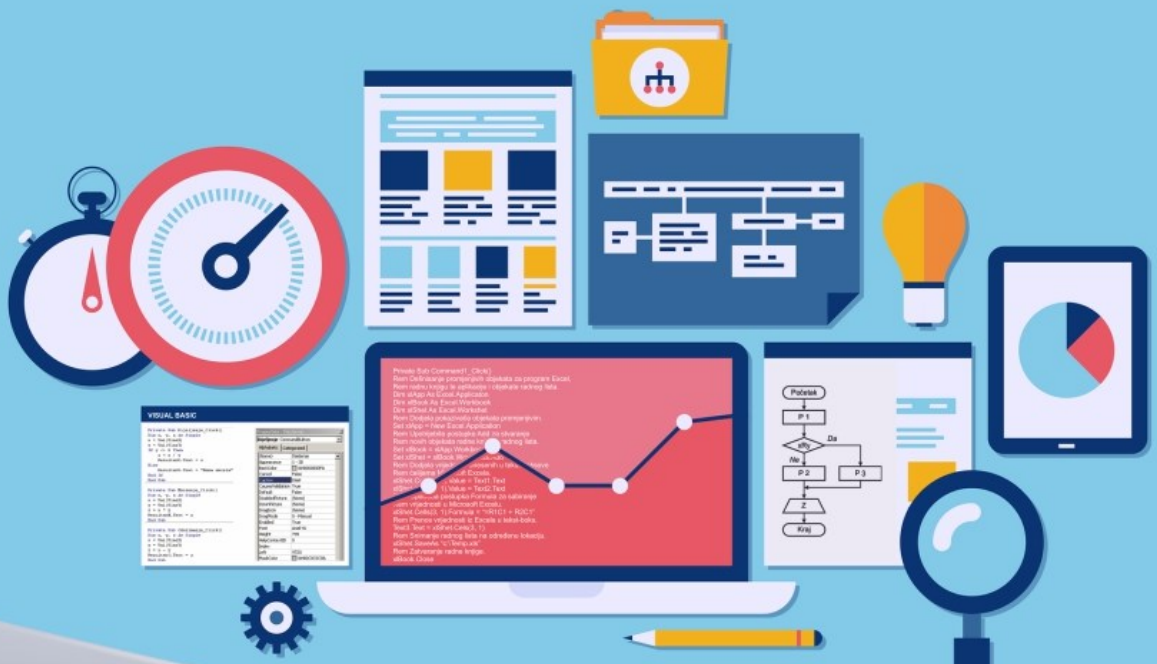
Dr Srđan Damjanović
Dr Predrag Katanić

PROGRAMSKI JEZIK VISUAL BASIC

ZBIRKA ZADATAKA



UNIVERZITET U ISTOČNOM SARAJEVU
FAKULTET POSLOVNE EKONOMIJE BIJELJINA

Dr Srđan Damjanović
Dr Predrag Katanić

PROGRAMSKI JEZIK *VISUAL BASIC*

ZBIRKA ZADATAKA

FAKULTET POSLOVNE EKONOMIJE
BIJELJINA, 2014.

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

Izdaje:
FAKULTET POSLOVNE EKONOMIJE
Bijeljina

Za izdavača:
Prof. dr Srđan Damjanović

Štampa:
GRAFIKA GOLE
Bijeljina

Tiraž:
200 primjeraka

ISBN: 978-99955-45-18-5

Posveta

Ovu knjigu posvećujem bakama Goši i Vukici.

Zahvaljujem se supruzi Slobodanki, bogatstvu naše ljubavi Teodori i Mihailu, koji su sa puno razumjevanja omogućili moj rad na ovoj knjizi.

Srđan Damjanović

S A D R Ž A J

UVOD

Visual Basic je objektno orjentisani programski jezik, koji je kao programski jezik prešao dug razvojni put od programskog jezika *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.

Knjiga je napisana zbog nedostatka na tržištu literature na srpskom jeziku sa praktičnim primjerima za programski jezik *Visual Basic*. Naime, postoje samo uputstva za rad sa raznim verzijama programskog jezika *Visual Basic*, 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. Na relativno skromnom broju stranica izložena je 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*. 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.

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 programskog jezika *Visual Basic,* kroz terijska 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 programa *Visual Basic* predstavljen je u drugom poglavlju. Na kraju je opisano, kako može izgledati postupak pravljenja prvog programa u ovom programskom jeziku.

U trećem poglavlju opisani su osnovni tipovi podataka, koji se koriste prilikom pisanja programa u programskom jeziku *Visual Basic*. Uz svaki tip podatka prikazan je primjer programa, kaji opisuje kako se on koristi.

Naredbe selekcije i iteracije u *Visual Basic*-u 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 Basic*-u, 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 gotovi objekti, koji postoje u standardnoj paleti objekata u programskom jeziku *Visual Basic*. Za svaki objekat su dati primjeri programa za korišćenje ovih objekata.

Primjeri programa napisanih u programskom jeziku *Visual Basic* predstavljeni su u osmom poglavlju. Prvo su predstavljeni primjeri programa, koji se mogu naći u *Help*-u programskog jezika *Visual Basic*. Zatim su predstavljeni praktični primjeri kodova 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 Basic*. 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 upotrebljavaju cifre i slova, koje jednim imenom nazivamo alfanumjerički znaci, za međusobno sporazumijevanje. 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 programskim jezicima,

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

da mogu raditi u različitim režimima korišćenja i pod različitim operativnim sistemima. Ovo je dovelo do toga, da pojam računar nedovoljno opisuje kompleksnost ovakvog sistema, pa se često pojam računara uopštava pojmom računarski sistem. Pod pojmom računarski sistem podrazumijevamo skup uređaja, čije su mogućnosti u obradi podataka rezultat jedinstvenog djelovanja njenog hardvera i softvera. Ovakav pristup računarskom sistemu nas dovodi do grubog rasčlanjenja računarskog sistema tj. računarski sistem se sastoji iz:

- ***hardvera,***
- ***softvera.***

Drugačije rečeno, jedinstveno djelovanje hardvera i softvera 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: ***centralna jedinica*** (kućište računara u koje su smješteni: matična ploča, procesor, memorijski uređaji, napojna jedinica, itd) i ***periferni uređaji*** (ulazni i izlazni). 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, 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 i razne vrste sekundarnih memorija.

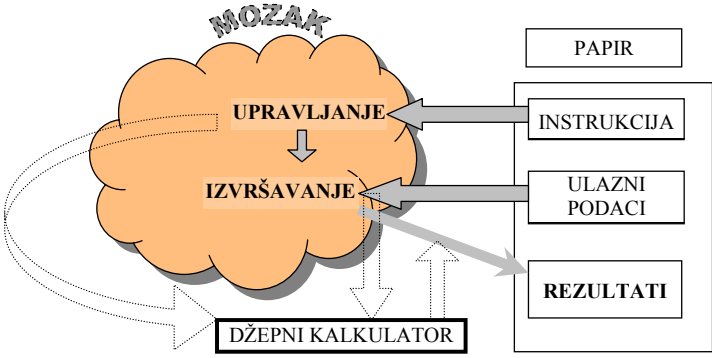
U ovomj knjizi ćemo se zadržati samo na centralnoj jedinici i njenim komponentama, koje su ujedno i osnovne komponente računarskog sistema.

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

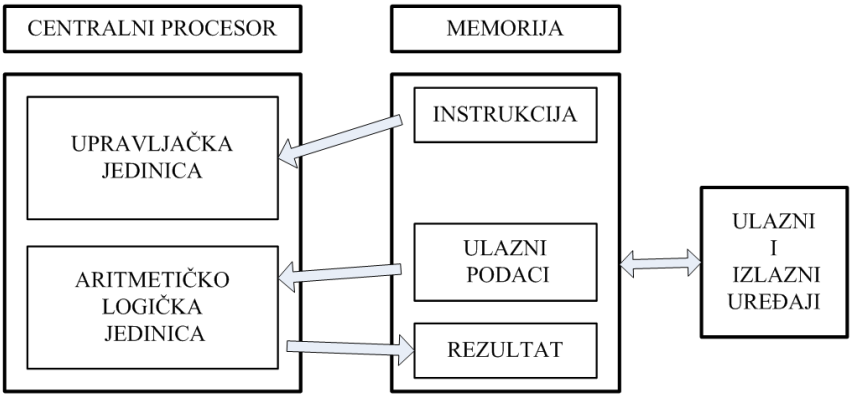
vezana za upravljanje, odnosno tumačenje instrukcije i donošenje odluke o operaciji koja će se izvršiti. Druga radnja je samo izvršavanje operacije, pri čemu se najćešće podaci potrebni za njeno izvršenje čitaju sa papira, a rezultat operacije se takođe zapisuje na papir. Blok dijagram koji predstavlja 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 aritmetičko logičku jedinicu.



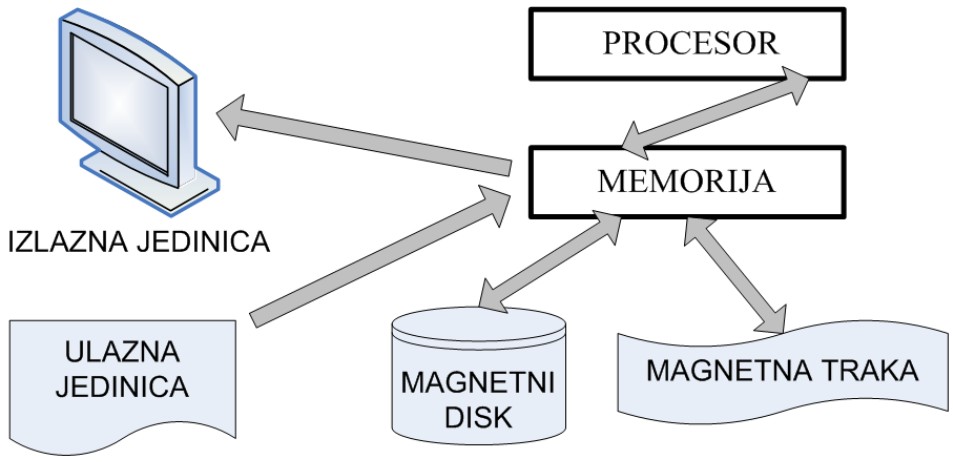
Slika 1.2. Osnovna blok šema računara

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. Glavna memorija računara (*RAM memory*) 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. Ulazni uređaji vrše pretvaranje podatak i instrukcija iz forme razumljive čovjeku u formu prilagođenu skladištenju u memoriji računara. Izlazni uređaji pretvaraju rezultate obrade iz forme u kojoj su zapisani u memoriji računaru, u formu razumljivu za čovjeka.

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 i magnetne trake kao eksterne memorije.

Naime, kapacitet 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 primarnoj memoriji, programi koji se trenutno izvršavaju i podaci nad kojima se vrši obrada, prebacuju se po potrebi iz sekundarnih memorija u primarnu memoriju.

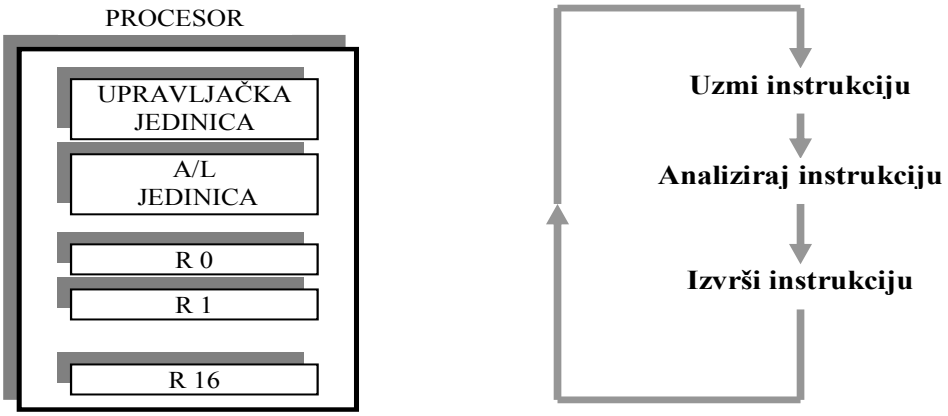
U nastavku izlaganja, kako bismo lakše shvatili suštinu funkcionisanja računarskog sistema za šta je zadužen operativni sistem, ograničićemo se na osnovni opis strukture i funkcije dvije ključne komponente računarskog sistema: centralnog procesora i primarne memorije.



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. Ciklični proces uzimanja, dekodiranja i izvršavanja instrukcije, takođe je prikazan na slici 1.4. On se često naziva uzmi-izvrši ciklus (*fetch-execute cycle*).



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. 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 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, kao npr. registri procesora koji sada predstavljaju internu memoriju takvih memorijskih dimenzija da omogućavaju mnogo rjeđe pristupe sporijoj *RAM*

memoriji. Ovo drastično povećava brzinu rada procesora (interna memorija procesora se naziva "keš" memorija). Današnji računarski sistemi imaju i do 130000 mikroprocesora ("srce računara"), koji rade brzinama reda GHz. 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. Primarna memorija

Primarna (operativna) 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). Očigledno je da redni broj registra predstav1ja njegovu jedinstvenu adresu. Uvedimo sada predpostavku da jedan memorijski registar moža da sadrži jednu i samo jednu instrukciju ili samo jedan podatak. Tada 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 upisao u prihvatni registar memorije.

Operacija upisivanja podatka u određenu memorijsku lokaciju odvija se na sličan način. Podatak koji treba da se upiše unosi se u prihvatni registar memorije, a adresa lokacije u koju se vrši upisivanje unosi se u adresni 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, što se razlikuje od situacije kod sekundarnih

memorija, gdje vrijeme pristupa nekoj lokaciji zavisi ne samo od fizičke pozicije te lokacije, već i od toga kojoj se lokaciji prethodno pristupilo. Zbog toga se primarna memorija često naziva memorija sa slučajnim pristupom. Ovaj naziv je zapravo doslovan i pomalo nespretan prevod originalnog izraza (*random access memory - RAM*).

1.2. S O F T V E R

Softver (*eng. software*) računara predstavlja skup programa i podataka, smještenih u memoriju računara, koji kao cjelina, povremeno ili stalno, kontroliše i upravljanja radom računara. Ovdje se pod memorijom podrazumijeva sva dostupna 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:

- operativni sistem (OS),
- uslužni programi.

Operativni sistemi su se razvijali sa razvojem računarskih sistema tj. potrebama čovjeka za sve većom količinom informacija. Sa stanovišta OS-a, hardverska komponenta računarskog sistema grubo se može podijeliti u dva dijela, sistemski dio hardvera (procesor, memorija i dr.) i dio hardvera koji čini interfejs sa korisnikom (monitor, tastatura, miš i dr.).

Slika 1.5. Struktura operativnog sistema

Za vrijeme pionirskog razvoja računarskih sistema, a samim tim i pionirskog razvoja OS-a, cijena sistemske komponente je bila neuporedivo veća od interfejsa (terminal) tj. rad "mašine" je bio daleko skuplji od rada korisnika. Iz ove činjenice nastaje ideja da prvi operativni sistemi budu tako koncipirani, kako bi se omogućilo 100% iskorićenje procesora.

Iz ovog ugla OS se posmatra na njegovom najnižem nivou, kada poput nadzornika pazi na disk, procesor, memoriju, štampače i dr.

Sa razvojem tehnologije izrade hardvera dolazi do pada cijene sistemskog hardvera. Rad korisnika računara dobija na vrijednosti, pa se ovoj činjenici prilagođava i operativni sistem tj. u prvi plan dolaze odgovornosti OS-a na nivou korisnika. Iz ovog se jasno vidi, da 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.

Kompleksnost operativnog sistema otežava preciznu definiciju. Sa zadržanom uopštenošću, može se reći, da pod operativnim sistemom podrazumijevamo skup sistemskih programa, koji djeluju kao posrednik između hardvera i korisnika. Pri tome pruža korisniku usluge, koje olakšavaju projektovanje, implementaciju i održavanje programa, a istovremeno, upravljaju 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, sa ciljem da omogući efikasan rad korisnika 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. Operativni sistemi mogu biti struktuirani na različite načine: kao monolitni sistemi, kao hijerarhija slojeva, kao virtuelne mašine i kao klijent server model. Jedna od mogućih struktura operativnog sistema prikazana je na slici 1.6. Navedimo neke od poslova koje obavljaju ove komponente operativnog sistema:

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

- upravljanje sistemom prekida računara i obradu prekida,
- planiranje procesa,
- manipulaciju nad procesima,
- 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 i izdavanje računa korisnicima za potrošene resurse.

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 svi 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, a to je zapravo element za projektovanje. 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[2]:

- 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. Preskakanje ove faze znači sigurno velike probleme u svim narednim fazama koje slijede. 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

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

neophodno definisati fizički i matematički model sistema ili procesa čije ponašanje se želi implementirati na računaru. Fizički model uvijek predstavlja model realnog sistema ili procesa, koji se odvija u prirodi. Matematički model je skup matematičkih postupaka i relacija kao i puteva njihovog rješavanja, koji moraju egzaktno odrediti postavljeni fizički model. 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, tj. ne možemo se potpuno osloniti na njegov automatizam u obavljanju različitih poslova.

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 izvjesne zaključke, koji predstavljaju rješenje zadatka. Ovakav skup informacija čini *ulazne veličine zadatka*. Kao rezultat zadatka dobijaju se *izlazne veličine zadatka*.

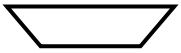
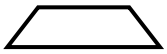
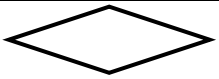
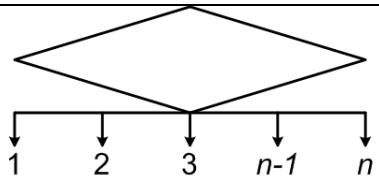
Izbor informacija koje čine ulazne veličine zadatka, kao i tačno formulisanje samog zadatka možemo zvati *postavka zadatka*. Jasno je da prije nego što se pristupi rješavanju, mora se izvršiti tačna postavka zadatka. Rad na postavci zadatka zahtijeva dobro poznavanje oblasti kojoj pripada zadatak.

Kada je izvršena postavka zadatka može se pristupiti rješavanju istog. Uređen skup pravila koja se formuliše u cilju rješavanja zadatka zove se *algoritam*. 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 koji 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 i zadnji algoritamski korak. Može postojati samo jedan prvi korak, a više krajeva algoritma, u zavisnosti od toha da li u algoritmu postoje strukture grananja. Svi ostali algoritamski koraci određeni su izvršenjem predhodnog algoritamskog koraka. Veze između algoritamskih koraka određuju *strukturu algoritama*.

1.3.2.1. Grafički zapis algoritma

Algoritmi za rješavanje pojedinih problema mogu da budu veoma složeni[3]. U takvim slučajevima algoritmi sa tekstualnim opisom pojedinih koraka dosta su nepregledni. Zato je za zapis algoritama pogodno koristiti njihovo grafičko prikazivanje. 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 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
	Definišu obradu podataka
	Uslovni algoritamski korak
	Definiše višestruku odluku
	Povezivanje ili konektor
	Tok algoritma
	Dvosmjerni prenos podataka

[3] 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 jedanputa, u toku jednog izvršavanja algoritma čini linijsku strukturu. Prema tome, karakteristika linijske strukture algoritma jeste da se poslije izvršenog jednog algoritamskog koraka, može preći samo na algoritamski korak koji nije

prethodno izvršen, u toku jednog izvršavanja algoritma. Linijska struktura algoritma može biti prosta ili razgranata.

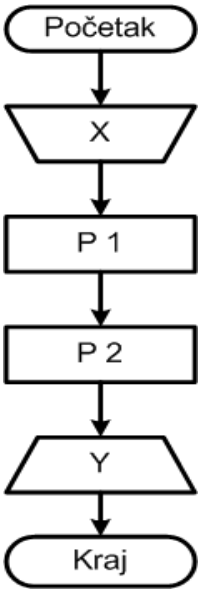
Prosta linijska struktura

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

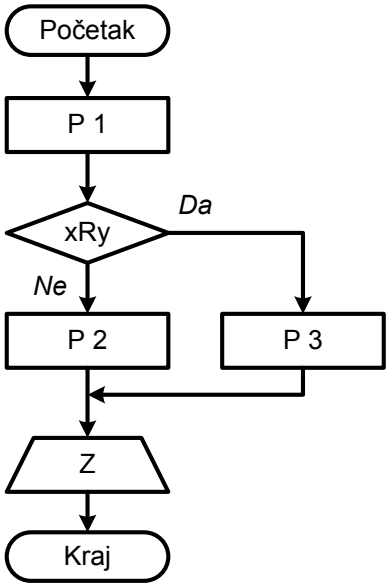
Algoritmi sa prostim linijskim strukturama, sastoje se isključivo od algoritamskih koraka ulaza, obrade ili izlaza. U ovakvim algoritamskim strukturama redoslijed izvršavanja algoritamskih koraka ne zavisi od ulaznih veličina, niti od među rezultata u procesu izvršavanja algoritma. Primjer ove strukture 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 jedanput u toku jednog izvršavanja algoritma. To znači da u razgranatoj algoritamskoj strukturi postoje algoritamski koraci koji se jedanput 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 nikada neće izvršiti.



Slika 1.6. Prosta linijska struktura



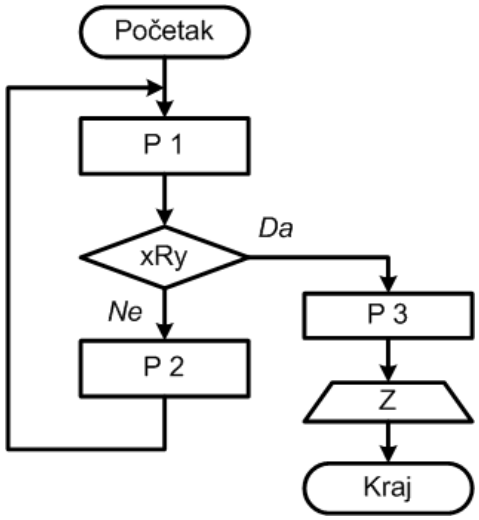
Slika 1.7. Razgranata linijska struktura

Elementarna razgranata struktura može se sastojati od minimalno 3 proste linijske strukture i jednog uslovnog algoritamskog koraka. Na slici 1.7. prikazana je elementarna razgranata struktura sa prostim algoritamskim koracima **P_1, P_2** i **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** može biti ispunjena (Da) i tada se prelazi na prosti algoritamski korak **P_3** ili neispunjena (Ne) i tada se prelazi na prosti algoritamski korak **P_2**. Važno je uočiti da se pri izvršavanju algoritma, na slici 1.7., izvršava uvijek samo jedan od prostih algoritamskih koraka **P_2** ili **P_3**, u zavisnosti od relacije **xRy**.

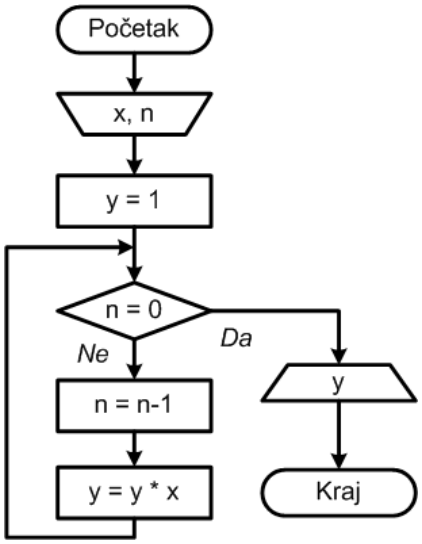
Veličine **x** i **y** mogu biti definisane sa ulaza ili to mogu biti međurezultati određeni u prostom algoritamskom koraku **P_1**. Relacija između veličina **x** i **y** odgovara prirodi ovih veličina. Ako su **x** i **y** brojne veličine može se ispitivati da li su jednake ili koja je od ovih veličina manja, odnosno veća.

Ciklična struktura algoritma

Niz algoritamskih koraka u kojem se jedan ili više algoritamskih koraka može izvršiti više od jedanput, 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 **P_1** i **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 **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. Ukažimo samo da kod iterativnih ciklusa broj prolazaka kroz ciklus nije unaprijed poznat.

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

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 $n - 1$ množenje 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 lijevoj strani, a zatim ovako izračunata vrijednost se dodjeljuje promjenljivoj na desnoj 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 nemože završiti.

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 nemora nikada izvršiti, jer se uslov za ulazak u ciklus ispituje na početku ulaska u ciklus.

Promjenljiva ciklična struktura

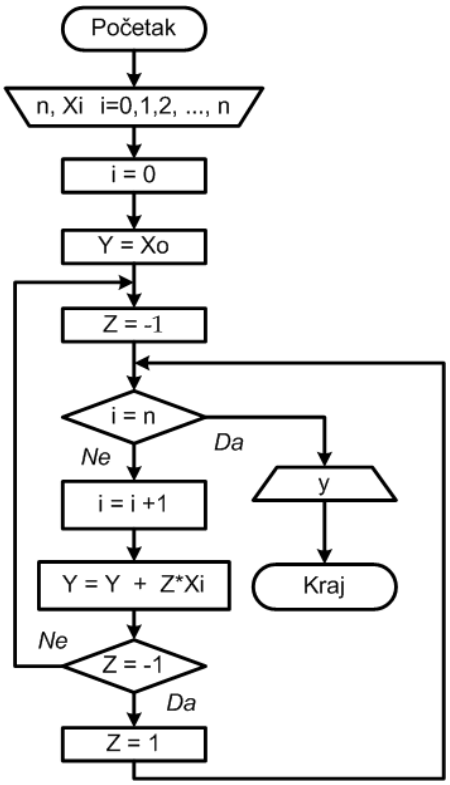
Ciklična struktura u kojoj dolazi do promjene zakona obrade, u jednom ili više algoritamskih koraka, koji se nalaze u ciklusu zove se promjenljiva ciklična struktura. Promjena zakona obrade može se odnositi na promjene operacija, koje vrše obradu informacija ili na promjene promjenljivih koje učestvuju 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 $n, x_0, x_1, x_2, ..., x_n$, a izlazna veličina y. Razvijanjem, suma (1.1.) se može zapisati u obliku

$$y = x_0 - x_1 + x_2 - ...$$



Slika 1.10. Promjenljiva struktura

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 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 $x_0, x_1, x_2, ..., x_n$.

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

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 *programiranje*.

Da bi program mogao da se izvrši na računaru mora biti zapisan na mašinskom jeziku računara. Međutim, pisanje programa na mašinskom jeziku predstavlja takvu detaljizaciju algoritma, da je to vrlo težak posao za čovjeka, a uz to i vrlo podložan greškama. 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.

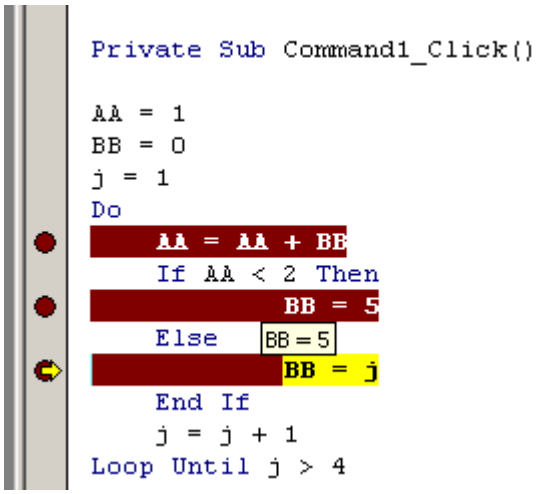
Programski jezici su vještački jezici konstruisani za pisanje progama, pri čemu, po pravilu nije potrebno poznavanje konstruktivnih osobina računara. Važno je uočiti da su programski jezici algoritamski orijentisani, tj. oni omogućuju korisniku da zapiše algoritam za rješavanje određenog zadatka pomoću 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. Iako postoje matematičke metode za takvo dokazivanje (koje se nazivaju verifikacija programa) u praksi se najčešće takve metode ne primjenjuju. Umjesto toga pristupa se 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 netestirani. 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 potrđuje 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 nastaju prilikom kucanja naredbi u programu. U programskom jeziku *Visual Basic*, to su naredbe koje se kucaju u kodu programa nad gotovim objektima od kojih se sastavlja program. Najčešće greške su izostavljanje nekog slova ili da 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. Da bi se otklonile ove 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 u kome su postavljene kontrolne tačke 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 ponovnim klikom na dugme **Start**.

Kada se program zaustavi na kontrolnoj tačci, mogu se provjeravati trenutne vrijednosti svih promjenljivih, koje se nalaze u kodu programa. Ova jednostavna provjera se vrši postavljanjem kursora miša na željenu promjenjivu u kodu programa, nakon čega se odmah prikaže trenutna vrijednost te promjenjive (u primjeru na slici 1.11. prikazana je vrijednost promjenjive 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"[4] 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 saki dio programa, tako što će testirati program za razne vrijednosti ulaznih podatka. Za

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

testiranje programa pored uobičajenih biraju i takozvane granične vrijednosoti. Granične vrijednosti su one vrijednosti podataka kod kojih se mogu očekivati pogrešni rezultati (nula, 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 psecifikaciji 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 testiranje 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 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 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"*. 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.

Kompajleri i interpreteri

Postoje dva načina kako se programi napisani u programskijm jezicima mogu izvršavati u računarima. To su kompajliranje i interpretacija. Kompajliranje se obavlja kompajlerima. Kompajliranje je postupak u kome se kompletan program napisan u izvornom obliku prevodi u mašinski. Kompajliranjem se otkrivaju sve eventualne greške u kodu kompletnog programa. Kao rezultat kompajliranja dobija se izvršna verzija programa. Izvršna verzija programa se zatim može pokretati na bilo kom računaru, nezavisno od razvojne komponente programa.

Interpretacija se obavlja interpreterima. Interpretacija je postupak u kome se samo dio programa napisanog u izvornom obliku prevodi mašinski jezike. Prevodi se samo onaj dio programa do koga se dođe tokom rada sa programo. Pomoću interpretera se otkrivaju i greške ali samo u kodu programa, koji se izvršavao. Kompajlirani programi obično rade brže. Prednost interpretera je što nije potrebno prolaziti sve djelove programa, kao kod kompajliranja da bi se program izvršio.

1.3.5. Distribucija vaših programa

Nakon što napravite *Visual Basic* program, možete ga distribuirati drugim korisnicima. Možete slobodno distribuirati svaki program, koju stvorite sa *Visual Basicom*, svakome ko upotrebljava *Microsoft Windows*. Vaše aplikacije možete distribuirati pomoću CD-a, USB-a, putem intraneta ili Interneta.

