)
PicNizU.Add(PicU1)
PicNizU.Add(PicU2)
PicNizU.Add(PicU3)

PicNizI.Add(PicI0)
PicNizI.Add(PicI1)
PicNizI.Add(PicI2)
PicNizI.Add(PicI3)
```

Organizovanje objekata u niz omogućuje lakše ponavljanje iste komande nad više sličnih objekata upotrebom *For* petlje. Na slici 7.11. prikazana je izvršna verzija ovog programa na kojoj se vide samo 4 okvira za slike, ali u kojima se stalno u istim vremenskim razmacima mijenjaju slike. Takođe se može uočiti da se u izvršnoj verziji programa ikonica objekata *Timer* ne pojavljuje. Kod ovog programa je sljedeći.

```
Private Sub Form_Load()
  Timer1.Enabled = False Rem zaustavljanje tajmera
End Sub
Private Sub Stop_Click()
    Rem dugme za zaustavljanje tajmera
  Timer1.Enabled = False
End Sub
Private Sub Mjesaj_Click()
   Rem dugme za pokretanje procesa mješanja karti
Timer1.Interval = Val(Text1.Text) * 1000
  Rem preuzimanje vremena ponavljanja iz TextBoxa
```

```
Timer1.Enabled = True
  Rem aktiviranje objekta Timer1
 For i = 0 To 3
      PicNizU(i).Visible = False
    Rem polazne 4 slike da se učine nevidljive
 Next i
End Sub
Private Sub Timer1_Tick()
   Rem procedura koja pokazuje šta tajmer radi
 For J = 0 To 3
     x = System.Math.Round(Rnd() * 3)
     PicNizI(J).Image = PicNizU(x).Image
        Rem slučajno pojavljivanje 4 karte
 Next J
End Sub
```



Slika 7.11. Izvršna verzija programa

Za navedeni kod programa provjerite da li sve četiri karte imaju istu vjerovatnoću pojavljivanja, ili neke karte imaju veću, a neke manju vjerovatnoću pojavljivanja? Koje su to karte u pitanju? Ako je potrebno, sami doradite kod da to provjerite.

Posebno značajnu ulogu ima primjena objekta *Timer* prilikom kreacije raznih animacija, u kombinaciji sa naredbom ***Point***. Naredbom *Point* definiše se kordinata

na koju se objekat (ili više njih) pomjera na formi, dok se uz pomoć *Timer*-a određuje brzina kretanja. Format naredbe ***Point*** ima sljedeći oblik:

imeobjekta. Location = New Point(kordinata po X-osi, kordinata po Y-osi)

Primjer primjene naredbe ***Point*** nad objektom pod imenom **Label1**:

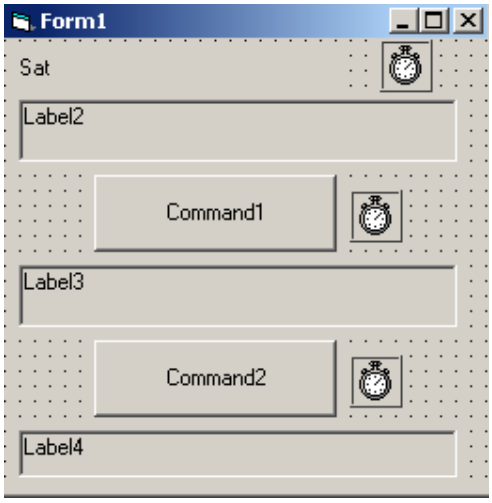
```
Label1.Location = New Point(20, 30)
```

gdje 20 i 30 predstavljaju koordinate u pikselima za vodoravno i vertikalno pomjeranje objekta labele pod imenom **Label1**. U tom slučaju podrazumijeva se da je gornje lijevo tjeme (vrh) forme koordinatni početak sistema čiji prvi broj predstavlja x-osu, a drugi broj *y*-osu. Da bi dobili kontinuirano pomjeranje objekta potrebno je da se vrijednost kordinata x i y mijenja u nekoj petlji ili pomoću objekta tajmer. Brzina pokreta objekta određuje se zadavanjem vrijednosti broja milisekundi osobini ***Interval*** objekta *Timer*.

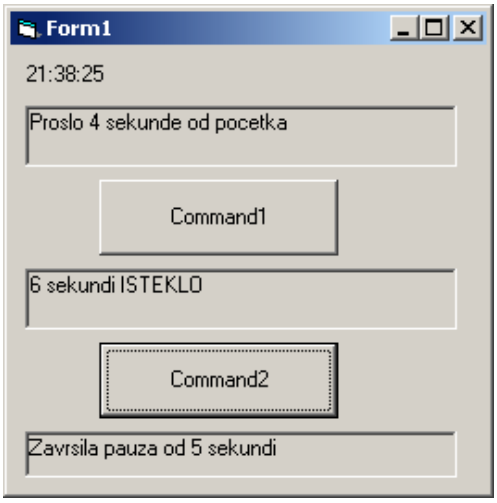
Slijedi primjer koda programa, koji omogućuje da se u objekat *Label* (pod imenom Label1) prvo postavi digitalni sat. Zatim se ovaj objekat kontinuirano pomjera po formi u jednakim vremenskim razmacima i u jednakim skokovima. Kada objekat dođe do ivica forme on se sam vraća u suprotnom smjeru. Na taj način se simulira odbijanje objekta od ivice. Što je vremenski period ponavljanja kraći i dužina skoka manja to se dobija bolja animacija, odnosno osjećaj da se objekat kontinuirano kreće.

```
      Dim DeltaX, DeltaY As Integer
      Dim StartX, StartY As Integer
      Dim Labx, LabY, LabH, LabW As Integer
      Dim Form1W, Form1H As Integer
Private Sub Form_Load()
      StartX = 5
      StartY = 5
      DeltaX = 50
      DeltaY = 50
      Form1W = Me.Size.Width
      Form1H = Me.Size.Height
      Timer1.Interval = 1000
      Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
        Label1.Text = CStr(TimeOfDay)
        Labx = Label1.Location.X
        LabY = Label1.Location.Y
        LabH = Label1.Height
        LabW = Label1.Width
```

```
            If Labx > Form1W Then
                DeltaX = -DeltaX
            End If
            If Labx < 0 Then
                DeltaX = -DeltaX
            End If
            If LabY > Form1H Then
                DeltaY = -DeltaY
            End If
            If LabY < 0 Then
                DeltaY = -DeltaY
            End If
            StartX = StartX + DeltaX
            StartY = StartY + DeltaY
            Label1.Location = New Point(StartX, StartY)
End Sub
```



Slika 7.12.a Forma sa 3 tajmera



Slika 7.12.b Izvršna verzija programa

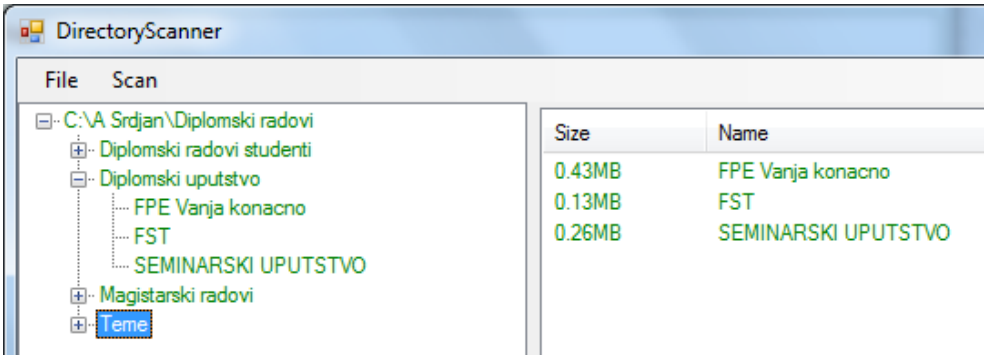
U jednom programu može da postoji više tajmera, koji mogu imati različita vremena i različite funkcije. Na slici 7.12. prikazan je primjer programa sa tri tajmera. Prvi tajmer se koristi za prikaz digitalnog sata u labeli. Drugi tajmer se koristi za kašnjenje od 4 sekunde. Treći tajmer unosi kašnjenje od 6 sekundi, a nakon toga se aktiviraju prva dva tajmera.

7.10. OBJEKAT *TREEVIEW*

Okvir za prikaz direktorijuma (*TreeView*) namjenjen je za prikaz i izbor svih direktorijuma, koji se trenutno nalaze na izabranoj disk jedinici. Ovaj objekat ispisuje korisničku strukturu direktorija i podirektorija. Struktura se prikazuje u obliku stabla kao u *Windows Exsplor*-u, a primjer korišćenja ovog objekta prikazan je na slici 7.13. Ovaj objekat se najčešće kombinuje sa objektom *ListView* za izbor fajlova. Ovaj objekat se postavlja preko objekta (veže se za njega) *Splitter* (ima jedan prozor) ili *SpliterContainer* (ima dva prozora).

Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Dock*** osobina koja služi za potpuno povezivanje sa objektom *Splitter* ili *SpliterContainer*, ako se izabere opcija *Fill*, a ako se izabere opcija *None* onda se može mijenjati veličina ovog objekta.
- ***Font*** koja omogućuje podešavanje veličine i oblika slova.
- ***PathSeparator*** koja se najčešće podešava na "\", a koristi se za izbor simbola, koji razdvaja imena dva direktorijuma.



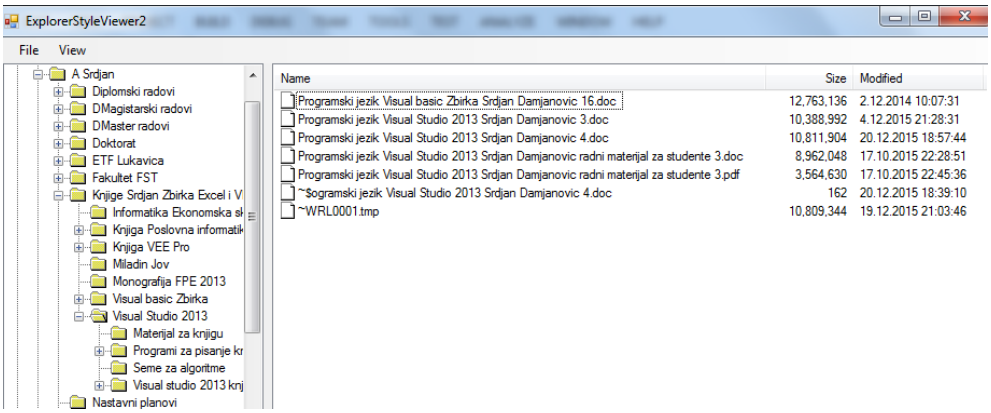
Slika 7.13. TreeView i ListView

7.11. OBJEKAT *LISTVIEW*

Okvir za prikaz fajlova (*ListView*) namjenjen je za prikaz veličine i imena direktorijuma, koji se trenutno nalaze na izabranom (aktivnom) direktorijumu. Ovaj objekat ispisuje fajlove u vidu liste. Primjer korišćenja ovog objekta prikazan je na slici 7.13. Ovaj objekat se najčešće kombinuje sa objektom *TreeView* za izbor direktorijuma. Ovaj objekat se postavlja preko objekta (veže se za njega) *Splitter* (ima jedan prozor) ili *SpliterContainer* (ima dva prozora).

Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Dock*** osobina koja služi za potpuno povezivanje sa objektom *Splitter* ili *SpliterContainer*, ako se izabere opcija Fill, a ako se izabere opcija None onda se može mijenjati veličina ovog objekta.
- ***Font*** koja omogućuje podešavanje veličine i oblika slova.
- ***View*** koja može biti: *Details* (najbolje), *LageIcon*, *SmallIcon*, *List* i *Title*.



Slika 7.14. TreeView i ListView

7.12. OBJEKTI *SPLITTER* I *SPLITERCONTAINER*

Objekat *Splitter* i *SpliterContainer* koristi se za pravljenje strukture eksplorera u više različitih varijanti. Moguće je pregledati imena, veličinu, vrijeme nastanka korisničkih fajlova kao na slici 7.14. Pomoću ovog objekta se mogu pratiti svi programi koji se izvršavaju trenutno na računaru. Ovom objektu nije moguće na formi dati novo ime, jer je na formu moguće postaviti samo ovaj objekat (i objekte za pravljenje padajućeg menija), koji zauzima prostor čitave forme. Ovim objektom se upravlja programski preko koda. Sada slijedi primjer koda za prikaz strukture eksplorera.

```
Private Sub ExplorerStyleViewer_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
 flvFiles=New FileListView() REM pravljenje nove liste
 SplitContainer1.Panel2.Controls.Add(flvFiles)
 flvFiles.Dock = DockStyle.Fill
  REM pravljenje strukture stabla u prvom prozoru
 dtvwDirectory = New DirectoryTreeView()
```

```
 SplitContainer1.Panel1.Controls.Add(dtvwDirectory)
 dtvwDirectory.Dock = DockStyle.Left
 AddHandler dtvwDirectory.AfterSelect, _
            AddressOf DirectoryTreeViewOnAfterSelect
    Rem Dodavanje View meni u padajući glavni meni.
 Dim menuView As New ToolStripMenuItem("&View")
 MenuStrip1.Items.Add(menuView)
       REM Dodavanje 4 padajuća menija u listu View
 Dim astrView As String() = {"Lar&ge Icons", "S&mall
Icons", "&List", "&Details"}
 Dim aview As View() = {View.LargeIcon, View.SmallIcon,
View.List, View.Details}
 Dim eh As New EventHandler(AddressOf MenuOnViewSelect)
 Dim i As Integer
   For i = 0 To 3
    Dim miv As New MenuItemView()
    miv.Text = astrView(i)
    miv.View = aview(i)
    miv.Checked = False
    AddHandler miv.Click, eh
    If i = 3 Then
       mivChecked = miv
       mivChecked.Checked = True
       flvFiles.View = mivChecked.View
    End If
    menuView.DropDownItems.Add(miv)
  Next i
End Sub
```

Sljedeći kod se koristi, da bi prikazali sve fajlove, koji postoje na izabranom direktorijumu u listi.