Kad distribuirate programa, postoje dva koraka kroz koje morate proći:

• Pakovanje – morate zapakovati datoteke vašeg programa u jednu ili više datoteka tipa ***.cab*** koje mogu biti raspakovane na mjestu koje odaberete i morate stvoriti aplikaciju podešavanja za određene tipove pakovanja. Datoteka tipa *.cab* je sažeta datoteka, koja je dobro opremljena za distribuciju na diskovima ili putem Interneta.

• Raspoređivanje (raspakivanje) – vaš zapakovani program morate premjestiti na mjesto sa koga ga korisnici mogu instalirati. To može značiti kopiranje paketa na lokalni ili mrežni disk, ili postavljanje paketa na *Web* stranicu.

Možete koristiti dva alata za pakovanje i raspoređivanje vaših programa. Čarobnjaka za pakovanje i raspakivanje (*Package and Deployment Wizard*, u opciji *Add-Ins Visual Basica*), ili alat za podešavanje (*Setup Toolkit*), koji dolaze sa vašom instalacijom *Visual Basica*. Čarobnjak za pakovanje i raspakivanje automatizuje puno postupaka uključenih u distribuiranje programa, predstavljajući vam mogućnosti sa kojima možete oblikovati vaše *.cab* datoteke. Alat za podešavanje dopušta vam prilagođivanje dijela onog što se događa tokom postupka instalacije.

Čarobnjak za pakovanje i raspakivanje (*Package and Deployment Wizard*) *Visual Basica* pomaže vam pri stvaranju *.cab* datoteka za vaš program, njihovom grupisanju u cjelinu, ili *paket*, koji sadrži sve informacije potrebne za instalaciju, te isporuku tih paketa krajnjim korisnicima. Čarobnjaka za pakovanje i raspakivanje možete upotrijebiti za stvaranje paketa, koji se distribuiraju na disketama, CD-ima, lokalnom ili mrežnom disku, ili Internetom. Čarobnjak za pakovanje i raspakivanje automatizuje većinu posla upletenog u stvaranje i raspakivanje tih datoteka. Čarobnjak za pakovanje i raspakivanje nudi sljedeće mogućnosti:

• Opcija ***Package*** pomaže vam pri pakovanju datoteka projekta u *.cab* datoteku, koja zatim može biti raspakivana. U nekim slučajevima ova opcija stvara program podešavanja koji instalira *.cab* datoteke. Čarobnjak ustanovljava koje datoteke

trebate pakovati i vodi vas kroz izbore, koji trebaju biti napravljeni kako bi se stvorila jedna ili više *.cab* datoteka za vaš projekt.

• Opcija ***Deploy*** pomaže vam pri isporuci vaših zapakovanih programa na pogodan medij distribucije, kao što su diskete, dio mreže, ili *Web* stranica.

• Opcija ***Manage Scripts*** omogućuje vam pregled i upravljanje skriptima koje ste snimili u prethodnim radovima pakovanja i raspakivanja sa čarobnjakom. Svaki put kad koristite čarobnjaka, snimate skript koji sadrže sve izbore koje ste napravili. Taj skript možete ponovno upotrijebiti u kasnijim upotrebama čarobnjaka, ako želite upotrijebiti slične postavke i napraviti iste izbore kao i prošli put.

Čarobnjak za pakovanje i raspakivanje vodi vas kroz stvaranje i distribuiranje profesionalnih programa podešavanja za vaše *Visual Basic* aplikacije. Osim stvaranja *.cab* datoteka za vašu aplikaciju, čarobnjak također stvara program podešavanja aplikacije prevođenjem projekta alata za podešavanje (*Setup Toolkit*) instaliranog sa *Visual Basic*-om. Program podešavanja se naziva *setup1.exe*.

Pokretanje čarobnjaka za pakovanje i raspakivanje

Možete ga pokrenuti iz *Visual Basica* kao dodatak. Ako pokrećete čarobnjaka kao dodatak, prvo morate postaviti potrebna uputstva u kreatoru dodataka kako bi učitali čarobnjaka. Kad upotrebljavate čarobnjaka kao dodatak, *Visual Basic* pretpostavlja da želite raditi sa projektom koga trenutno imate otvorenog. Ako želite raditi sa drugim projektom, morate otvoriti taj projekt prije pokretanja čarobnjaka, ili morate upotrijebiti čarobnjaka kao samostalan dio.

Nakon što pokrenete čarobnjaka, niz ekrana će od vas tražiti informacije o vašem projektu i omogući će vam biranje opcija za pakovanje. Svaki ekran objašnjava kako se upotrebljava, uključujući neobavezne informacije, te informacije koje moraju biti upisane prije nego što se možete pomaknuti na idući ekran. Ako trebate više informacija o bilo kom ekranu, pritisnite tipku *F1* ili kliknite dugme *Help*. Posebno napominjemo da bi trebali snimiti i prevesti svoj projekt u *EXE* verziju prije nego što pokrenete čarobnjaka za pakovanje i raspakivanje.

Pokretanje čarobnjaka za pakovanje i raspakivanje iz *Visual Basic*-a vrši se u sljedećim koracima:

1) Otvorite projekt koji želite zapakovati ili rasporediti korištenjem čarobnjaka. (Napomena: Ako radite u projektnoj grupi ili imate više istovremeno učitanih projekata, projekat koji želite zapakovati ili rasporediti mora biti trenutan projekt prije nego što pokrenete čarobnjaka.)

2) Upotrijebite menadžera dodacima (*Add-In Manager*) za učitavanje čarobnjaka za pakovanje i raspakivanje. U meniju *Visual Basic*-a ***Add-Ins*** odaberite stavku ***Add-In Manager***, odaberite stavku ***Package and Deployment Wizard*** sa popisa, i kliknite ***OK***.

3) Odaberite stavku ***Package and Deployment Wizard*** iz menija ***Add-Ins*** za pokretanje čarobnjaka.

4) Na glavnom ekranu odaberite jednu od sljedećih opcija:

• Ako želite stvoriti standardno pakovanje, pakovanje za Internet ili datoteku zavisnosti za projekt, kliknite ***Package***.

• Ako želite raspakovati projekt, kliknite ***Deploy***.

• Ako želite pregledati, mijenjati ili obrisati skript, kliknite ***Manage Scripts***.

5) Nastavite kroz ekrane čarobnjaka.

Pokretanje čarobnjaka za pakovanje i raspakivanje, kao samostalan dio vrši se na sljedeći način:

1) Ako je projekt koji želite zapakovati otvoren, snimite ga i zatvorite *Visual Basic*.

2) Kliknite glavno dugme ***Start***, pa kliknite stavku ***All Programs*** pa ***Microsoft Visual Studio 6.0,*** pa zatim ***Microsoft Visual Studio 6.0 Tools***, pa zatim iz liste opciju ***Package and Deployment Wizard***.

3) U popisu ***Project*** na uvodnom ekranu, odaberite projekt koji želite zapakovati. (Napomena: Možete kliknuti dugme ***Browse*** ako vaš projekt nije na popisu.)

4) Na glavnom ekranu, odaberite jednu od sljedećih opcija:

• Ako želite stvoriti standardno pakovanje, pakovanje za Internet ili datoteku zavisnosti za projekt, kliknite ***Package***.

• Ako želite raspakovati projekt, kliknite ***Deploy***.

• Ako želite pregledati, mijenjati ili obrisati skripte, kliknite ***Manage Scripts***.

5) Nastavite kroz ekrane čarobnjaka.

Možete slobodno distribuirati svaku aplikaciju ili sastavni dio koji ste stvorili sa *Visual Basicom*. Uz izvršne (*.exe*) datoteke, vaša aplikacija može zahtijevati druge datoteke, kao što su dinamičke biblioteke (*DLL*), *ActiveX* kontrole (*.ocx* datoteke) ili slike (*.bmp, .jpg, .ico* ... datoteke).

Nezavisno o tipu pakovanja koji stvarate ili alatu koji upotrebljavate kako bi ga stvorili, postoje neki koraci koji moraju biti preduzeti.

1) **Odredite tip paketa koji želite stvoriti.** Možete stvoriti standardni paket za aplikacije temeljene na *Windows*-ima koje će biti distribuirane na disketama, CD-ima ili putem mreže. Umjesto toga, možete stvoriti Internet paket za aplikacije koje će biti distribuirane putem *Web*-a. Možete također odabrati stvaranje samo datoteke zavisnosti.

2) **Odredite datoteke koje trebate distribuirati.** Čarobnjak mora odrediti projektne datoteke i datoteke zavisnosti za vašu aplikaciju prije nego što može stvoriti paket. Projektne datoteke su datoteke koje su uključene u sam projekt – na primjer, *.vbp* datoteka i njezin sadržaj. Datoteke zavisnosti su datoteke ili sastavni dijelovi izvođenja, koje vaša aplikacija zahtijeva za izvođenje. Informacije o

zavisnosti su spremljene u datoteci *VB6dep.ini*, ili u raznim *.dep* datotekama, koje odgovaraju sastavnim dijelovima vašeg projekta.

3) **Odredite gdje na računaru korisnika instalirati datoteke.** Programske datoteke i datoteke podešavanja se obično instaliraju u poddirektorij direktorija *Program Files*, dok se sistemske datoteke i datoteke zavisnosti obično instaliraju u direktorije *\Windows\System* ili *\Winnt\System32*. Vaš program podešavanja to mora uzeti u obzir i odrediti gdje instalirati svaku datoteku.

4) **Stvorite svoj paket.** Čarobnjak stvara paket i program podešavanja (*setup1.exe*) za njega, ukazujući na sve potrebne datoteke. Krajnji rezultat ovog koraka je jedna ili više *.cab* datoteka i sve potrebne datoteke podešavanja.

5) **Raspakujte svoj paket.** Postupak raspakivanja sadrži stvaranje vašeg medija distribucije i kopiranje svih potrebnih datoteka na mjesto gdje mu korisnici mogu pristupiti.

Postoji nekoliko datoteka koje su uvijek dio vaših standardnih paketa. Tu su uključeni:

• **Datoteka *setup.exe*.** Ova datoteka djeluje kao izvršna prije instalacije. Datoteka ***setup.exe*** je prva stvar koja se izvodi na uređaju korisnika u postupku instaliranja i izvodi potrebne obrade, koje se moraju pojaviti prije početka glavne instalacije.

• **Datoteka *setup1.exe*.** Ova datoteka djeluje kao glavni program podešavanja za vašu aplikaciju.

Kad korisnik instalira vašu aplikaciju, program podešavanja kopira uslužni dodatak *St6unst.exe* za uklanjanje aplikacije u direktorije *\Windows* ili *\Winnt*. Svaki put kad upotrijebite *Visual Basic* program podešavanja za instalisanje aplikacije, stvara se evidencijska datoteka za uklanjanje aplikacije (***St6unst.log***) u direktorijumu u kome je aplikacija instalirana.

Datoteka tipa ***.log*** sadrži unose koji ukazuju na:

- Direktorije koji su stvoreni tokom instalacije.
- Instalirane datoteke i njihove položaje.

Ovaj popis sadrži sve datoteke u programu podešavanja, čak i ako neke datoteke nisu instalisane na korisnikov računar, jer je već postojala novija verzija iste datoteke. Evidencijska datoteka označava je li datoteka djeljiva te ako jeste, je li zamijenila postojeću datoteku.

1.3.6. 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 dati 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[5] 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 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.

1.3.7. 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 proizvodnji 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.

---

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

Čak i softveri koji služe za rješavanje nekih matematičkih problema, koji imaju izuzetnu vrijednost i mogu se koristiti sa visokim stepenom pouzdanosti daju ovakve izjave. Ipak, 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 njihovi sastavni djelovi), 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 uputno 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.

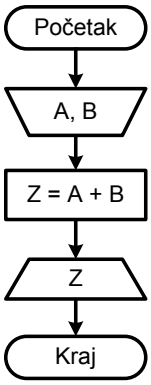
Lako je uvidjeti da ne postoji nikakva čvrsta definicija i ujednačeni pogled na to nakon koliko modifikacija treba da se pojavi nova verzija programa. Tako da označavanje brojeva modifikacija i verzija u priličnoj mjeri zavisi od mjere, ukusa i tradicije koja je svojstvena nekom proizvođaču programa. 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

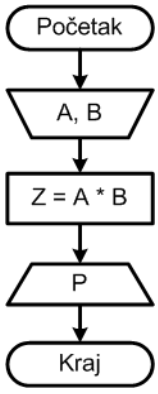
U ovom poglavlju su dati zadaci, za koje je potrebno napraviti grafički algoritam. Za većinu zataka su predloženi grafički algoritmi, koji predstavljaju moguća rješenja.

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.



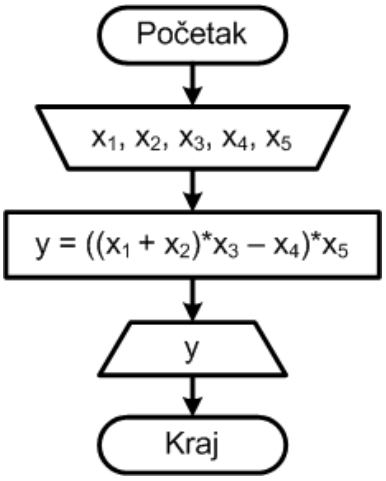
Slika 1.13. Zadatak 1.2.

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.

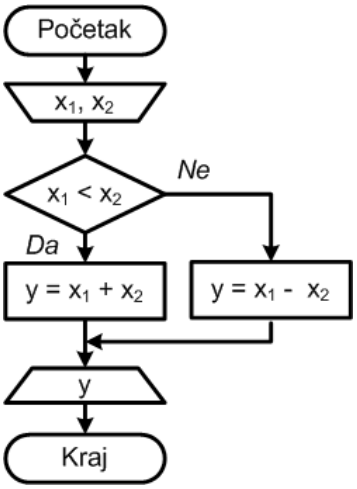
Zadatak 1.4.
Sastaviti algoritam za izračunavanje vrijednosti ***y*** po formuli

$y = x_1 + x_2$ *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.

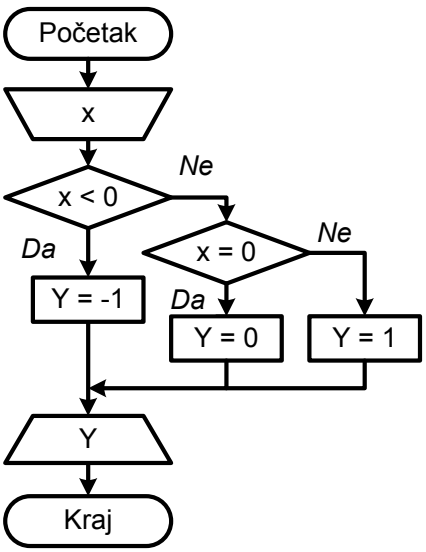
Zadatak 1.5.

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

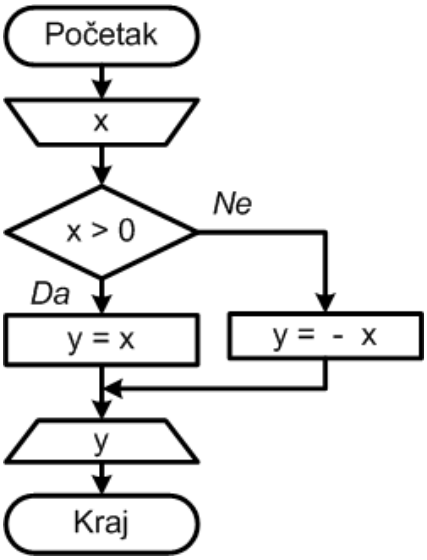
$y = -1$ ako je $x < 0$
$y = 0$ ako je $x = 0$
$y = 1$ ako je $x > 0$.

Rješenje:

Rješenje ovog algoritma dato je na slici 1.16.



Slika 1.16. Zadatak 1.5.

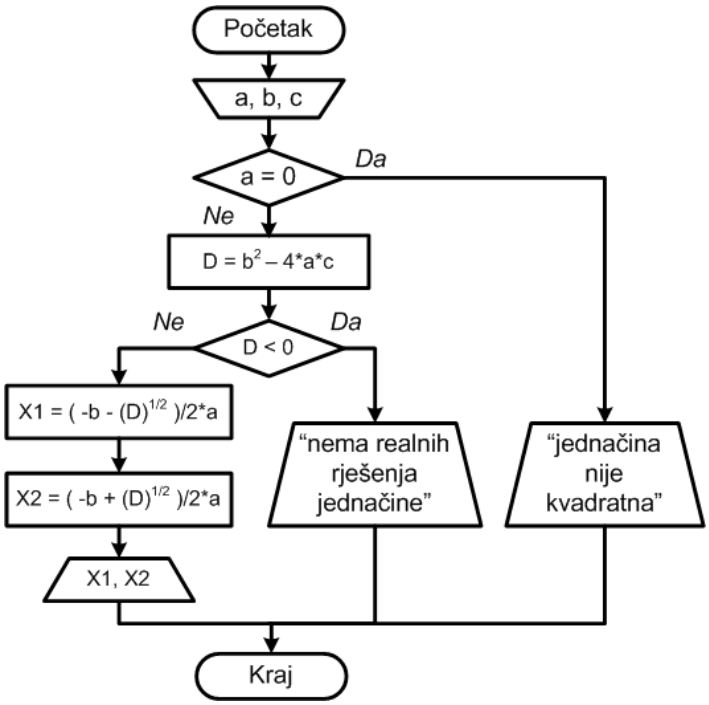


Slika 1.17. Zadatak 1.6.

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

Zadatak 1.7.
Sastaviti algoritam za izračunavanje funkcije $\boldsymbol{y} = || x | - 1|$.
Rješenje 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, 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 veća ili jednaka od nule, onda kvadratna jednačina ima dva realna rješenja X1 i X2.

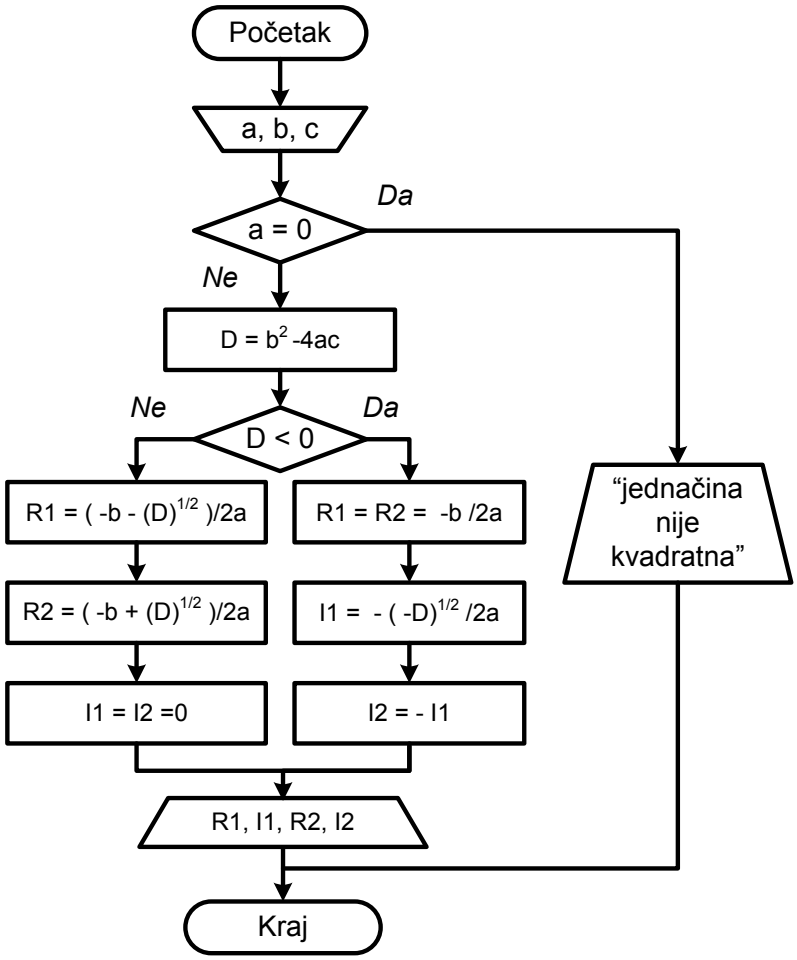


Slika 1.18. Zadatak 1.8.

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.



Slika 1.19. Zadatak 1.9.

Zadatak 1.10.

Sastaviti algoritam za izračunavanje kvadratnog korena $y = \sqrt{x}$ po Njutnovoj iterativnoj formuli $x_{i+1} = (x_i + x / x_i)/2$, $i = 0, 1, 2, ...$
gde je $x_0 = (x + 1)/2$. Proces računanja prekinuti kada se dostigne zadata tačnost *e,* tako da je $x_i - x_{i+1} < e$.

Rješenje:

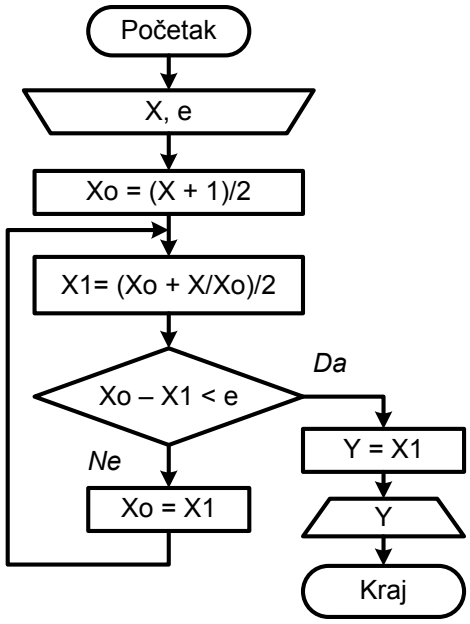
Rješenje ovog algoritma dato je na slici 1.20.

Zadatak 1.11.

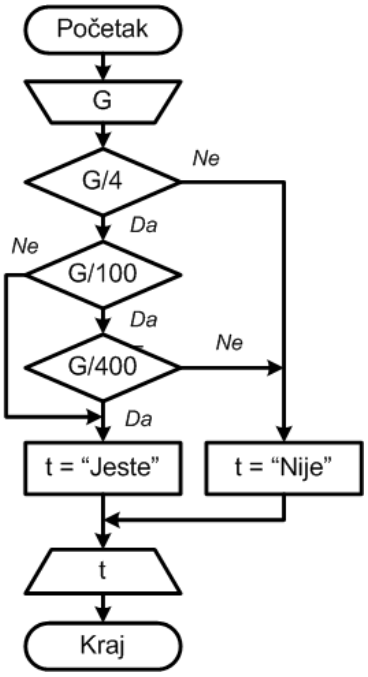
Sastaviti algoritam za određivanje prestupnosti godine (godina nije prestupna ako nije djeljiva sa 4 ili je djeljiva sa 100, a nije sa 400).

Rješenje:

Rješenje ovog algoritma dato je na slici 1.21.



Slika 1.20. Zadatak 1.10.



Slika 1.21. Zadatak 1.11.

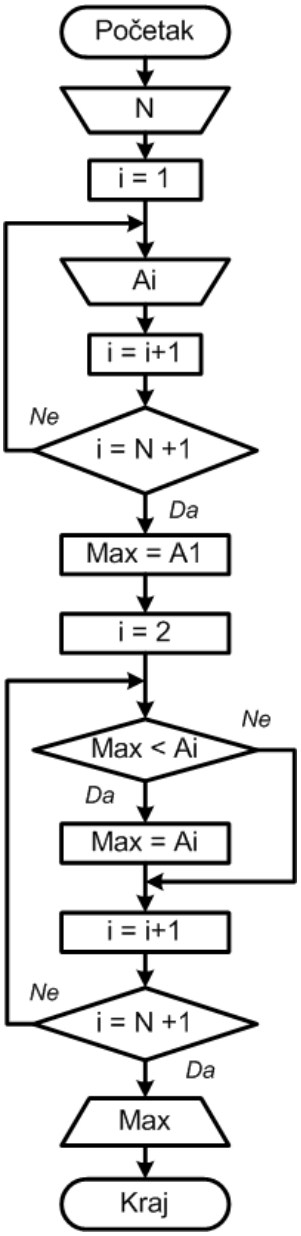
Zadatak 1.12.

Sastaviti algoritam za unos niza A_i koji ima N elemenata, $i = 1...N$. Zatim pronaći najveću vrednosti (Max) unesenog niza A_i.

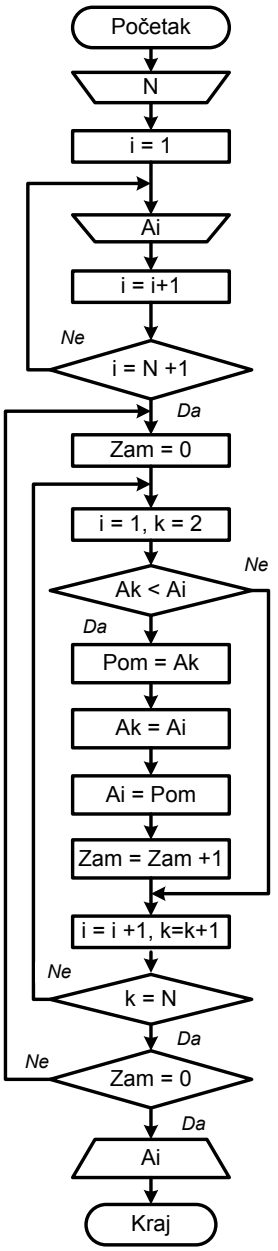
Rješenje:

Rješenje ovog algoritma dato je na slici 1.22. Prvo se preko petlje unese N članova niza. Zatim se prvi član niza proglasi da ima najveću vrijednost (Max = A1). Zatim

se uzima drugi član niza i poredi sa najvećom vrijednošću. Ako drugi član niza ima veću vrijednost od maksimalne, onda se on proglašava za maksimalnu vrijednost. Ovaj postupak poređenja se ponavlja u petlji, dok se ne dođe do kraja niza.



Slika 1.25. Zadatak 1.12.



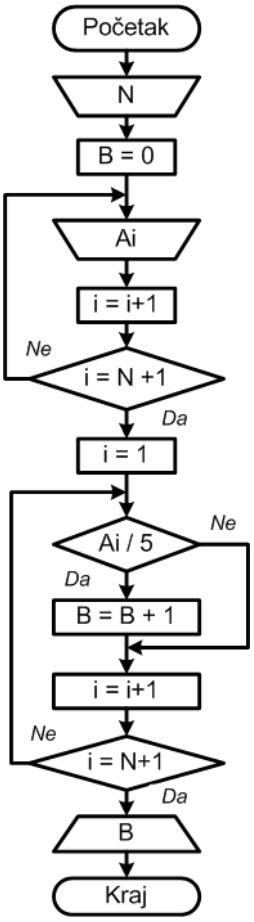
Slika 1.26. Zadatak 1.13.

Zadatak 1.13.
Sastaviti algoritam za unos niza A_i koji ima N elemenata, $i = 1...N$. 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. Prvo se ulazi u spoljnu petlju. Promjenjivoj (*Zam*), koja služi za brojanje zamjena mjesta u nizu, se 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 promjenjive *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 promjenjiva *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 promjenjivoj *Zam* ponovo dodjeli vrijednost nula. Tada se postupak poređena susjednih članova niza ponavlja ponovo od prvog člana niza u unutrašnjoj petlji. Ako promjenjiva *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.

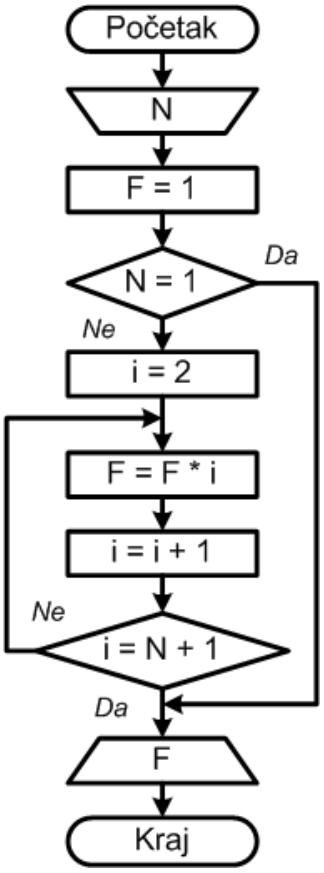
Zadatak 1.14.
Sastaviti algoritam za unos niza A_i koji ima N elemenata, $i = 1...N$. Zatim sastaviti algoritam za ispisivanje niza u nerastućem (opadajućem) poretku.
Rješenje analogno prethodnom zadatku.

Zadatak 1.15.
Sastaviti algoritam za pretraživanje niza A_i koji ima N elemenata, $i = 1...N$ i ispisivanje 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 cijeli broj N, koji je veći od nule.
F = 1*2*3* ... *N
Rješenje:
Rješenje ovog algoritma dato je na slici 1.25.



Slika 1.24. Zadatak 1.15.



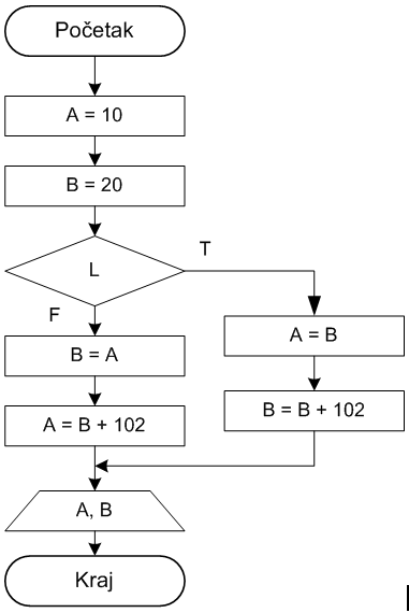
Slika 1.25. Zadatak 1.16.

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:

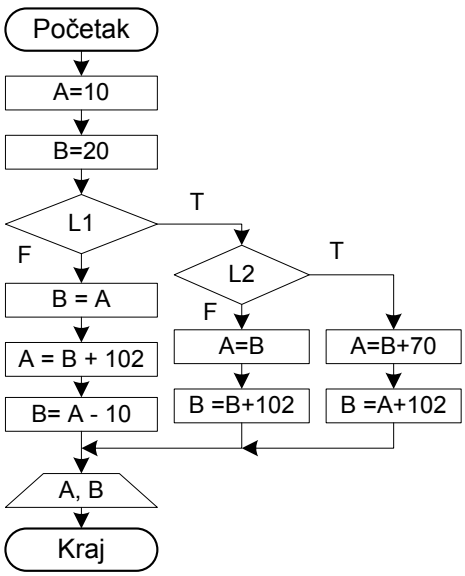
1) Ciklus A= 10, B= 20, Uslov zadovoljen (T), A= 20, B= 122
Na kraju: A = 20, B= 122

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

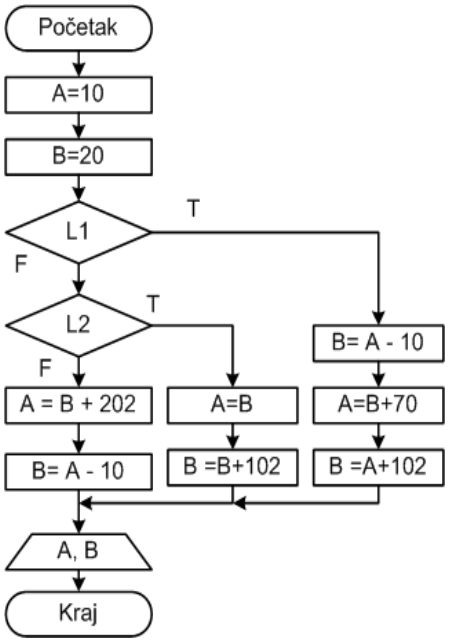
Zadatak 1.23.