```
Public Class ExplorerStyleViewer
    Private dtvwDirectory As DirectoryTreeView
    Private flvFiles As FileListView
    Private mivChecked As MenuItemView

Sub DirectoryTreeViewOnAfterSelect(ByVal obj As Object,
ByVal tvea As TreeViewEventArgs)
       flvFiles.ShowFiles(tvea.Node.FullPath)
End Sub
```

7.13. OBJEKAT *PICTUREBOX*

Ovaj tip okvira za sliku se koristi za postavljanje slika na formu, direktno preko osobina objekta ili programski kroz kod programa. Slike koje se postavljaju trebaju biti snimljene negdje na računaru, a poželjno je da budu na istom direktorijumu gdje je snimljen i projekat, u nekom od sljedećih formata: *Bitmap, Icon, Jpeg* i *Gif*.

Najbitnije osobine ovih objekata su:

- ***Name*** za definisanje imena objekta.
- ***Image*** za povezivanje objekata sa gotovom slikom (meni sa 3 tačke).
- ***SizeMode*** omogućuje mijenjanje veličine okvira za sliku tako da bude prikazana kompletna slika, kada se ova osobina postavi na *AutoSize*. Ako je ova osobina postavljena na *Normal*, onda je okvir uvijek fiksan bez obzira na veličinu slike. Ako je slika veća od okvira, biće prikazan samo njen gornji lijevi dio. Ako je ova osobina postavljena na *StretchImage*, onda je omogućeno mijenjanje veličine slike, tako da ona u potpunosti popuni okvir za sliku i tada može doći do deformacije slike po visini ili širini. Ako je ova osobina postavljena na *CenterImage*, onda se prikazuje samo centar slike, do veličine postavljenog okvira za sliku. Ako je ova osobina postavljena na *Zoom*, onda se prikazuje kompletna slika, ali umanjeno tako da stane u predviđeni okvir za sliku na formi.
- ***Size*** omogućuje definisanje širine slike (preko osobine ***With***) i visine slike (preko osobine ***High***t) u pikselima.

Programski se slika može postavljati u okvir za slike preko procedure **System.Drawing.Image.FromFile()**. Sintaksa koda kada se slika pod imenom "Pr1.gif" preuzima sa tačno poznatog direktorijuma (“c:\slike\Pr1.gif”) u okvir za sliku pod nazivom **Slika**, je sljedeća:

```
Slika.Image=System.Drawing.Image.FromFile(“c:\slike\Pr1.gif”)
```

Sintaksa koda kada se slika pod imenom "Leptir1.bmp" preuzima u okvir za sliku pod nazivom **Slika2,** sa direktorijuma na kome se nalazi snimljen projekta u kome se trenutni program piše, ali na njegovom pod direktorijumu **\bin**, je sljedeća:

```
Slika2.Image=System.Drawing.Image.FromFile
(My.Application.Info.DirectoryPath & "\Leptir1.bmp")
```

Programski se može podešavati način prikaza slike u okviru za slike preko promjene osobine **SizeMode**. Sintaksa koda kada se slika prilagođava veličini okvira za sliku pod nazivom **Slika2**, je sljedeća:

```
Slika2.SizeMode = PictureBoxSizeMode.StretchImage
```

Programski se može podešavati dimenzija (širina i visina) okvira za slike u pikselima preko promjene osobine **ClientSize**. Sintaksa koda kada se okviru za sliku pod nazivom **Slika2** širina okvira podešava na 300 piksela, a visina na 200 piksela, je sljedeća:

```
Slika2.ClientSize = New Size(300, 200)
```

Nad ovim objektima se najčešće koriste događaji ***Click*** i ***DblClick,*** a to je jednostruki i dvostruki klik miša na ovaj objekat.

7.14. OBJEKAT *DATAGRIDVIEW*

Objekat ***DataGridView*** se može koristiti kao tabela, u koju se mogu upisivati podaci i iz koje se mogu čitati podaci. Ovaj objekat se može koristiti i za direktno povezivanje sa tabelama postojećih baza podataka. Tabelarno prikazivanje podataka predstavlja najpregledniji način prikaza podataka u tekstualnim editorima, pa samim tim i na ekranu računara. Tabelarni prikaz raznih podataka posebno dobija na značaju, ako se zna da je danas veoma rasprostranjena upotreba relacionih baza podataka, koje su organizovane u obliku dvodimenzionalnih tabela. Upotreba struktura *For... Next* i *Do... Loop,* u radu sa velikim dvodimenzionalnim nizovima, može značajno da uštedi vrijeme upisa i čitanja podataka iz tabela. Upotrebom programskih petlji prilikom rada sa tabelama, značajno se skraćuje programski kod.

Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***BackgroundColor*** za izbor boje pozadine tabele.
- ***BorderStyle*** za izbor stila okvira tabele.
- ***ColumnHeadersHeight*** za izbor visine kolone u zaglavlju u pikselima.
- ***ColumnHeadersHeightSizeMode*** najbolje postaviti na ***AutoSize*** da bi se veličina automatski prilagođavala unesenom tekstu.
- ***DataSource*** za izbor tabele iz baze podataka sa kojom se povezuje objekat.
- ***Enable*** da objekat bude dostupan za rad ako je izabrano ***True***.
- ***GridColor*** za izbor boje linija u tabeli.
- ***ReadOnly*** podesiti na ***False*** ako želimo da u tabelu možemo da upisujemo podatke, a ***True*** ako je tabela samo za gledanje podataka.
- ***ScrollBars*** je najbolje podesiti na ***Both***, da bi se pojavljivala oba klizača, kada tabela ima više podatake nego što može da stane u predviđeni prostor.

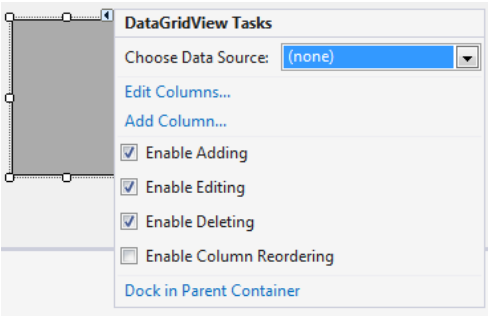
Sve osobine se mogu podešavati preko prozora *Properties*, ali i preko koda.

Nad ovim objektom se najčešće koriste sljedeći događaji:

- ***CellClick*** klik miša bilo gdje na ćeliju tabele.
- ***CellDoubleClick*** duli klik miša bilo gdje na ćeliju tabele.

- ***CellContentClick*** klik miša na sadržaj u ćeliji tabele.
- ***CellContentDoubleClick*** dupli klik miša na sadržaj u ćeliji tabele.
- ***CellMouseMove*** prelazak miša preko ćelija tabele.
- ***KeyPress*** pritisak na tastaturu kada je objekat u fokusu.

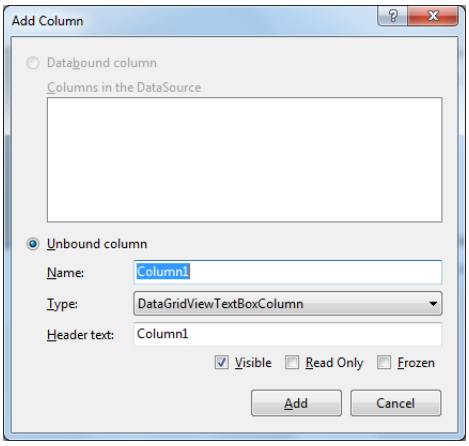
Kada se objekat postavi na formu pojavljuje se prozor kao na slici 7.15. i može se vidjeti da ovaj objekat uopšte ne liči na tabelu. Da bi dobili tabelu kao na slici 7.16. potrebno je prvo u ovaj objekat dodati kolone preko opcije ***Add Column...***, nakon čega se otvara prozor kao na slici 7.17. U ovom prozoru se daju imena kolonama, a kolona se dodaje klikom na dugme **Add**. Objekat *DataGridView* može biti tabela sa proizvoljnim brojem kolona i redova. Jednom napravljene kolone se mogu mijenjati ili brisati preko menija ***Edit Columns,*** koji je prikazan na slici 7.18.



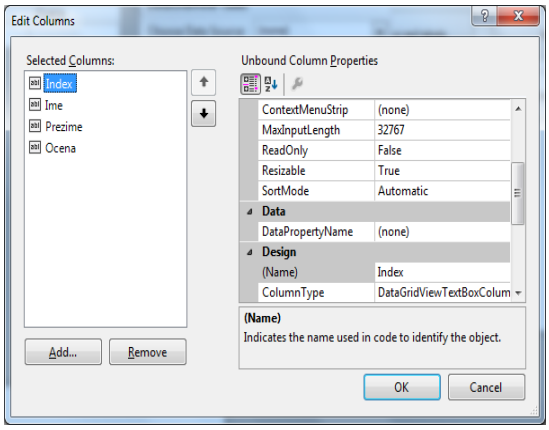
Slika 7.15. Objekat DataGridView

	Index	Ime	Prezime	Ocena
▶	045/15	Ilija	Rosić	6
	101/15	Mirko	Damjanović	8
	185/15	Stefan	Čović	7
	002/15	Vuka	Škrba	10
	001/15	Goša	Papaz	10
	103/15	Risto	Toplik	9
*				

Slika 7.16. Tabela sa podacima



Slika 7.17. Dodavanje kolone



Slika 7.18. Izmjena kolona

Nakon formiranja i imenovanja kolona potrebno je prvo formirati redove tabel. To se obično radi kroz proceduru Form_Load, a sljedi primjer koda za upis fiksnih tekstualnih vrijednosti u 4 kolone i 6 redova.

```
Private Sub Form1_Load(sender As Object, e As
EventArgs) Handles Me.Load
DataGridView1.Columns(0).AutoSizeMode=
DataGridViewAutoSizeColumnMode.DisplayedCells
DataGridView1.Columns(3).AutoSizeMode=
DataGridViewAutoSizeColumnMode.DisplayedCells
DataGridView1.Columns(3).DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight
Rem upis fiksnih podataka u 4 kolone i 6 redova
        DataGridView1.Rows.Add(New           String()
{"045/15", "Ilija", "Rosić", "6"})
        DataGridView1.Rows.Add(New           String()
{"101/15", "Mirko", "Damjanović", "8"})
        DataGridView1.Rows.Add(New           String()
{"185/15", "Stefan", "Čović", "7"})
        DataGridView1.Rows.Add(New           String()
{"002/15", "Vuka", "Škrba", "10"})
        DataGridView1.Rows.Add(New           String()
{"001/15", "Goša", "Papaz", "10"})
        DataGridView1.Rows.Add(New           String()
{"103/15", "Risto", "Toplik", "9"})
        Me.SetFormatting()
End Sub
```

Dodavanje novog reda u tabelu bez unosa podataka (prazan red) ima sintaksu

imeobjekta.Rows.Add(New String() {})

Ako se želi dodati novi red u tabelu i pri tome se unose vrijednosti u taj novoformirani red koristi se sljedeća sintaksa

imeobjekta.Rows.Add(New String() {vrednost1, ..., ...})

Sljedi primjer koda, koji se pokreće klikom na dugme **ButNoviRed**, a koji pravi jedan prazan red i 3 reda sa fiksnim vrijednostima pomoću **For** petlje.

```
Private Sub ButNoviRed_Click(sender As Object, e As
EventArgs) Handles ButNoviRed.Click
    Dim StIndex, StIme, StPrezime, StOcena As String
    Dim ii As String
              REM dodam jedan prazan red
    DataGridView1.Rows.Add(New String() {})
```

```
For i = 1 To 3
  ii = i
  StIndex = ii + "/02"
  StIme = ii + "Mira"
  StPrezime = ii + "Rosić"
  StOcena = i + 5
        REM dodavanje 3 nova reda preko For petlje
  DataGridView1.Rows.Add(New String() {StIndex, StIme,
  StPrezime, StOcena})
Next i
End Sub
```

Klikom miša na ćeliju u tabeli može se preuzimati vrijednost selektovane ćelije ili vrijednosti svih ćelija u tom redu. Da bi ovo bilo moguće potrebno je deklarisati promjenljivu tipa ćelije (Dim Celija1 As DataGridViewCell) i promjenljivu tipa red (Dim Red1 As DataGridViewRow). Preuzimanje vrijednosti selektovane ćelije omogučuje sistemska promjenljiva (e As DataGridViewCellEventArgs), koja pamti podatke o selektovanoj ćeliji (kolona (e.ColumnIndex) i red (e.RowIndex)). Sljedi primjer koda, koji se pokreće klikom na ćeliju tabele pod imenom DataGridView1, a koji preuzima vrijednost ćelije i reda i upisuje ih u odgovarajuće TextBox-ove.

```
Private Sub DataGridView1_CellClick(sender As Object, e As
DataGridViewCellEventArgs) Handles DataGridView1.CellClick
If e.RowIndex >= 0 Then
   Dim Red1 As DataGridViewRow
   Dim Celija1 As DataGridViewCell
   Dim Kol2, Red2 As Integer
   Red1 = Me.DataGridView1.Rows(e.RowIndex)
   Red2 = e.RowIndex
   Kol2 = e.ColumnIndex
   Celija1 = Me.DataGridView1(Kol2, Red2)
   CelijaTb.Text = Celija1.Value
   IndexTb.Text= Red1.Cells("Index").Value.ToString
   ImeTb.Text= Red1.Cells("Ime").Value.ToString
   PrezimeTb.Text= Red1.Cells("Prezime").Value.ToString
   OcenaTb.Text= Red1.Cells("Ocena").Value.ToString
End If
End Sub
```

U ćelije nekog reda mogu se upisivati vrijednosti tako što se red direktno poziva preko rednog broja reda (treba voditi računa da postoji nulti red i nulta kolona u tabeli). Slijedi primjer koda, koji se pokreće klikom na komandno dugme pod

imenom ButUpisi, a koji upisuje vrijednosti u ćelije jednog reda (koji se zadaje preko broja reda).

```
Private Sub ButUpisi_Click(sender As Object, e As
EventArgs) Handles ButUpisi.Click
   Dim RedUpisi As DataGridViewRow
    RedUpisi = Me.DataGridView1.Rows(3)   REM treci red
    RedUpisi.Cells(0).Value = "13/04"  REM nulta kolona
    RedUpisi.Cells(1).Value = "Risto"
    RedUpisi.Cells(2).Value = "Skrba"
    RedUpisi.Cells(3).Value = "9"
  End Sub
```

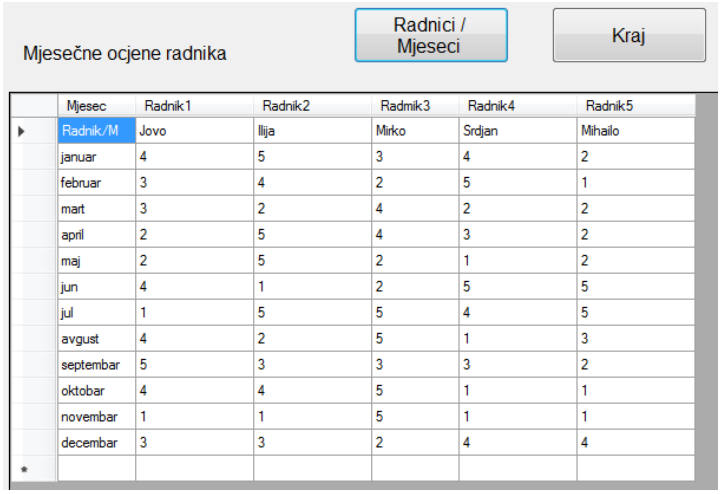
Svaka ćelija tabele se može direktno pozicionirati preko koda. Prilikom pozicioniranja čelije u tabeli prvo se navodi redni broj kolone, a zatim redni broj reda. Slijedi primjer koda, koji se pokreće klikom na komandno dugme pod imenom ButCelija, a koji upisuje vrijednosti u ćeliju i koji čita vrijednost iz ćelije.