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?

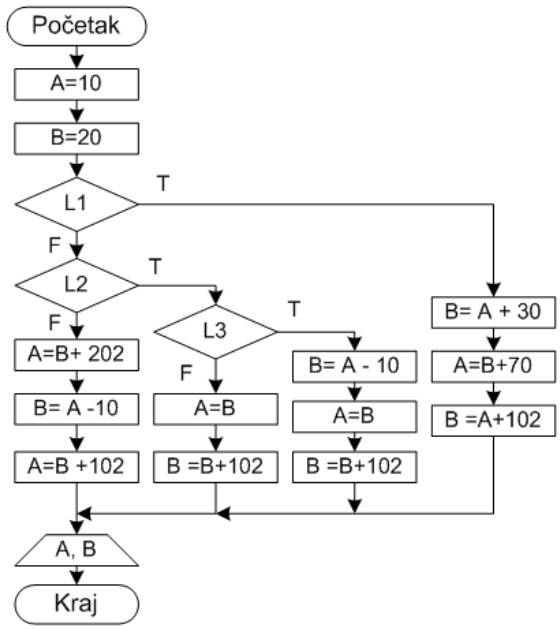
Rješenje:

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?

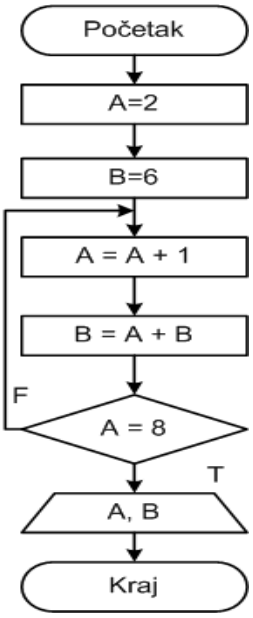
Rješenje:

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

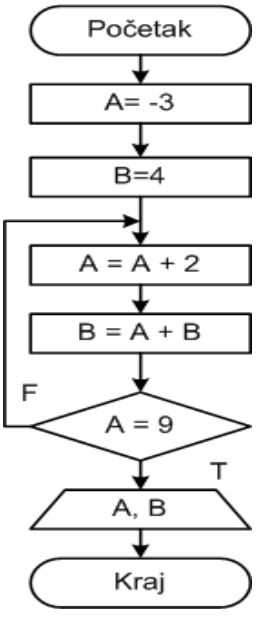
Zadatak 1.29.
Koliko ciklusa će se u algoritmu prikazanom na slici 1.30. izvršiti blok (B = A + B) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?
Rješenje:

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 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:

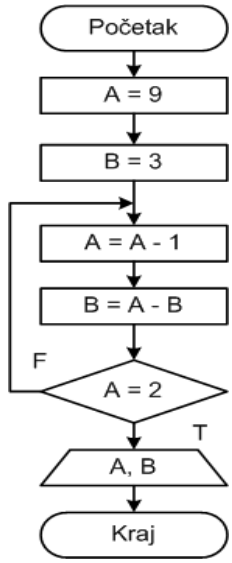
1) Ciklus A= -1, B= 3, Uslov nije zadovoljen (F)
2) Ciklus A= 1, B= 4, Uslov nije zadovoljen (F)
3) Ciklus A= 3, B= 7, Uslov nije zadovoljen (F)
4) Ciklus A= 5, B= 12, Uslov nije zadovoljen (F)
5) Ciklus A= 7, B= 19, Uslov nije zadovoljen (F)
6) Ciklus A= 9, B= 28, Uslov zadovoljen (T)

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

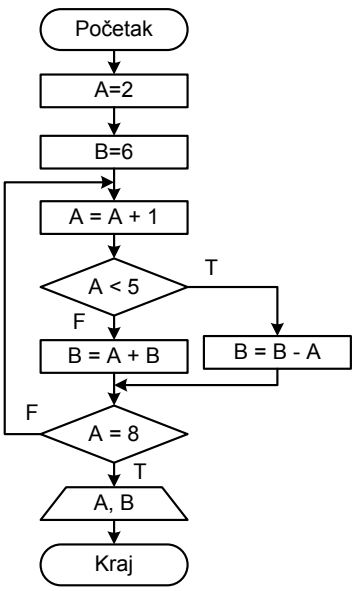
Zadatak 1.31.
Koliko ciklusa će se u algoritmu prikazanom na slici 1.32. izvršiti blok (B = A - B) i koje vrijednosti će na kraju poprimiti izlazi algoritma A i B?
Rješenje:

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 1.30.



Slika 1.33. Zadatak 1.31.

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:

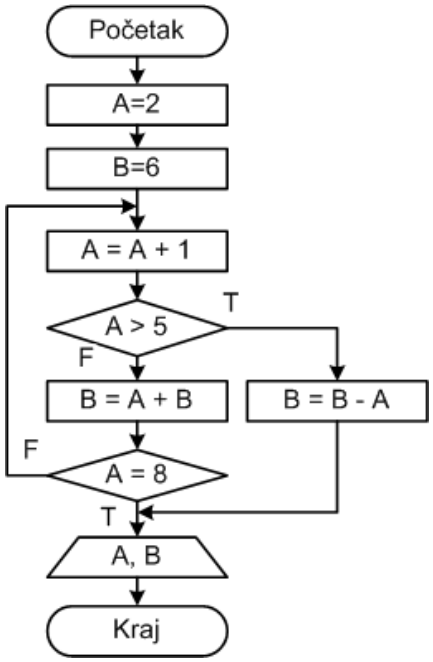
1) Ciklus A= 3, Uslov1 je zadovoljen (T), B= 3, Uslov2 nije zadovoljen (F)
2) Ciklus A= 4, Uslov1 je zadovoljen (T), B= -1, Uslov2 nije zadovoljen (F)
3) Ciklus A= 5, Uslov1 nije zadovoljen (F), B= 4, Uslov2 nije zadovoljen (F)
4) Ciklus A= 6, Uslov1 nije zadovoljen (F), B= 10, Uslov2 nije zadovoljen (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 *Visuel Basic*-u da realizujete zadnji algoritam. Vidjećete da vam to neće poći za rukom pomoću jedne *IF THEN ELSE* strukture i jedne *WHILE* petlje. To govori da zadnji algoritam nije struktuiran.
Za prikazane ulazne podatke uslov2 se neće nikada ostvariti. Za vježbu modifikovati uslove u postojećem zadatku, kako bi postigli da uslov 2 nekada bude zadovoljen.

2. UVOD U PROGRAMSKI JEZIK *VISUAL BASIC*

Visual Basic je već dugo jedan od najpopularnijih objektno orjentisanih 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 orjentisanih 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 (*Code Window*). U ovom prozoru pišemo kod programa, koji želimo da se izvrši nad postavljenim objektima. Na slici 2.15. 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 *Visual Basic*-a, koja je takođe napravila revoluciju u programiranju. *Activex* kontrole su omogućile direktno povezivanje

programa napisanog u *Visual Basic*-u 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, slično kao i pomoću programskog jezika *Java*.

Visual Basic 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 orjentisani programski jezik ima više prednosti, koje ga izdvajaju od ostalih programskih jezika. Osnovne prednosti[6] upotrebe programskog jezika *Visual Basic* su:

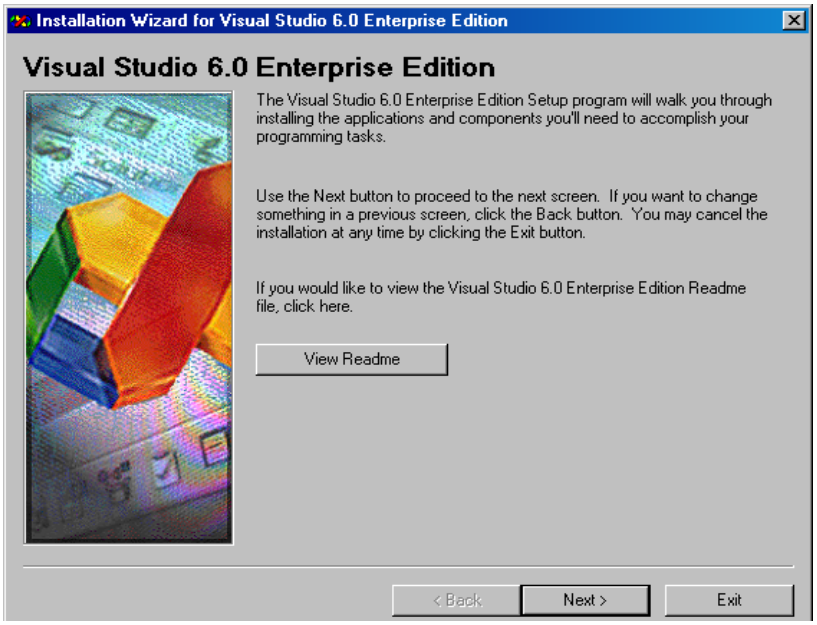
1) Jednostavnost korišćenja je jedna od osobina, koja ga izdvaja od ostalih programskih jezika.
2) Lako učenje preko velikog broja urađenih primjera programa, koji se nalaze u *Help*-u ovog programa.
3) Veliki broj gotovih i raznolikih objekata za upravljanje.
4) Funkcionalan i raznolik grafički interfejs 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 BASIC* PROGRAMA

Za instaliranje programskog jezika *Visual Basic 6.0* postoje tri instalaciona diska. Pored glavnog instalacionog diska postoje još dva diska, na kojima se nalaze prateće biblioteke za ovaj program. Ove biblioteke je takođe poželjno da se instaliraju, jer se na njima nalazi pomoćni dokumenti (*Help*), u kojima su detaljno

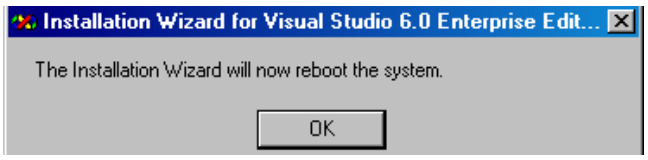
[6] Dr Tihomir Latinović, *Osnove programiranja (Visual Basic)*, Biblioteka Informacione tehnologije, Banja Luka 2007, str. 13

opisane sve komande, funkcije i procedure koje posjeduje *Visual Basic*. Za većinu navedenih komandi su dati i praktični primjeri korišćenja u kodu programa. Program *Visual Basic 6.0* se može instalirati na računar automatski, kada se ubaci glavni instalacioni disk. Prvo se pojavi ekran prikazan na slici 2.1., na kome se mogu saznati osnovni podaci o ovom programu.

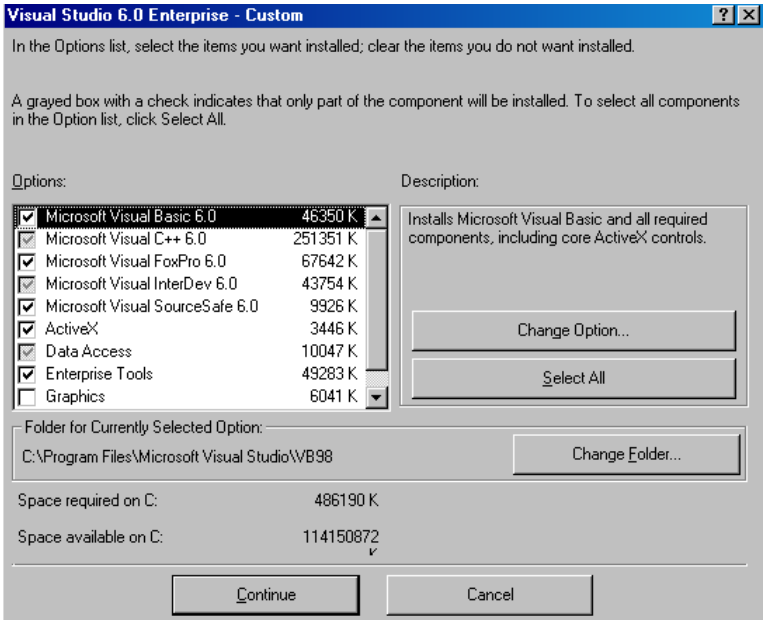


Slika 2.1. Instaliranje Visual Basic 6.0 programa

Klikom na dugme ***Next*** prelazi se na naredni prozor ***End User Licence Agrement***, na kome se korisnik upoznaje sa uslovima pod kojima se ovaj program može koristiti. Da bi se nastavilo dalje sa instaliranjem programa, potrebno je izabrati opciju ***I accept the agreement***, a zatim pritisnuti dugme ***Next***. Nakon toga se pojavljuje prozor na kome se unosi ime korisnika i ime kompanije. Klikom na dugme ***Next*** prelazi se na naredni prozor ***Microsoft Virtual Machine for Java***. Na ovom prozoru korisnik može čekirati opciju ***Update Microsoft Virtual Machine for Java,*** kojom se ako već ne postoji, vrši instaliranje virtuelne mašine za *Java* programski jezik. Ako je ova opcija čekirana, klikom na dugme ***Next*** pojaviće se prozor prikazan na slici 2.2., a zatim klikom na dugme ***OK*** dolazi do resetovanja računara.



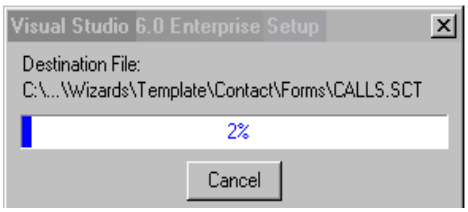
Slika 2.2. Potvrda resetovanja računara



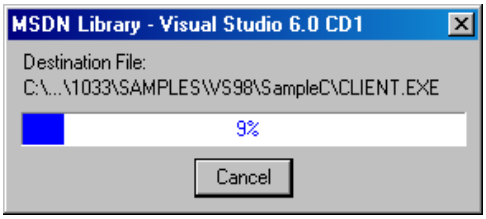
Slika 2.3. Izbor opcija koje se instaliraju

Ako ova opcija nije čekirana ili ako se računar resetovao, pojaviće se prozor ***Visual Studio 6.0 Enteprise Edition.*** Na ovom prozoru potrebno je izabrati jednu od tri ponuđene opcije: *Costum, Product* ili *Server Applications*. Ako se program instalira na personalnom računaru potrebno je izabrati opciju ***Costum***. Klikom na dugme ***Next*** prelazi se na prozor ***Choose Common Install Folder***. Na ovom prozoru se bira direktorijum na kome će program biti instaliran. Inicijalno se pojavljuje putanja **"C:\Program Files\Microsoft Visual Studio\Common"**. Ova putanja se može promjeniti klikom na dugme ***Browse***. Klikom na dugme ***Next*** prelazi se na prozor ***Visual Studio 6.0 Enteprise Setup***, za početak instaliranja programa na hard disk računara. Izborom dugmeta ***Continue*** na ovom prozoru, nastavlja se sa postupkom instaliranja programa i pojavljuje se prozor prikazan na slici 2.3. Izborom dugmeta ***Exit Setup***, odustaje se od daljeg postupka instaliranja programa.

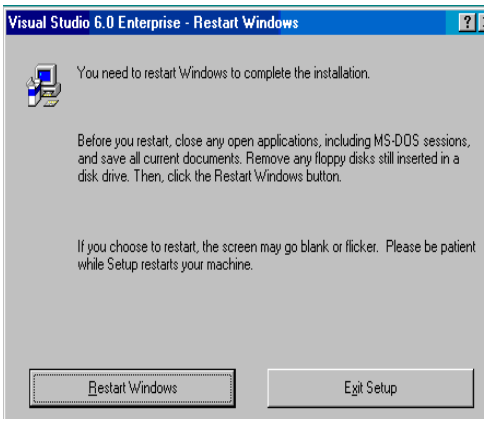
Na prozoru prikazanom na slici 2.3. biraju se opcije programskog jezika *Visual Basic 6.0,* koje se žele instalirati. Ako na računaru imate dovoljno memorije poželjno je čekirati sve ponuđene opcije, jer neznate kada vam neka opcija može zatrebati prilikom programiranja. Izborom dugmeta ***Continue*** na ovom prozoru, prelazi se na prozor ***Setup Environment Variables***. Na ovom prozoru se može čekirati opcija ***Register Environment Variables***, koja omogućava pokretanje *Visual C++* pomoći pri programiranju. Klikom na dugme ***OK*** pojavljuje se prozor prikazan na slici 2.4. na kome se vidi tok instaliranja glavnog programa.



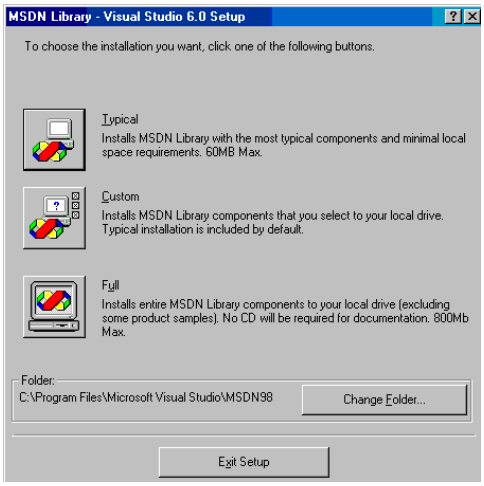
Slika 2.4. Instaliranje glavnog programa



Slika 2.5. Instaliranje biblioteka



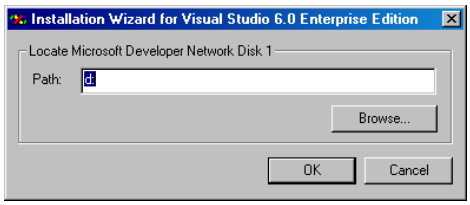
Slika 2.6. Restart nakon instaliranja



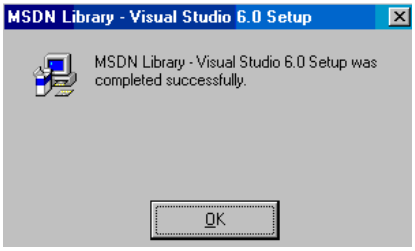
Slika 2.7. Izor instaliranja biblioteka

Kada se završi postupak instaliranja svih fajlova glavnog programa, pojavljuje se prozor prikazan na slici 2.6. Na ovom prozoru je potrebno pritisnuti dugme ***Restart Windows***, kako bi se omogućio restart računara. Kada se računar ponovo pokrene pojavljuje se prozor ***Install MSDN***, na kome je potrebno čekirati opciju *Install MSDN*, da bi se omogućilo snimanje pratećih biblioteka za programski jezik *Visual Basic 6.0.* Klikom na dugme ***Next*** prelazi se na naredni prozor, prikazan na slici 2.8., koji pokazuje da treba ubaciti novi instalacioni disk 1. Kada se obaci traženi disk, pojavljuje se prozor ***MSDN Library - Visual Studio 6.0 Setup***, koji nas obavještava da počinje proces instaliranja pomoćnih biblioteka na disk računara. Za nastavak instaliranja potrebno je pritisnuti dugme ***Continue***. Nakon toga se pojavljuje prozor ***Licence Agreement***, na kome se korisnik upoznaje sa uslovima pod kojima se ove biblioteke mogu koristiti. Da bi se nastavilo dalje sa instaliranjem biblioteka, potrebno je pritisnuti dugme ***I Agree***. Nakon toga se pojavljuje prozor prikazan na slici 2.7., na kome se nude tri nivoa instaliranja biblioteka: *Typical* (najčešće korišćene), *Custom* (pojedinačan izbor) ili *Full* (sve raspoložive). Preporučujemo da se izabere opcija *Full,* koja omogućuje korišćenje svih biblioteka koje posjeduje programski jezik *Visual Basic 6.0.* Nakon izbora

jedne od ove tri ponuđene opcije, otpočine proces instaliranja ovih biblioteka na disk računara, što je prikazano na slici 2.8. Kada je završeno instaliranje svih fajlova sa diska 1, pojaviće se poruka u kojoj se od korisnika se traži da ubaci drugi instalacioni disk sa bibliotekama. Kada se disk 2 ubaci, automatski se nastavlja proces instaliranja biblioteka. Kada je završeno instaliranje svih fajlova sa diska 2, pojaviće se prozor prikazan na slici 2.9., koji obavještava korisnika da su biblioteke uspješno instalirane. Time je završen postupak instaliranja programskog jezika *Visual Basic.*



Slika 2.8. Disk 1 biblioteka



Slika 2.9. Biblioteke instalirane

2.3. POKRETANJE *VISUAL BASIC* PROGRAMA

Aktiviranje programa *Visual Basic 6.0* u *Windows* operativnim sistemima vrši se najlakše kroz Start meni (*Start→Programs→ Microsoft Visual Studio 6.0→ Microsoft Visual Basic 6.0*). Naravno, da ovo nije jedini način da se pokrene program *Visual Basic*. 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.

Pokretanjem *Visual Basic 6.0* programa, 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.

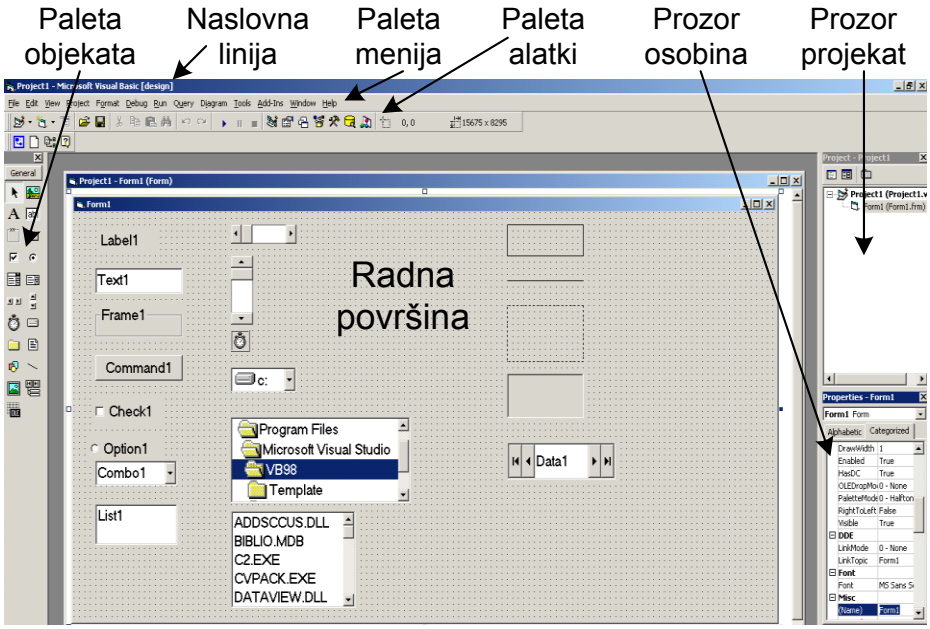
Nakon pokretanja programa pojavljuje se prozor prikazan na slici 2.10. Na njemu se može izabrati jedna od sljedećih opcija:

- ***New*** - za pravljenje potpuno novog projekta. Ponuđeno je više opcija za početak pravljenja programa. Za početnike je najbolje da izaberu opciju *Standard EXE*. Iskusniji programeri mogu da izaberu neko već ranije pripremljeno radno okruženje. Na taj način ubrzavaju proces programiranja.
- ***Existing*** - 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 mjesto (direktorijum) gdje je program snimljen.

Slika 2.10. Početni ekran nakon pokretanja programa Visual Basic

Izborom novog programa pojavljuje se prozor prikazan na slici 2.11. Tu je prikazana radna površina (*Form1*) na koju se mogu dodavati objekti za naš novi program. 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 (*Project Explorer*) se pojavljuje sa desne strane ekrana i u njemu se prikazuju: forme, moduli i klase koje postoje u aktivnom programu. Paleta objekata (*ToolBox*) se obično nalazi na lijevoj strani ekrana. Prikazuje standardne objekte, koji programeru stoje na raspolaganju. Ova paleta se može obagatiti sa dodatnim objektima, što će biti opisano u 5 poglavlju. Ako se greškom zatvori ova paleta, ponovo se postavlja na radni ekran preko ikonice  , koja se nalazi u paleti alatki.



Slika 2.11. Radna površina za pisanje programa

2.4. SIMBOLI U PROGRAMU *VISUAL BASIC*

Visual Basic, kao i svaki drugi programski jezik, ima svoju gramatiku. Osnovne dvije grane gramatike *Visual Basic* 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 *Visual Basic* jezika pomoću kojih se grade složeniji jezički oblici 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) za povezivanje objekata sa njihovim osobinama (npr. Dugme.Enabled = False) i
3) povezivanje imena fajla sa njegovim tipom (npr. Strana1.frm).

Znak za podvlačenje (_) se upotrebljava ako sadržaj komande ne može da stane u jedan programski red. Onda 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 *Visual Basic* jeziku služi za navođenje komentara. Tekst komentara je u kodu obojen zelenom bojom. Primjer:

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

Komentar može da se napiše i sa komandom *Rem*. Primjer:

```
Rem Kraj programa
```

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

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

Simboli (!) i (**|**) povezuju objekte sa kolekcijom, odnosno u listi filtera.

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"
```

Tačka-zapeta (;) omogućava da se dva podatka iskažu u razmaku od jednog karaktera prilikom korišćenja komande print.

Zapeta (,) koristi se i za razdvajanje argumenata kod funkcija i procedura. Ovaj simbol se nesmije koristiti za pisanje realnih brojeva, koji se isključivo pišu korištenjem simbola tačka (.). Primjer:

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

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

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

Male zagrade ( ) služe za grupisanje više izraza u jednu cjelinu i za postavljanje argumenata funkcija i procedura. 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 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 (*) posjeduje značenje operatora za množenje i znaka za popunu skrivenog teksta u *TextBox*-u (*PasswordChar*).

Operator (^) 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 u naredbama za dijeljenje *(a/b)*, kada se kao rezultat dobija realni broj. Odnosno za cjelobrojno dijeljenje *(a\b),* kada kao rezultat dobija cijeli broj. 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 poređivanja dva izraza i redovno se koriste u naredbama selekcije i iteracije.

Riječi u *Visual Basic-u,* 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) i 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 PROGRAMU *VISUAL BASIC*

Visual Basic posjeduje veliki broj gotovih objekata, koji stoje programeru na raspolaganju. U standardnoj paleti alatki 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.
- *CommandButton* komandno dugme.
- *CheckBox* i *OptionButton* za izbor opcija.
- *Frame* pravljenje posebnog okvira sa objektima na formi.
- *ListBox* i *ComboBox* za prikaz vrijednosti u padajućim listama.
- Klizne trake *HScrollBar* i *VScrollBar* za izbor vrijednosti preko kliznih traka.
- *Timer* za korišćenje sistemskog datuma, vremena i vremenska podešavanja.
- *DriveListBox* za prikaz i izbor drajvera na računaru.
- *DirectoryListBox* za prikaz i izbor direktorijuma na aktivnom drajvu.
- *FileListBox* za prikaz i izbor fajlova na aktivnom direktorijumu.
- *Line* i *Shape* za crtanje po radnoj površini.
- *Image* i *Picturebox* za prikaz slika raznih formata.
- *Data* za povezivanje sa bazama podataka.
- OLE za povezivanje sa drugim programima.

Svi koji žele da pronađu dodatne objekte, mogu ih pronaći u *Help*-u *Visual Basica*. Da bi se ovi dodatni objekti mogli koristiti u programiranju, potrebno ih je prvo dodati u paletu objekata (*ToolBox*). Dodavanje dodatnih objekata se vrši preko glavnog menija u kome se izabere ***Project***, nakon čega se otvara list opcija, od kojih treba odabrati opciju ***Components***, a zatim čekirati željene objekte. Objekti se postavljaju na radnu površinu (formu), tako što se prvo klikne mišem na odgovarajuću ikonicu objekta u paleti objekata. Potom, uz zadržavanje pritisnutog tastera, u odgovarajućoj veličini i na željenoj poziciji na formi razvije končanica oblika paravougaonika, koja sadrži traženi objekat. Nakon otpuštanja tastera ostaje objekat okružen linijom pravougaonika, koji na tjemenima i sredinama stranica ima označene tačke. Dovođenjem kursora miša na jednu od tih tačaka 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 tačke 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 *Visual Basic*-u 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 dvije opcije: *Palette* i *System*. One omogućavaju da se sa palete izvrši željeni izbor boje. Kada se izabere željena boja prikazuje se njena vrijednost u heksadecimalnom kodu. Heksadecimalni kod boje je za većinu programera nerazumljiv broj i koristi se uglavnom kod dodjeljivanja boja preko koda na sljedeći način:

```
TxtRezultatZ.BackColor = &H000000FF&
```

Ovo je primjer koda gdje je boja pozadine *TextBox*-a pod imenom TxtRezultatZ promjenjena u crvenu 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 i 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:

0 - *Left Justify* (poravnanje u lijevo),
1 - *Right Justify* (poravnanje u desno),
2 - *Center* (centriranje teksta) i
3 - *Justify* (obostrano poravnanje).

Osobina ***Appearance*** posjeduje dvije vrijednosti: 0 - *Flat* i 1 - 3D, koje omogućavaju prikaz objekta u jednoj dimenziji ili trodimenzionalno.

AutoRedraw je osobina forme, a mogu joj se dodijeliti dvije vrijednosti: *False* i *True.* Dodjelom vrijednosti *True* omogućava se da *Visual Basic* automatski ispisuje rezultate dobijene pomoću naredbe *Print,* kada se ponovo prikaže forma. U protivnom, ako se dodijeli osobina *False,* prilikom novog prikazivanja forme rezultati bi izostali. Ostali objekti prikazuju se automatski bez obzira na izabranu vrijednost osobine.

Klikom na osobinu ***BackColor****,* na desnoj strani ove osobine, pojavljuju se tri tačke (...). Klikom na njih 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:

0 - *None* (bez okvira),
1 - *FixedSingle* (objekat uokviren jednom linijom),
2 - *Sizable* (linija je istaknuta i deblja) i
3 - *FixedDialog* (može se izabrati željeni stil linije).

Pomoću osobine ***BorderColor*** dodjeljuje se boja graničnoj liniji objekta.

Pored osobine ***Caption*** upisuju se tekst, koji će pisati na dotičnom objektu. Ovu osobinu treba razlikovati od imena objekta.

Osobina ***Connect*** dodjeljuje se ime formata baze podataka kojima će se pristupati (npr. *Acces, Excel, dBase* i sl.) radi obrade podataka.

Aktiviranjem osobine ***DatabaseName*** otvara se okvir za dijalog u kojem treba izabrati naziv fajla (baze podataka) i njegovu putanju, kako bi mu se moglo pristupiti.

Izborom osobine ***DataField*** pojavljuje se lista kolona izabrane tabele baze podataka. Klikom se bira kolona, čiji sadržaj treba prikazati u predviđenon objektu.

DataFormat omogućava da se pomoću okvira za dijalog, podatku dodijeli jedna od ponuđenih vrijednosti: *General, Number, Currency, Date, Time, Percentage, Boolean, Scientific* ili *Picture.*

Aktiviranjem osobine ***DataSource*** pojavljuje se strelica. Klikom na strelicu dobija se padajuća lista sa spiskom objekata *Data,* preko kojih je ostvarena veza sa konkretnom tabelom u bazi podataka, od kojih treba izabrati odgovarajuću.

Enabled je važna osobina brojnih objekata, 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.

Osobina ***FillColor*** odnosi se, obično, na određivanje boje kojom treba da bude popunjen objekat dobijen na opisani način (preko palete ili sistema boja).

FillStyle je osobina pomoću koje se određuje uzorak za popunjavanje objekta. Uobičajene vrijednosti ove osobine su:

0 - *Solid* (puna popunjenost),
1 - *Transparent* (neznatno popunjen),
2 - *HorizontalLine* (horizontalne linije),
3 -*VerticalLine* (vertikalne linije),
4 - *UpwardDiagonal* (dijagonale odozdo na dole i slijeva nadesno),
5 - *Down Diagonal* (dijagonale odozdo na gore),
6 - *Cross* (vodoravno - vertikalna mreža) i
7 - *DiagonalCross* (koso - dijagonalna mreža).

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 ***Left*** određuje se udaljenost objekta u pikselima (*pixels)* od lijeve ivice ekrana.

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

Osobina ***Picture*** određuje tip slike, na primjer, *bmp, gif, jpg, wmf, emf, ico* i dr.

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

Aktiviranjem osobine ***RecordSource*** otvara se okvir za dijalog, u kojem treba izabrati naziv tabele. Naziv tabele se može birati samo ako je u predhodno izabrana bazi podataka preko osobine *DatabaseName*.

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

0 - *None* (bez klizača),
1 - *Horizontal,*
2 - *Vertical* i
3 - *Both* (obostrani).

Osobina ***Stretch*** takođe ima dvije vrijednosti: *True* i *False.* Posebno je značajna upotreba vrijednosti *True* čijim se izborom daje zadatak *Visual Basic*-u da

prilagodi originalnu dimenziju importovane slike, prema dimenzijama objekta za prikaz slike na formi.

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:

0 - Manual (ručno postavljanje),
1 - CenterOwner (centriranje prema objektu na formi),
2 - CenterScreen (centriranje prema čitavom ekranu),
3 - WindowsDefoult (pozicioniranje preko čitavog ekrana).

2.6. SKRAĆENICE U PROGRAMU *VISUAL BASIC*

U *Visual Basic*-u programiranje se vrši 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, koje se mogu koristiti u programu. 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.

Skraćenica	**Opis radnje**
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.
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.

Skraćenica	**Opis radnje**
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.*
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

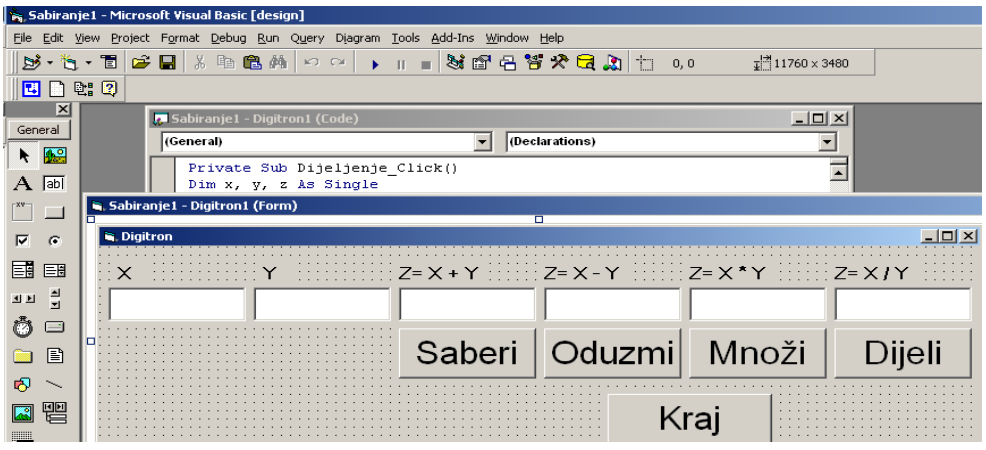
U ovom poglavlju opisani su osnovni koraci za kreiranje programa, kroz izradu jednostavnog primjera. Programski jezik *Visual Basic* se najlakše uči kroz rješavanje praktičnih problema. Kroz ovaj primjer početnici u radu sa ovim programskim jezikom mogu da uoče veliku jednostavnost i lakoću pravljenja 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 za kreiranje 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.
5) Pravljenje izvršne verzije programa (*Make Digitron1.exe*).

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.

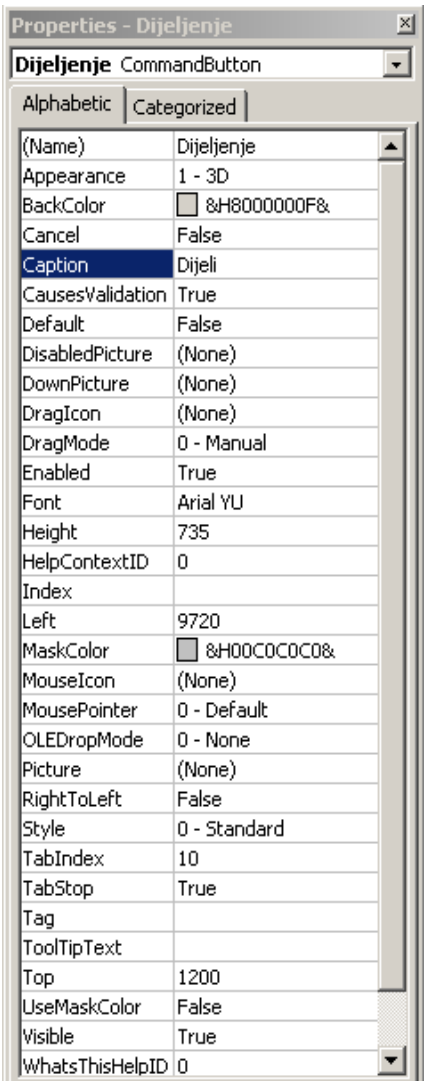


Slika 2.12. Postavljanje potrebnih objekata na radnu površinu

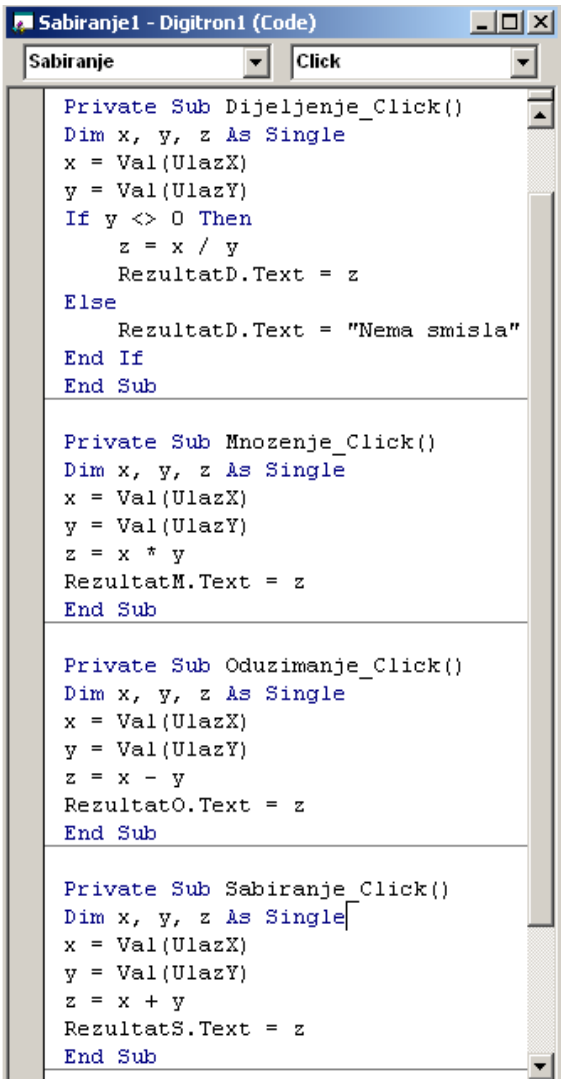
Za pisanje programa u programskom jeziku *Visual Basic*, sa kojim želimo da riješimo postavljeni problem, potrebni su nam sljedeći objekti:

- 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 (*CommandButton*).

Svi ovi objekti se biraju iz palete *ToolBox* i postavljaju na radnu površinu (*Form1*) za pravljenje programa, a prikazani su na slici 2.12.



Slika 2.13. Osobine objekta



Slika 2.14. Kod programa

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 2.13.

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:

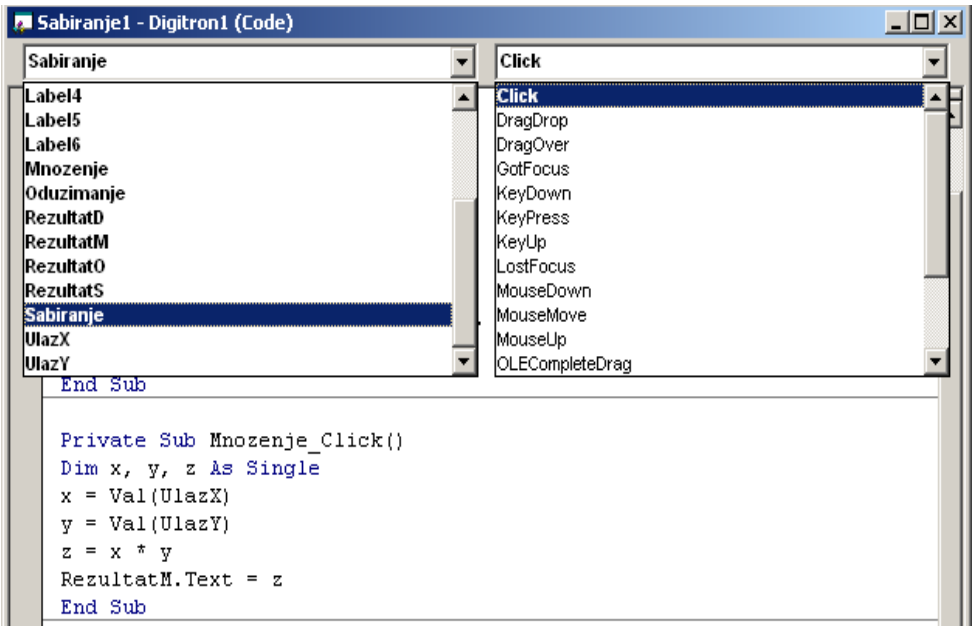
- ime objekta,
- natpis na objektu,
- oblik objekta,
- font, boja i veličina slova,
- boje pojedinih dijelova objekta,
- dimenzije objekta,
- veze sa drugim objektima,
- vidljivost ili nevidljivost objekta,
- dostupnost objekta.

2.7.3. Međusobno povezivanje objekata

Kada su objekti postavljeni na radnu površinu i kada su im postavljene željene osobine, potrebno je izvršiti povezivanje svih 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 radnoj formi i izbor opcije *View Code*. Izgled prozora za pisanje koda programa, koji omogućuje izvršavanje osnovnih matematičkih operacija, prikazan je na slici 2.14. Na slici 2.15. 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 promjenjive realnog tipa. Zatim slijede dvije kodne linije u kojima se iz objekata *TextBox* preuzimaju unesene vrijednosti u promjenjive. 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 prilikom izvršenja programa pojavila neka greška.



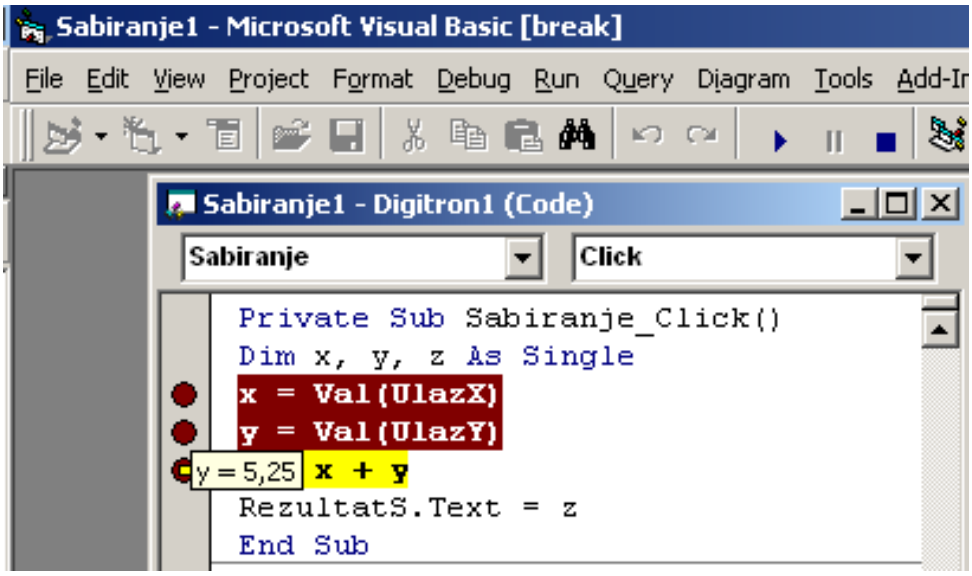
Slika 2.15. Lista postojećih objekata i događaja nad njima

2.7.4. Testiranje programa i pravljenje izvršne verzije programa

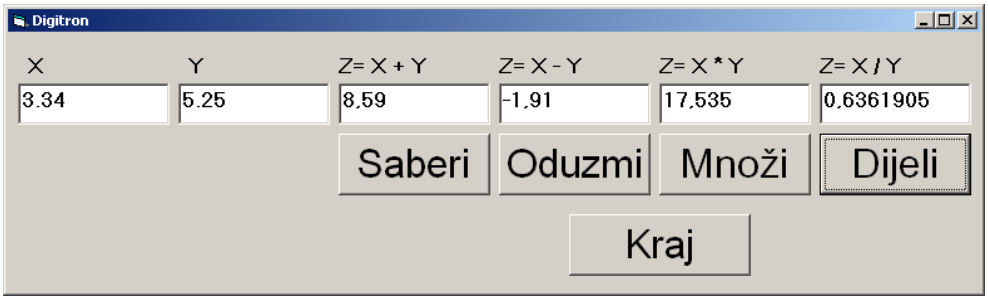
Kada se program napiše, potrebno je prvo uraditi njegovo testiranje. Prilikom testiranja programa otkrivaju se greške u pisanju programa. Greške mogu da budu sintaktičke i logičke. Sintaktičke greške nastaju prilikom kucanja naredbi u kodu programa. Najčešće greške su izostavljanje nekog slova ili da 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. Jedan od načina otklanjanja logičkih grešaka je da se u programu postave kontrolne tačke, kao na slici 2.16. 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. Zatim da se program izvršava postepeno od jedne do druge kontrolne tačke. Pri tome se provjeravaju vrijednosti promjenljivih dolaskom miša na pojedine promjenjive. U primjeru sa slike 2.16. to je promjenjiva **Y** koja ima vrijednost 5.25 (y=5.25). Na ovaj način se i najlakše dolazi do otkrivanja grešaka, a zatim i ispravljanja grešaka u programu.

Kada se program u potpunosti istestira, potrebno je napraviti izvršnu verziju programa preko opcije *Make Digitron1.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"*. Korisnik bi trebao 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 objekata. Sa druge strane pisac programa štiti svoj autorski rad isporučivanjem korisniku samo izvršne verzije programa. Na slici 2.17. prikazan je izgled izvršne verzije ovog programa.



Slika 2.16. Kontrolne tačke u kodu programa



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 promjenjive. Tako promjenjivu možemo shvatiti kao kontejner za vrijednosti, koje mogu da se mijenaju tokom izvršavanja programa. Uvijek treba da se trudite da promjenjivim 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 promjenjivim u *Visual Basic*-u su:

- Ime promjenjive mora činiti jedna riječ.
- Ime promjenjive 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 promjenjive.
- Ime promjenjive ne smije da bude neka od službenih riječi *Visual Basic*-a, kao što su reči ***Then, For, Public*****,** itd.

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

- Ime promjenjive treba da bude sa značenjem koje opisuje promjenjivu, a najbolje je da bude neka imenica.
- Kod imenovanja uobičajeno je označavanje gde je prvo slovo u imenu veliko a sva ostala mala.
- Ime promjenjive 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 izvjesne 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.

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 *Visual Basic*-u dati su u tabeli 3.1.

Tabela 3.1. Tipovi podataka

Red. br.	Naziv tipa	Skraćeni naziv	Vrsta podatka	Vrijednosti
1.	*Boolean*		Logička vrijednost	*True* ili *False* (1 bit)
2.	*Integer*	%	Cijeli brojevi	-32768 do 32767 sa 16 bita
3.	*Long*	&	Cijeli brojevi	-2147483648 do 2147483647 sa 32 bita
4.	*Single*	!	Realni brojevi	±3.40282347E±38 sa 32-bita
5.	*Double*	#	Realni brojevi	±1.7976931348623157E308 sa 64 bita
6.	*Decimal*		Realni brojevi	+/-7.92281625142643375935439503 35 sa 112 bita
7.	*String*	$	Karakteri	1 do 65400 znakova
8.	*Currency*	@	Realni brojevi	±922 337 203 685 477,5808 sa 64 bita
9.	*Date*		Datum i vrijeme	Realni broj sa 64 bita
10.	*Variant*		Svi tipovi	Proizvoljna

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.

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

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, osim za univerzalni tip podataka *variant*. Definisanje tipa promjenjive ima sljedeću sintaksu

Public [Dim] ImePromjenjive As TipPodatka ili

Public [Dim] ImePromjenjive (Indeks) Skraćeni naziv

Obavezna riječ *Public* upotrebljava se kada se promjenjiva definiše u okviru standardnog modula i dostupna je svim procedurama u programu. Ključna riječ *Dim* piše se onda kada se promjenjiva primjenjuje samo u procedurama u kojima je i definisana.

```
Dim A1 As Integer    ili Dim A1%
Public A2 As Double  ili Public A2#
```

3.1. LOGIČKI TIP (*BOOLEAN*)

Logičke promjenjive (eng. *Boolean*) predstavljaju najjednostavnije promjenjive. 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 objektu 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* postoje sljedeće definicije cjelobrojnog tipa podataka:

1) ***Integer*** je 16-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-32768 do 32767).
2) ***Long*** (*long integer*) je 32-bitno predstavljanje, a ovaj tip podataka sadrži pozitivne, ali i negativne cjelobrojne brojeve u rasponu (-2147483648 do 2147483647).

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 promjenjive i broja)
Z = X + Y (sabiranje vrijednosti dvije promjenjive)
X = X + 1 (uvećanje trenutne vrijednosti promjenjive 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 *Visual Basic*-u 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, kao na Primjer:

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

Provjeriti da li se dobije isti rezultat u slijedećim 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* postoje sljedeće definicije realnog tipa podataka:

1) ***Single*** (*single-precision floating-point*) je 32-bitno predstavljanje, a ovaj tip podataka sadrži realne brojeve u rasponu (±3.40282347E±38), sa približno 8 značajnih decimalnih mjesta).
2) ***Double*** (*double-precision floating-point*) je 64-bitno predstavljanje, a ovaj tip podataka sadrži realne brojeve u rasponu (±1.7976931348623157E308), sa približno 16 značajnih decimalnih mjesta).
3) ***Decimal*** je 112-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 Basic* postoji dosta funkcija, koje rade sa ovim tipom podataka. Neke od funkcija koje se koriste za konverziju realnog u cijeli broj su:

1) ***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 vrijednost.
3) ***Fix***(*number*) - uzimanje samo cjelobrojne vrijednosti realnog broja, pri čemu se zaokruživanje radi na manju vrijednost, ali po apsolutnoj vrijednosti za negativne brojeve.

Primjer:
Round([23.0, 23.1, 23.9, 23.5, (-23.5), 24.5]), daje [23, 23, 24, 24, -24, 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, 23.1, 23.9, 23.5, (-23.5), (-24.9)]), daje [23, 23, 23, 23, -24, -25].
Fix([23.0, 23.1, 23.9, 23.5, (-23.5), (-24.9)]), daje [23, 23, 23, 23, -23, -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 *Visual Basic*-u promjenjive ovog tipa podatka mogu imati od 1 do 65400 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 razlikovao od naziva promjenljivih, ulaza ili izlaza iz objekata potrebno je da stoji između apostrofa ("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 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.

U programskom jeziku *Visual Basic* 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]

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

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 veći", "je"), daje 7] i InStr(8, "Mirko je veći", "e"), daje 10]

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

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 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 ("STUDENT", "<@@@@@@@@") ; daje "student".

- > 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 sa 100 i dodavanje znaka %)
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. NOVČANI TIP (*CURRENCY*)

Novčani tip podataka (eng. *Currency*) predstavlja podskup skupa realnih brojeva sa 4 značajna decimalna mjesta. *Currency* je 64-bitno predstavljanje brojeva u rasponu (±922 337 203 685 477,5808). 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.

3.6. 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 1900. 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.

Vremenu 5 sati, 25 minuta i 35 sekundi, na dan 21. 07. 2002. godine pridružuje se broj 37458, 226099537.

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

1) ***now()*** - uzima sistemsko vrijeme sa računara, a na svom izlazu daje realni broj u formatu *Real64*. Dobijeni realni broj predstavlja broj sekundi od 1. januara 1900. godine, do trenutka aktiviranja ove funkcije u programu.
2) ***DateValue(string)*** - koja uzima string i od njega pravi datum. [DateValue("7/21/02 5: 25:35 PM"), daje #7/21/02#]
3) ***TimeValue(string)*** - koja uzima string i od njega pravi vrijeme. [DateValue("7/21/02 5: 25:35 PM"), daje #5:25:35 PM#]
4) ***CDate(broj)*** - od realnog broja pravi datum.

[CDate (37458.22610) = #7/21/02 5:25:35 AM#]

5) ***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.
6) ***Month(***Datum***)*** - daje broj mjeseca u trenutnoj godini. Opseg vrijednosti izlaza ove funkcije je od 1 do 12.
7) ***Year(***Datum***)*** - daje broj godina u trenutnom datumu.
8) ***Hour(***Datum***)*** - izdvaja dio datuma koji sadrži sat.
9) ***Minute(***Datum***)*** - izdvaja dio datuma koji sadrži minute.
10) ***Second(***Datum***)*** - izdvaja dio datuma koji sadrži sekunde.
11) ***DateSerial(d,m,y)*** - formira datum od 3 ulazna parametra. Parametar ***d*** je dan u mjesecu. Parametar ***m*** je mjesec u godini. Parametar ***y*** je godina.
12) ***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.
13) ***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.
```

3.7. NEODREĐENI TIP (*VARIANT*)

Variant je neodređeni tip podataka, koji postoji u programskom jeziku *Visual Basic*. Kada se neka promjenljiva deklariše kao varijant, 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. Razlog leži u činjenici, što ovaj tip promjenljive zauzima najviše memorije od svih ostalih tipova podataka, a to je 128-bitno predstavljanje brojeva.

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

3.8. STATIČKI NIZOVNI TIP (*ARRAY*)

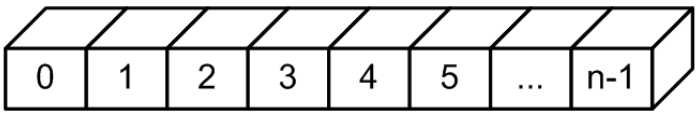
Statički nizovi služe za realizaciju jednoindeksnih i višeindeksnih nizova. Skup ***S*** iz grupe elemenata istog tipa podataka i koji imaju isti smisao, naziva se niz. Elementi niza se u programu obično obrađuju na sličan ili isti način. Elementi niza mogu biti i objekti, koji pripadaju jednoj formi 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.

Nizovi su nastali radi smanjenja broja deklarisanih promjenjivih, 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 formu

```
Public [Dim] ImeNiza (Indeks) As TipPodatka
```

Obavezna riječ *Public* upotrebljava se kada se niz definiše 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 Variant.* 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

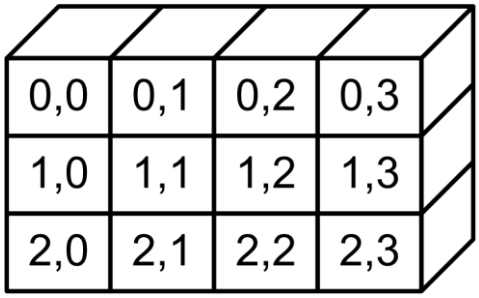
```
Public Promjenljiva1 () As Variant
```

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 tog razloga, 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.

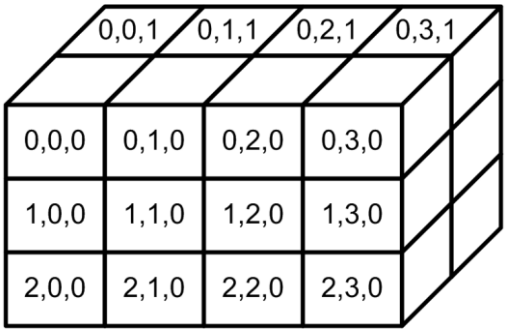
Osnovna prednost u radu sa nizovima sastoji se u efikasnosti izvršavanja programa i u preglednosti izrade programskog koda. Promjenljive deklarisane kao

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



Slika 3.2. Dvodimenzioni niz



Slika 3.3. Trodimenzioni niz

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.

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 ' Promjenjive 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 element.
```

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

```
Dim i, j, k As Integer ' Promjenjive 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.

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* 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 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

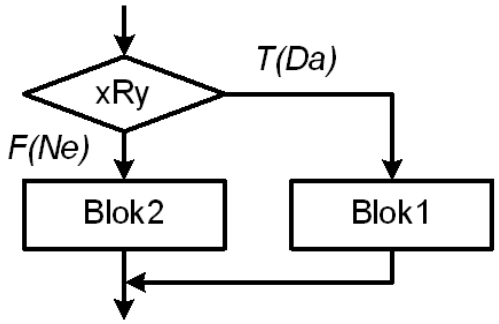
If Then predstavlja struktura grananja, koja se koristi u skoro svim programskim jezicima. Standardni dijagram ove strukte prikazan je na slici 4.1., a struktuirani na slici 4.2. Osnovu ovog grananja čini logički uslov. Ako je ovaj logički uslov zadovoljen (Da-*True*) izvršava se jedna naredba (ili blok 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

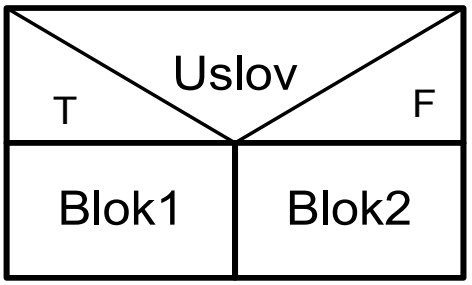
pri čemu je:

- **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 **uslov1**, tačan izvršava se samo jedna naredba 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 moguće je samo onda, ako je imenilac razlomka različit od nule. Slijedi kod programa, koji prokazuje, kako se ovaj problem nedefinisanog razlomka rješava u *Visual Basic*-u pomoću strukture grananja *If Then*.

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

U slučaju da uslov nije zadovoljen (Imenilac = 0) navedena naredba biće ignorisana (preskočena) i počeće izvršavanje sljedećeg programskog reda.

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.
- **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
```

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 i u ovom slučaju se nebi mogla upotrijebiti *If Then* struktura.

```
Brojilac = Text1
Imenilac = Text2
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 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.

```
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
```

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 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 **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 ' naredbe koje se izvršavaju ako nije ispunjen nijedan prethodni uslov
End If

Pri čemu se **Blok1** naredbi izvršava, samo ako je **uslov1** ispunjen. Ako **uslov1** nije ispunjen, tada se provjerava da li je ispunjen **uslov2** i ako je on uspunjen izvršava se **Blok2** naredbi. Ako ni **uslov2** nije ispunjen, tada se provjerava da li je ispunjen **uslov3** i ako je on uspunjen izvršava se **Blok3** naredbi. Međutim, ako nije ispunjen ni jedan od postavljenih uslova u *ElseIf* strukturama, tada se izvršava **Blok4** naredbi koji se nalazi u strukturi *Else*. Broj *ElseIf* blokova (odnosno 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 uslovni izrazi moraju postaviti u rastućem redoslijedu promjenljive, 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 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 promjenjive i kako funkcionišu grananja i petlje.

Jedan 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.
Napraviti uslov grananja pomoću operatora *Or* i *And* sa tri promjenjive.

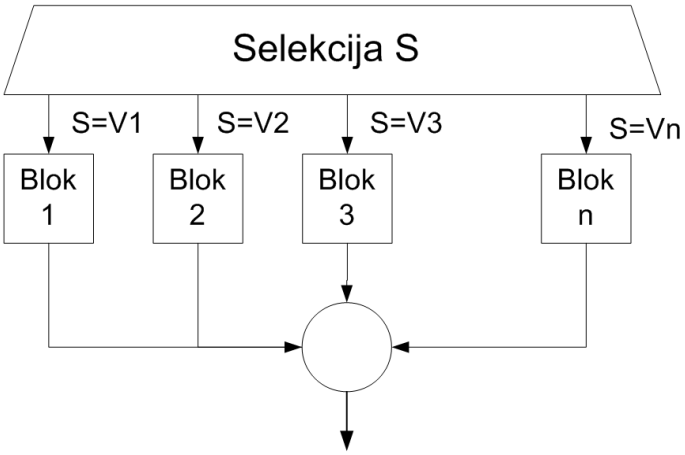
```
If X<5 Or Y=7 Or Z>10 Then
  ' kod kada je X manje od 5 ili Y jednako 7 ili Z veće
  od 10, odnosno kada je zadovoljen bar jedan od uslova
Else
  ' kod kada nije zadovoljen nijedan od tri gornja
uslova
End If

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 nisu zadovoljena sva tri gornja 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 nije zadovoljen jedan od 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 modernu 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 različitim smjerovima 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 računar 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
Case vrijednost2
    Blok2
Case vrijednost3
    Blok3

:
.
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 promjenjiva po kojoj se ispituje *Select Case* struktura poprimila tačno **vrijednost1**. **Blok2** naredbi se izvršava, samo ako promjenjiva po kojoj se ispituje *Select Case* struktura poprimila **vrijednost2**. ***Case Else*** je opcioni dio *Select Case* struktura, koji će se izvršiti, samo ako promjenjiva 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 4.7.

Napisati program koji ispisuje poruku sa ocjenom koju je dobio učenik, na osnovu ocjene koja se upisuje preko *Textbox*-a pod imenom Text1.