```
Private Sub ButCelija_Click(sender As Object, e As
EventArgs) Handles ButCelija.Click
   Dim Celija1 As DataGridViewCell
      REM uzimanje podatka iz celije 3 kolona 2 red
   Celija1 = Me.DataGridView1(3,2)
   OcenaTb.Text = Celija1.Value
      REM upis podataka u celiju 1 kolona 3 red
   Celija1 = Me.DataGridView1(1, 3)
   Celija1.Value = "Ilija"
End Sub
```

Programski kod za popunjavanje ćelija tabele (DataGridView1) preko dvostruke *For* petlje i funkcije *Rnd* (generisanje slučajnih brojeva), u primjeru prikaza mjesečnih ocjena za 5 radnika, imao bi sljedeću formu. Rezultat ovog koda je prikazan na slici 7.19.

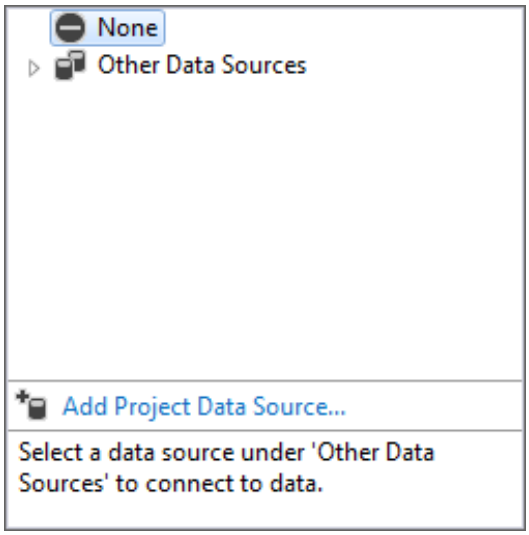
```
Dim Celija1 As DataGridViewCell
Dim i, j As Integer
For i = 1 To 13
    DataGridView1.Rows.Add(New String() {})
Next
Celija1 = Me.DataGridView1(0, 0)
Celija1.Value = "Radnik/M"
Celija1 = Me.DataGridView1(1, 0)
```

```
Celija1.Value = "Jovo"
Celija1 = Me.DataGridView1(2, 0)
Celija1.Value = "Ilija"
Celija1 = Me.DataGridView1(3, 0)
Celija1.Value = "Mirko"
Celija1 = Me.DataGridView1(4, 0)
Celija1.Value = "Srdjan"
Celija1 = Me.DataGridView1(5, 0)
Celija1.Value = "Mihailo"
Celija1 = Me.DataGridView1(0, 1)
Celija1.Value = "januar"
Celija1 = Me.DataGridView1(0, 2)
Celija1.Value = "februar"
Celija1 = Me.DataGridView1(0, 3)
Celija1.Value = "mart"
Celija1 = Me.DataGridView1(0, 4)
Celija1.Value = "april"
Celija1 = Me.DataGridView1(0, 5)
Celija1.Value = "maj"
Celija1 = Me.DataGridView1(0, 6)
Celija1.Value = "jun"
Celija1 = Me.DataGridView1(0, 7)
Celija1.Value = "jul"
Celija1 = Me.DataGridView1(0, 8)
Celija1.Value = "avgust"
Celija1 = Me.DataGridView1(0, 9)
Celija1.Value = "septembar"
Celija1 = Me.DataGridView1(0, 10)
Celija1.Value = "oktobar"
Celija1 = Me.DataGridView1(0, 11)
Celija1.Value = "novembar"
Celija1 = Me.DataGridView1(0, 12)
Celija1.Value = "decembar"
For i = 1 To 5
    For j = 1 To 12
        Celija1 = Me.DataGridView1(i, j)
        Celija1.Value = Int(Rnd() * 5 + 1)
    Next j
Next i
```

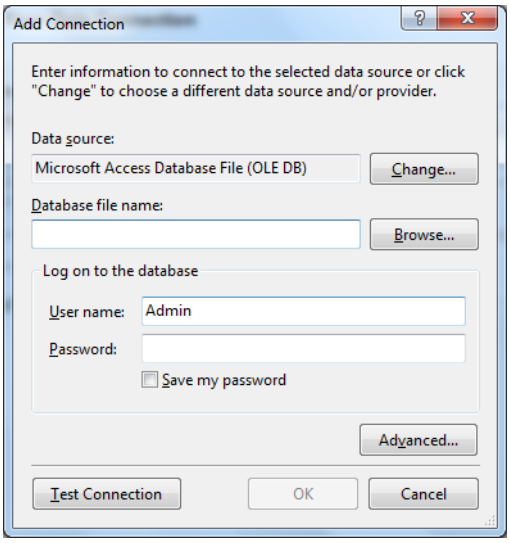


Mjesec	Radnik1	Radnik2	Radnik3	Radnik4	Radnik5
Radnik/M	Jovo	Ilija	Mirko	Srdjan	Mihailo
januar	4	5	3	4	2
februar	3	4	2	5	1
mart	3	2	4	2	2
april	2	5	4	3	2
maj	2	5	2	1	2
jun	4	1	2	5	5
jul	1	5	5	4	5
avgust	4	2	5	1	3
septembar	5	3	3	3	2
oktobar	4	4	5	1	1
novembar	1	1	5	1	1
decembar	3	3	2	4	4

Slika 7.19. Program za prikaz mjesečnih ocjena 5 radnika



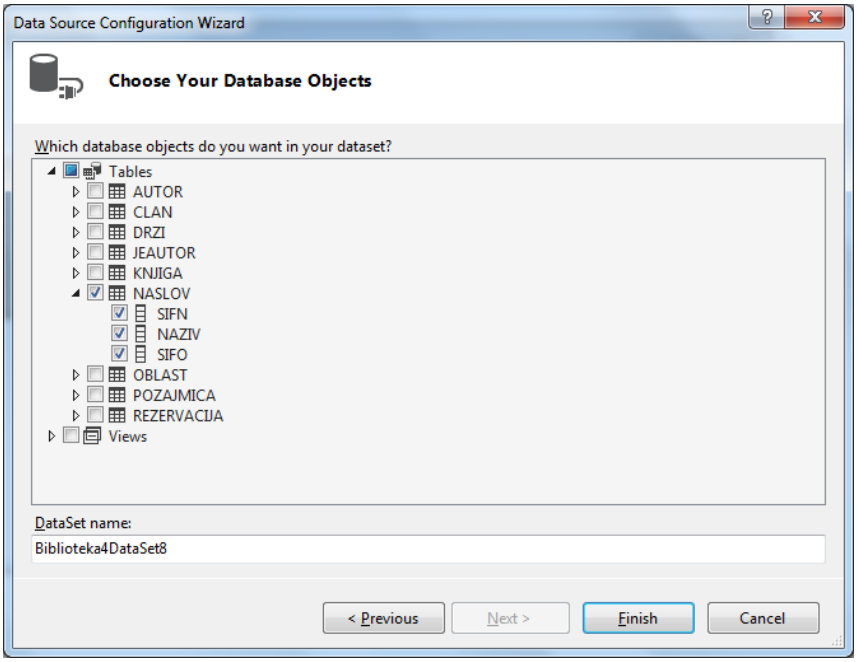
Slika 7.20. Dodavanje baze u projekat



Slika 7.21. Izbor baze

Rad sa bazama podataka veoma je značajan za svaki programski jezik, pa i za *Visual Basic* 2013. Pomoću baza podataka može se čuvati velika količina podataka, koji su različitog tipa i koji su organizovani u tabele. *DataGridView* objekat se može koristiti za direktnu komunikaciju sa raznim tipovima baza podataka. Kada se objekat postavi na formu pojavljuje se prozor kao na slici 7.15. Može se vidjeti da ovaj objekat uopšte ne liči na standardnu tabelu sa kolonama i redovima. Da bi

povezali ovaj objekat sa tabelom u postojećoj bazi podataka, potrebno je na prozoru prikazanom na slici 7.15. izabrati parametar ***Chose Data Source,*** nakon čega se pojavljuje prozor kao na slici 7.20. Na ovom prozoru je potrebno kliknuti na link ***Add Project Data Source***. Nakon toga se otvara prozor pod nazivom ***Data Source Configuration Wizard***, na kome je potrebno markirati ikonicu ***Database***, a zatim kliknuti na dugme Next. Nakon toga se otvara novi prozor pod nazivom ***Data Source Configuration Wizard***, na kome je potrebno kliknuti na dugme ***New Conection...***, nakon čega se otvara prozor kao na slici 7.21.



Slika 7.22. Izbor tabele i kolona u tabeli

Na ovom prozoru se bira fajl i putanja do fajla baze podataka preko dugmeta ***Browse...***, a poželjno je da ta baza bude na istom direktorijumu na kome se nalazi i projekat. Kada se izabere željena baza podataka potrebno je da se provjeri komunikacija programa sa bazom klikom na dugme ***Test Connection***. Ako je uspješno uspostavljena komunikacija sa bazom podataka pojaviće se prozor sa porukom: ”***Test connection succeeded***”. Kliokom na dugme ***OK*** otvara se prozor kao na slici 7.22. pod nazivom ***Data Source Configuration Wizard***, na kome se selektuje tabela i kolone tabele, koje se žele prikazati u objektu *DataGridView*, kada se forma pokrene. Ako se na ovom prozoru selektuju dvije ili više tabela iz baze podataka, tada se u objektu neće prikazati nijedna tabela sa podacima. Klikom

na dugme ***Finish*** završava se proces povezivanja objekta *DataGridView* sa bazom podataka i ako je proces uspješno završen u zaglavlju objekta *DataGridView* se prikažu nazivi kolona tabele, sa kojom se objekat povezuje. Sadržaj baze podata će biti prenesen u objekat *DataGridView,* kada se program pokrene.

7.15. OBJEKAT *PROGRESSBAR*

Objekat *ProgressBar* predstavlja horizontalni BarGraf, koji se programski popunjava. Popunjavanje BarGrafa se može vezati za neki tajmer ili za neku drugu vremensku funkciju u programu, ali najčešće se koristi prilikom kopiranja nekih fajlova. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Maximum*** za određivanje maksimalne vrijednosti, koju može imati ovaj objekat.
- ***Minimum*** za određivanje minimalne vrijednosti, koju može imati ovaj objekat.
- ***Step*** za određivanje koraka sa kojim se mijenja vrijednost grafa.
- ***Value*** trenutna vrijednost koju zauzima BarGraf.

Povećanje vrijednosti BarGrafa za vrijednost osobine *Step* se može vršiti sa metodom **.PerformStep().**

Sljedi primjer koda u kome se BarGraf puni do vrijednosti 60 aktiviranjem tajmera pod imenom TimerBar svake sekunde, nakon čega se tajmer isključuje.

```
Private Sub TimerBar_Tick(sender As Object, e As
EventArgs) Handles TimerBar.Tick
  Dim BarV As Integer
  ProgressBar1.Minimum = 0
  ProgressBar1.Maximum = 60
  ProgressBar1.Step = 1
  ProgressBar1.PerformStep()
  BarV = ProgressBar1.Value
  Rem prikaz vrijednosti u TextBox-u
  TexBar.Text=Str(BarV)
  If BarV = 60 Then
     TimerBar.Enabled = False
  End If
End Sub
```

7.16. OBJEKATI *COMMONDIALOG* ZA STANDARDNI MENI

Visual Basic 2013 pruža standardan skup dijaloških okvira za operacije, kao što su otvaranje i spašavanje datoteka, određivanje opcija za štampu, te izbor boja i veličine i oblika slova. Ovaj objekat takođe ima sposobnost prikaza pomoći, pokretanjem mehanizma za *Windows* pomoć (*Help*).

Kako bi kreirali dijaloški okvir, potrebno je da ga izaberete iz palete *Toolbox* i dodate na formu. Tokom izrade programa, objekati okvira za dijalog se prikazuje kao ikonica u posebnom prostoru ispod forme, a ne na samoj formi.

U tabeli 7.3. prikazani su dijaloški okviri, koji postoje kao objekti u programu *Visual Basic* 2013.

Tabela 7.3. Naredbe u standardnom okviru za dijalog

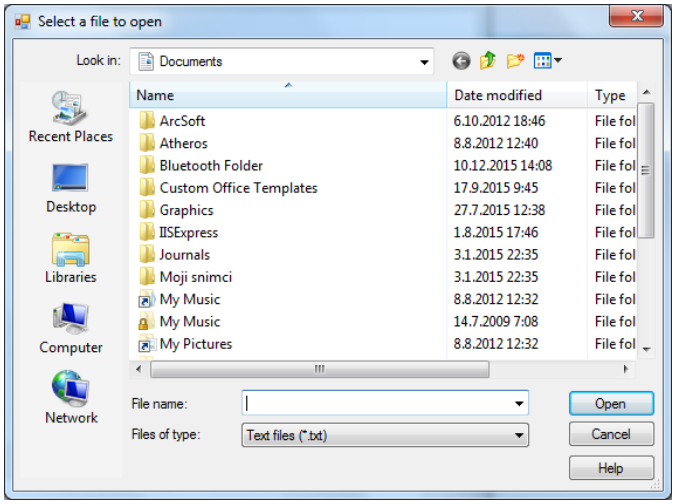
Red. br.	Dijaloški okvir	Naziv objekta dijaloškog okvira
1.	Open	OpenFileDialog
2.	Save As	SaveFileDialog
3.	Color	ColorDialog
4.	Font	FontDialog
5.	Explorer	FolderBrowserDialog
6.	Print	PrintDialog
7.	PrintPreview	PrintPreviewDialog
8.	PageSetup	PageSetupDialog
9.	Windows Help	HelpProvider

7.16.1. Dijaloški okviri *OpenFileDialog* i *SaveFileDialog*

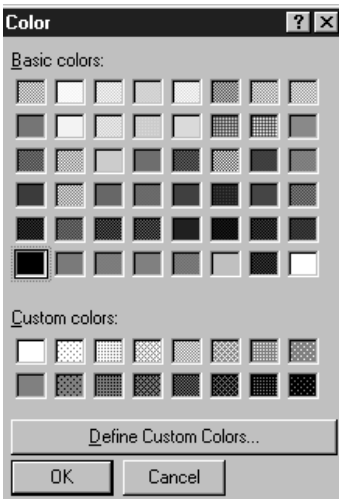
Dijaloški okvir *OpenFileDialog* prikazan je na slici 7.23. Omogućuje korisniku određivanje direktorijuma, nastavka imena datoteke (ekstenzije) i imena datoteke. Dijaloški okvir *SaveFileDialog* izgleda isto kao i dijaloški okvir *Open*, osim naslova dijaloga i imena datoteka koje su ispisana zasjenjeno. Tokom rada aplikacije, kad korisnik odabere datoteku i zatvori dijaloški okvir, osobina *FileName* se koristi za čuvanje imena odabrane datoteke.

Sljedi primjer koda za preuzimanje imena fajla u dijaloškom prozoru *OpenFileDialog*, pod imenom OpenD1. Izabrano ime fajla, koji seželi otvoriti, se prikazuje u *TextBox*-u pod imenom TxtFile1. Ovako izabrano ime fajla se može koristiti za otvaranje bilo kog dokumenta iz bilo kog programa, koji se može pokrenuti iz programa *Visual Basic* 2013.

```
If OpenD1.ShowDialog=Windows.Forms.DialogResult.OK Then
   Try
   TxtFile1.Text=My.Computer.FileSystem.ReadAllText(.FileName)
   Catch fileException As Exception
   Throw fileException
   End Try
End If
```



Slika 7.23. Dijaloški okvir Open



Slika 7.24. Okvir Color

Sljedi primjer koda programa za preuzimanje imena fajla u dijaloškom prozoru *SaveFileDialog*, pod imenom SaveD1. Sadržaj teksta koji se želi iskoristiti kao ime pod kojim se želi spasiti neki fajl, preuzima se iz *TextBox*-a pod imenom TxtFile1.

```
If SaveD1.ShowDialog()=Windows.Forms.DialogResult.OK Then
   My.Computer.FileSystem.WriteAllText(SaveD1.FileName,
   TxtFile1.Text, False)
End If
```

Za dijaloški okvir *OpenFileDialog* može se postaviti filter nad tipom fajla (***Files of type***) za prikaz fajlova samo određenog tipa. To se može napraviti određivanjem osobine ***Filter*** koristeći sljedeći oblik:

```
opis1 |filter1 |opis2 |filter2 ...
```

Opis1 je string, koji se ispisuje u okviru sa popisom (ComboBox) – na primjer "Tekstualne datoteke (*.doc)". *Filter* je stvarni filter – na primjer "*.doc". Svaki skup ***opis* I *filter*** mora biti razdvojen simbolom vertikalne linije (|). Primjer koda:

```
OpenD1.Filter = "Text files (*.txt)|*.txt|All files|*.*"
```

7.16.2. Dijaloški okvir *ColorDialog*

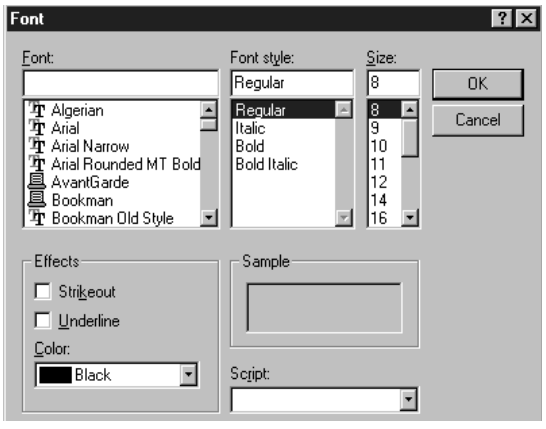
Dijaloški okvir *ColorDialog* prikazan je na slici 7.24. Omogućuje korisniku izbor boje iz palete ili stvaranje i izbor korisničke boje. Tokom rada programa, kad korisnik odabere boju i zatvori dijaloški okvir, treba upotrijebiti osobinu *Color* za dobijanje odabrane boje. Sljedeći kod prikazuje korištenje dijaloškog okvira *ColorDialog* pod imenom Boja1, za promjenu boje slova teksta u *TextBox*-u pod imenom TxtFile1.

```
        Rem Preuzimanje inicijalne boje u dijaloški okvir
  Boja1.Color = TxtFile1.ForeColor
  Boja1.CustomColors = CustomColors
  If Boja1.ShowDialog()=Windows.Forms.DialogResult.OK Then
     TxtFile1.ForeColor = Boja1.Color
     CustomColors = Boja1.CustomColors
  End If
  Boja1.Reset()
```

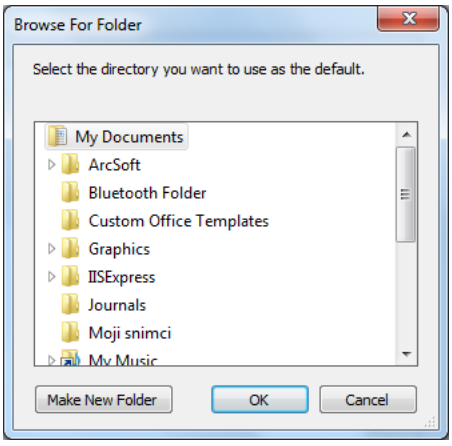
7.16.3. Dijaloški okvir *FontDialog*

Dijaloški okvir ***FontDialog*** prikazan je na slici 7.25. Omogućuje korisniku podešavanje teksta kao u *Word*-u, određivanjem njegove veličine, boje i stila. Jednom kada korisnik napravi izbor u dijaloškom okviru *Font*, sljedeće izabrane osobine će biti zapamćene: *FontName*, *Font style*, *Size* .... Sljedeći kod prikazuje korištenje dijaloškog okvira *FontDialog* pod imenom Font1.

```
If Font1.ShowDialog = Windows.Forms.DialogResult.OK Then
  TxtFile1.Font = Font1.Font
End If
```



Slika 7.25. Dijaloški okvir Font



Slika 7.26. Okvir FolderBrowser

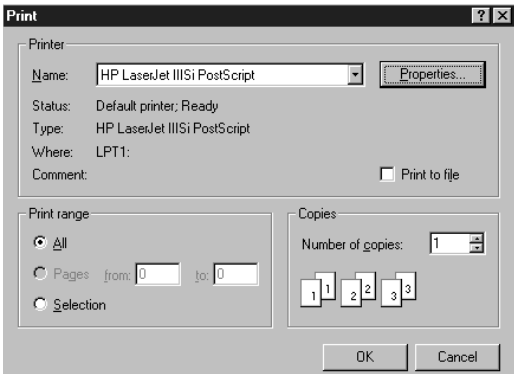
7.16.4. Dijaloški okvir *FolderBrowserDialog*

Dijaloški okvir *FolderBrowserDialog* prikazan je na slici 7.26. Omogućuje korisniku izbor direktorijuma, koji se želi koristiti dalje u programu. Sljedeći kod prikazuje korištenje dijaloškog okvira *FolderBrowserDialog* pod imenom Folder1, za prikaz putanje do selektovanog direktorijuma u *TextBox*-u.

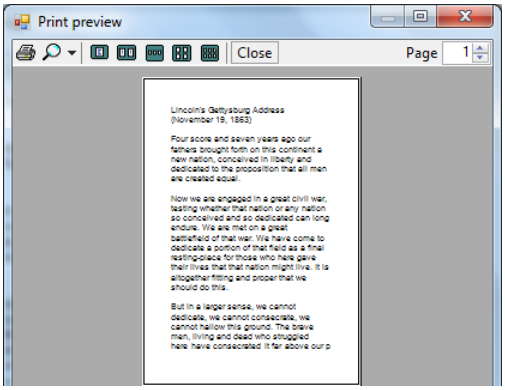
```
   Rem izbor početnog direktorijuma
Folder1.RootFolder = Environment.SpecialFolder.MyComputer
   Rem Kucanje teksta poruke u dijaloškom prozoru
Folder1.Description = "Select the directory you want to use
as the default."
   Rem da se aktivira dugme za dodavanje novog direktorijuma
Folder1.ShowNewFolderButton = True

If Folder1.ShowDialog = Windows.Forms.DialogResult.OK Then
    TxtDirectory.Text = Folder1.SelectedPath
End If
```

7.16.5. Dijaloški okvir *PrintDialog, PrintPreview i PageSetup*

Dijaloški okvir *PrintDialog* prikazan je na slici 7.27. Omogućuje korisniku određivanje izgleda izlaza na štampač. Korisnik može odrediti opseg strana koje će biti odštampane, kvalitet štampe, broj kopija i tako dalje. Ovaj dijaloški okvir takođe prikazuje informacije o trenutno instaliranom štampaču, omogućuje korisniku instaliranje ili reinstaliranje novog inicijalnog štampača.



Slika 7.27. Dijaloški okvir Print

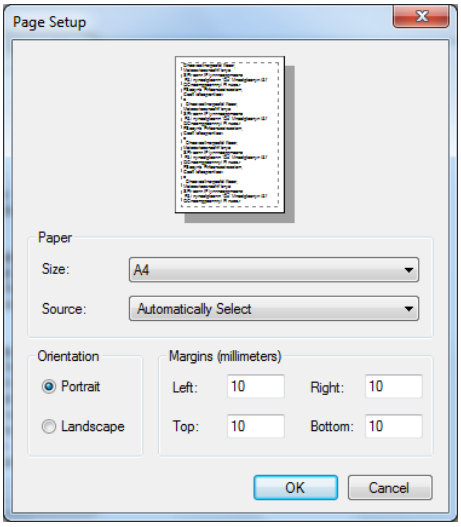


Slika 7.28. Dijaloški okvir Print

Dijaloški okvir *PrintPreview* prikazan je na slici 7.28. Omogućuje korisniku da vidi kako će izgledati dokument, koji se šalje na štampač. Korisnik može izabrati da vidi jednu ili više stranica koje se žele štampati.

Dijaloški okvir *PageSetup* prikazan je na slici 7.29. Omogućuje korisniku da podesi orjentaciju patira i margine teksta, koji se želi štampati.

Uz ova 3 dijaloška okvira ide zajedno i objekat *PrintDocument*, koji definiše objekat sa koga se šalje dokument na štampač, kada se štampa sa *Windows Forms application*.



Slika 7.29. Dijaloški okvir PageSetup

7.17. IZRADA SOPSTVENOG MENIJA *MENUSTRIP*

Pored standardnog *Visual Basic* 2013 omogućuje da se napravi sopstveni padajući meni na srpskom jeziku. Na taj način program, koji se pravi, će izgledati puno profesionalnije, preglednije i lakši za korišćenje. Vlastiti padajući meni se pravi preko objekta *MenuStrip.* Postavljanjem objekta *MenuStrip* na formu, na vrhu forme se pojavljuje jedno prazno polje (oblika *ComboBox Type Here*) za unos teksta vlastitog menija, kao što je prikazano na lijevoj strani slike 7.30.

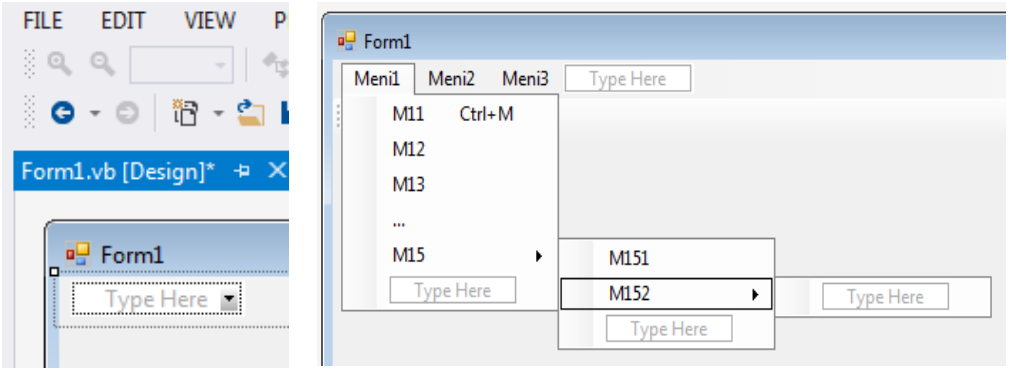
U ovo polje se unosi tekst, koji želimo da se pojavljuje kao padajući meni. Unošenjem jednog teksta menija, automtski se pojavljuje novo polje za novi meni (sa desne strane postojećeg unesenog menija) i novo polje za unos pod menija (sa donje strane postojećeg unesenog menija). Na desnoj strani slike 7.30. prikazan je

meni koji ima tri glavna menija (Meni1, Meni2 i Meni3), a glavni meni Meni1 ima 5 pod menija (M11 do M15), a pod meni M15 ima još svoja 2 pod menija (M151 i M152). *MenuStrip* omogućava dodavanje novih ili modifikaciju i brisanje postojećih menija. Svaki napravljeni meni i pod meni se ponaša kao zaseban objekat, koji ima svoje osobine, koje se odvojeno podešavaju. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Text*** za ispis teksta, koji se pojavljuje na objektu.
- ***Cheked*** potvrda da je komanda aktivirana i može biti *True* ili *False*, a upotrebljava se za komande čije funkcionisanje zavisi od znaka za potvrdu. Ako je u objekat unesen znak za potvrdu (√ znak čekiranja) komanda će biti uključena u meni, dok u protivnom nije aktivna.
- ***Enabled*** dostupnost komande u meniju i može biti *True* ili *False*, a trenutno nefunkcionalna komanda prepoznaje se po tome, što je njen zapis znatno bljeđi u odnosu na ostale komande.
- ***Visible*** da se komanda učini potpuno ne vidljiva u meniju i može biti *True* ili *False*.
- ***ShortcutKeys*** definisanje prečice za pristup komandi putem tastature. Iz razvijene liste ponuđenih direktnih pristupa, najčešće ***Ctrl*** + neko slovo, bira se željeni pristup klikom na odgovarajuću opciju.

Najbitnija osobina ovog objekta je *Name* (ime), jer se preko ovog imena pristupa tom objektu u kodu programa. Elementima menija treba dodjeljivati imena uz zadovoljavanje uslova:

- da su kratka i jasna,
- da asociraju na akciju koja se od njih očekuje i
- da su im, po mogućnosti, početna slova različita.

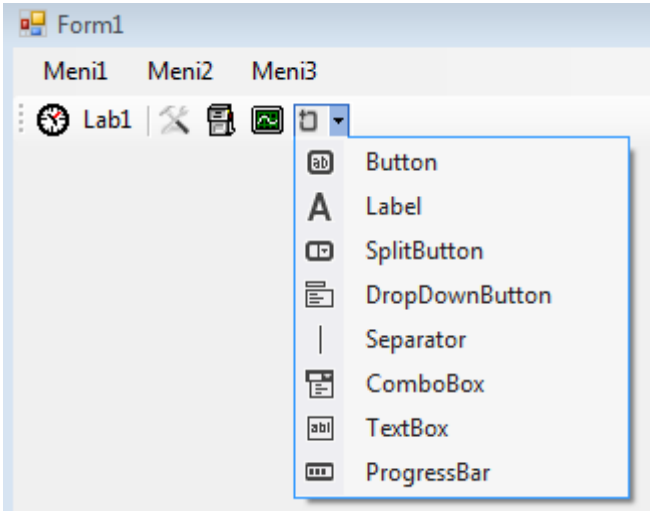


Slika 7.30. Izrada sopstvenog menija

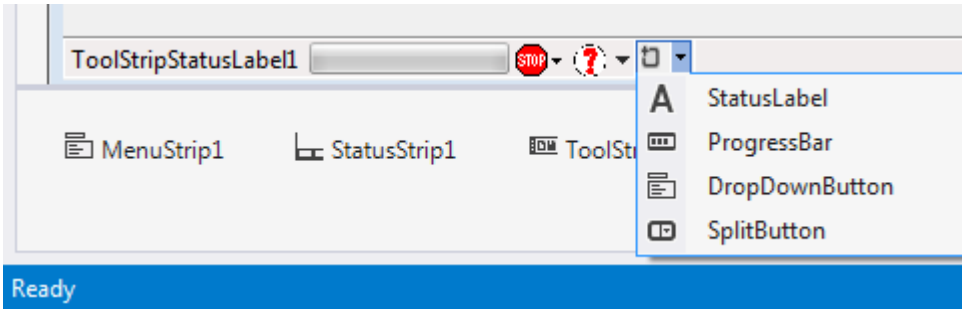
7.18. IZRADA SOPSTVENOG *TOOLBAR-A*

Kada se prave složeni programi često imamo potrebu da napravimo sopstveni *ToolBar* (na vrhu forme) pomoću objekta *ToolStrip* ili *StatusBar* (na dnu forme) pomoću objekta *StatusStrip*.

Postavljanjem objekta *ToolStrip* na formu, na vrhu forme ispod padajućeg menija se pojavljuje jedno prazno polje (oblika *ComboBox*) za unos ikonice vlastitog menija, kao što je prikazano na slici 7.31. Za ove ikonice se mogu unositi razni tipovi slika i objekata, kao što se vidi na slici. Unošenjem jedne ikonice u meni, automtski se pojavljuje novo polje za novu ikonicu (sa desne strane postojeće unesene ikonice).



Slika 7.31. Pravljenje ToolBar-a



Slika 7.32. Pravljenje StatusBar menija

Postavljanjem objekta *StatusStrip* na formu, na dnu forme se pojavljuje jedno prazno polje (oblika *ComboBox*) za unos ikonice vlastitog menija, kao što je prikazano na slici 7.32. Za ove ikonice se mogu unositi razni tipovi objekata kao što se vidi na slici. Unošenjem jedne ikonice u meni, automtski se pojavljuje novo polje za novu ikonicu (sa desne strane postojeće unesene ikonice).

Svaka napravljena ikonica se ponaša kao zaseban objekat, koji ima svoje osobine, koje se odvojeno podešavaju. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Text*** za ispis teksta, koji se pojavljuje na objektu.
- *ToolTip**Text*** za ispis teksta, koji se pojavljuje, kada se kursor miša dovede na objekat.
- ***Enabled*** dostupnost ikonice u meniju da se sa njom može izvršiti neka akcija u programu. Može imati vrijednost *True* ili *False*. Trenutno nefunkcionalna ikonica (***Enabled= False***) prepoznaje se po tome, što je njena boja znatno bljeđa u odnosu na ostale ikonice.
- ***Visible*** da se ikonica učini vidljivom ili potpuno ne vidljiva u meniju. Može imati vrijednost *True* ili *False*.

7.19. KRETANJE KROZ OBJEKTE SA TAB TASTEROM

Tabulatorni red (*tab order*) je redoslijed, po kojem se korisnik može kretati od jednog do drugog objekta na formi, korištenjem tipke **TAB**. Svaka forma ima svoj tabulatorni red. U pravilu, tabulatorni red odgovara redoslijedu kreiranja objekata. Na primjer, pretpostavimo da ste kreirali dva okvira sa tekstom, **Text1** i **Text2**, te zatim komandno dugme **Command1** kao na slici 7.42. Kad se program pokrene, objekat **Text1** će imati fokus. Pritisak na tipku TAB fokus se prebacuje na druge objekte, redoslijedom kojim su kreirani.

Slika 7.24. Tabulatorni reda

Tabela 7.4. Promjena TabIndexa

Objekat	TabIndex prije zamjene	TabIndex poslije zamjene
Text1	0	1
Text2	1	2
Command1	2	0

Za promjenu postojećeg tabulatornog reda objekta, podesite osobinu *TabIndex* (preko prozora *Properties*). Osobina *TabIndex* objekta određuje, gdje je postavljen u tabulatornom redu. U pravilu, prvi kreirani objekat ima vrijednost osobine *TabIndex* =0, drugi *TabIndex* =1 i tako dalje.

Kad promijenite redoslijed objekta u tabulatornom redu, *Visual Basic* 2013 automatski mijenja mjesta ostalih objekata u tabulatornom redu, tako da odražava ubacivanja i brisanja. Na primjer, ako objekat **Command1** postavite kao prvi u tabulatornom redu, osobina TabIndex ostalih objekata će automatski biti povećana za jedan, kao što je prikazano u tabeli 7.4.

Najveća vrijednost osobine *TabIndex* je uvijek za jedan manja od broja objekata u tabulatornom redu (jer brojanje počinje od 0). Čak i ako osobini *TabIndex* dodijelite vrijednost veću od broja objekata, *Visual Basic* 2013 tu vrijednost pretvara u broj za jedan manji od broja objekata.

Objekti koji ne mogu dobiti fokus, kao i objekti koji su onemogućeni ili nevidljivi, nemaju osobinu *TabIndex* i nisu uključeni u tabulatorni red. Kad korisnik pritisne tipku TAB, takvi objekti se preskaču.

Objekat možete isključiti iz tabulatornog reda, postavljanjem njegove osobine *TabStop* na *False* (0). Objekat čija je osobina *TabStop* postavljeno na *False* i dalje zadržava svoje mjesto u tabulatornom redu, iako će biti preskočena kruženjem po objektima tipkom TAB.

Grupa dugmadi izbora ima jedinstven izbor tipkom TAB. Potvrđeno dugme (ono čije je osobina *Value* postavljeno na *True*) automatski ima osobina *TabStop* postavljeno na *True*, a ostala dugmad u grupi imaju osobinu *TabStop* postavljenu na *False*.

7.20. KOPIRANJE FAJLOVA

Prilikom rada sa bazama podataka zbog sigurnosti bitno je povremeno praviti rezervnu kopiju baze. Rezervna kopija bilo kog fajla na računaru se kreira pomoću sljedeće sintakse:

My.Computer.FileSystem.CopyFile(Adresalzvora, AdresaCilja)

Primjer 1.

Kopiranje *Word* dokumenta pod imenom "Tea1.doc", koji se nalazi na lokaciji C: diska, na istu lokaciju, ali pod imenom "Kopija1.doc".