```
Ocjena = Text1
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 promjenjiva po kojoj se ispituje *Select Case* struktura poprimila vrijednost, koja zadovoljava logički uslov zadat pomoću relacionog operatora. **Blok2** naredbi se izvršava, samo ako promjenjiva 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 4.8.

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 promjenjiva 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 promjenjiva 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 4.9.

Jedna trgovačka kuća prodaje robu uz različite iznose popusta. Za iznos robe koji je između 100 KM i 499 KM uz 8% popusta. Za iznos robe koji je između 500 KM i 2000 KM uz 12% popusta. Za iznos robe koji je veći od 2000 KM 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 uslov grananja. Uslov je zadovoljen i kada promjenjiva 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 4.10.

```
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

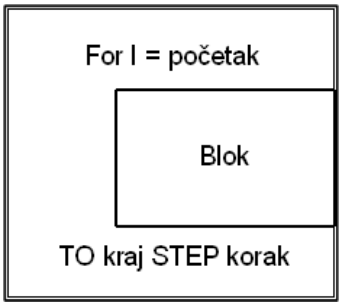
Programski jezik *Visual Basic* ima nekoliko struktura iteracije (ponavljanja). Ovdje će biti predstavljene samo neke strukture:

- *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. Koja će se struktura izabrati zavisi prvenstveno od toga da li se neka struktura ponavlja tačno određeni broj puta, ili promjenjiv broj puta.

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 1 onda ga nije potrebno 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 4.11. opisano je kako funkcioniše jedna *For* petlja, koja ponavlja dvije naredbe u 4 ciklusa.

Primjer 4.11.

```
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 4.12 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 4.12.

```
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 4.13 opisano je kako funkcioniše jedna *For* petlja koja ima 4 ciklusa ponavljanja jedne naredbe i jednog *Case* grananja (sa 10 kodnih linija).

Primjer 4.13.

```
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 4.14. 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 *MSFlexGrid1* dvodimenzionu tabelu.

Primjer 4.14.

```
AA = 0
BB = 3
With MSFlexGrid1
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
```

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,

```
      If AA > 5 Then
        .TextMatrix(I, J) = AA
      Else
        .TextMatrix(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 4.15.

```
AA = 0
BB = 3
With MSFlexGrid1
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
        .TextMatrix(K,I,J)= AA
      Else
        .TextMatrix(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 *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 4.15 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

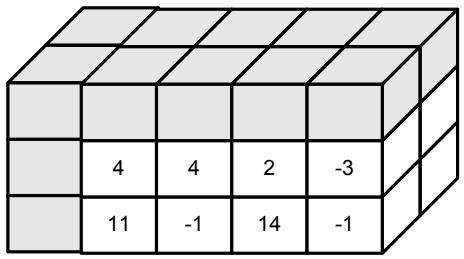
Tabela u prvom redu (K = 0)

	4	4	2	-3
	11	-1	14	-1

Tabela u drugom redu (K = 1)

	19	0	21	-1
	27	2	-23	52

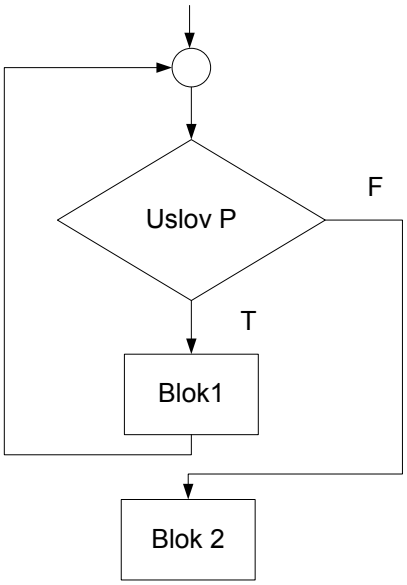
ili trodimenziono kada se vidi samo prvi dio



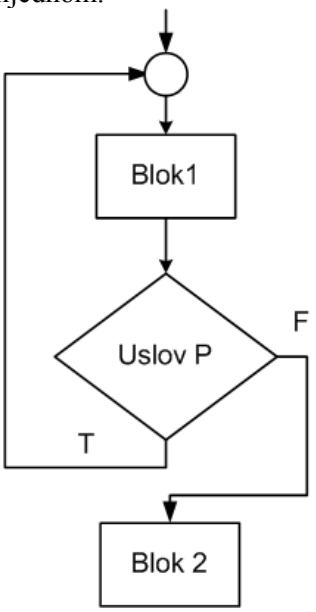
4.2.2. *Do While ... Loop* struktura

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 dotle 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 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 kompletnog programa *Visual Basic*. 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 nijednom.



Slika 4.5. Dijagram Do While



Slika 4.6. Dijagram Do Loop While

Sintaksa ove naredbe glasi:

DO WHILE logički uslov
  Iterativne naredbe
 [LOOP]
 [EXIT]
ENDDO

U ovoj strukturi **logički uslov** (**Uslov P**) predstavlja izraz sastavljen od jedne ili više promjenjivih 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. Ovo kao i naredba ***EXIT*** predstavlja opcionu komandu u ovoj

strukturi. ***EXIT*** predstavlja komandu koja omogućuje trenutni izlazak iz ove petlje, bez obzira da li je početni **logički uslov** (**Uslov P**) ispunjen.

U primjeru 4.16. opisano je kako funkcioniše jedna *Do While ... Loop* petlja koja ponavlja tri naredbe u 4 ciklusa.

Unutar *Do While ... Loop* petlje može postojati jedna, ali i više *If Then* ili *Case* struktura grananja. U primjeru 4.17. opisano je kako funkcioniše *Do While ... Loop* petlja, koja ima 4 ciklusa ponavljanja dvije naredbe i jedno *If Then* grananja (sa 5 kodnih linija).

Primjer 4.16.

```
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 4.17.

```
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

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 promjenjivih 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, ali samo ako je logički uslov (**Uslov P**) ispunjen.

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

U primjeru 4.18. opisano je kako funkcioniše jedna *Do ... Loop While* petlja, koja ima 4 ciklusa ponavljanja tri naredbe.

Primjer 4.18.

```
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
```

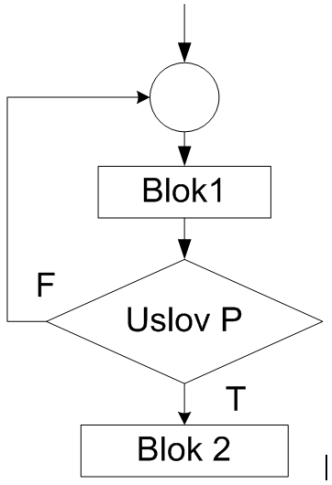
Rješenje

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.



Slika 4.7. Dijagram Do Loop Until

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 promjenjivih koje su povezane logičkim operatorima (=, <,>, < >, >= ili <=).

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. ***LOOP UNTIL*** predstavlja ključnu riječ, koja označava da se od ove naredbe program vraća na početak petlje, ali samo ako logički uslov nije ispunjen. Kod ove strukture treba takođe voditi računa, da ne dođe do pojave koja se zove beskonačna petlja.

U primjeru 4.19. opisano je kako funkcioniše jedna *Do ... Loop Until* petlja, koja ima 4 ciklusa ponavljanja tri naredbe.

Primjer 4.19.

```
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

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*-a, tj. nalaze se u nekoj biblioteci jezika (npr. MSDN) 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 ***bas***, koja je od imena odvojena tačkom, npr. "Ime_Modula.bas".

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

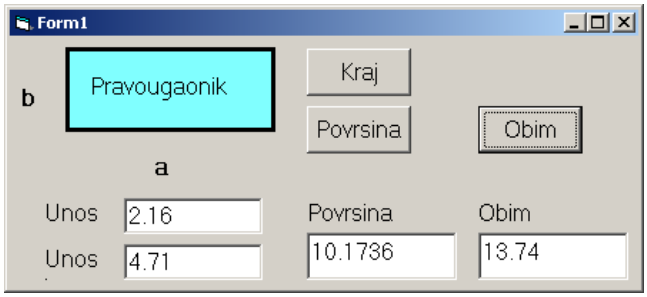
[Public ili **Private] Function** ImeFunkcije **([**argumenti**] [As Tip]) [As Tip]**
    Sekvencija naredbi
    ImeFunkcije = izraz ili vrijednost
End Function

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 promjenjive 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*.

Prva kodna linija u funkciji je obično deklaracija lokalnih promjenjivih, 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 promjenjivu 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 promjenjivoj **a**, a vrednost 2.0 dodeljena je promjenjivoj **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 Window),* za neki objekat ili za grupu objekata obavlja se pomoću procedure osobina. Slijedi primjer koda za programsku dodjelu osobine ***Text*** i ***BackColor*** objektu *TextBox*, koji nosi ime **Text1**:

```
Text1.Text = "Unos teksta u objekat"
Text1.BackColor = &H8080FF
```

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 promjenjive **PovrsinaP** i **ObimP,** koje su deklarisane u posebnom modulu.

```
Public PovrsinaP, ObimP As Single
Rem deklaracija 2 globalne promjenjive 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 promjenjive 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 promjenjivih. Prva kodna linija u proceduri je obično deklaracija lokalnih promjenjivih, koje se koriste samo u toj proceduri. Zatim se piše radni kod procedure, u kome globalne promjenjive moraju da poprime neku vrijednost. Preko globalnih promjenjivih 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 *VB* jezik.

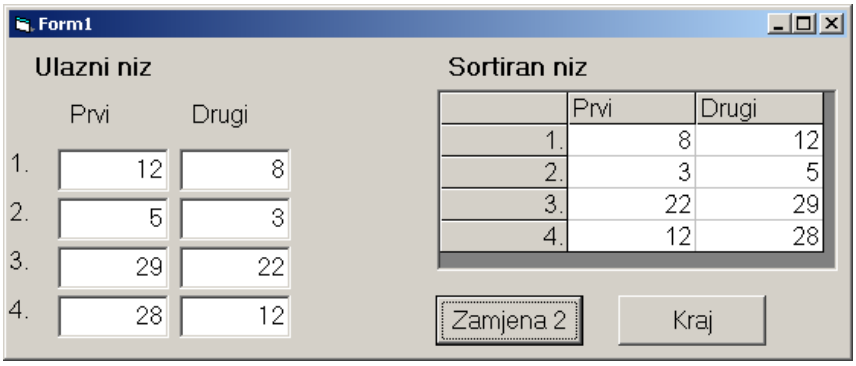
Sljedeći Primjer pokazaće postupak, kojim će se moći uređivati parovi realnih brojeva i to tako, da je na prvom mestu manji broj, a na drugom veći. Ovo uređivanje redoslijeda brojeva, se drugim riječima može nazvati sortiranje brojeva prema rastućem redoslijedu. Procedura u *Visual Basic*-u, kako je 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
    MSFIzlaz.TextMatrix(j + 1, 1) = Niz1(j)
    MSFIzlaz.TextMatrix(j + 1, 2) = Niz2(j)
Next j
End Sub
```

Nakon izvršenja program će dati ispis uređenih parova u *MSFlexGrid* 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.

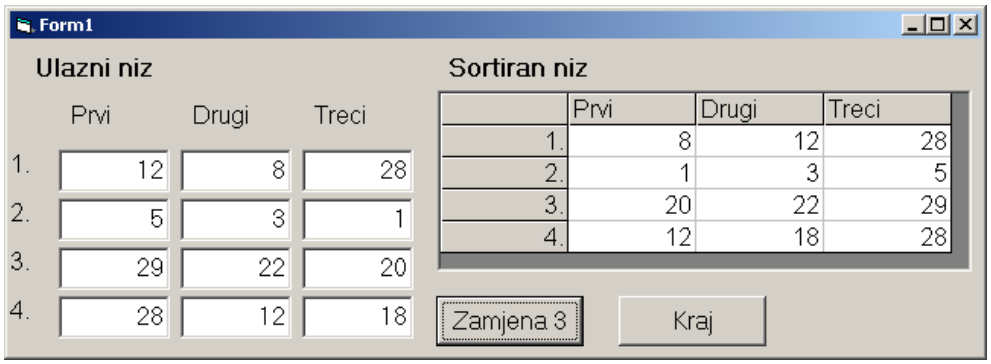
```
Public Prvi, Drugi As Single
Rem deklarisanje dvije globalne promjenjive

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
        MSFIzlaz.TextMatrix(j + 1, 1) = Niz1(j)
        MSFIzlaz.TextMatrix(j + 1, 2) = Niz2(j)
        MSFIzlaz.TextMatrix(j + 1, 3) = Niz3(j)
    Next j
    End Sub
```

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

Nakon izvršenja program će dati ispis uređenih parova u *MSFlexGrid* 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 promjenjive 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 promjenjivih, a u memoriji ostaju samo vrijednosti globalnih promjenjivih. 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* ima veliki broj gotovih matematičkih funkcija, koje stoje programerima na raspolaganju. Sve ove funkcije su detaljno opisane u *Help*-u, u kome se dati i primjeri korišćenja ovih funkcija. U tabeli 5.1. date su najčešće primjenjivane funkcije u *Visual Basic*-u. 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
Sqr (x)	Izračunava kvadratni korijen broja ≥ 0.
Exp (x)	Izračunava vrijednost stepena e^x.
Log (x)	Izračunava prirodni logoritam ln x , gdje je x >0.
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(99.2); 99=Fix(99.8); -99=Fix(-99.2); -99=Fix(-99.8)
Sgn (x)	Daje znak broja ili izraza.
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 = 12 Mod 5; 0 = 12 Mod 4.2 = 12 Mod 4
div (x,y)	Cjelobrojno dijeljenje broja x sa brojem y. 2=div(11,4)=11\4
Rnd [(x)]	Daje slučajan broj veći ili jednak 0 i manji od 1.
Round(x[, y])	Zaokružuje broj x na y broj decimalnih mjesta. 3.25=Round(3.25145,2)

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

5.4. FINANSIJSKE FUNKCIJE

Programski jezik *Visual Basic* 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.
Rješenje:

Iznos pojedinačnih otplata duga (kredita) računa se u *Visual Basic*-u 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 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).

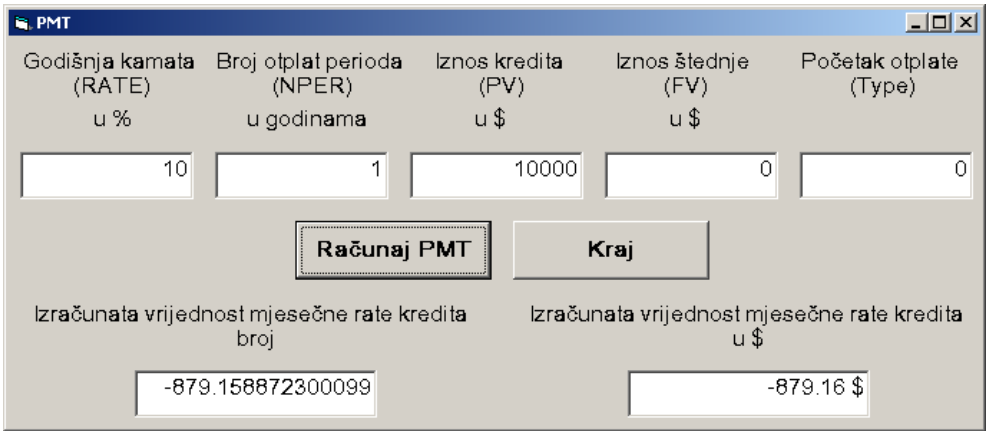
Nper predstavlja preostali broj otplatnih perioda (u našem slučaju to je 12 broj mjeseci koliko kredit treba da se vraća).

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
```

5.8. *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 je korisnik uneo. U slučaju da nije uneo ništa ili odustao od dijaloga biće vraćen

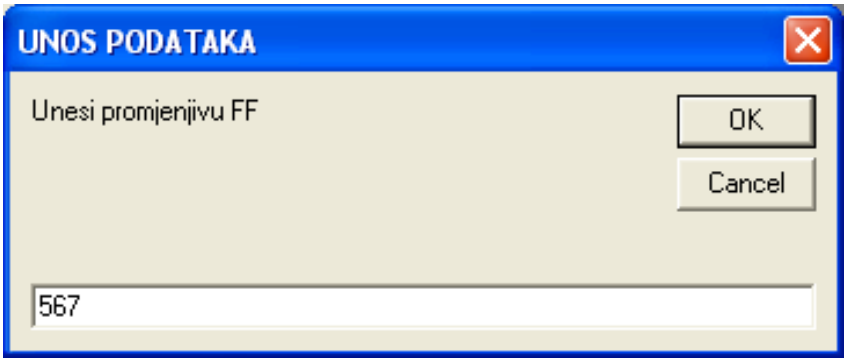
prazan tekst. Uneseni podaci mogu se memorisati u promjenljivoj definisanoj za tu svrhu ili prikazati na ekranu. Sintaksa ove funkcije je:

Promjenjiva=**InputBox(***prompt***[,** *title*][**,** *default*][**,** *xpos*][**,** *ypos*][**,** *helpfile***,** *context*]**)**

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.
- ***helpfile*** je tekst, koji predstavlja *Help* fajl, koji se poziva pokretanjem ove funkcije.
- ***Context*** je numerička vrijednost, koja identifikuje sadržaj *Help* fajla.

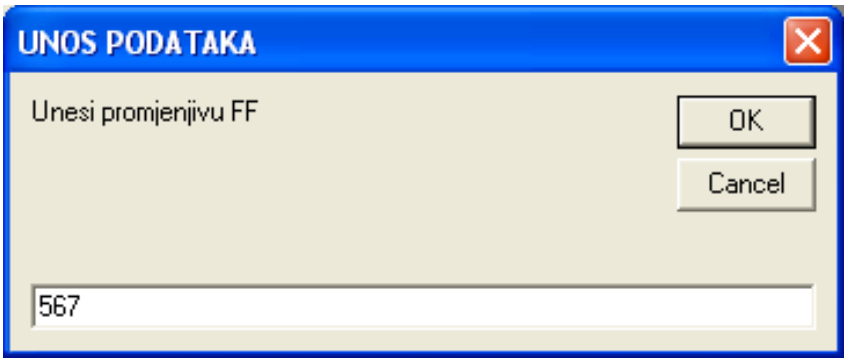
Za bolje razumijevanje ove funkcije izvršena je njena prezentacija na primjeru:

```
FF = InputBox("Unesi promjenjivu 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 promjenjivu 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 promjenjive *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")).
```

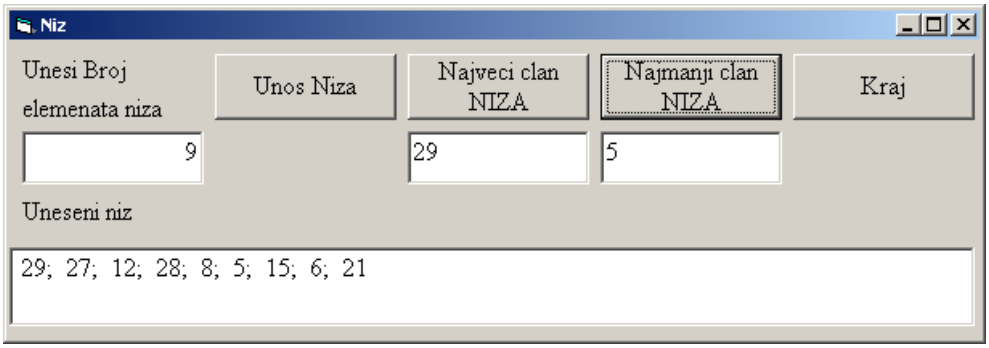
Parametri *Promt* i *Title InputBox* funkcije mogu da se unose i preko tekstualnih promjenjivih, koje su definisane van ove funkcije. Tako da kod predhodnog primjera možemo napisati i na sljedeći način:

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



Slika 5.6. Nalaženje maksimalnog elementa niza

5.9. *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 promjenjive.
- ***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 promjenjive.

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.

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 promjenjivoj ***Dugme3***. Ova promjenjiva 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
```

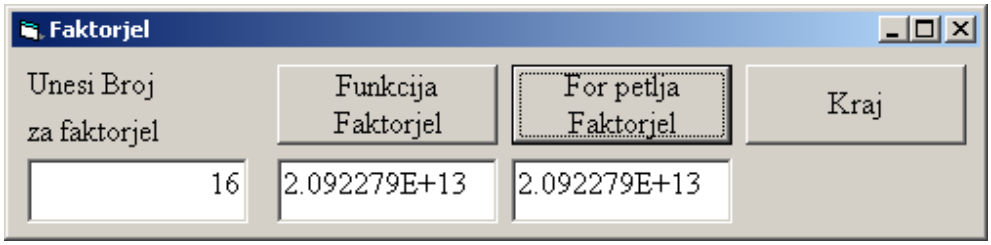
5.10. REKURZIVNE FUNKCIJE I PROCEDURE

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 u *Visual Basic*-u sa sljedećim kodom, a izgled rješenja ovog proglema za faktorjel broja 16 prikazan je na slici 5.7. U ovom primjeru promjenjiva **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 ORJENTISANO 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*-a*),* 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 orjentisanom 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 orjentisanom 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 *VISUAL BASIC*-A

Visual Basic posjeduje veliki broj gotovih objekata, koji stoje programeru na raspolaganju. U ovom poglavlju biće predstavljeni samo neki objekti, koji se najčešće koriste u *Visual Basic*-u. Za većinu objekata su dati primjeri programa, kako se navedeni objekti mogu koristiti prilikom pisanja programa. Svi oni koji žele da pronađu dodatne objekte, mogu ih lako pronaći u *Help*-u *Visual Basica*. Da bi se dodatni objekti mogli koristiti u programiranju, potrebno ih je prvo dodati u paletu alatki.

7.1. OBJEKAT *FORM*

U pogramskom jeziku *Visual Basic* 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. 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.
- ***BorderStyle*** za definisanje vrste okvira i ponašanje forme. Moguće vrijednosti su:
  0-*None*: Forma je bez okvira.
  1-*Fixed Single*: okvir jednostruke debljine.
  2-*Sizable*: okvir koji omogućava izmjenu dimenzije forme (inicijalna vrednost).
  3-*Fixed Dialog*: forma ima fiksni okvir kome se ne mogu mijenjati dimenzije.
  4-*Fixed ToolWindow*: forma ima izgled *toolbox*-a (manji font se koristi za naslov forme) i nije moguće mijenjati joj dimenzije.
  5-*Sizable ToolWindow*: isto kao 4 samo je moguće mijenjati dimenzije forme.
- C***aption*** 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).
- ***MaxButton*** prikaz dugmeta za maksimizaciju forme (*True* ili *False*).

- ***MinButton*** prikaz dugmeta za minimizaciju forme (*True* ili *False*).
- ***Moveable*** određuje da li korisnik može da pomjera formu. Vrijednosti *True* (može) ili *False* (ne može je pomjerati).
- ***Picture*** omogućava da na formu postavimo sliku.
- ***WindowState*** određuje dimenzije prozora prilikom otvaranja:
  0-*Normal*: dimenzije prozora su iste, kao one koje su postavljene u dizajn režimu.
  1-*Minimized*: prozor će biti prikazan minimiziran.
  2-*Maximized*: prozor će biti prikazan preko cijelog ekrana.

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.
- ***Unload*** uklanja 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 inicijalno se uvijek pravi prazna forma pod imenom **Form1.frm**. 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.frm 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 *.vbp*) u kome tu formu koristimo. Ovo posebno ističemo, jer 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.

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* ne dozvoljava programeru, da u jednom programu budu dvije forme sa istim imenom. U jednom programu 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, bolje 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 Form* u meniju *Project,* sa linije menija. Forme mogu da budu

- modalne i
- nemodalne.

Modalna forma se mora iskoristiti čim se pojavi na ekranu i na njoj se zadržava fokus, sve dok se pomoću neke komande ne ukloni sa ekrana. Forma je nemodalna, ako se može napuštati po želji i ponovo upotrebljavati. Ovakve forme su fleksibilnije za upotrebu.

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

ImeForme.Show broj

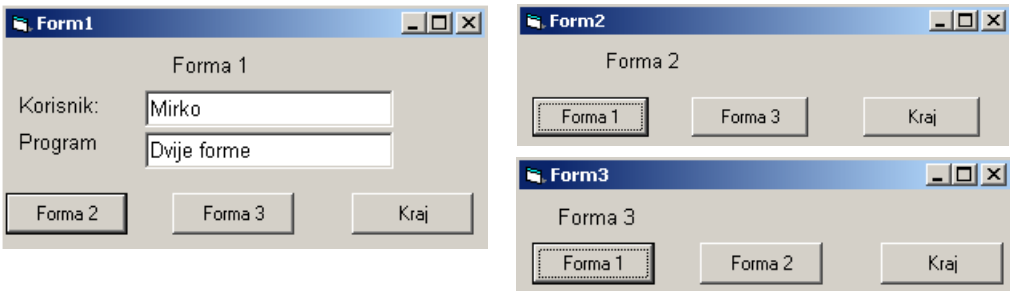
gdje je broj jednak 0 za nemodalne, a 1 za modalne forme. Ako se broj izostavi podrazumijeva se da je forma nemodalna, na primjer, *Form2.Show.*

Brisanje forme postiže se primjenom naredbe ***UnloadForm2.*** Uklanjanjem forme iz memorije biće odstranjeni i svi njemu pripadajući objekti, promjenljive i osobine, koje su vezane za tu formu.

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

1) Prvo se piše komanda za zatvaranje forme na kojoj se trenutno nalazimo.
2) Zatim se piše komanda za otvaranje nove forme.

```
Unload Imeforme1
NovaForma2.Show vbModal
```



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()
Unload Form1         Rem Zatvaranje forme
Form2.Show vbModal   Rem prelazak na formu
End Sub

Private Sub ComForma3_Click()
Unload Form1         Rem Zatvaranje forme
Form3.Show vbModal   Rem prelazak na formu
End Sub

Private Sub ComKraj_Click()
End
End Sub
```

Slijedi kod koji opisuje rad komandnih dugmadi na drugoj formi.

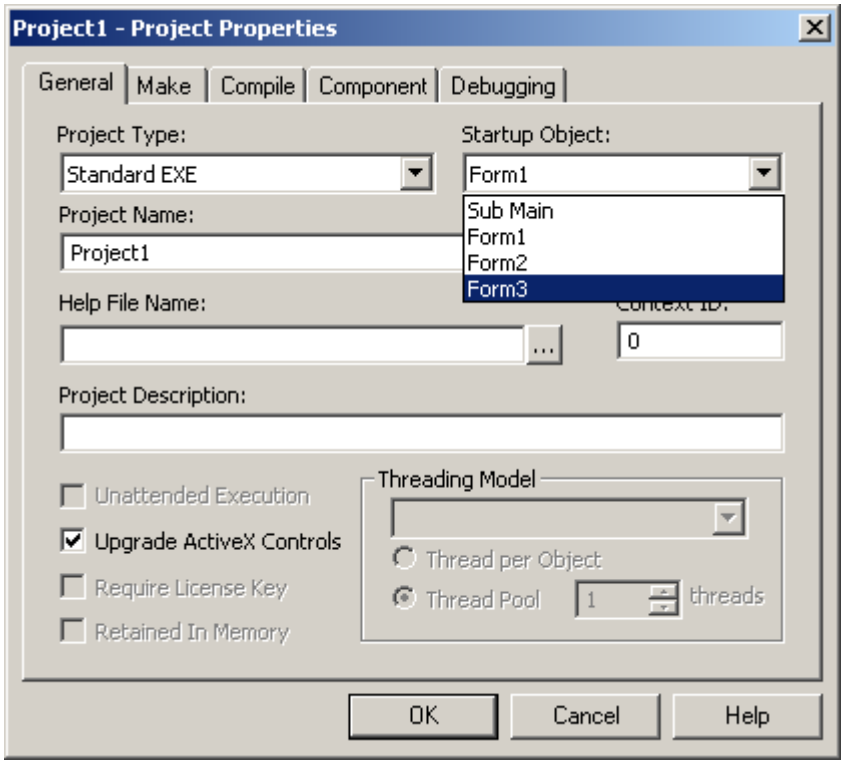
```
Private Sub ComForma1_Click()
Unload Form2         Rem Zatvaranje forme
Form1.Show vbModal   Rem prelazak na formu
End Sub

Private Sub ComForma3_Click()
Unload Form1         Rem Zatvaranje forme
Form3.Show vbModal   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 **Project Properties**. Nakon toga se pokazuje prozor prikazan na slici 7.2.
2) Zatim odaberite paletu **General**.
3) U komboboks listi **Startup Object**, odaberite formu koju želite kao novu početnu formu.
4) Nakon izbora željene početne forme odaberite dugme **OK**.



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* 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.
Unload frmUvod  Rem Prikaz uvodne slike.
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 zbirku *Forms* i naredbu *Unload*. 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()
Unload Me
End Sub
```

Ako vaš program koristi više formi, možete izbaciti sve forme, postavljanjem koda u potprogram događaja *Unload* vaše glavne forme. Možete upotrijebiti niz Forms, kako bi bili sigurni da ste pronašli i zatvorili sve svoje otvorene forme. Sljedeći programski kod koristi niz formi, za zatvaranje svih formi.

```
Private Sub Form_Unload (Cancel As Integer)
Dim i As Integer
  Rem Petlja za prolaz kroz niz formi
  Rem i zatvaranje svake forme.
For i = Forms.Count -1 To 0 Step -1
  Unload Forms(i)
Next
End Sub
```

U nekim slučajevima treba završiti rad programa, bez obzira na stanje postojećih formi ili objekata. *Visual Basic* 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* neće izvršiti potprograme događaja *QueryUnload, Unload* ili *Terminate* nijedne forme. Pokazivači objekata biće oslobođeni, *Visual Basic* 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 objektima. Tako da pri ponovnom pokretanju može doći do neplaniranih grešaka u radu programa.

7.2. OBJEKAT *LABEL*

U pogramskom jeziku *Visual Basic* objekat *Label*  se uglavnom koristi za prikaz teksta i kod nas se još zove obilježje. 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 može se mijenjati preko osobina (*Properties*) ili preko programa kroz kod. 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.
- ***Alignment*** za izbor poravnanja ispisa teksta u objektu (lijevo, desno, centrirano ili uz obadvije strane).
- ***Appearance*** za izbor 3-D ili ravnog izgleda ivica objekta.
- ***AutoVel*** omogućuje automatsko proširenje širine objekta na onu veličinu koja odgovara unesenom tekstu, ako je izabrana osobina *True*.
- C***aption*** 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. *TextBox* je vezani objekat i daje mogućnost pregleda ili izmjena vrijednosti polja iz baze podataka. Inicijalno, naziv prvog objekta je postavljen na *Text1*, sljedeći *Text2* itd. Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***Alignment*** za izbor poravnanja ispisa teksta u objektu (lijevo, desno, centrirano ili uz obadvije strane).
- ***Appearance*** za izbor 3-D ili ravnog izgleda ivica objekta.

- ***AutoVel*** omogućuje automatsko proširenje širine objekta na onu veličinu koja odgovara unesenom tekstu, ako je izabrana osobina *True*.
- ***DataField*** za izbor kolone tabele baze podataka iz koje se preuzimaju podaci.
- ***DataSource*** za izbor baze podataka sa kojom se vrši povezivanje.
- ***DataFormat*** za izbor tipa podataka u kome će se prikazivati podaci.
- ***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 "*".
- ***Text*** za upis teksta, koji će biti ispisan u ovom objektu.

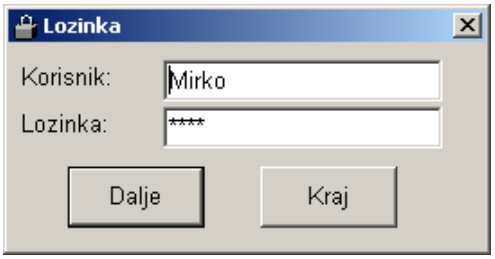
Nad ovim objektom se najčešće koriste sljedeći događaji:
- ***Change*** se pokreće kada se promjeni osobina *Text* u objektu.
- ***LastFocus*** se pokreće kada se napusti ovaj objekat.
- ***KeyPress*** se pokreće pritiskom tastera na tastaturi, kada se kursor nalazi u ovom objektu.
- ***MouseUp, MouseDown, 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 promjenjivu (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

PasswordChar	*
RightToLeft	False
ScrollBars	0 - None
TabIndex	3
TabStop	True
Tag	
Text	Mira

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. *TextBoks* 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 *TextBoks* 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 obadva
TextBoxsa kada 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. Probaj ponovo!", 0,
     "Lozinka"
     End If
  Else
     MsgBox "Korisniku nije dozvoljen pristup!", 0, "Lozinka"
  End If Rem otkazivanje ulogovanja
End Sub
```

7.4. OBJEKAT *COMMANDBUTTON*

Pomoću komandnog dugmeta *CommandButton* 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:

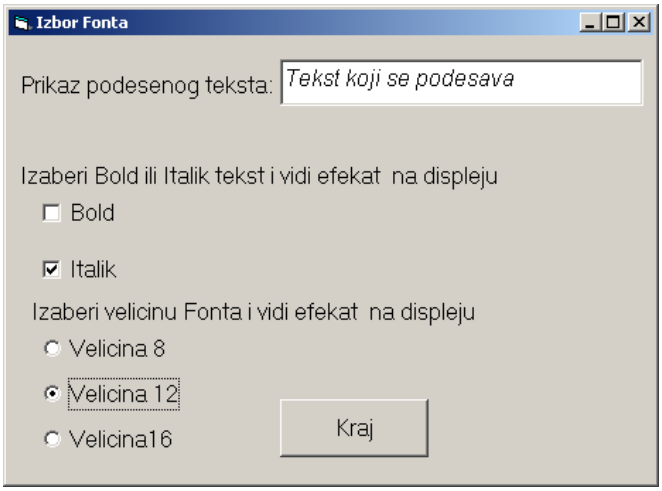
- ***Name*** za definisanje imena objekta.
- ***Appearance*** za izbor 3-D ili ravnog izgleda ivica objekta.
- C***aption*** za upis teksta koji će biti ispisan na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.

Nad ovim objektom se najčešće koristi događaj ***Click***, a to je jednostruki klik miša na ovaj objekat.

7.5. OBJEKTI *CHECKBOX* I *OPTIONBUTTON*

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 *OptionButton* 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 *Frame*, koji služi za razdvajanje grupa. Najbitnije osobine ovih objekata su:
- ***Name*** za definisanje imena objekta.
- C***aption*** za upis teksta, koji će biti ispisan na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.
- ***Value*** za izbor objekta *CheckBox* (može biti ne izabran 0-*vbUnchecked*, izabran 1-*vbChecked*, ili uslovno izabran 2-*Grayed*) i objekta *OptionButton* (može biti izabran *True* ili ne izabran *False*).

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 OptionButton

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 *OptionButton* 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 = vbChecked*). 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.Value=vbChecked Then Rem ako je potvrđen
    TxtDisplay.Font.Bold = True
Else      Rem ako nije potvrden
    TxtDisplay.Font.Bold = False
End If
End Sub
Private Sub ChkItalik_Click()
If ChkItalik.Value=vbChecked Then    Rem potvrđen
    TxtDisplay.Font.Italic = True
Else      Rem ako nije potvrden
    TxtDisplay.Font.Italic = False
End If
End Sub

Private Sub Opt8_Click()
  TxtDisplay.Font.Size = 8   Rem izabrana opcija 8
End Sub
Private Sub Opt12_Click()
  TxtDisplay.Font.Size = 12  Rem izabrana opcija 12
End Sub

Private Sub Opt16_Click()
  TxtDisplay.Font.Size = 16  Rem izabrana opcija 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.Value = False
  ChkItalik.Value = False
  Opt8.Value = True
  Opt12.Value = False
  Opt16.Value = False
End Sub
```

Ako se na jednoj formi želi istovremeno selekovati više objekata *OptionButton,* onda ih je potrebno postaviti u različite okvire (*Frame*). Na slici 7.6 prikazan je izgled programa, u kome se bira jelo sa prilozima.

7.6. OBJEKAT *FRAME*

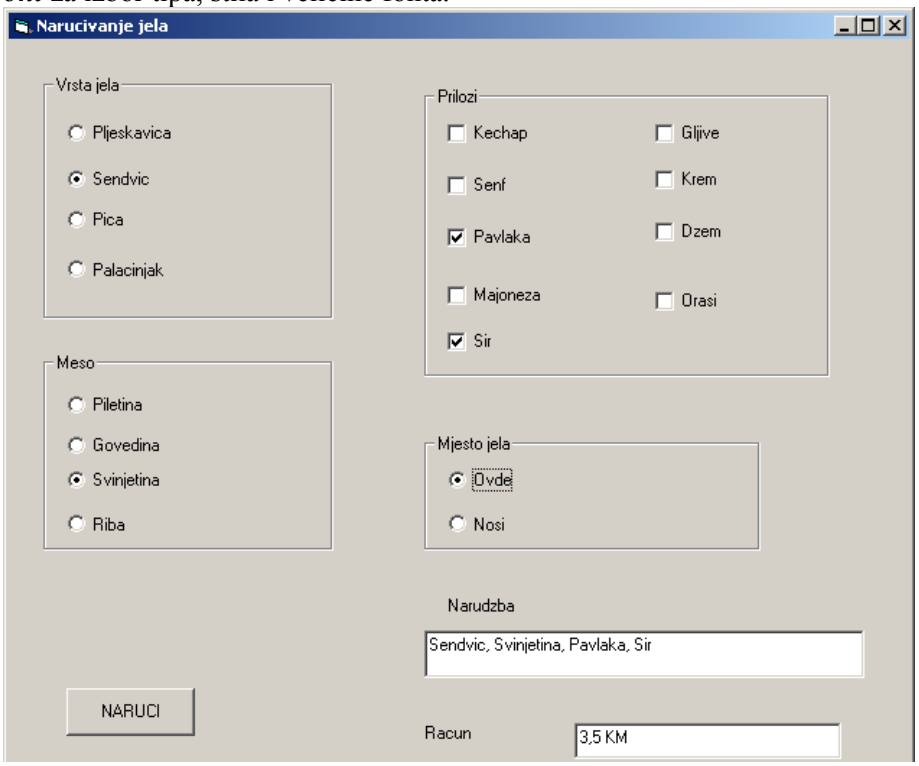
Objekat okvir *Frame*  se koristi za grupisanje drugih objekata najčešće istog tipa na jednoj formi. Na jednoj formi može postojati više okvira, kao u primjeru programa prikazanom na slici 7.6. Postupak grupisanja objekata *OptionButton* u okviru odvija se u sljedećim koracima:

1) Odaberite objekat okvira (*frame*) u paleti objekata i kreirajte okvir na formi.
2) Odaberite objekat *OptionButton* u paleti objekata i kreirajte ga unutar okvira.
3) Ponavljajte 2. korak za svaki novi objekat *OptionButton,* koji želite dodati u isti okvir.

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 okvira.

Najbitnije osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- C***aption*** za upis teksta, koji će biti ispisan kao naslov na ovom objektu.
- ***Font*** za izbor tipa, stila i veličine fonta.



Slika 7.6. Upotreba objekta CheckBox i OptionButton u više okvira

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.

Ovi objekti se mogu koristiti za direktno povezivanje sa bazama podataka. Obično se koriste za prikaz podataka iz baze podataka, ali se mogu koristiti i za punjenje podataka u bazu. Jedna lista se koristi za prikaz podataka iz jedne kolone tabele baze podataka.

Slika 7.7. Objekat list box

Slika 7.8. Objekat combo box

Najbitnije osobine ovih objekata su:
- ***Name*** za definisanje imena objekta.
- ***Appearance*** za izbor 3-D ili ravnog izgleda ivica objekta.
- ***List*** polje u koje se unose stavke koje se nalaze u listi.
- ***ItemData*** broj stavke koja može biti izabrana u listi.
- ***MultiSelekt*** pokazuje koliko se u listi stavki može izabrati, 0 - samo jedna stavka, 1 - više stavki, ili 2 - grupa stavki.
- ***Sorted*** za sortiranje elemenata u listi po *Ascii* kodu (ako je postavljeno *True*).
- ***Text*** za prikaz izabrane stavke u kombinovanom okviru (*ComboBox*).

U *Visual Basic*-u postoji nekoliko komandi za programsko upravljanje listama:
- ***AddItem*** - programski dodavanje novog člana liste.
- ***Clear*** - briše sve stavke iz liste.
- ***Removeitem*** - briše pojedinačne stavke iz liste.

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.RemoveItem 3 Rem brisanje 4 člana liste,
  jer prvi član liste ima redni broj 0
End Sub
Private Sub Brisisve_Click()
  List1.Clear       Rem brisanje svih članova liste
End Sub
Private Sub Dodaj_Click()
 List1.AddItem "Dodatak1" Rem dodavanje novog člana
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*. Nakon upisa svake stavke, pritisnite tipke *CTRL* + *ENTER* za prelaz u iduću liniju liste. 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 podataka
Do While Rezervoar.EOF = False
   priv = Rezervoar("Regbroj")
   Rem Preuzimanje vrijednosti u promjenjivu priv
iz kolone Regbroj tabele Rezervoar
   List1.AddItem (priv)
   Combo1.AddItem (priv)
   Rezervoar.MoveNext
Loop
```

Podaci se iz liste mogu preuzimati u promjenjive 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
End Sub
```

Kako bi stvorili listu sa više kolona i mogućnošću višestrukog izbora, trebate podesiti njegove osobine *Columns* i *MultiSelect*.

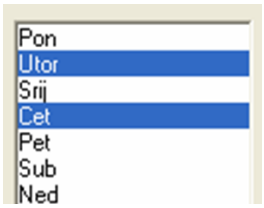
```
Columns = 2 (za dvije kolone)
Columns = 3 (za tri kolone)
```

Ako stvorite listu dovoljno veliku da prikazuje sve stavke u jednoj koloni, druga kolona će biti prazna, ostale stavke će biti prelomljene, a vodoravna klizna traka će se automatski pojaviti samo ako okvir s popisom nije dovoljno dug.

Svaki zapis u listi odgovara stavci popisa i postavljen je na *True,* ako je stavka izabrana, ili na *False*, ako nije odabrana. Nakon što korisnik odabere stavke iz popisa, provjerava se svaki zapis matrice, kako bi se vidjelo je li odabran (True).

Izbor istovremeno dvije stavke (druge i četvrte) u listi pod imenom *LstSedmica,* programski preko koda se vrši naredbama

```
LstSedmica.Selected(1) = True
LstSedmica.Selected(3) = True
```



Slika 7.9. Izbor u listi

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. Također 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 *small change* (mala izmjena).
- Klikom na lijevu ili desnu, odnosno gornju ili donju stranu objekta u odnosu na aktuelnu poziciju klizača. Ovaj način se zove *large change* (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).

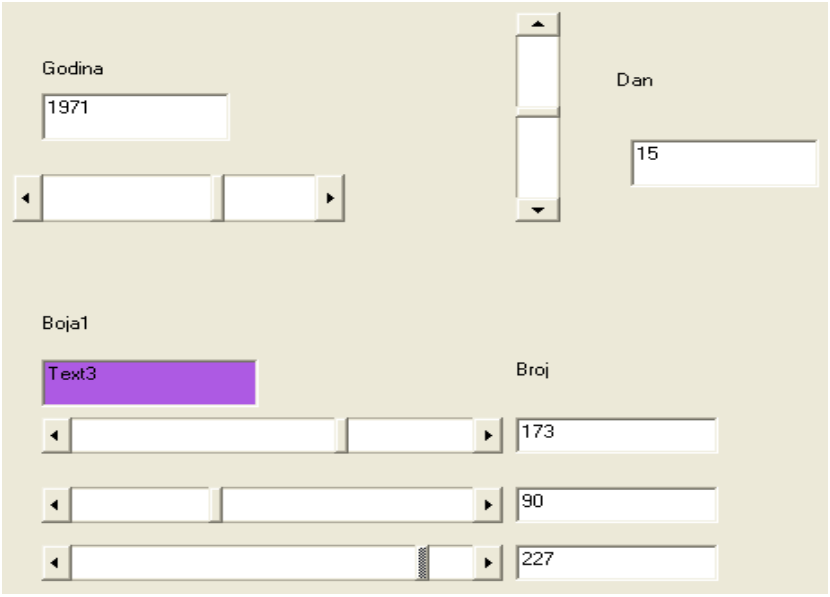
Najbitnije osobine kliznih traka su:

- ***Name*** za definisanje imena objekta.
- ***Max*** je maksimalna vrijednost klizača (opseg vrijednosti od -32768 do 32768).
- ***Min*** je minimalna vrijednost klizača (opseg vrijednosti od -32768 do 32768).
- ***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.
- ***Scroll*** je događaj, kada se klizač na kliznoj traci pomjera mišem.



Slika 7.10. Primjer korišćenja kliznih traka

Na slici 7.10. 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
```

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()
Rem na početku se klizači postavljaju na nulu
Dim x, y, z As Integer
x = 0
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 = RGB(x, y, z)
Rem pravljenje boje od 3 broja (0-255) funkcijom RGB
End Sub

Private Sub HScBoja1_Scroll()
Text4.Text = HScBoja1.Value
x = CInt(Text4.Text)
y = CInt(Text5.Text)
z = CInt(Text6.Text)
Text3.BackColor = RGB(x, y, z)
End Sub
```

7.9. OBJEKAT *TIMER*

Pomoću objekta *Timer*  u *Visual Basic*-u 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. 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*. Završetak izvršavanja procedure može se podesiti isključivanjem objekta *Timer,* 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 ***Timer.*** U njega se upisuje kod, koji želimo da se izvrši kada se objekat *Timer* starta.

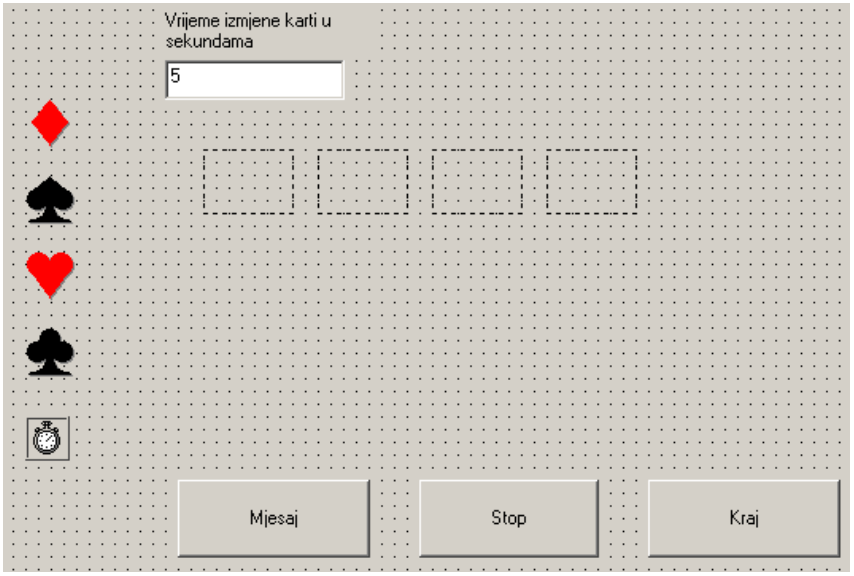
Jedna od primjena objekta *Timer* je postavljanje digitalnog sata na radnoj formi u labeli namijenjenoj za te svrhe. Digitalni sat pokazuje sistemsko vrijeme u formatu: sat:minuta:sekunda (**16:07:21**). Da bi sat funkcionisao pod nadzorom objekta *Timer* i uz sistemsko vrijeme računara objektu *Timer* 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_Timer()
  Label1.Caption = Time
  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
```

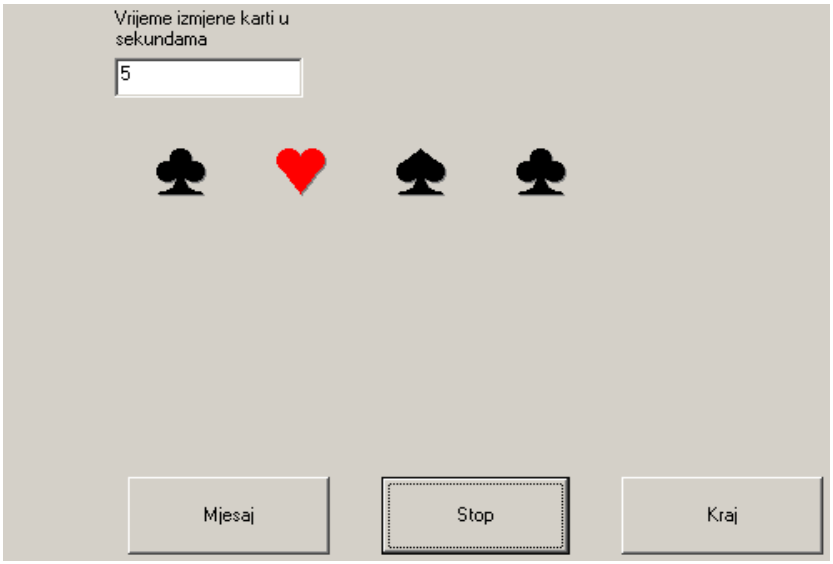
```
Private Sub Command1_Click()
    Command1.Visible = False
    Timer1.Enabled = True
End Sub
Private Sub Command2_Click()
      Command2.Visible = False
      Timer2.Enabled = True
End Sub
Private Sub Timer1_Timer()
    Label1.Caption = Time
    Command1.Visible = True
End Sub
Private Sub Timer2_Timer()
   Label1.Caption = "drugo dugme"
   Command2.Visible = True
   Timer2.Enabled = False
End Sub
```



Slika 7.11. Izgled razvojne forme

Slijedi primjer programa u kome se objekat *Timer* koristi za periodično mješanje 4 karte. Na slici 7.11. 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 od po četiri okvira (Karta(0), Karta(1), Karta(2), Karta(3), Ulaz(0), Ulaz(1), Ulaz(2) i Ulaz(3)).

Organizovanje objekata u niz omogućuje lakše ponavljanje iste komande nad više sličnih objekata upotrebom *For* petlje. Na slici 7.12. 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.



Slika 7.12. Izvršna verzija programa

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
        Ulaz(i).Visible = False
        Rem polazne 4 slike da se učine nevidljive
   Next i
End Sub
```

```
Private Sub Timer1_Timer()
   Rem procedura koja pokazuje šta tajmer radi
 For J = 0 To 3
     x = Round(Rnd * 3)
     Karta(J).Picture = Ulaz(x).Picture
        Rem slučajno pojavljivanje 4 karte
 Next J
End Sub
```

Za navedeni kod programa provjerite da li sve četiri karte imaju istu vjerovatnoću pojavljivanja? Ako je potrebno, sami doradite kod da to provjerite.

Posebno značajnu ulogu ima primjena objekta *Timer* prilikom kreacije raznih animacija u kombinaciji sa naredbom ***Move***. Naredbom *Move* definiše *se* smjer i pravac pomjeranja objekta (ili više njih), dok se uz pomoć *Timer*-a određuje brzina kretanja. Format naredbe *Move* ima sljedeći oblik:

imeobjekta.Move left[, top[, width[, height]]]

gdje je

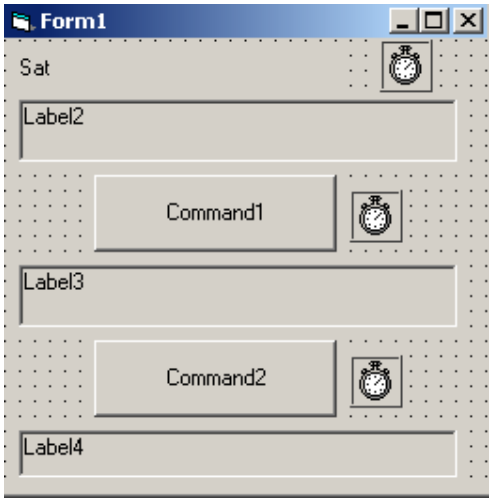
left - udaljenost od lijeve ivice forme,
top - udaljenost od gornje ivice forme,
width - definisanje širine objekta koji se pomjera,
height - definisanje visine objekta koji se pomjera.
Primjer primjene naredbe *Move* nad objektom pod imenom **Picture1**:

```
Picture1.Move X
Picture1.Move X,Y
Picture1.Move Picture1.Left X, Picture1.Top Y
```

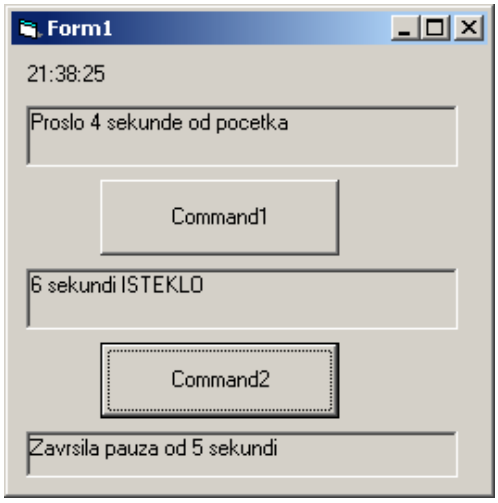
gdje X i Y predstavljaju koordinate u pikselima za vodoravno i vertikalno pomjeranje objekta slike pod imenom **Picture1**. U tom slučaju podrazumijeva se da je gornje lijevo tjeme (vrh) forme koordinatni početak sistema čiji prvi red tačaka predstavlja x-osu, a prva kolona tačaka obrasca *y-osu.* Da 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.

```
Private Sub Form_Load()
   Timer1.Interval = 500   Rem postavljanje vremena
    DeX = 100   Rem dužina skoka po x osi
    DeY = 100   Rem dužina skoka po y osi
End Sub
Private Sub Timer1_Timer()
  Label1.Caption = Time   Rem postavljanje sata
  Label1.Move Label1.Left + DeX, Label1.Top + DeY
    Rem pomjeranje objekta
  If Label1.Left < ScaleLeft Then
    Rem objekat nije došao do desne ivice forme
    DeX = 100
  End If
  If Label1.Left+Label1.Width>ScaleWidth+ScaleLeft Then
      DeX = -100   Rem promjena smjera kretanja
  End If
  If Label1.Top < ScaleTop Then
       DeY = 100
  End If
  If Label1.Top+Label1.Height>ScaleHeight+ScaleTop Then
       Rem objekat nije došao do donje ivice forme
      DeY = -100 Rem promjena smjera kretanja
  End If
End Sub
```



Slika 7.13.a Forma sa 3 tajmera



Slika 7.13.b Izvršna verzija programa

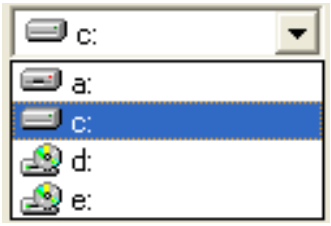
U jednom programu može da postoji više tajmera, koji mogu imati različita vremena i različite funkcije. Na slici 7.13. prikazan je primjer programa sa tri

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. Slijedi primjer koda programa koji omogućuje da se izračuna vrijeme trajanja nekog procesa pomoću *DoWhile* petlje.

```
Private Sub Command2_Click()
PauseTime = 5   ' postaviti cekanje na 5 sekundi.
Start = Timer    ' Postaviti start time.
Do While Timer < Start + PauseTime
    Label4.Caption = "Pocela pauza od 5 sekundi"
    DoEvents    ' Rezultat drugog procesa.
Loop
Finish = Timer    ' Postaviti vrijeme kraja.
TotalTime =Finish-Start 'Racunanje ukupnog vremena.
Label4.Caption = "Zavrsila pauza od 5 sekundi"
MsgBox "Pauza trajala " & TotalTime & " secondi"
End Sub
```

7.10. OBJEKAT *DRIVELISTBOX*

Okvir za prikaz drajva (*DriveListBox*) namjenjen je za prikaz i izbor svih disk jedinice u sistemu, gdje se program izvršava. Prilikom prikaza su uključujučeni i mapirani mrežni diskovi, koji se trenutno nalaze na računaru (a:, c:, d:, e:, itd.). Sa lijeve strane oznake disk jedinice se nalazi odgovarajuća ikona, zavisno od toga da li je u pitanju *flopy drajv, hard disk, CD Rom* uređaj ili mapirani mrežni disk.



Slika 7.14. Drive List Box

Ovaj objekat se ponaša kao *ComboBox*, kod koga je osobina *Style* podešena na 2 - *Dropdown List*. Logično, ova lista je samo za čitanje. Ovaj objekat se najčešće kombinuje sa objektom *DirectoryListBox* za izbor direktorijuma i *FileListBox* za izbor fajlova. Najbitnija osobina ovog objekta je ***Drive,*** pomoću koje se vraća sadržaj trenutno izabranog drajva.

Nad ovim objektom se najčešće koristi događaj ***Change,*** koji se starta kada se u ponuđenoj listi drajvova računara izabere željeni drajv. Primjer koda programa za upis izabranog drajva iz objekta pod imenom **DrivLista**, u objekat za izbor direktorijuma sa imenom **DirLista** je sljedeći.

```
Private Sub DrivLista_Change()
Rem procedura se aktivira izaborom novog drajva
  DirLista.Path = DrivLista.Drive
End Sub
```

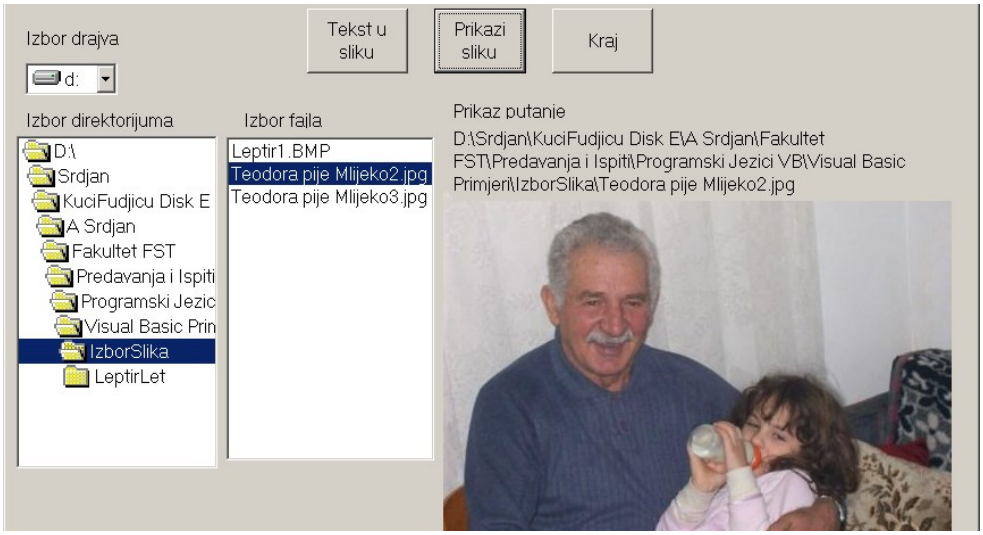
7.11. OBJEKAT *DIRECTORYLIST BOX*

Okvir za prikaz direktorijuma (*DirectoryListBox*) 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 7.15. Ovaj objekat se najčešće kombinuje sa objektom *Drive List Box* za izbor disk jedinice i *FileListBox* za izbor fajlova.

Najbitnija osobina ovog objekta je ***Path*** koja sadrži tekuću putanju do direktorijuma, koji je trenutno selektovan.

Nad ovim objektom se najčešće koristi događaj ***Change,*** koji se starta kada se u ponuđenoj listi izabere željeni direktorijum. Primjer koda programa za upis izabranog direktorijuma u objekat za izbor fajlova je sljedeći

```
Private Sub DirLista_Change()
  FilLista.Path = DirLista.Path
End Sub
```



Slika 7.15. Drive List Box, DirectoryListBox i FileListBox

7.12. OBJEKAT *FILELISTBOX*

Okvir za prikaz fajlova (*FileListBox*) namjenjen je za prikaz i izbor svih fajlova, koji se trenutno nalaze na izabranom (aktivnom) direktorijumu. Ovaj objekat ispisuje fajlove u vidu liste. Moguće je podesiti koji fajlovi će biti prikazani u zavisnosti od njihovih atributa i ekstenzije. Primjer korišćenja ovog objekta prikazan je na slici 7.15. Ovaj objekat se najčešće kombinuje sa objektom *Drive List Box* za izbor drajva i *DirectoryListBox* za izbor direktorijuma.

Najbitnija osobina ovog objekta je ***Path,*** koja sadrži tekuću putanju do direktorijuma, za fajl koji je trenutno selektovana, a ostale bitne osobine su:
- ***Name*** za definisanje imena objekta.
- ***Pattern*** koja sadrži tekst, koji određuje koja će grupa fajlova biti izabrana. Ova osobina podržava * i ? karaktere. (Primjer: *.bmp;*.gif;*.jpg).
- ***Multiselect*** koja omogućuje istovremeni izbor više fajlova (0, 1 ili 2).