```
My.Computer.FileSystem.CopyFile_
("C:\Tea1.doc","C:\Kopija2.doc")
```

Primjer 2.

Kopiranje *Word* dokumenta pod imenom "Tea1.doc", koji se nalazi na lokaciji C: diska, na istu lokaciju, ali pod imenom "Kopija1.doc", pri čemu je dozvoljeno prebrisavanje dokumenta pod već postojećim imenom "Kopija1.doc".

```
My.Computer.FileSystem.CopyFile_
("C:\Tea1.doc","C:\Kopija2.doc",overwrite:=True)
```

Primjer 3.

Kopiranje *Word* dokumenta pod imenom " Tea1.doc", koji se nalazi na lokaciji C: diska, na drugi direktorijum sa putanjom "C:\Proba", ali pod istim imenom.

```
My.Computer.FileSystem.CopyFile_
("C:\Tea1.doc","C:\Proba\Tea1.doc",overwrite:=True)
```

Primjer 4.

Kopiranje *Word* dokumenta "Tea1.doc", koji se nalazi na lokaciji C: diska, na direktorijum na kome se nalazi tekući projekat *Visual Studio* 2013 sa kojim radimo i to na njegov pod direktorijum **\bin**, ali pod istim imenom kao i originalni fajl.

```
My.Computer.FileSystem.CopyFile("C:\Tea1.doc,_
My.Application.Info.DirectoryPath&"\Tea1.doc",_
overwrite:=True)
```

Primjer 5.

Kopiranje *Word* dokumenta "Tea1.doc", koji se nalazi na direktorijum na kome se nalazi tekući projekat *Visual Studio* 2013 sa kojim radimo i to na njegovom pod direktorijumu **\bin**, na isti direktorijum ali pod drugim imenom "\Tea5.doc".

```
My.Computer.FileSystem.CopyFile_
(My.Application.Info.DirectoryPath&"\Tea1.doc",_
My.Application.Info.DirectoryPath&"\Tea5.doc",_
overwrite:=True)
```

8. PRIMJERI PROGRAMA U *VISUAL BASIC*-U 2013

U ovom poglavlju prvo će biti predstavljeni primjeri programa napisanih u programskom jeziku *Visual Basic* 2013, za komunikaciju sa *Microsoft Excel* dokumentima. Zatim će biti predstavljeni načini povezivanja programa *Visual Basic* 2013 sa *Microsoft Access* bazom podataka, prvo preko gotovih objekata, a zatim i preko koda. Predstavljeno je i nekoliko primjera *SQL* upita, koji se mogu napisati direktno u programu *Visual Basic* 2013. Na kraju su dati primjeri zadataka za samostalan rad studenata, a većina ovih zadataka se pojavljuje i na praktičnom dijelu ispita.

8.1. PRISTUP *EXCEL* DOKUMENTU

Opisaćemo primjer pristupa *Microsoft Excel* dokumentu sa više stranica iz programa *Visual Basic* 2013. Prvo će biti opisano otvaranje postojećeg *Excel* dokumenta, upis podataka u fiksne ćelije na željenoj stranici *Excel* dokumenta, zatim čitanje podataka iz *Excel* dokumenta i upisivanje u promjenljivu u *Visual Studio*. Na kraju će biti opisan postupak spašavanje *Excel* dokumenta.

Da bi iz programa *Visual Studio* mogli pristupiti *Microsoft Excel* dokumentu, potrebno je nakon pokretanja svakog novog projekta **(*File*—> *New* —>*Project*—> *Templates*—> *Visual Basic*—> *Windows*—> *Windows Form Applications*)**, prvo dodati referencu (biblioteku) za vezu programa sa *Excelom.* U glavnom meniju *Visual Basic* 2013, potrebno je izabrati ***PROJECT*—> *Add Reference...* —> *COM***, a zatim čekirati opciju

Microsoft Excel 14.0 Object Library.

Tek onda je moguć rad sa *Excel* dokumentima iz programa *Visual Basic*. Poslije toga počinje pisanje koda u editoru forme pod nazivom **Form1Excel2**, tako što se prvo definišu objekti, promjenljive i konstante. Sljedi zatim dio koda programa za otvaranje već postojećeg *Excel* dokumenta pod nazivom **"ProbaExcel1.xls"**, koji ima dvije stranice sa nazivima **"Strana1"** i **"Srana2"**.

U prvoj kodnoj liniji programa treba otkucati kod, koji označava pristup bazi podataka.

```
Imports Excel = Microsoft.Office.Interop.Excel
```

Zatim se unutar klase forme, koju definiše sam projekat (u našem slučaju je to forma pod nazivom **Form1Excel2**), deklarišu globale (javne) promjenljive za vezu sa *Excel* programom, radnom knjigom i radnim listovima.

```
Public Class Form1Excel2
    Dim APP As New Excel.Application
    Dim workbook As Excel.Workbook
```

```
Dim worksheet1 As Excel.Worksheet
Dim worksheet2 As Excel.Worksheet
Dim strSaveFilename As String
Dim strFileName As String
```

Sljedi kod programa, koji predstavlja proceduru koja se pokreće nakon pokretanja forme, a pomoću koje se otvara već postojeći *Excel* dokument pod nazivom **"ProbaExcel1.xls"**. Promjenljiva **strFileName** označava punu putanju *Excel* dokumenta na disku računara. Ova putanja se može upisivati direktno (na primjer **strPutBaz = "D:\Mirko\ProbaExcel1.xls"**). Ali se može pojaviti problem kada se program prenosi na drugi računar, jer u tom slučaju putanje na kojima se nalazi program tada moraju biti identične na obadva računara. Umjesto direktnog načina definisanja putanje može se uraditi i apsolutno definisanje putanje. Kod apsolutnog definisanja putanje potrebno je da *Excel* dokument i program u kome se radi sa tim dokumentom budu na istom direktorijumu, odnosno *Excel* dokument treba postaviti na pod direktorijum **\bin\Debug** koji *Visual Basic* 2013 sam izgeneriše. Preuzimanje putanje na kojoj se nalazi program, u kome se trenutno radi, vrši se pomoću komande **My.Application.Info.DirectoryPath&**, a onda se iza nje pod dvostrukim navodnicima navede ime i ekstenzija dokumenta kome se pristupa. Apsolutni način definisanja putanje se prepuručuje pri programiranju u *Visual Basic* 2013.

```
Private Sub Form1Excel2_Load(sender As Object, e As_
EventArgs) Handles MyBase.Load
strSaveFilename = ""
APP.Visible = True
  Rem da se vidi excel na startnoj liniji
APP.DisplayFullScreen = True
  Rem da se vidi excel u pozadini forme koja je aktivna
strFileName=My.Application.Info.DirectoryPath&_
"\ProbaExcel1.xlsx"
  Rem Definisanje apsolutne putanje do dokumenta
workbook = APP.Workbooks.Open(strFileName)
worksheet1 = workbook.Worksheets("Strana1")
worksheet2 = workbook.Worksheets("Strana2")
End Sub
```

Zatvaranje predhodno otvorenog *Excel* program se u *Visual Basic* 2013 vrši pomoću sljedećeg koda.

```
Try
  APP.Quit()
Catch ex As Exception
End Try
```

Sljedi kod u kome je opisan postupak upisa vrijednosti iz *TextBox*-a pod nazivom **TexUpis** u stranicu pod imenom **"Strana1"** i fiksnog teksta **"Zaglavlje"** u stranicu pod imenom **"Strana2"** *Excel* dokumenta **"ProbaExcel1.xls"**. Prilikom navođenja ćelije (*Cells*) u koju se upisuje željena vrijednost prva brojna vrijednost je redni broj reda, a druga brojna vrijednost je redni broj kolone.

```
worksheet1.Cells(2,3).Value=TexUpis.Text
Rem 2 red 3 kolona
worksheet2.Cells(4, 5).Value = "Zaglavlje"
Rem 4 red 5 kolona
```

Sličan je kod u kome je opisan postupak upisa vrijednosti u *TextBox*-a pod nazivom **TexCitaj** i **TexUpis** iz stranice pod imenom **"Strana1"** i **"Strana2"** *Excel* dokumenta **"ProbaExcel1.xls"**.

```
TexCitaj.Text = worksheet1.Cells(1, 1).Value
TexUpis.Text = worksheet2.Cells(4, 5).Value
```

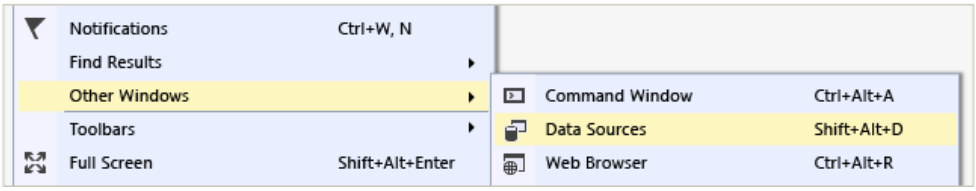
Sljedi kod, koji omogućuje da se otvoreni *Excel* dokument spasi pod drugum imenom, a da se to novo ime dokumenta unosi preko *TextBox*-a (**TexImeEx**). Nakon što se dokument spasi pod drugim imenom potrebno ga je i zatvoriti. Kod je napisan u proceduri, koja se aktivira klikom na dugme **ButSpasiExcel**.