Nad ovim objektom se najčešće koriste sljedeći događaji:
- ***DblClick*** kada se klikne dva puta brzo na neki fajl.
- ***PathChange*** kada se promjeni putanja do fajla.

Slijedi primjer koda programa, koji selektovanu sliku u okviru *FileListBox* prikazuje u objektu *Image* (sa imenom **ImgSlika1**), a putanju do izabrane slike u objektu *Label* (sa imenom **LabSlika**). Ovaj kod se pokreće klikom na dugme (sa imenom **ComPrikaz**).

```
Private Sub ComPrikaz_Click()
Dim ImeSlike As String
Rem provjera korjena direktorijuma
If Right(FilLista.Path, 1) = "\" Then
    ImeSlike = FilLista.Path + FilLista.FileName
Else
    ImeSlike=FilLista.Path+ "\" + FilLista.FileName
End If
LabSlika.Caption = ImeSlike
Rem upis putanje u labelu
ImgSlika1.Picture = LoadPicture(ImeSlike)
Rem upis izabrane slike u objekat Image
End Sub
```

U kodu programa ili prilikom podešavanja nekih osobina objekata u *Visual Basic*-u primjenjuju se sljedeći zamjenski karakteri:

? - simbol koji mjenja jedan karakter.

* - simbol koji mjenja jedan ili više karaktera.

- simbol koji mjenja jedan broj (0–9).

[*niz karaktera*] - bilo koji jednostruki karakter iz niza karaktera (d-s, rezultat mogu biti sva slova između slova d i s).

[**!***niz karaktera*] - bilo koji jednostruki karakter koji nije iz navedenog niza karaktera. (!d-s, rezultat mogu biti sva slova koja nisu između slova d i s).

Slijedi primjer korišćenja zamjenskih karaktera u kombinaciji sa funkcijom *Like* i promjenjivom, koja je logičkog tipa.

```
Dim Provjera as Boolean
Provjera = "aBBBa" Like "a*a"          Rem daje True.
Provjera = "F" Like "[A-Z]"            Rem daje True.
Provjera = "F" Like "[!A-Z]"              Rem daje False.
Provjera = "a2a" Like "a#a"               Rem daje True.
```

```
Provjera = "aM5b" Like "a[L-P]#[!c-e]"   Rem True.
Provjera = "BAT123khg" Like "B?T*"       Rem True.
Provjera = "CAT123khg" Like "B?T*"       Rem False.
```

7.13. OBJEKTI *LINE* I *SHAPE*

Objekat *Line* koristi se za crtanje prave linije, a objekat Shape za crtanje figura u obliku kvadrata, krugova i ovala na formi. Moguće je izabrati tip geometrijskog oblika, boju, ispunu, tip linije kojom se crta, kao i nekoliko predefinisanih vrsta šrafiranja. Osnovna namjena ovih objekata je estetsko uređenje forme na koju se postavljaju. Najbitnije osobine ovih objekata su:

- ***Name*** za definisanje imena objekta.
- ***BorderColor*** za izbor boje linije objekta.
- ***BorderStyle*** za izbor tipa linije.
- ***BorderWith*** za izbor debljine linije.
- ***FillColor*** za izbor boje, kojom je popunjen objekat *Shape*.
- ***Shape*** za izbor figure (pravougaonik, krug ili oval) kod objekta *Shape*.

Na žalost, nema događaja koji se mogu aktivirati nad ovim objektima. Nije moguće u izvršnom režimu jednostavno detektovati klik miša na njih.

Crtanje linija i osnovnih oblika se može vršiti i programski preko koda. U sljedećim primjerima prikazano je crtanje osnovnih objekata, pri čemu su x i y brojne vrijednosti u pikselima kordinata na formi. Boja linije se zadaje u heksadecimalnom kodu ili preko imena boje prema tabeli 7.1.

U primjerima crtanja objekata koji slijede **Ime objekta** označava na kom objektu se crta željeni oblik, a to je obično glavna forma (*Form1*). Programski se mogu crtati sljedeći oblici:

- Crtanje tačke
imeobjekta.PSet (X,Y), Boja
- Crtanje linije
imeobjekta.Line (X1, Y1)-(X2, Y2), Boja
- Pravljenje rama
imeobjekta.Line (X1, Y1)-(X2, Y2), Boja, B
- Popunjavanje rama
imeobjekta.Line (X1, Y1)-(X2, Y2), Boja, B, BF
- Crtanje krive linije
imeobjekta.Line (X1, Y1)-(X2, Y2), Boja
imeobjekta.Line -(X2, Y3), Boja
- Crtanje kruga (x i y kordinate centra kruga)
imeobjekta.Circle (x, y), poluprečnik, Boja
- Brisanje svih grafičkih objekata koji su nacrtani na objektu radi se sa funkcijom CLS
imeobjekta.Cls

Tabela 7.1. Naziv boja

Naziv	Vrijednost	Opis
vbBlack	&h00	crna
vbRed	&hFF	crvena
vbGreen	&hFF00	zelena
vbYellow	&hFFFF	žuta
vbBlue	&hFF0000	plava
vbCyan	&hFFFF00	cyan
vbWhite	&hFFFFFF	bijela

Primjer koda programa za crtanje na formi koja nosi ime **Form1**.

```
Rem Crtanje tačke
  Form1.PSet (50, 150), &HFF&
Rem Crtanje linije
  Form1.Line (50, 50)-(1300, 1300), &HFF&
Rem pravljenje rama
  Form1.Line (150, 150)-(2300, 2300), &HFF&, B
Rem pravljenje rama koji je popunjen bojom
  Form1.Line (550, 550)-(3300, 3300), &HFF&, BF
Rem pravljenje krive linije od dvije prave linije
koje su spojene
  Form1.Line (5000, 5000)-(7000, 8000), &HFF&
  Form1.Line -(7000, 5000), &HFF&
Rem pravljenje tri kruga koji imaju isti centar
  Form1.Circle (4000, 5000), 1000, vbGreen
  Form1.Circle (4000, 5000), 500, vbBlue
  Form1.Circle (4000, 5000), 200, vbRed
```

7.14. OBJEKTI *IMAGE* I *PICTUREBOX*

Ova dva tipa okvira za sliku se koriste za postavljanje slika na formu, direktno preko osobina objekta ili programski kroz kod programa. Slike koje se postavljaju trebaju biti snimljene negdje na računaru u nekom od sljedećih formata: *Bitmap, Icon, Jpeg* i *Gif*.

Najbitnije osobine ovih objekata su:

- ***Name*** za definisanje imena objekta.
- ***AutoSize*** omogućuje mijenjanje veličine okvira za sliku (za *PictureBox*) tako da bude prikazana kompletna slika, kada se ova osobina postavi na *True*. Ako je ova osobina postavljena na *False*, onda je okvir uvijek fiksan bez obzira na veličinu slike. Ako je slika veća od okvira, biće prikazan samo njen gornji lijevi dio.
- ***Picture*** za povezivanje ovih objekata sa gotovom slikom.
- ***Strech*** omogućuje mijenjanje veličine slike tako da ona u potpunosti popuni okvir za sliku (objekta *Image*), kada se ova osobina postavi na *True*. Tada može doći do deformacije slike. Ako je ova osobina postavljena na *False,* onda se bez obzira na veličinu okvira, slika prikazuje u punoj veličini. Ako je slika velika može se desiti da ona onda prekrije ostale objekte na formi.

Programski se slika može postavljati u okvir za slike preko procedure *LoadPicture*. Sintaksa koda kada se slika "vjezba.gif " preuzima sa tačno poznatog direktorijuma u okvir pod nazivom **Slika1**, je sljedeća:

```
Slika1.Picture=LoadPicture("c:\slike\vjezba.gif")
```

Sintaksa koda kada se slika preuzima sa direktorijuma na kome se nalazi snimljen projekta u kome se trenutni program piše, je sljedeća:

```
slika2.Picture =LoadPicture(App.Path & "\vjezba.gif ")
```

Nad ovim objektima se najčešće koriste događaji ***Click*** i ***DblClick,*** a to je jednostruki i dvostruki klik miša na ovaj objekat.

Nad ovim objektima se mogu koristiti i metode:

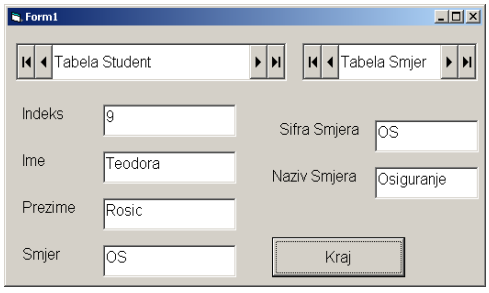
- ***Cls*** za brisanje teksta koji je predhodno ispisan preko slike.
- ***Print*** za ispis željenog teksta preko slike.

Primjer koda je sljedeći:

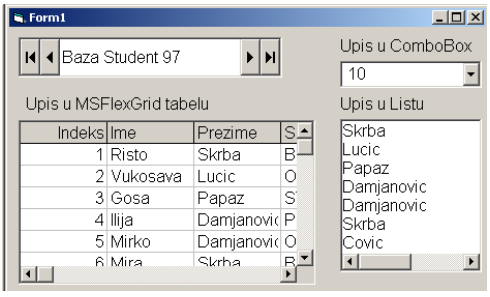
```
ImeOkvira.Cls
ImeOkvira.Print "proba"
```

7.15. OBJEKAT *DATA*

Rad sa bazama podataka veoma je značajan za svaki programski jezik, pa i za *Visual Basic*. Pomoću baza podataka može se čuvati velika količina podataka, koji su različitog tipa i koji su organizovani u tabele. Osnovni objekat za komunikaciju sa bazom *MS Access-a* je *Data*, koji omogućava pristup poljima i slogovima baze direktno na formi. U verziji programa *Visual Basic* 6.0 objekat *Data* može se povezati samo sa *MS Access* bazom koja je snimljena u verziji 97. Sa novijim verzijama *MS Access-a* ne može ostvariti komunikaciju. Na slici 7.16. prikazan je primjer korišćenja objekta *Data* za prikaz podataka u *TextBox*-u, a na slici 7.17. prikaz podataka *MSFlexGrid* tabeli, *List*-i i *ComboBox*-u. Strelice na objektu *Data* služe za pomjeranje slogova (redova u tabeli) u bazi podataka. Unutrašnje strelice vrše pomjeranje po jedan slog unaprijed ili unazad, a spoljašnje za prelazak na prvi ili posljednji slog u bazi. Objekat *Data,* u suštini, čini sponu između *Access*-ovih baza podataka i *Basic*-ovih objekata, odnosno omogućava prikaz i aktivnosti sa podacima iz baze.



Slika 7.16. Objekat Data i TextBox



Slika 7.17. Objekat Data i Tabela

Najbitnije osobine ovog objekta su:
- ***Name*** za definisanje imena objekta.
- ***Appearance*** za izbor 3-D ili ravnog izgleda ivica objekta.
- ***Caption*** za upis teksta koji će biti ispisan u ovom objektu.
- ***Conect*** za izbor baze sa kojom se vrši povezivanje (to je uglavnom Access).
- ***DataBaseName*** za izbor putanje i imena baze podataka iz koje se preuzimaju podaci (D:\Mira\Fakultet FPE\Programski Jezici VB\ Data\Student97.mdb).
- ***Font*** za izbor tipa, stila i veličine fonta.
- ***RecordsetType*** za određivanje načina kotišćenja podataka sa kojim se povezuje ovaj objekat (može biti: *0 - Table , 1 - Dynaset (difolt), 2 - Snapshot).*
- ***RecordSource*** za izbor odgovarajuće tabele iz padajuće liste, a na osnovu predhodno izabrane baze podataka.

Nad ovim objektom se najčešće koristi događaj ***Validate,*** koji se pokreće kada se klikne mišem na strelice objekta.

Sve osobine se mogu podešavati preko prozora *Properties*, ali i preko koda kao u sljedećem primjeru

```
Data1.RecordsetType = vbRSTypeDynaset
Data1.DatabaseName = "Student97.MDB"
Data1.RecordSource = "Student"
Data1.Refresh
```

Podaci iz baza podataka mogu se prikazivati, a zatim i obrađivati pomoću niza *VB* objekata, a posebno pomoću objekata:

- *Text Box,*
- *List Box,*
- *Label,*
- *FlexGrid,*
- *Image* i dr.

Za povezivanje *TextBox* objekata sa odgovarajucim poljima iz baze podataka, da bi se podaci iz tih polja smjestili u objekte za tekst, neophodno je podesiti dvije osobine *TextBox*-a: *DataSource* i *DataField.* Podešavanje *DataSource* osobine *TextBox* objekta vrši se preko objekta *Data,* pomoću kojeg je već uspostavljena veza sa bazom podataka.

Postupak dodjele osobine teče sljedećim redom. Prvo se odabere željeni *TextBox* objekat, pa se, zatim, u prozoru *Properties* klikne na osobinu *DataSource.* Klikom na strelicu koja se pojavljuje u produžetku reda navedene osobine razvija se lista svih objekata *Data* od kojih treba odabrati odgovarajući. Na sličan način podešava se i osobina *DataField.* U istom prozoru *Properties* se bira osobina *DataField* koje ima strelica okrenuta nadole. Klikom na ovu strelicu pojavljuje se spisak svih kolona iz tabele i potrebno je odabrati onu kolonu, koje će imati interakciju sa odgovarajucim tekstualnim poljem. Tekstualnim objektima mogu se,

između ostalog, odrediti osobinu N*ame,* kojom se imenuje dotična kolona, i osobinu *Text* koja je, uglavnom, prazna. Slijedi primjer koda, koji prikazuje podešavanje objekta *Data* i objekata *TextBox.*

```
Data3.RecordsetType = vbRSTypeDynaset
Data3.DatabaseName = "Student97.MDB"  ' izbor baze
Data3.RecordSource = "Student"   ' izbor tabele
Data3.Caption = "Unos preko koda"
Text1.DataField = "Indeks"       ' kolona tabele
Text2.DataField = "Ime"          ' kolona tabele
Text3.DataField = "Prezime"      ' kolona tabele
Text4.DataField = "Smjer"        ' kolona tabele
```

Pod osnovnim operacijama sa bazama podataka obično podrazumijevamo:

- pronalaženje određenog sloga i eventualne izmjene na podacima,
- dodavanje novog sloga,
- brisanje postojećeg sloga,
- pregled baze,
- kopiranje i, eventualno
- štampanje.

Zadatak pojedinih operacija definiše se, skoro redovno, pomoću komandnog dugmeta. Naredbe za tu namjenu temelje se, uglavnom, na osobinama objekta *RecordSet,* na primjer:

```
With ImeBaze.RecordSet
.Index = ImePolja Rem određuje se polje po kome će
biti obavljeno indeksiranje i pretraga.
.Seek = Vrijednost Rem zadata vrijednost po kojoj
se vrši traženje.
.NoMatch    Rem znači da slog nije pronađen.
.MoveFirst  Rem prelazak na prvi slog.
.MoveNext   Rem prelazak na sljedeći slog.
.MoveEOF ili .MoveLast Rem prelazak na zadnji slog.
.AddNew     Rem dodavanje novog sloga, pri čemu se
naredbom
ImeObjektaTekst.Fokus Rem postavlja pokazivac u ono
tekstualno polje od kojeg počinje upis podataka za
novi slog.
.Delete  Rem briše se odgovarajući slog. Naravno,
da bi slog bio izbrisan potrebno ga je prethodno
pronaći.
```

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:

```
FileCopy Adresalzvora, AdresaCilja.
```

Primjer 1.

Kopiranje *Word* dokumenta pod imenom "Evidencija.doc", koji se nalazi na lokaciji C: diska, na istu lokaciju, ali pod imenom "Kopija1.doc".

```
FileCopy "C:\Evidencija.doc", "C:\Kopija1.doc"
```

Primjer 2.

Kopiranje *Word* dokumenta pod imenom "Evidencija.doc", koji se nalazi na lokaciji C: diska, na drugi direktorijum sa putanjom "C:\Proba", ali pod istim imenom.

```
FileCopy "C:\Evidencija.doc", "C:\Proba\Evidencija.doc"
```

Primjer 3.

Kopiranje *Word* dokumenta "Proba3.doc", koji se nalazi na lokaciji C: diska, na direktorijum na kome se nalazi tekući projekat *Visual Basica* sa kojim radimo, ali pod istim imenom kao i originalni fajl.

```
FileCopy "C:\Proba3.doc", App.Path & "\Proba3.doc"
```

Primjer 4.

Kopiranje *Word* dokumenta "Proba.doc", koji se nalazi na direktorijum na kome se nalazi tekući projekat Visual Basica sa kojim radimo, na isti direktorijum na kome se nalazi tekući projekat Visual Basica sa kojim radimo, ali pod drugim imenom "\Kopi4.doc".

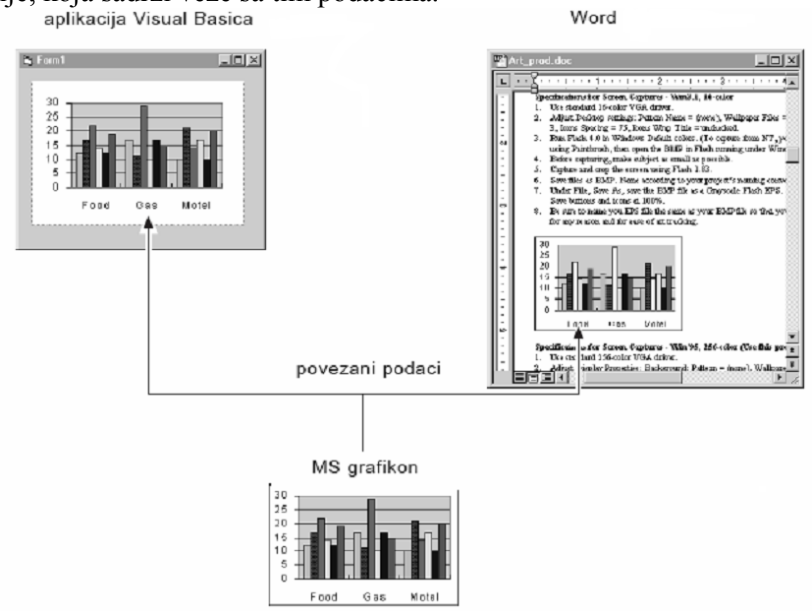
```
FileCopy App.Path & "\Proba.doc",App.Path & "\Kopi4.doc"
```

7.16. OBJEKAT *OLE*

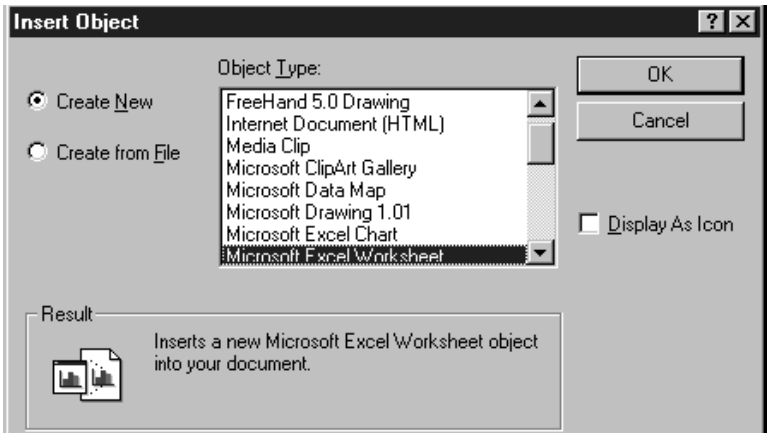
OLE objekat  (*OLE Container*) može povezati ili umetnuti bilo koji objekat, koji podržava automatizaciju. Koristeći ovaj objekat, vaša *Visual Basic* aplikacija može prikazati i upravljati podacima iz drugih *Windows* temeljenih aplikacija, kao što su *Microsoft Excel* i *Microsoft Word*. *OLE* objekat koristi se za stvaranje aplikacije usmjerene na dokumente. U takvoj aplikaciji, korisnik kombinuje podatke iz različitih aplikacija, kako bi stvorio jedan dokument. Takav tip aplikacije može biti uređivač teksta, koji korisniku omogućuje upis teksta te, zatim umetanje *Excel* tabele ili grafikona kao na slici 7.18.

Kada povežete objekat, u svoju aplikaciju ubacujete oznaku mjesta (a ne same stvarne podatke) za povezan objekat. Na primjer, kada povežete niz ćelija *Excel* tabele sa *Visual Basic* aplikacijom, podaci pridruženi ćelijama spremljeni su u drugoj datoteci. U *OLE* objektu nalazi se samo veza prema podacima i slika

podataka. Tokom rada sa vašom *Visual Basic* aplikacijom, korisnik može aktivirati povezani objekat (na primjer dvoklikom na objekat), a aplikacija *Excel* tabele će se automatski pokrenuti. Korisnik zatim može mijenjati te ćelije *Excel* tabele koristeći aplikaciju *Excel* tabele. Kod mijenjanja povezanog objekata, izmjena se obavlja u odvojenom prozoru izvan kontrole *OLE* kontejnera. Kada je objekat povezan sa *Visual Basic* aplikacijom, trenutni podaci objekata mogu se vidjeti iz bilo koje aplikacije, koja sadrži veze sa tim podacima.



Slika 7.18. Povezani dokumenti pomoću OLE objekata



Slika 7.18. Izbor dokumenata za povezivanje sa OLE objektom

Svaki put kada se na formi stvori *OLE* objekat, *Visual Basic* prikazuje dijaloški okvir ***Insert Object*** kao na slici 7.19. Taj dijaloški okvir se upotrebljava za ubacivanje povezanih ili umetnutih objekata tokom izrade aplikacije. Dijaloški okvir *Insert Object* predstavlja listu dostupnih objekata koje možete povezati ili umetnuti u svoju aplikaciju. Postupak povezivanja odvija se kroz sljedeće korake:
1) Postaviti *OLE* objekat na formu. Pojaviće se dijaloški okvir ***Insert Object***. Taj dijaloški okvir se može, takođe, prikazati klikom desnom tipkom miša na *OLE* objekat, te izborom naredbe *Insert Object*.
2) Odaberite dugme izbora ***Create from File***.
3) Odaberite dugme ***Browse***. Pojaviće se dijaloški okvir *Browse* sličan *Windows Explorer*-u.
4) Odaberite datoteku sa kojom se želi povezati *OLE* objekat.
5) Kliknite ***Insert*** za povratak u dijaloški okvir ***Insert Object***.
6) Odaberite kontrolnu kućicu ***Link*** u dijaloškom okviru ***Insert Object***, te odaberite dugme ***OK*** za stvaranje povezanog objekata.

Kad koristite povezani objekat, svaki korisnik koji pokrene vašu aplikaciju mora imati pristup (valjanu putanju) povezanoj datoteci i kopiju aplikacije, koja je stvorila datoteku. Inače, kad se pokrene vaša aplikacija, pojaviće se slika originalnih podataka, ali korisnik neće biti u mogućnosti mijenjati podatke, niti će vidjeti promjene, koje su napravili drugi u povezanim podacima. To može biti važno, ako se vaša aplikacija izvodi u mreži. Ako vaša aplikacija sadrži povezani objekat, moguće je da će podaci tog objekata biti promijenjeni u drugoj aplikaciji, dok se vaša aplikacija ne izvodi. Idući put kad se vaša aplikacija pokrene, promjene u izvoru neće se automatski pojaviti u *OLE* objektu. Kako bi prikazali trenutne podatke u *OLE* objektu, upotrijebite postupak *Update* objekta:

```
ImeOLEObjekta.Update
```

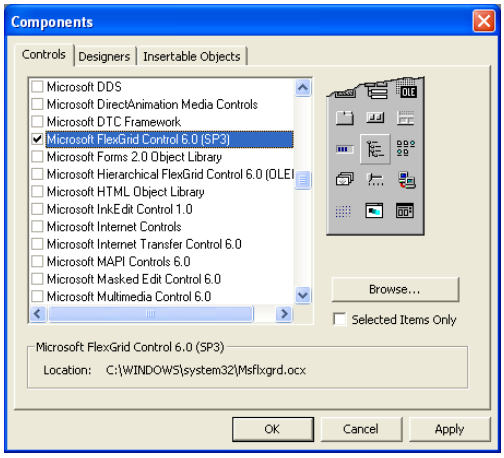
Ako korisnik želi snimiti promjene u povezanom objekatu, mora ih snimiti iz menija *ActiveX* sastavnog dijela. Postupak *SaveToFile OLE* objekta primjenjiv je samo za umetnute objekate.

Drugi način stvaranja objekata tokom izrade aplikacije je korištenje dijaloškog okvira *Paste Special*. Ovaj dijaloški okvir je koristan, samo ako želite upotrijebiti dio datoteke, na primjer, skup ćelija iz *Exsel* tabele, ili odlomak iz *Word* dokumenta. Korišćenje dijaloškog okvira *Paste Special* se vrši kroz sljedeće korake:
1) Pokrenite aplikaciju koja sadrži podatke koje želite povezati ili umetnuti.
2) Odaberite podatke koje želite povezati ili umetnuti.
3) U meniju ***Edit*** ActiveX sastavnog dijela, odaberite ***Copy***. Podaci će se spremiti za kopiranje.
4) U *Visual Basic*-u, kliknite na željeni *OLE* objekat sa desnim tasterom miša, i odaberite naredbu ***Paste Special***.

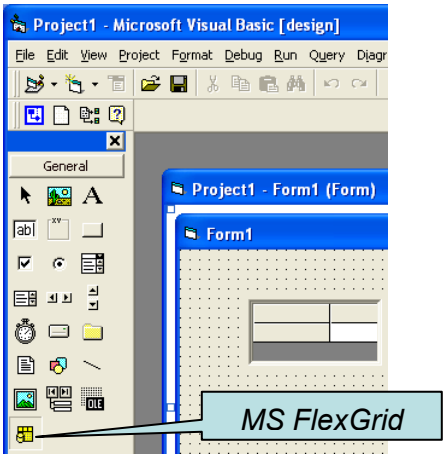
5) Odaberite dugme izbora ***Paste,*** ako želite stvoriti umetnuti objekat, ali koji nije povezan sa izvornim dokumentom. Ako odaberite dugme izbora ***Paste Link*** stvara se povezan objekat. Ako već postoji objekat, koji je umetnut ili povezan, poruka će vas upitati želite li obrisati taj postojeći objekat i stvoriti novi na njegovom mjestu. Odaberite dugme ***OK*** za stvaranje objekata.

7.18. OBJEKAT *MSFLEXGRID* ZA TABELARNI PRIKAZ

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.



Slika 7.20. Izbor novih kontrola

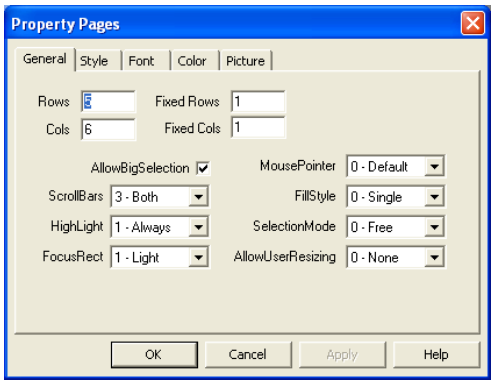


Slika 7.21. Objekat MSFlexGrid

Za tabelarni prikaz podataka *Visual Basic* posjeduje nekoliko objekata, ali je najznačajniji *MSFlexGrid* objekat. Budući da su podaci u relacionim bazama podataka uređeni u obliku dvodimenzionalnih tabela, *MSFlexGrid* objekat može posluziti i za prikazivanje informacija iz takvih baza podataka. *MSFlexGri* tabele, koriste se za tabelarni prikaz raznih podataka, kao i za prikazivanje podataka iz dvodimenzionalnih promjenljivih (dvodimenzioni niz). *MSFlexGrid* objekat se ne nalazi na standardnoj listi objekata (paleta alatki), već ga treba prethodno pronaći i postaviti. Potrebno je prvo klikom miša na meni ***Project*** (na liniji menija) otvariti

listu opcija. Od ponuđenih treba odabrati opciju ***Components***. Nakon toga, pojavljuje se okvir za dijalog sa spiskom svih objekata (kontrola) kao na slici 7.20. Čekiranjem komponente ***Microsoft FlexGrid 6.0*** i klikom na dugmi ***OK***, na paleti objekata pojaviće se objekat ***MSFlexGrid*** kao na slici 7.21. Pomoću dobijenog objekta kreira se, na uobičajen način, tabela *MSFlexGrid* na Formi. Tako dobijena tabela ima samo dva reda i dvije kolone. Potreban broj kolona i redova određuje se upisom vrijednosti u osobine ***Cols*** i ***Rows***. Na slici 7.22. prikazan je izgled prozora u kome se podešavaju osnovne osobine tabele.

Prvi, odnosno nulti, red predviđen je za zaglavlje tabele i prva (nulta) kolona predviđena je za upis imena redova. Ćelije koje predstavljaju zaglavlje su osjenčene sivom bojom, da bi bile lakše uočljive. Ćelije u *MSFlexGrid* tabeli posjeduju indekse, kao i niz dvodimenzionalnih promjenljivih. Prvi indeks označava broj reda, a drugi broj kolone. Indeksi počinju sa 0. Nulta vrsta i kolona mogu se, po potrebi, i rasjenčiti, ako se osobinama *FixedCols* i *FixedRows* dodijele vrijednosti 0, umjesto 1. U tabeli može biti osjenčeno više kolona i redova, ali to nema veliku praktičnu primjenu. Preko prozora za podešavanje osobina tabele se mogu podešavati i druge osobine, kao što su fontovi i boje.