```
Private Sub ButSpasiExcel_Click(sender As Object, e As_
EventArgs) Handles ButSpasiExcel.Click
Dim NovoIme As String
NovoIme = TexImeEx.Text
NovoIme = "\" + NovoIme + ".xlsx"
strSaveFilename=My.Application.Info.DirectoryPath& NovoIme
workbook.SaveAs(strSaveFilename,_
Excel.XlFileFormat.xlWorkbookDefault)
workbook.Close(False)
APP.Quit()
End Sub
```

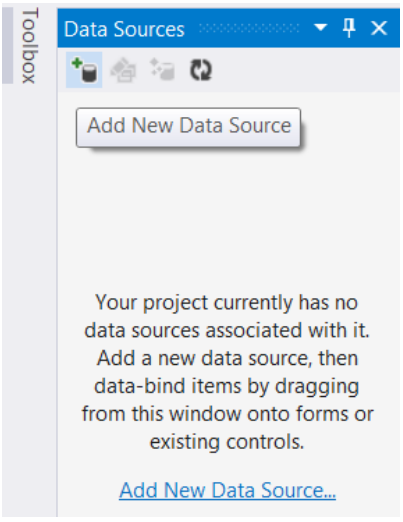
8.2. POVEZIVANJE SA *ACCESS* BAZOM PODATAKA

Visual Studio ima mogućnost povezivanja sa raznim vrstama baza podataka uključujući *Access*, *Microsoft SQL Server* i *Oracle*. U ovom poglavlju biće opisan *ADO.NET (ActiveX Data Objects)* pristup *Access* bazama podataka iz programa *Visual Studio*. Nakon pokretanja svakog novog projekta, pojavljuje se jedna prazna forma. Da bi izvršili povezivanje programa napisanog u *Visual Studiju* i baze podataka napravljene u *Access*-u (koja ima ekstenziju .mdp) potrebno je napraviti

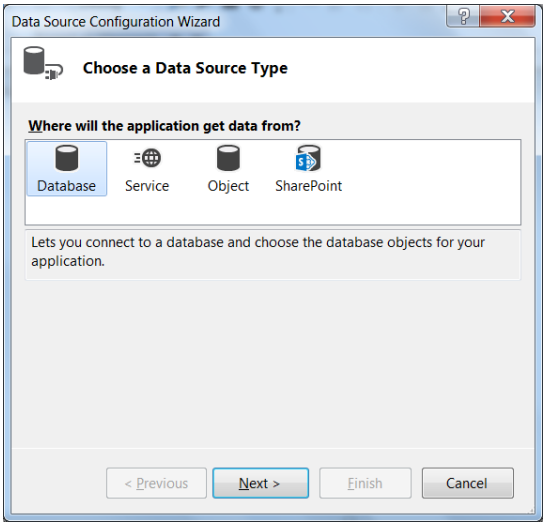
novu strukturu *DataSet*, koja treba da se pojavi u prozoru *Solution Explorer*. U jednom programu može postojati više različitih *DataSet* struktura, što znači da se jedan program može povezivati sa više različitih baza podataka. Poželjno je da baza podataka bude na istom direktorijumu kao i ostali elementi projekta, kako ne bi nastao problem pri presnimavanju programa sa jednog direktorijuma na drugi. Da bi napravili novu *DataSet* strukturu potrebno je u glavnom meniju *Visual Basic*, izabrati *View —> Other Windows —> Data Sources*, kao na slici 8.1. Nakon toga se pojavljuje prozor kao na slici 8.2. Dodavanje nove *Data Sorces* strukture vrši se klikom na ikonicu  pod nazivom *Add New Data Sorce*. Nakon toga se pojavljuje prozor kao na slici 8.3., na kome treba izabrati opciju *Database* i kliknuti na dugme *Next*. Pojavljuje se prozor kao na slici 8.4., na kome treba izabrati opciju *Dataset* i kliknuti na dugme *Next*. Pojavljuje se prozor *Add Connection* kao na slici 8.5. U polju *Data Source* se bira tip baze sa kojom se želi izvršiti povezivanje (u našem slučaju je to *Microsoft Access Database File*). U polje *Database file name* potrebno je upisati potpunu putanju do baze podataka, kao i ime baze sa njenom ekstenzijom. Ovo je lakše uraditi klikom na dugme *Brovse..*, nakon čega se pojavljuje standardni prozor za izbor fajla sa slike 8.6.



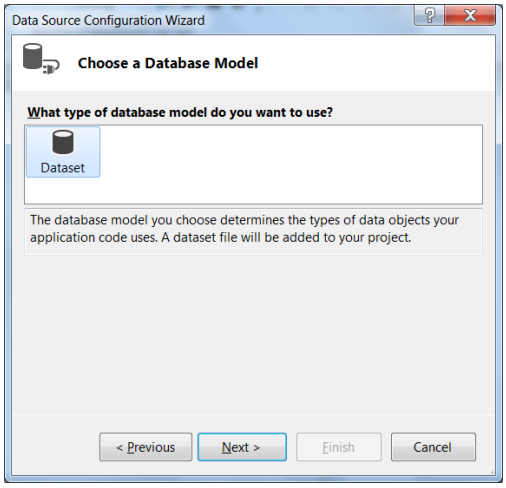
Slika 8.1. Dolazak do prozora Data Sources



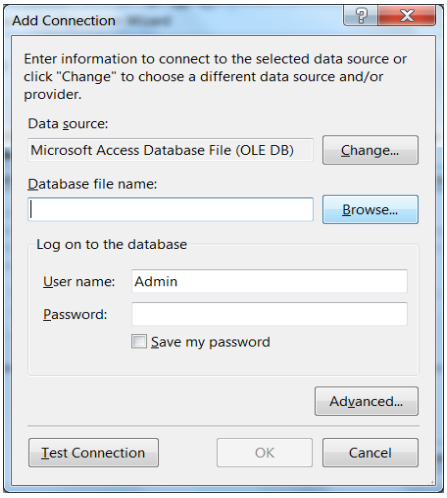
Slika 8.2. Prozor Data Sources



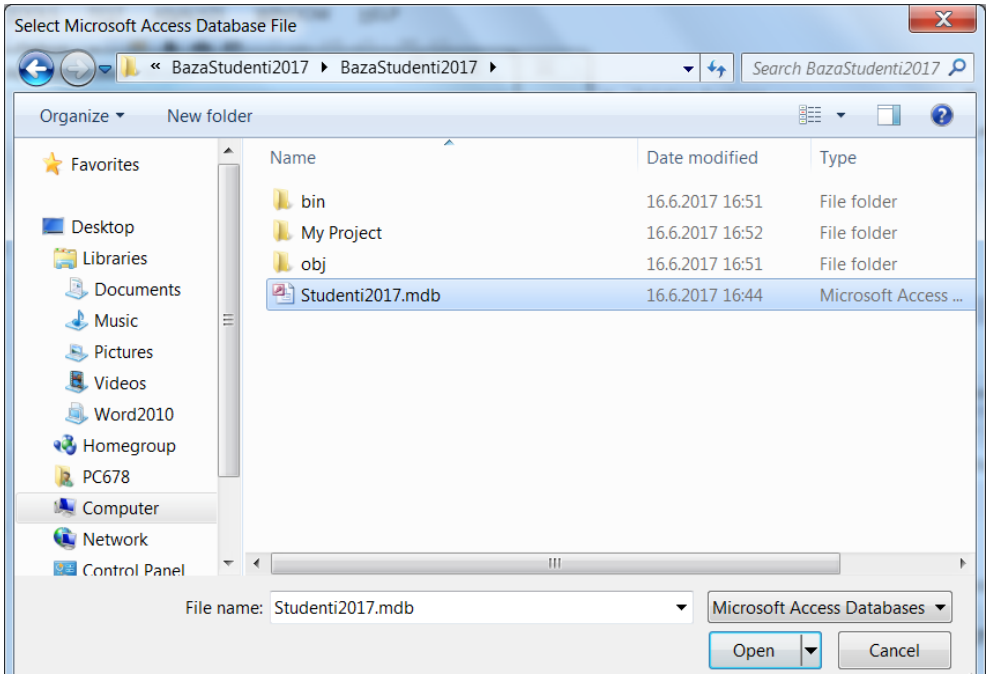
Slika 8.3. Viza sa bazom



Slika 8.4. Izbor Dataset



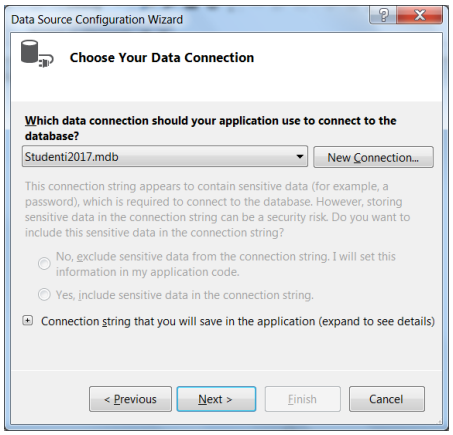
Slika 8.5. Prozor za izbor baze



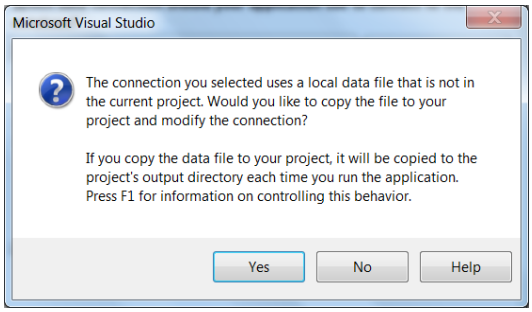
Slika 8.6. Prozor za izbor fajla

Na prozoru prikazanom na slici 8.6. se bira putanja do baze, kao i ime baze. Klikom na dugme *Open* u polju *Database file name* se pojavljuje putanja do baze, kao i ime baze. Klikom na dugme *Test Connection* može se izvršiti probno

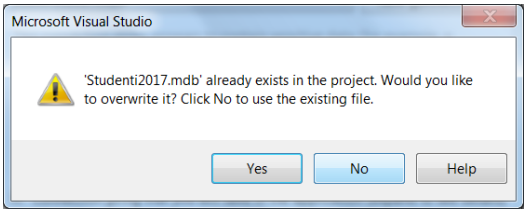
povezivanje sa bazom podataka. Ako je proba povezivanja uspjela, onda se pojavljuje prozor sa porukom *Test connection succeeded*. Zatim je potrebno na prozoru sa slike 8.5. kliknuti na dugme OK, nakon čega se pojavljuje prozor kao na slici 8.7. Tu se vidi ime baze i njena ekstenzija. Klikom na dugme *Next* pojavljuje se poruka sa slike 8.8. Ovde je potrebno kliknuti na dugme *YES* kako bi se izabrana baza iskopirala na pod direktorijum **bin\Debug** kompletnog projekta. Ako je nekada ranije već urađeno povezivanje sa izabranom bazom podataka, onda se pojavljuje prozor kao na slici 8.9. Klikom na dugme YES izvršiće se prekopiranje nove preko stare baze podataka. Klikom na dugme NO zadržaće se stari podaci u bazi podataka. Nakon toga se dodjeljuje ime konekcije kao na slici 8.10.



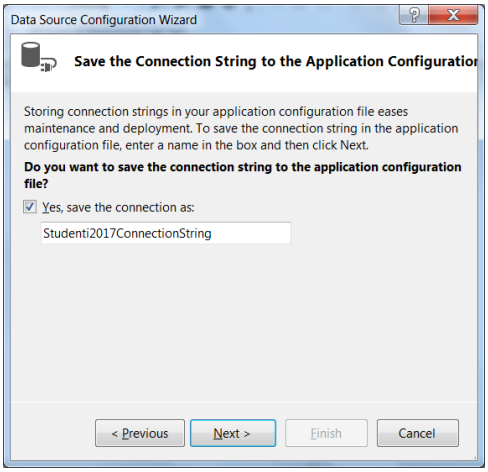
Slika 8.7. Data Source konfiguracija



Slika 8.8. Kopiranje u projekat

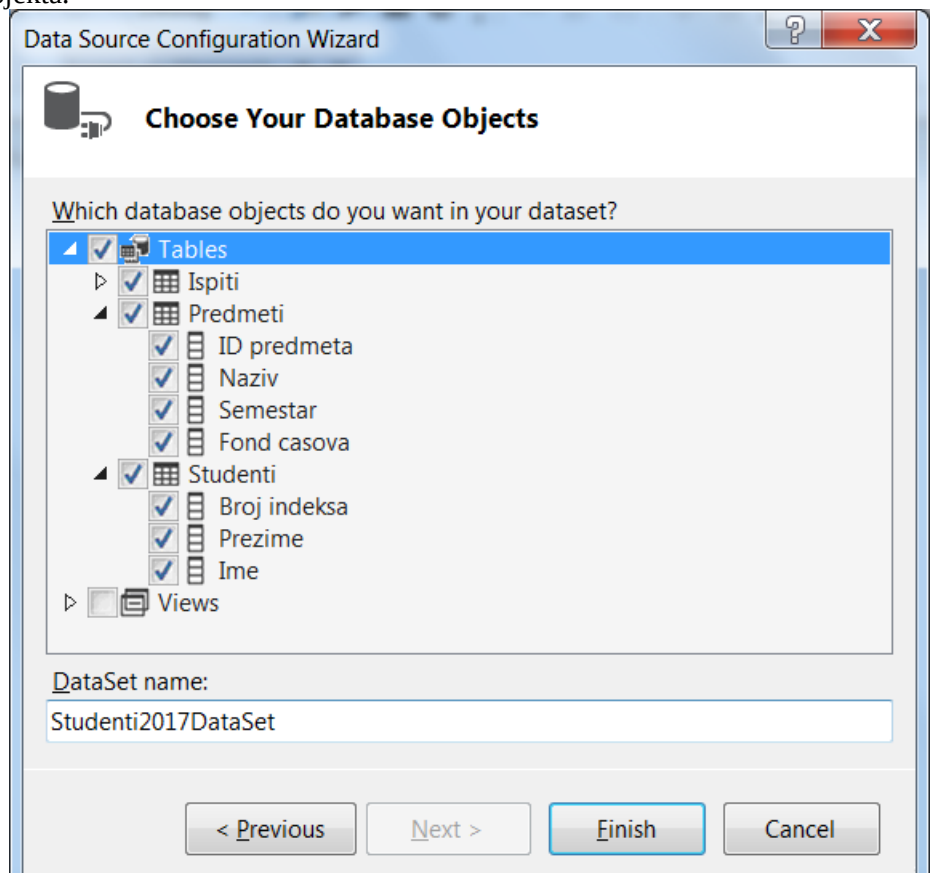


Slika 8.9. Data Source konfiguracija

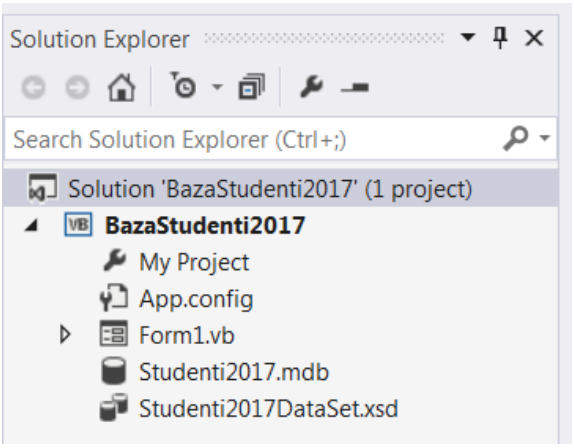


Slika 8.10. Kopiranje u projekat

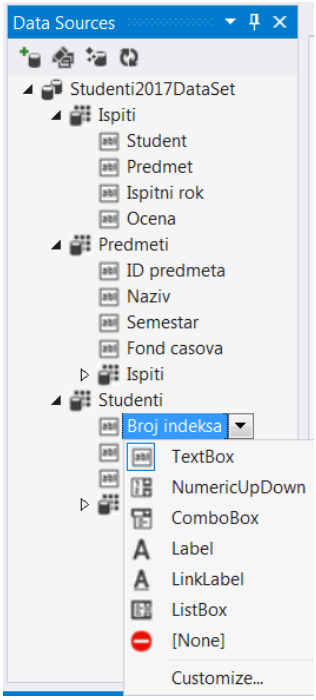
Klikom na dugme *Next* pojavljuje se poruka sa slike 8.11. U ovom prozoru se pojavljuju svi objekti, koji postoje u izabranoj bazi podataka. Jedna ispod druge se navode sve tabele i pogledi. Klikom na trougao ispred imena tabele otvaraju se polja sa imenima svih kolona u toj tabeli. Čekiranjem u kvadratiću ispred imena tabele, automatski se čekiraju sve kolone u toj tabeli, a to znači da se može izvršiti povezivanje sa podacima u tim kolonama baze podataka. U polje *DataSet name* se upisuje ime objekta za povezivanje sa bazom podataka, koje će se pojaviti u prozoru *Solution Explorer*, nakon klika na dugme *Finish*, kao na slici 8.12. Sada se u prozoru *Data Source* sa slike 8.13. svakoj koloni može pridružiti tip objekta u kome se mogu prikazatu podaci iz te kolone. Podaci iz pojedinačnih kolona se mogu prikazivati u objektima: *TextBox, ComboBox, Label, ListBox*. Čitava tabela se može prikazati u objektu *DataGridView*. Kada se završi dodjeljivanje objekata svakoj koloni ili čitavoj tabeli, onda se prevlačenjem kolona mišem na formu automatski pojavljuje dodjeljeni objekat i labela sa imenom kolone ispred tog objekta.



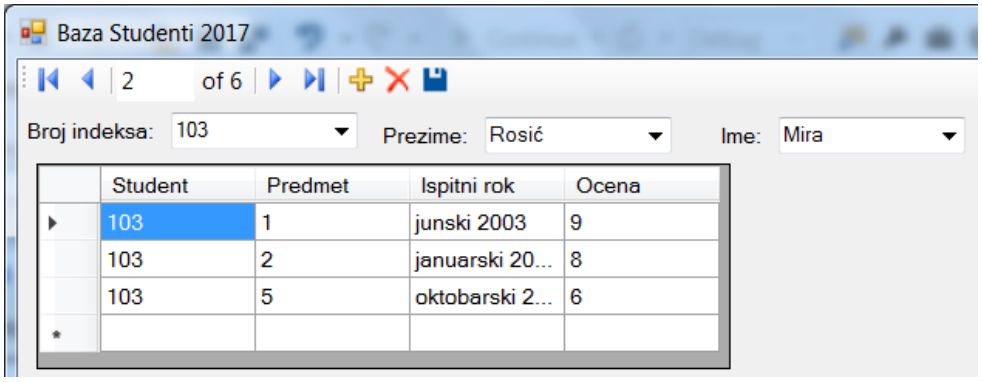
Slika 8.11. Prozor za izbor fajla



Slika 8.12. DataSet objekat



Slika 8.13. Data Source struktura



Slika 8.14. Preuzeti podaci iz baze

Kada se na formu postave objekti, koji su povezani sa bazom podataka, na vrhu forme se pojavljuje automatski objekat *BindingNavigator*, kao na slici 8.14. Ovaj objekat se koristi za izbor podatka iz kolona, koji će biti prikazani u postavljenim objektima na formu. Izbor od prvog do zadnjeg podatka u koloni ili čitavoj tabeli se vrši preko strelica lijevo/desno. Ove strelice i ikonice (za brisanje i spašavanje) postaju aktivne kada se program pokrene sa ikonicom ▶ ***Start*** (ili tipka F5).

8.3. *VISUAL BASIC I SQL* UPITI

Visual Basic omogućava razne vidove komunikacije sa relacionim bazama podataka, kao i da se u njemu prave *SQL* upiti. U ovom poglavlju pokazaćemo, kako se preko koda programa može direktno pristupiti *Access* bazi podataka i kako napraviti razne *SQL* upite. Rezultati *SQL* upita su u većini slučajeva opet tabele pa se najčešće prikazuju u objektu *DataGridView*, ali se mogu prikazivati i u drugim objektima (*List*, *ComboBox* i *TextBox*).

SQL predstavlja danas standardni jezik relacionih baza podataka. Naziv potiče od naziva na engleskom jeziku *Structured Querry Language*, što u prevodu glasi "Strukturirani upitni jezik". U *SQL*-jeziku osnovni objekti manipulacija su tabele, a rezultat toga su isto tabele, čak i kada se kao rezultat manipulacije dobija skup vrijednosti ili samo jedna vrijednost.

SQL jezik podržava tri osnovne funkcije:

- definicija baze podataka;
- manipulacija bazom podataka;
- kontrola pristupu podacima.

SQL podržava četiri manipulativna iskaza:

- ***SELECT*** – pretraživanje;
- ***INSERT*** – umetanje;
- ***UPDATE*** – ažuriranje;
- ***DELETE*** – brisanje.

Naredba ***SELECT*** za upite predstavlja najznačajniju i najčešće korišćenu *SQL* naredbu za manipulaciju podacima. U principu, kod svakog upita zadajemo:

- koje podatke tražimo kao rezultat;
- iz kojih tabela to tražimo;
- koji uslov treba da zadovolje podaci da bi bili uključeni u rezultat;
- po kom redoslijedu želimo prikazati rezultat.

Osnovni oblik iskaza pretraživanja u *SQL*-u ima sljedeću strukturu:

```
SELECT lista-kolona
FROM ime-tabele
WHERE logički-izraz
```

Lista-kolona je spisak svih kolona za koje pravimo upit iz tabele pod imenom ***ime-tabele***. ***Logički-izraz*** uključuje poređenja vrijednosti kolona, konstanti i izraza (odgovarajućeg tipa) u kojima kolone i konstante učestvuju. Pripadne relacijske operacije su: =, <>, <, <=, >, >=, *IN*, *BETWEEN*, *LIKE*, *IS NULL* (*NOT IN*, *NOT BETWEEN*, *NOT LIKE*, *IS NOT NULL*).

SELECT iskaz može biti dosta složen, i može da uključi sljedeće linije, obavezno u navedenom redoslijedu:

SELECT ListaKolona
FROM Tabela
[**WHERE** uslov koji svaki red u tabeli **Tabela** mora da zadovoljava da bi bio uzet u postupak grupisanja]
GROUP BY ListaKolona po kojima radimo grupisanje
[**HAVING** uslov koji svaki red formiran svođenjem mora da zadovolji da bi bio uključen u rezultat]
[**ORDER BY** način sortiranja podataka u koloni [ASC ili DESC]]

Opisaćemo pravljenje *SQL* upita kroz bazu podataka **Biblioteka.mdb**. Potrebno je prvo gotovu bazu podataka snimiti na željeni direktorijum na disku računara. Baza podataka se može postaviti na proizvoljni direktorijum na računaru, ali se ipak preporučuje da se baza podataka uvijek postavi na pod direktorijum projekta **\bin** koji *Visual Basic* 2013 sam izgeneriše, prilikom pravljenja novog projekta. Na taj način će se obezbjediti prenošenje baze sa svim ostalim elementima projekta, kada se projekat prebacuje sa jednog računara na drugi. Zatim je kao u predhodnom poglavlju 8.2. potrebno dodati bazu podataka u projekat. Da bi napravili novu *DataSet* strukturu potrebno je u glavnom meniju *Visual Basic*, izabrati *View —> Other Windows —> Data Sources*. Dodavanje nove *Data Sorces* strukture vrši se klikom na ikonicu pod nazivom *Add New Data Sorce*. Nakon toga se treba izabrati opciju *Database,* pa izabrati opciju *Dataset*. U polju *Data Source* se bira tip baze sa kojom se želi izvršiti povezivanje (u našem slučaju je to *Microsoft Access Database File*). U polje *Database file name* potrebno je upisati potpunu putanju do baze podataka, kao i ime baze sa njenom ekstenzijom. Ovo je lakše uraditi klikom na dugme *Brovse...* Kada se završi postupak pravljena *DataSet* strukture tek tada se može preći na povezivanje sa bazom podataka preko koda.

U prvoj kodnoj liniji programa treba otkucati kod, koji označava **Ole** pristup bazi podataka.

```
Imports System.Data.OleDb
```

Zatim se unutar klase forme, koju definiše sam projekat (u našem slučaju je to forma pod nazivom FormBaza1), prvo definiše promjenjljiva **Con** preko koje se definiše tip baze podataka kojoj se pristupa, a zatim i putanja na kojoj se nalazi baza podataka. Prvo sljedi kod kada se pristupa bazi, koja se nalazi na fiksnom direktorijumu **D:\Biblioteka.mdb.**

```
Public Class FormBaza1
Dim Con As OleDbConnection = New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Dat
a source=D:\Biblioteka.mdb;")
```

Umjesto direktnog načina definisanja putanje može se uraditi i apsolutno definisanje putanje. Kod apsolutnog definisanja putanje potrebno je da baza i program u kome se radi sa tim dokumentom budu na istom direktorijumu, odnosno bazu treba postaviti na pod direktorijum **\bin** , a koji *Visual Basic* 2013 sam izgeneriše. Apsolutni način definisanja putanje se prepuručuje pri programiranju u *Visual Basic*-u 2013.

```
Public Class FormBaza1
Dim Con As OleDbConnection = New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Dat
a source=..\Biblioteka.mdb;")
```

Deklariše se promjenljiva **Cmd** kao baza podataka, a zatim promjenljiva **DT** kao tabela. Ako želimo da se sa bazom podataka radi na svim formamam i procedurama u programu, onda se umjesto lokalne deklaracije promjenljive sa **Dim** radi globalno deklarisanje sa **Public**.

```
Dim Cmd As New OleDbCommand
Dim DT As New DataTable
```

Ako se rezultat *SQL* upita želi prikazati u objektu *DataGridView*, onda je poželjno da se uvijek na početku programa prebrišu stari podaci u ovom objektu, a to se može uraditi preko sljedećeg koda.

```
DT.Clear()
DataGridView1.DataSource = DT
```

Otvaranje postojeće baze podataka da bi iz nje mogli da čitamo ili u nju da upisujemo nove podatke vrši se preko sljedeće dvije komande.

```
Con.Open()
Cmd.Connection = Con
```

Kada je baza otvorena mogu se nad njom sada praviti razne vrste *SQL* upita. Prvo sljedi SQL upit kojim se iz tabele KNJIGA uzimaju svi podaci i upisuju prvo u promjenljivu "**DT"** (tabela). Zatim se iz ove promjenljive prebroji broj redova (komandom *Count*), pa se iz 3 reda tabele (tabela ima i 0 red) uzimaju sadržaji kolone "SIFN" i kolone "SIFK". Preko tri promjenljive dobijene vrijednosti se prikazuju u tri *TextBox*-a.

```
Cmd.CommandText = "SELECT * FROM KNJIGA"
DT.Load(Cmd.ExecuteReader)
DataGridView1.DataSource = DT
Var1 = DT.Rows.Count
Var2 = DT.Rows(2)("SIFN").ToString
Var3 = DT.Rows(2)("SIFK").ToString
```

```
TexVar1.Text = Var1
TexVar2.Text = Var2
TexVar3.Text = Var3
```

Svaki put nakon otvaranja baze potrebno je bazu zatvoriti, kako ne bi došlo do greške pri pokušaju ponovnog otvaranja baze, a koja predhodno nije zatvorena. Zatvaranje baze se radi preko sljedećeg koda.

```
Con.Close()
```

Sljedi primjer *SQL* upita, kojim se kao rezultat dobija samo jedan red tabele zbog uslova u *WHERE* strukturi.

```
DT.Clear()
DataGridView1.DataSource = DT
Con.Open()
Cmd.Connection = Con
Cmd.CommandText = "SELECT * FROM KNJIGA WHERE SIFK=9"
DT.Load(Cmd.ExecuteReader)
DataGridView1.DataSource = DT
Var2 = DT.Rows(0)("SIFN").ToString
Var3 = DT.Rows(0)("SIFK").ToString
TexVar2.Text = Var2
TexVar3.Text = Var3
Con.Close()
```

Sljedi primjer *SQL* upita sa korišćenjem agregatne funkcije ***Avg*** za računaje srednje vrijednosti u jednoj koloni (koja mora da sadrži samo brojeve).

```
DT.Clear()
DataGridView1.DataSource = DT
Con.Open()
Cmd.Connection = Con
Cmd.CommandText = "SELECT Round(Avg(DANA),4) as
Srednja FROM POZAJMICA"
DT.Load(Cmd.ExecuteReader)
DataGridView1.DataSource = DT
var1 = DT.Rows(0)("Srednja").ToString
TexVar1.Text = Var1
Con.Close()
```

Sljedi primjer *SQL* upita za unos novog reda sa novim podacima u postojeću bazu podataka. U kodu je prikazana i provjera da li već postoji podatak, koji se želi unijeti. Ovo je potrebno uraditi kako bi se spriječilo da se pokuša unijeti dupli podatak u kolonu koja je definisana kao primarni ključ tabele.

```
DT.Clear()
DataGridView1.DataSource = DT
Con.Open()
Cmd.Connection = Con
Cmd.CommandText = "SELECT SIFO FROM OBLAST WHERE
SIFO='S5'"
DT.Load(Cmd.ExecuteReader)
BrRed = DT.Rows.Count
If BrRed = 0 Then
   Cmd.CommandText = "INSERT INTO OBLAST (SIFO,NAZIV)
VALUES ('S5','TEST5')"
   DT.Load(Cmd.ExecuteReader)
Else
   MsgBox("Ovaj podatak vec ima u bazi", , "Greska")
End If
DataGridView1.DataSource = DT
Con.Close()
```

Sljedi primjer *SQL* upita za izmjenu podataka u postojećoj bazi podataka (u tabeli pod imenom POZAJMICA, u koloni SIFP pronaći red u kome ova kolona ima vrijednost 6, a zatim u tom pronađenom redu u koloni DANA staru vrijednost zamjaniti sa novom vrijednosti 13).

```
DT.Clear()
DataGridView1.DataSource = DT
Con.Open()
Cmd.Connection = Con
Cmd.CommandText = "UPDATE POZAJMICA SET DANA=13 WHERE
SIFP=6"
DT.Load(Cmd.ExecuteReader)
DataGridView1.DataSource = DT
Con.Close()
```

Sljedi primjer *SQL* upita za brisanje reda sa podacima u postojećoj bazi podataka. U kodu je prikazana i provjera da li već postoji podatak, koji se želi izbrisati. Ovo je potrebno uraditi kako bi se pojavila poruka greške, kada se pokuša brisati podatak, koji ne postoji.

```
DT.Clear()
DataGridView1.DataSource = DT
Con.Open()
Cmd.Connection = Con
Cmd.CommandText = "SELECT SIFO FROM OBLAST WHERE
SIFO='S5'"
DT.Load(Cmd.ExecuteReader)
```

```
BrRed = DT.Rows.Count
If BrRed = 1 Then
     Cmd.CommandText = "DELETE FROM OBLAST WHERE
SIFO='S5'"
     DT.Load(Cmd.ExecuteReader)
Else
     MsgBox("Ovaj podatak nema u bazi", , "Greska")
End If
DataGridView1.DataSource = DT
Con.Close()
```

8.5. ZADACI ZA SAMOSTALNI RAD

U ovom poglavlju biće dati problemi, koji se mogu riješiti pomoću programa napisanih u *Visual Basic*. Da bi se riješili ovi problemi, potrebno je prvo da se izaberu potrebni objekti i dodaju na radnu površinu. Preporučuje se da se svakom novom objektu prvo da i novo ime. Za svaki problem bi trebao i da se napravi algoritam, a tek zatim se piše odgovarajući kod programa.

8.5.1. Zadaci sa grananjem

Sljede zadaci, koji u sebi sadrže razna grananja. Svi oni se mogu riješiti uz upotrebu *If Then Else* strukture, a neki i uz upotrebu *Case* strukture.

Zadatak 1.

Napraviti program, koji se pokreće klikom na komandno dugme. Unesite preko preko *TextBox*-a proizvoljan realan broj RBA i broj ZBI. Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 10). Naći zbir i razliku broja RBA i broja ZBI, a rezultat prikazati u dva *TextBox*-a. Podjeliti i pomnožiti broj RBA sa brojem ZBI, a rezultat prikazati u druga dva *TextBox*-a. Manji broj (RBA ili ZBI) pomnožiti sa ZBI, a rezultat prikazati u petom *TextBox*-u.

Zadatak 2.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko preko *TextBox*-a jedan proizvoljan realan broj X i broj ZBI. Gdje je Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 2). Za ovu unesenu vrijednost X i ZBI izračunati vrijednost funkcije Y= $\sqrt{(X+5)*(ZBI-X-1)}$. Rezultat prikazati u jednom *TextBox*-u.

Zadatak 3.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko preko *TextBox*-a jedan proizvoljan realan broj X i broj ZBI. Gdje je Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 3). Za ovu unesenu vrijednost X i ZBI izračunati vrijednost funkcije Y= $\sqrt{X + \frac{X^3 + ZBI - 2}{COS(X)}}$.

Rezultat prikazati u jednom *TextBox*-u.

Zadatak 4.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko preko *TextBox*-a proizvoljan realan broj X i broj ZBI. Gdje je Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 4). Izračunati vrijednost funkcije Y

$$\text{Y}= \frac{\text{X}^3 + 2*\text{X}^2 - 5*\text{X} + 8}{(\text{X} + \text{ZBI})*(\text{X} - \text{ZBI})}.$$

Rezultat prikazati u jednom *TextBox*-u.

Zadatak 5.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko preko *TextBox*-a jedan proizvoljan realan broj X i broj ZBI. Gdje je Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 5). Za ovu unesenu vrijednost X i ZBI izračunati vrijednost funkcije

$$\text{Y}= \begin{bmatrix} 3X, \_\_\_\_\_ za \_\_ X < 2 \\ \sqrt{X^2 - ZBI}, za \_\_ 2 \le X < 7 \\ |ZBI - X|, \_\_ za \_\_ X \ge 7 \end{bmatrix}.$$

Rezultat prikazati u jednom *TextBox*-u.

Zadatak 6.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox*-a iznos iznos broja bodova koje je student osvojio na testu. Preko drugog *TextBox*-a unesite broj bodova koje je student osvojio na prisustvo nastavi (maksimalno 5 bodova). Do 55 osvojenih bodova, student dobija ocjenu 5. Od 56 do 65 bodova, student dobija ocjenu 6. Od 66 do 73 boda, student dobija ocjenu 7. Od 74 do 82 boda, student dobija ocjenu 8. Od 83 do 91 bod, student dobija ocjenu 9. Preko 92 boda, student dobija ocjenu 10. Na osnovu unesenog broja bodova, izračunati ocjenu koju student treba da dobije. Rezultat prikazati u jednom *TextBox*-u.

8.5.2. Zadaci sa upotrebom gotovih funkcija

Zadaci nevedeni u ovom poglavlju se rješavaju upotrebom funkcija, koje su već ugrađene u *Visual Studio* 2013.

Zadatak 7.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko preko *TextBox*-a iznos kredita (PV=5000 KM), iznos godišnje kamate na uzeti kredit (RATE=15,3%), vrijeme koliko se kredit vraća (NPER= 1 godina). Izračunati mjesečnu ratu kredita pomoću finansijske funkcije PMT. Rezultat prikazati u jednom *TextBox*-u.

Zadatak 8.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox*-a iznos iznos kredita (PV) koji uzimate kod banke na 6 godina. Za kredite do 200 KM, iznos godišnje kamate na uzeti kredit je 18,3%. Za kredite od 200 KM do 600 KM, iznos godišnje kamate na uzeti kredit je 14,8%. Za kredite od 600 KM do 900 KM, iznos godišnje kamate na uzeti kredit je 13,5%. Za kredite od 900 KM do 1500 KM, iznos godišnje kamate na uzeti kredit je 12,3%. Za kredite preko 1500 KM, iznos godišnje kamate na uzeti kredit je 11,8%. Izračunati mjesečnu ratu kredita pomoću finansijske funkcije PMT.

Rezultat prikazati u jednom *TextBox*-u.

Zadatak 9.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox*-a proizvoljan realan broj X, koji ima 7 decimalna mjesta. Naći cjelobrojnu vrijednost broja X, a rezultat prikazati u jednom *TextBox*-u. Naći realnu vrijednost broja X (vrijednost iza decemalne zapete), a rezultat prikazati u drugom *TextBox*-u. Zaokružiti vrijednost broja X na 5 decimalna mjesta, a rezultat prikazati u trećem *TextBox*-u.

Zadatak 10.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox*-a proizvoljan tekst, koji ima maksimalno 9 karaktera. Naći dužinu unesenog teksta, a rezultat prikazati u jednom *TextBox*-u. Pretvoritu uneseni tekst tako da ima sve velika (mala) slova, a rezultat prikazati u drugom *TextBox*-u. Sa lijeve (desne) strane unesenog teksta izdvojiti 4 karaktera, a rezultat prikazati u trećem *TextBox*-u. Ulazni tekst napisati u obrnutom redoslijedu, a rezultat prikazati u četvrtom *TextBox*-u.

Zadatak 11.

Napraviti program, koji se sastoji od dvije forme. Na prvoj formi postaviti osnovne podatke o studentu i dugme za prelazak na narednu formu. U sredini druge forme postaviti jedan digitalni sat, koji pokazuje sistemsko vrijeme, koje mjenja vrijednost svake 0,4 sekunde. Postaiti drugi digitalni sat, koji pokazuje sistemsko vrijeme, koje mjenja vrijednost svake 1,4 sekunde i pri tome se pomjera po formi od jedne do druge ivice.

Zadatak 12.

Napraviti program, koji simulira generisanje trocifrenog cijelog broja. Svaka cifra se generiše pomoću odvojenog komandnog dugmeta, uz upotrebu generatora slučajnih brojeva. Generisani broj podijeliti sa brojem 13. Naći i prikazati u odgovarajućim *TextBox*-ovima rezultat realnog dijeljenja sa 3 decimalna mjesta, rezultat cjelobrojnog dijeljenja i ostatak cjelobrojnog dijeljenja.

Zadatak 13.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox* -a niz tri cijela broja A, B i C. Napraviti program, kojim se izračunava zbir samo negativnih brojeva, a rezultat prikazati u jednom *TextBox*-u. Izračunati proizvod samo pozitivnih brojeva, a rezultat prikazati u drugom *TextBox*-u.

Zadatak 14.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox* -a niz tri cijela broja A, B i C. Napraviti program, kojim se izračunava ukupan broj negativnih brojeva, a rezultat prikazati u jednom *TextBox*-u. Izračunati zbir samo pozitivnih brojeva, a rezultat prikazati u drugom *TextBox*-u.

Zadatak 15.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko dva *TextBox*-a dva datuma. Napraviti program, kojim se izračunava kolika je razlika između ta dva datuma u sekundama, minutama, satima i danima, a rezultat prikazati u četiri *TextBox*-a.

Zadatak 16.

Napraviti program, koji se pokreće klikom na komandno dugme. Učitajte preko *TextBox*-a vrijeme u sekundama. Napraviti program, kojim se izračunava koliko je to sati, minuta i sekundi, a rezultat prikazati u tri *TextBox*-a.

Zadatak 17.

Učitajte preko dva *TextBox*-a stranicu a i b pravougaonika. Napraviti program, kojim se izračunava dijagonala, obim i površinu pravougaonika, a rezultate prikazati u tri *TextBox*-a.

Zadatak 18.
Učitajte preko dva *TextBox*-a stranicu a i b pravouglog trougla. Napraviti program, kojim se izračunava hipotenuzu c, obim i površinu pravouglog trougla, a rezultate prikazati u tri *TextBox*-a.

Zadatak 19.
Učitajte preko tri *TextBox*-a koeficijente a, b i c kvadratne jednačine ($Y=aX^2+bX+C$). Provjeriti da li za unesene koeficijente kvadratna jednačina ima realna rješenja i ako ima rezultate prikazati u dva *TextBox*-a.

8.5.3. Zadaci sa petljama

Sljede zadaci, koji u sebi sadrže razna ponavljanja. Svi oni se mogu riješiti uz upotrebu *For, Do While* ili *Do Until* petlje.

Zadatak 20.
Napraviti program, koji se pokreće klikom na komandno dugme. Napraviti program, kojim za sve brojeve od 1 do 55 izračunava zbir brojeva, koji su djeljivi sa 5, a rezultat prikazati u jednom *TextBox*-u. Izračunati proizvod brojeva, koji su djeljivi sa 9, a rezultat prikazati u drugom *TextBox*-u.

Zadatak 21.
Napraviti program, koji se pokreće klikom na komandno dugme. Unijeti preko dva *TextBox*-a broj N i broj M. Učitajte preko *InputBox*-a niz AA od N = 5 + ZBI elemenata i niz CC od M = 3 + ZBI elemenata. Gdje je ZBI zadnji broj indeksa studenta, koji se unosi preko *TextBox*-a (ukoliko je nula, onda se uzima 10). Elementi niza mogu biti proizvoljni cijeli brojevi. Sabrati sve elemente niza AA (ZbirA), a rezultat prikazati u jednom *TextBox*-u. Sabrati svaki drugi element (1, 3, 5, ...) niza CC (ZbirC2), a rezultat prikazati u drugom *TextBox*-u. Sabrati svaki peti element niza CC (ZbirC5), a rezultat prikazati u trećem *TextBox*-u. Sve elemente niza prikazati u jednoj koloni *DataGridView* tabele.

Zadatak 22.
Napraviti program, koji se pokreće klikom na komandno dugme. Unijeti preko dva *TextBox*-a broj N i broj M. Učitajte preko *InputBox*-a niz AA od N =(14 - ZBI) elemenata i niz BB od M = 5 + ZBI elemenata. Gdje je ZBI zadnji broj indeksa studenta, koji se unosi preko *TextBox*-a (ukoliko je nula, onda se uzima 5). Elementi niza mogu biti proizvoljni realni brojevi. Naći srednju vrijednost niza AA (SredA), a rezultat prikazati u jednom *TextBox*-u. Naći najveći element niza BB (MaxB), a rezultat prikazati u drugom *TextBox*-u. Naći najmanji element niza BB (MinB), a rezultat prikazati u trećem *TextBox*-u. Zatim pomnožiti SredA sa MaxB,

a rezultat prikazati u četvrtom *TextBox* -u. Sve elemente niza prikazati u jednoj koloni *DataGridView* tabele.

Zadatak 23.

Napraviti program, koji se pokreće klikom na komandno dugme. Unijeti preko dva *TextBox*-a broj N i broj M. Učitajte preko *InputBox*-a niz AA od N =(15 - ZBI) elemenata. Gdje je ZBI zadnji broj indeksa studenta, koji se unosi preko *TextBox*-a (ukoliko je nula, onda se uzima 4). Elementi niza mogu biti proizvoljni cjeli brojevi veći od 5 i manji od 100. Naći zbir elemenata niza AA, koji su manji od 13 (JedA), a rezultat prikazati u jednom *TextBox*-u. Naći zbir elemenata niza A koji su veći ili jednaki 13 i manji od 21 (DvaA), a rezultat prikazati u drugom *TextBox*-u. Naći zbir elemenata niza A, koji su veći ili jednaki 21 i manji od 55 (TriA), a rezultat prikazati u trećem *TextBox*-u. Naći zbir elemenata niza A, koji su veći ili jednaki 55 (CetA), a rezultat prikazati u četvrtom *TextBox*-u. Sve elemente niza prikazati u jednoj koloni *DataGridView* tabele.

Zadatak 24.

Napraviti program, koji se pokreće klikom na komandno dugme. Unijeti preko *TextBox*-a broj N. Učitajte preko *InputBox*-a niz AA od N = (7 + ZBI) elemenata. Gdje je ZBI zadnji broj indeksa studenta, koji se unosi preko *TextBox*-a (ukoliko je nula, onda se uzima 6). Elementi niza mogu biti proizvoljni realni brojevi. Naći najmanji i najveći element niza AA, a rezultate prikazati u dva *TextBox*-a. Sortirati elemente niza prema opadajućem (rastućem) redoslijedu, a rezultat prikazati u trećem Text box-u (ili u tabeli). Sve elemente niza prikazati u jednoj koloni *DataGridView* tabele.

Zadatak 25.

Napraviti program, koji uz pomoć generatora slučajnih brojeva popunjava (realnim brojevima sa tri decimalna mjesta u rasponu od 1 do 5) *DataGridView* tabelu sa 6 kolona i 4 reda. Treba onemogućiti da dođe do ponavljanja brojeva. Zatim pronaći maksimalni i minimalni broj iz tabele, a rezultate prikazati u odgovarajućim *TextBox*-ovima.

Zadatak 26.

Napraviti program, koji uz pomoć generatora slučajnih brojeva popunjava (realnim brojevima sa dva decimalna mjesta u rasponu od 1 do 10) *DataGridView* tabelu sa 3 kolona i 5 reda. Treba onemogućiti da dođe do ponavljanja brojeva. Zatim pronaći maksimalni broj u svakoj koloni. Za ova 3 pronađena najveća broja u tri kolone, naći najmanji od njih. Rezultate prikazati u odgovarajućim *TextBox*-ovima.

Zadatak 27.

Napraviti program, koji uz pomoć generatora slučajnih brojeva popunjava (realnim brojevima sa jednim decimalnim mjestom u rasponu od 1 do 33) *DataGridView* tabelu sa 5 kolona i 4 reda. Treba onemogućiti da dođe do ponavljanja brojeva. Zatim pronaći najmanji broj u svakom redu. Za ova 4 pronađena najmanja broja u redovima, naći najveći od njih. Rezultate prikazati u odgovarajućim *TextBox*-ovima.

Zadatak 28.

Unijeti preko jednog *TextBox*-a broj N. Učitajte preko *InputBox*-a niz AA od N = (4 + ZBI) elemenata. Gdje je ZBI zadnji broj indeksa studenta, koji se unosi preko *TextBox*-a (ukoliko je nula, onda se uzima 7). Klikom na dugme napuniti novi niz BB koji ima 2*N članova. Parni članovi niza BB su članovi niza AA uvećani za 3, a neparni članovi niza BB su članovi niza AA pomnoženi sa 2. Uneseni niz AA i dobijeni niz BB prikazati u odgovarajućim *TextBox*-ovima ili u *DataGridView* tabeli.

Zadatak 29.

Klikom na dugme izračunati zbir prirodnih brojeva od 1 do 100 čija je zadnja cifra 3 ili 8 (3, 8, 13, 18, 23, 28, ... , 93, 98) i rezultat prikazati u odgovarajućim *TextBox-u*.

Zadatak 30.

Unijeti preko jednog *TextBox*-a prirodan broja N, koji može biti samo od 0 do 9. Zatim klikom na dugme izračunati zbir prirodnih brojeva od 1 do 1000 čija je zadnja cifra N i rezultat prikazati u odgovarajućim *TextBox*-u.

Zadatak 31.

Klikom na dugme izračunati zbir prirodnih brojeva od 1 do 100, koji su djeljivi sa 3, a rezultat prikazati u jednom *TextBox-u*. Klikom na drugo dugme izračunati zbir prirodnih brojeva od 1 do 100, koji su djeljivi sa 7, a rezultat prikazati u drugom *TextBox*-u. Veći zbir podjeliti sa manjim i prikazati u trećem *TextBox*-u.

Zadatak 32.

Unijeti preko jednog *TextBox*-a prirodan broja od 5 cifara. Klikom na dugme napraviti broj sa istim ciframa ali u inverznom poretku, a rezultat prikazati u drugom *TextBox*-u (na primjer od 13579 treba dobiti 97531).

Zadatak 33.

Unijeti preko jednog *TextBox*-a prirodan broja od 2 do 9 cifara. Klikom na dugme napraviti broj sa istim ciframa ali u inverznom poretku, a rezultat prikazati u drugom *TextBox*-u (na primjer od 13579 treba dobiti 97531).

8.5.4. Zadaci za pristup *Excel* dokumentu i *Access* bazi

Sljede zadaci gdje je potrebno iz *Visual Studio* 2013 pristupiti postojećem *Excel* dokumentu, a zatim i zadatak u kome je potrebno izvršiti povezivanje sa *Access* bazom podataka.

Zadatak 34.

Potrebno je napraviti novi *Excel* dokument pod imenom **Ispit.xls**, koji ima jednu stranicu, koja se zove **List1**. U ćeliju **A3** upisati tekst **Excel22.** Ovaj dokument je potrebno snimiti na direktorijumu na, koji se snimi i glavni program.

U *Visual Studio* 2013 naprviti tri *TextBox*-a i četiri dugmeta. Klikom miša na prvo dugme potrebno je da se otvori *Excel* dokument **Ispit.xls** i u ćeliju **B5** upiše sadržaj prvog *TextBox*-a, a u ćeliju **C5** sadržaj drugog *TextBox*-a. Klikom na drugo dugme potrebno je iz Excela preuzeti sadržaj ćelije **A3** u treći *TextBox*. Pomoću trećeg dugmeti izvršiti prenos formule za sabiranje tri ćelije (=D4+D5+D6) u ćeliju D7. Četvrto dugme iskoristiti za zatvaranje otvorenog *Excel* dokumenta i kompletnog programa.

Zadatak 35.

Napraviti bazu podataka **Ispit.mdb** u *Access*-u. Baza treba da ima jednu tabelu pod imenom **Student**, sa 3 kolone **RedBroj** (za unos rednog broja studenta), **Ime** (za unos imena studenta) i **Prezime** (za unos prezimena studenta). Zatim u napravljenu bazu podataka unijeti podatke za 10 studenata. Bazu je potrebno snimiti na direktorijumu, na koji se snimi i glavni program u *Visual Studio* 2013.

Pomoću komandnog dugmeta cijeli sadržaj tabele **Student** prikazati u jednoj *DataGridView* tabeli. Iz tabele **Student** u koloni **RedBroj** pronaći broj **4.** Za taj pronađeni broj (taj RECORDSET) ispisati sadržaj kolone **Ime** u jedan *TekstBox* i sadržaj kolone **Prezime** u drugi *TekstBox*.

Pomoću komandnog dugmeta iz tabele **Student** u koloni **Ime** pronaći studenta pod imenom **Ilija.** Za to pronađeno ime zamjeniti sadržaj kolone **Prezime** u Rosic.

Pomoću komandnog dugmeta iz tabele **Student** izbrisati sve studente, koji imaju prezime Covic. Izbrojati koliko je izbrisano studenata iz baze podataka, a dobijeni broj prikazati u jednom *TekstBox*-u.

Pomoću komandnog dugmeta dodati novi red u tabelu **Student**, sa sljedećim podacima: 12, Mihailo, Rosic.

Pomoću komandnog dugmeta izmjenjeni sadržaj tabele **Student** prikazati u drugoj *DataGridView* tabeli.

Pomoću komandnog dugmeta napraviti *SQL* upit nad tabelom **Student,** tako da se prikažu samo imena i prezimena onih studenata, koji imaju redne brojeve veće od 3 i manje od 8. Rezultat *SQL* upita prikazati prikazati u trećoj *DataGridView* tabeli. Napraviti dugme za zatvaranje *Access* baze podataka i cijelog programa.

L I T E R A T U R A

[1] Microsoft Visual Studio 2013 Help, Microsoft, 2013.
[2] Bruce Johnson, *Professional Visual Studio 2013*, JohnWiley & Sons, Inc., Indianapolis, Indiana, 2014.
[3] Michael Halvorson, *Developer Step by Step Microsoft Visual Studio 2013 2013*, O'Reilly Media, Inc., California, 2013.
[4] Srđan Damjanović, Predrag Katanić, *Programski jezik Visual Basic Zbirka zadataka*, Fakultet poslovne ekonomije, Bijeljina, 2014.
[5] Dr Jozo J. Dujmović, *Programski jezici i metode programiranja*, Akademska misao, Beograd, 2003.
[6] Dr Tihomir Latinović, *Osnove programiranja (Visual Basic)*, Biblioteka Informacione tehnologije, Banja Luka, 2007.
[7] Dr Lazar Miličević, Mr Lazar Radovanović, *Programiranje (Visual Basic)*, Ekonomski fakultet, Brčko, 2005.
[8] Slobodan Obradović, *Veština dobrog programiranja*, Viša elektrotehnička škola Beograd, 1997.
[9] Srđan Damjanović, Predrag Katanić, Borislav Drakul, *Zbirka zadataka iz poslovne informatike*, Fakultet spoljne trgovine, Bijeljina, 2008.
[10] Srđan Damjanović, Predrag Katanić, *Programski jezik VEE Pro*, Elektrotehnički Fakultet, Istočno Sarajevo, 2011.
[11] Dr Milan Popović, *Osnove programiranja*, BSP, 2007.
[12] Ascii kod, http://frank.harvard.edu/aoe/images/t10r3.pdf, 28.01.2011.

CIP - Каталогизација у публикацији
Народна и универзитетска библиотека
Републике Српске, Бања Лука

004.432.2(075.8)

ДАМЈАНОВИЋ, Срђан, 1971-
Integrisano razvojno okruženje Visual Studio 2013 / Srđan Damjanović, Predrag Katanić. - Bijeljina : Fakultet poslovne ekonomije, 2017 (Bijeljina : Grafika Gole). - 213 str. : ilustr. ; 25 cm

Tiraž 200. - Bibliografija: str. 213.

ISBN 978-99955-45-23-9

1. Катанић, Предраг, 1970- [аутор]

COBISS.RS-ID 6641176