Slika 7.22. Podešavanje osobina tabele

Slika 7.23. Primjer upisa u tabelu

Adresa odgovarajuće ćelije u tabeli određuje se pomoću imena objekta i osobine ***TextMatrix (i, j)***, gdje ***i*** predstavlja broj reda, a ***j*** predstavlja broj kolone, čijim presjecima u tabeli ćelija pripada. Na taj način, svakoj ćeliji u tabeli (matrici) može se dodijeliti željena vrijednost, na primjer:

```
ImeTabele.TextMatrix (2, 3) = "Mirko"
```

što znači da će se u ćeliju, koja se nalazi na presjeku trećeg reda i četvrte kolone upisati tekst "Mirko". Slijedi primjer koda za upis podataka u tabelu pod imenom **MSFlexGrid1**, a rezultat ovog koda je prikazan na slici 7.23.

```
Private Sub Command1_Click()
  MSFlexGrid1.TextMatrix(0, 0) = "r.b."
  MSFlexGrid1.TextMatrix(0, 1) = "Kolona1"
  MSFlexGrid1.TextMatrix(1, 0) = "Red1"
  MSFlexGrid1.TextMatrix(0, 3) = "Ime"
  MSFlexGrid1.TextMatrix(2, 3) = "Mirko"
End Sub
```

Inače, ćelije se mogu birati u kodu i pomoću osobina *Row* i *Col,* na primjer:

```
ImeTabele.Row = 1       'Drugi red
ImeTabele.Col = 2       'Treca kolona
```

Prilikom nabrajanja osobina za jednu ili više ćelija, a koje se odnose na isti objekat (tabelu) može se koristiti zapis:

```
With ImeObjekta
  Osobine
End With
```

Najčešće osobine koje se podešavaju za tako odabrane ćelije, obično su sljedeće:

```
With ImeTabele
•CelFontBold = True (False)      'Tamna slova
•CelFontltalic = True (False)    'Kosa slova
•CelFontUnderline = True (False)'Podvučena slova
•CelFontSize = 12                'Veličina fonta
•CelFontName = "Times"        'Izbor fonta
•CelAlignment = FlexAlignCenterCenter
     'Vertikalno i horizontalno  poravnanje
•CelBackColor = "White"       'Boja pozadine
•CelForeColor = "Blue"        'Boja teksta ili slike
•RowHeight (i) = 1500         'Visina celije
•ColWidth (j) = 2000          'Širina kolone
End With
```

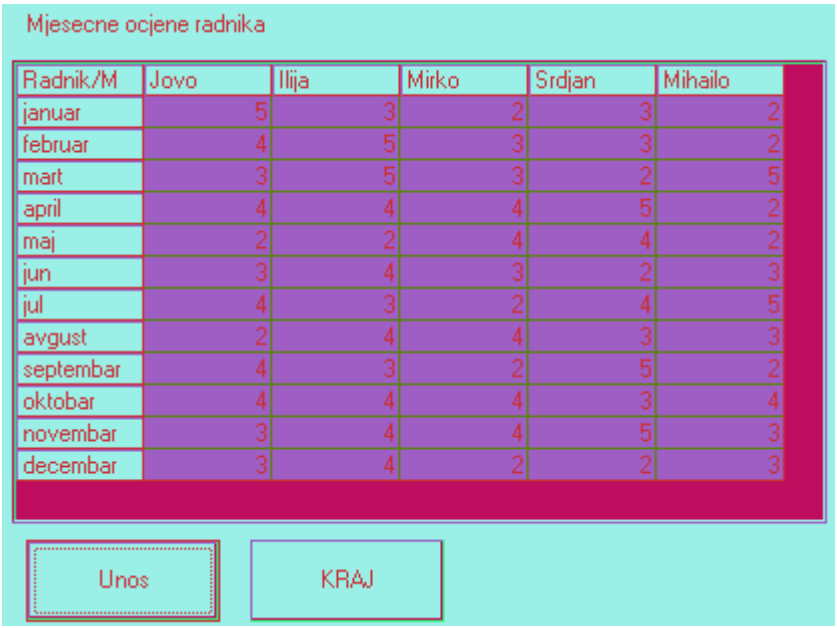
Ako se istovremeno postavljaju osobine za više ćelija, onda se prvo obavlja selekcija ćelija na koje se te osobine odnose, pa se nakon toga definišu osobine. Izbor ćelija vrši se na sljedeći način:

```
With ImeTabele
  •Row = i
  •Col = j
  •RowSel = k
  •ColSel = m
  •FillStyle = FlexFillRepeat
End With
```

gdje su ***(i, j)*** koordinate gornje lijeve ćelije, ***(k, m)*** koordinate donje desne ćelije. Vrijednost osobine *FlexFillRepeat* znači da će se novonavedena osobina odnositi na sve ćelije u okviru pravougaonika sa navedenim dijagonalnim tjemenima.

Karakteristike *MSFlexGrid* tabele (matrice) obično se daju u okviru procedure *FormLoad,* tako da je tabela spremna za prikaz i unos podataka čim se ta forma pojavi na ekranu.

Programski kod za popunjavanje ćelija tabele (**MSFGOcjene**) iz predhodno popunjenog dvodimenzionog niza, u primjeru prikaza mjesečnih ocjena za 5 radnika, imao bi sljedeću formu. Rezultat ovog koda je prikazan na slici 7.24.



Slika 7.24. Program za prikaz mjesečnih ocjena 5 radnika

```
With MSFGOcjene
    .TextMatrix(0, 0) = "Radnik/M"
    .TextMatrix(0, 1) = "Jovo"
    .TextMatrix(0, 2) = "Ilija"
    .TextMatrix(0, 3) = "Mirko"
    .TextMatrix(0, 4) = "Srdjan"
    .TextMatrix(0, 5) = "Mihailo"
    .TextMatrix(1, 0) = "januar"
    .TextMatrix(2, 0) = "februar"
    .TextMatrix(3, 0) = "mart"
    .TextMatrix(4, 0) = "april"
    .TextMatrix(5, 0) = "maj"
```

```
        .TextMatrix(6, 0) = "jun"
        .TextMatrix(7, 0) = "jul"
        .TextMatrix(8, 0) = "avgust"
        .TextMatrix(9, 0) = "septembar"
        .TextMatrix(10, 0) = "oktobar"
        .TextMatrix(11, 0) = "novembar"
        .TextMatrix(12, 0) = "decembar"
        For i = 1 To 12
          For j = 1 To 5
            .TextMatrix(i, j) = Ocjene(i, j)
            j = j + 1
          Next j
        Next i
    End With
```

U radu sa tabelama potrebno je definisati poravnanje sadržaja u ćelijama. Funkcija za poravnanje teksta u koloni tabele je **ColAlignment,** a njena sintaksa glasi

***imeobjekta*.ColAlignment(*number*) [= *value*]**

gdje je

number - redni broj kolone

value - broj koji definiše način poravnanja prema sljedećoj tabeli 7.2.

Tabela 7.2. Način poravnanja teksta u MSFlexGrid Tabeli

Vrijednost	Način poravnanja
0	Lijevo gore (*Left Top*)
1	Lijevo sredina (*Left Center*) (podrazumjevano za tekst)
2	Lijevo dole (*Left Bottom*)
3	Sredina gore (*Center Top*)
4	Sredina sredina (*Center Center*)
5	Sredina dole (*Center Bottom*)
6	Desno gore (*Right Top*)
7	Desno sredina (*Right Center*) (podrazumjevano za brojeve)
8	Desno dole (*Right Bottom*)
9	Opšte: Lijevo sredina za tekst, Desno sredina za brojeve (*General*)

Funkcija za poravnanje teksta u fiksnoj koloni tabele je ***ColAlignmentFixed*,** a njena sintaksa glasi

imeobjekta.**ColAlignmentFixed**(*index*) [=*value*]

gdje je

index - redni broj kolone

value - broj koji definiše poravnanje prema tabeli 7.2.

Broj fiksnih kolona i redova u tabeli se može podešavati direktno preko prozora *Properties*, ili preko koda korišćenjem sljedećih sintaksi:

imeobjekta.**FixedCols**[= *broj*] (Primjer: FG1.FixedCols = 3)
imeobjekta.**FixedRows** [= *broj*] (Primjer: FG1. FixedRows = 2)

Za tabele je često potrebno precizno definisati širinu kolone i visinu reda. Definisanje fiksne širine kolone vrši se pomoću funkcije ***ColWidth***, a visine reda pomoću funkcije ***RowHeight***, koje imaju sljedeću sintaksu:

imeobjekta.**ColWidth(** *number* **)** [= *value*]
imeobjekta.**RowHeight(** *number* **)**[= *value*]

gdje je:
number - redni broj kolone ili reda
value - broj koji definiše širinu/visinu u pikselima

Primjer za 1 i 3 kolonu

```
FG1.ColWidth(0) = 600
FG1.ColWidth(2) = 2000
```

Primjer za 2 i 4 red

```
FG1.RowHeight(1) = 400
FG1.RowHeight(3) = 800
```

Širina jedne kolone se može progmraski sužavati ili proširivati. Primjer koda za sužavanje na polovinu druge kolone tabele pod imenom Fg2.

```
Fg2.ColWidth(1) = Fg2.ColWidth(1) / 2
```

Pomoću funkcije ***SetFocus*** može se pokazivač vratiti na prvu ćeliju tabele. Primjer koda za vraćanje fokus na prvu ćeliju tabele pod imenom Fg1.

```
FG1.SetFocus
```

Pomoću funkcije ***Sort*** može se vršiti sortiranje sadržaja kolone u tabeli, a sintaksa ove funkcije je

imeobjekta.**Sort** [=*value*]

gdje je:
value - broj koji definiše sortiranje(1-rastuće, 2-opadajuće)

Primjer za rastuce sortiranje po trećoj koloni.

```
FG1.Col = 2
FG1.Sort = 1
```

Primjer za opadajuće sortiranje po četvrtoj koloni.

```
FG1.Col = 3
FG1.Sort = 2
```

Dodavanje novog reda u tabeli se vrši pomoću funkcije ***AddItem*****,** koja ima sintaksu

```
imeobjekta.AddItem [ "tekst koji se želi upisati" ]
```

Brisanje sadržaja cijele tabele vrši se pomoću funkcije ***Clear***, koja ima sintaksu

```
imeobjekta.Clear
```

7.19. OBJEKAT *COMMONDIALOG* ZA STANDARDNI MENI

Standardni okvir za dijalog (*CommonDialog* ) pruža standardan skup dijaloških okvira za operacije, kao što su otvaranje i spašavanje datoteka, određivanje opcija za ispis, te izbor boja i pisama. Ovaj objekat takođe ima sposobnost prikaza pomoći pokretanjem mehanizma za *Windows* pomoć.

Standardni okvir za dijalog pruža međusklop između *Visual Basic*-a i naredbi u dinamičkoj biblioteci *Microsoft Windowsa Commdlg.dll*. Kako bi kreirali dijaloški okvir, korištenjem ovog objekta, datoteka *Commdlg.dll* mora se nalaziti u direktorijumu *Windows\System*. Standardni okvir za dijalog se upotrebljava u programu, tako da ga dodate formi i odredite osobine objekta. Dijalog koje se prikazuje pozivom tog objekta, određen je postupcima objekta. Tokom rada programa, kad se pozove odgovarajući postupak, prikazuje se dijaloški okvir ili se izvršava mehanizam pomoći. Tokom izrade programa, objekat standardni okvir za dijalog se prikazuje kao ikonica na formi. Veličina te ikonice je fiksna i ne može se mijenjati. Standardni okvir za dijalog omogućuje vam prikaz sljedećih obično korištenih dijaloških okvira:

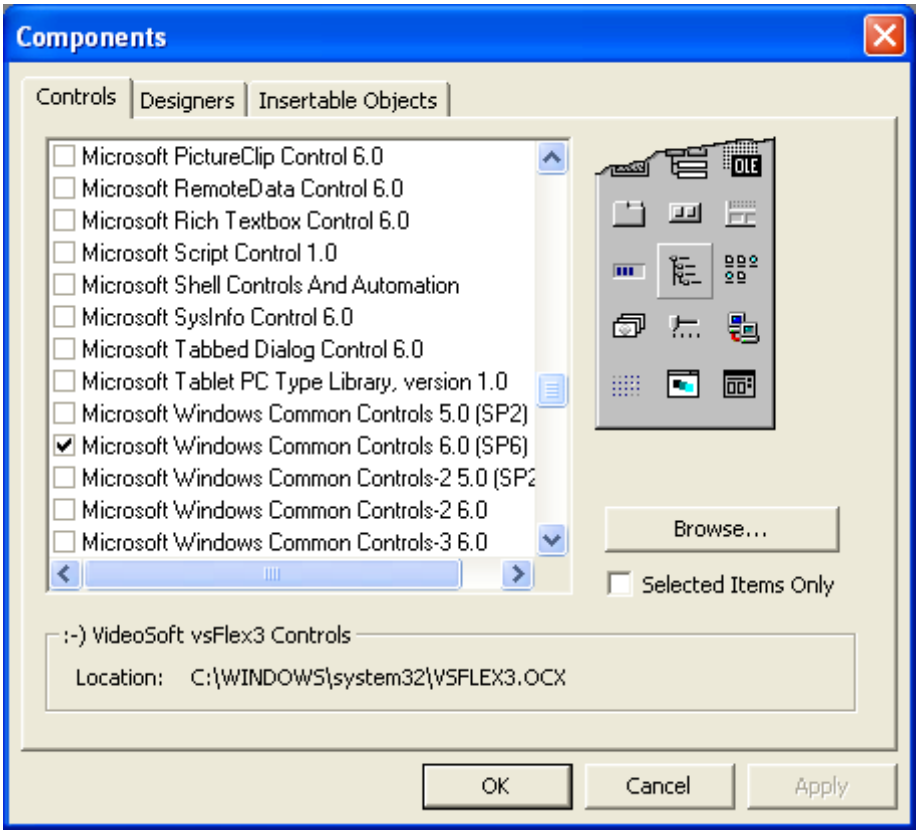
- Open
- Save As
- Color
- Font i
- Print

Objekat standardnog okvira za dijalog se na nalazi u standardnoj paleti objekata *Visuel Basic*-a, već ga je potrebno dodati svaki put, kada se pravi novi program. Postavljanje ovog objekta u standardnu paletu objekata se vrši kroz sljedeće korake:

1) Ako to već niste napravili, dodajte objekat standardnog okvira za dijalog u paletu objekata, izborom stavke ***Components*** u meniju ***Project***. Pojaviće se lista svih objekata, koje posjeduje *Visual Basic*, kao na slici 7.25. Čekiranjem komponente ***Microsoft Common Control 6.0*** na listi u kartici ***Controls*** i klikom na dugme ***OK*** u paleti alatki pojavljuje se objekat ***CommonDialog*** koji se, po potrebi, prenosi na formu. Čekiranjem komponente ***Microsoft Windows Common Control 6.0*** na listi u kartici ***Controls*** i klikom na dugme ***OK*** u paleti objekata pojavljuje se više novih objekata koji se mogu kombinovati sa standardnim okvirom za dijalog. Ti dodatni objekti koji se pojavljuju u paleti objekata su: *TabStrip, Toolbar, Statusbar, Progressbar, Imagelist, Slider* i *ImageCombo*.

2) U paleti objekata, kliknite na objekat ***CommonDialog*** i kreirajte ga na formi. Kada stvarate ovaj objekat na formi, objekat će automatski odrediti svoju veličinu. Kao i objekat mjerača vremena (*Timer*) i objekat standardnog okvira za dijalog je nevidljiv tokom izvršenja programa.

3) Standardni okviri za dijalog se u programu pozivaju pomoću naredbi prikazanih u tabeli 7.3. Ove naredbe se u programu obično pozivaju preko komandnih dugmadi, menija napravljenog pomoću *Menu Editor*-a ili ikonice koju postavimo u vlastitom *ToolBar*-u.



Slika 7.25. Dodavanje novih objekta u paletu alatki

Tabela 7.3. Naredbe u standardnom okviru za dijalog

Red. br.	Dijaloški okvir	Naredba kojom se poziva dijaloški okvir
1.	Open	ShowOpen
2.	Save As	ShowSave
3.	Color	ShowColor
4.	Font	ShowFont
5.	Print	ShowPrint
6.	Windows Help	ShowHelp

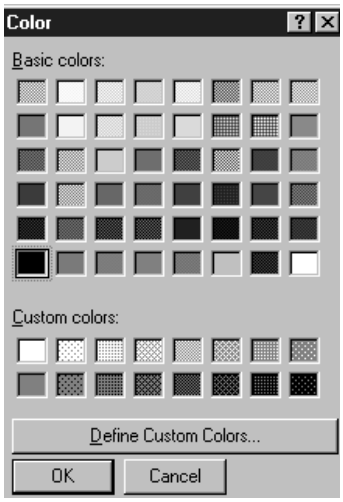
7.19.1. Dijaloški okviri *Open* i *Save As*

Dijaloški okvir *Open* prikazan je na slici 7.26. Omogućuje korisniku određivanje direktorijuma, nastavka imena datoteke (ekstenzije) i imena datoteke. Dijaloški okvir *Save As* 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.
Za dijaloški okvir *Open* 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 (|).

Slika 7.26. Dijaloški okvir Open



Slika 7.27. Dijaloški okvir Color

7.19.2. Dijaloški okvir *Color*

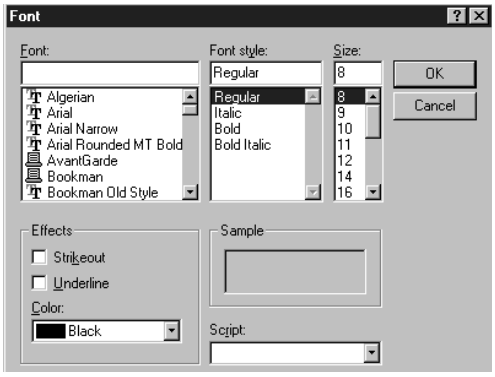
Dijaloški okvir *Color* prikazan je na slici 7.27. Omogućuje korisniku izbor boje iz palete ili stvaranje i izbor korisničke boje. Tokom rada programa, kad korisnik odabere boju i zatvori dijaloški okvir, treba upotrijebiti osobinu *Color* za dobijanje odabrane boje. Sljedeći kod prikazuje korištenje dijaloškog okvira *Color*.

```
' Prikaz dijaloškog okvira Color.
CommonDialog1.ShowColor
' Postavljanje boje pozadine forme na odabranu boju.
Form1.BackColor = CommonDialog1.Color
```

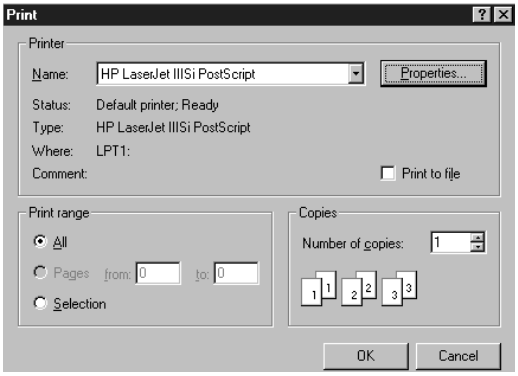
7.19.3. Dijaloški okvir *Font*

Dijaloški okvir ***Font*** prikazan je na slici 7.28. Omogućuje korisniku izbor pisma određivanjem njegove veličine, boje i stila. Jednom kad korisnik napravi izbore u dijaloškom okviru *Font*, sljedeće osobine: *FontName*, *Font style*, *Size* ..., sadrže informacije o izboru korisnika. Sljedeći kod prikazuje korištenje dijaloškog okvira *Font*.

```
Dialog1.ShowFont
Text1.Font.Name = Dialog1.FontName
Text1.Font.Size = Dialog1.FontSize
Text1.ForeColor = CommonDialog1.Color
```



Slika 7.28. Dijaloški okvir Font



Slika 7.29. Dijaloški okvir Print

7.19.3. Dijaloški okvir *Print*

Dijaloški okvir *Print* prikazan je na slici 7.29. 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. Ovaj dijaloški okvir ne šalje stvarno podatke štampaču. Tokom rada programa, kad korisnik

napravi izbor u dijaloškom okviru *Print*, sljedeće osobine sadrže informacije o izborima korisnika.

- *Copies* Broj kopija koje će se ispisati.
- *FromPage* Početna stranica ispisa.
- *ToPage* Posljednja stranica ispisa.
- *Orientation* Orijentacija stranice (uspravna ili položena).

7.19.3. Dijaloški okvir *Help*

Dijaloški okvir *Help* omogućuje korisniku pozivanje fajla, koji služi kao pomoć prilikom rada sa programom. Komanda *ShowHelp* prikazuje ovaj dijaloški prozor.

```
Dialog1.HelpCommand = cdlHelpForceFile
' Određivanje datoteke pomoći.
Dialog1.HelpFile = "c:\Windows\Help\Access.hlp"
' Prikaz mehanizma Windows Help
Dialog1.ShowHelp
```

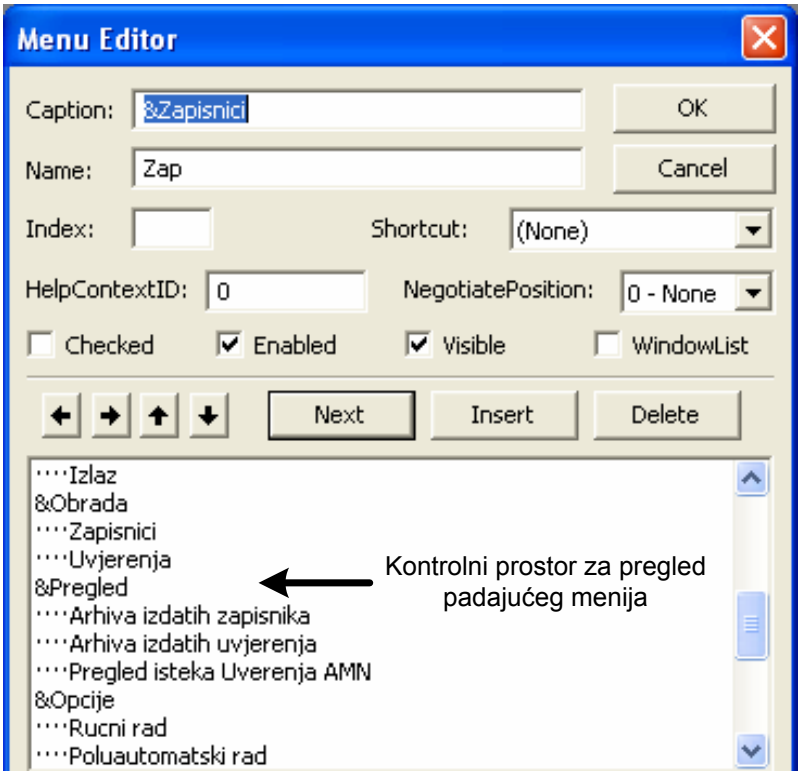
7.20. IZRADA SOPSTVENOG MENIJA

Pored standardnog *Visual Basic* omogućuje da se napravi sopstveni padajući meni na srpskom jeziku. Na taj način program, koji se pravi, će izgledati puno profesionalnije. Vlastiti padajući meni se pravi preko dijaloškog prozora *Menu Editor. Menu Editor* se pokreće pomoću komande *MenuEditor* iz menija *Tools* sa linije menija *Visual Basic*-a. Pozivanjem *Menu Editor-a* otvara se dijaloški prozor prikazan na slici 7.30. Ako se u programu preko *Menu Editor-a* napravi vlastiti padajući meni kao na slici 7.30., pokretanjem programa pojaviće se prozor kao na slici 7.31. *Menu Editor* omogućava dodavanje novih ili modifikaciju i brisanje postojećih menija. Mogu se praviti razne prečice za direktno aktiviranje pojedinih komandi iz menija putem tastature. Nakon formiranja menija pristupa se pisanju programskog koda za eventualno aktiviranje odabranih komandi. Na *Menu Editor* nalazi se niz veoma korisnih objekata za izradu i podešavanje sopstvenog padajućeg menija.

U polje za tekst *Caption* (natpis) upisuju se imena menija, koja će se pojaviti na ekranu, kada se program pokrene. Dok se u polje za tekst *Name* (ime) upisuje ime te komande, preko koga će se pristupati toj komandi u kodu programa. Sadržaj ova dva polja može, ali i ne mora biti isti. Istovremeno, dok se vrši upis naziva menija u polje *Caption*, pojaviće se isti takav natpis na vrhu kontrolnog prostora za pregled padajućeg menija. Na taj način, u ovom polju će se formirati lista (spisak)

svih napravljenih glavnih menija, podmenija i stavki koje su u sastavu podmenija. 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 u MenuEditor-u

Slika 7.31. Izgled vlastitog padajućeg menija i Tool Bar-a u programu

U kontrolnom prostoru za pregled padajućeg menija, po pravilu, zapisuju se natpisi glavnog menija od samog početka (ruba lijeve ivice). Natpisi stavki koje se u okviru menija pomjeraju za četiri karaktera (tačkice) udesno, predstavljaju podmeni glavnog menija, koji se nalazi prvi iznad tog natpisa. Natpisi stavki koje se u okviru menija pomjeraju za još četiri karaktera (tačkice) udesno u odnosu na podmeni, predstavljaju podmeni u podmeniju, koji se nalazi prvi iznad tog natpisa. Na ovaj način se dobije struktura stabla u padajućem meniju.

Vodoravno pomjeranje natpisa vrši se prvo klikom miša (markiranjem) na izabranu stavku, koja se pomjera i zatim na strelicu sa lijevim (⇐) ili desnim (⇒) usmjerenjem.

Aktivnost zamjene mjesta pojedinim stavkama vrši se prvo klikom miša (markiranjem) na izabranu stavku, koja se pomjera i zatim na strelicu nagore (⇑), ako tu stavku želimo da pomjerimo unaprijed za jedno mjesto u padajućem meniju ili na strelicu nadole (⇓), ako se stavka premješta za jedno mjesto nadole.

Ubacivanje nove stavke u meni vrši se tako, da se klikne mišem na stavku ispred koje treba izvršiti unos, a zatim aktivirati dugme ***Insert***. Kao rezultat izvedenih akcija u meniju se pojavljuje prazan programski red, u koji treba unijeti novu komandu istom tehnikom, kao što su uneseni i ostali elementi menija.

Stavka iz menija se briše, ako se prvo markira, a zatim aktivira dugme ***Delete***.

Izbor pojedinih komandi iz menija može se obavljati, pored otvaranja menija klikom miša na komandu i definisanjem odgovarajućih prečice putem simbola dodijeljenih tipkama na tastaturi. To se čini tako što se prilikom upisa natpisa u polje *Caption,* ispred željenog karaktera, stavi simbol (&). Tada se pomoću tastera kome je dodijeljen karakter, ispred koga se nalaz simbol (&), može aktivirati ta komanda. Obično se bira prvo slovo imena elementa iz menija, pod uslovom da imena nemaju ista početna slova. U protivnom bira se neko drugo slovo, koje je karakterističnо za tu stavku. Na primjer, slovo *x* za opciju *Exit* u meniju *File.*

Za naznačene (markirane) stavke iz menija mogu se definisati prečice klikom na strelicu opcije *Shortcut*. Iz razvijene liste ponuđenih direktnih pristupa, najčešće ***Ctrl*** + neko slovo, bira se željeni pristup klikom na odgovarajuću opciju.

Poslije unošenja natpisa u polju *Caption,* pritiskom na tipku ***Tab*** ili ***Enter*** na tastaturi, kursor se premješta u naredno polje *Name.* U polju *Name* se upisuje ime, koje će se koristiti u programskom kodu. Obično se daju ista imena kao i u polju *Caption,* s tim što se ispred imena, prema konvenciji, dodaje prefiks *Mnu,* koji asocira na meni. Uobičajeno je da se u *Visual Basic* jeziku prefiksi dodaju i imenima vezanim za neke druge objekte, kako bi se jasnije razlikovali jedni od drugih i kako bi se ukazalo na njihovu namjenu i ulogu. U narednoj tabeli dat je pregled prefiksa za pojedine objekte.

Tabela 7.3. Prefiksi imena za objekte u Visual Basic-u

Objekat	Prefiks	Objekat	Prefiks
CheckBox	*Chk*	*Image*	*Img*
ComboBox	*Cbo*	*Label*	*Lbl*
CommandButton	*Cmd*	*Line*	*Lin*
CommonDialog	*Cdi*	*ListBox*	*Lst*
Data	*Dat*	*Menu*	*Mnu*
DataCombo	*Dco*	*MSFlexGrid*	*Msf*
DataGrid	*Dgr*	*MSHFlexGrid*	*Mshf*
DataList	*Dls*	*Ole*	*Ole*
DirectoryList Box	*Dir*	*OptionButton*	*Opt*
DriveListBox	*Drv*	*PictureBox*	*Pic*
FileListBox	*Fil*	*Shape*	*Shp*
Form	*Frm*	*TextBox*	*Txt*
Frame	*Fra*	*Timer*	*Tmr*
HorizontalScrollBar	*Hsb*	*VerticalScrolBar*	*Vsb*

Objekat ***Cheked*** u okviru za dijalog *MenuEditor-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.

U nekim momentima obrade programa pojedine stavke (komande) treba učiniti nefunkcionalnim. Na primjer, nepoželjno je koristiti komandu *Close,* ako novi podaci nisu spašeni. Deaktiviranje (isključivanje) pojedinih stavki obavlja se klikom na određene stavke i brisanjem znaka za potvrdu (dečekiranje) u polju ***Enabled***. Trenutno nefunkcionalna komanda prepoznaje se po tome, što je njen zapis znatno bljeđi u odnosu na ostale komande.

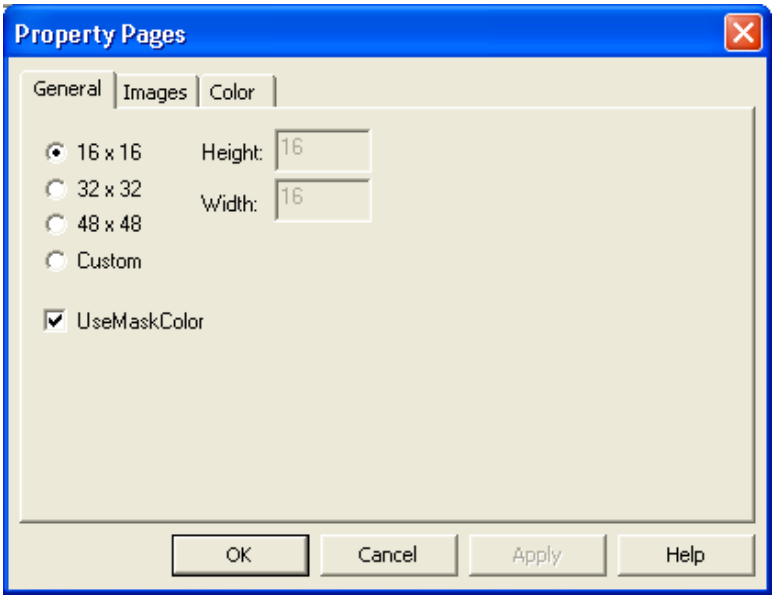
Komanda se može učiniti potpuno nevidljivom, ako se prethodno markira, a potom iz polja ***Visible*** poništi znak za potvrdu.

Poslije unošenja imena menija ili stavke u polja *Caption* i *Name* i poslije izbora opisanih opcija iz okvira za dijalog, slika 7.29, treba kliknuti na dugme ***Next***. Time se potvrđuje unos obrađivanog elementa i automatski prazne pomenuta polja za unos nove stavke. Postupak se, potom, redom nastavlja, sve dok se ne unesu svi elementi menija. Lista i struktura unesenih elemenata data je na dnu u kontrolnom prostoru za pregled menija.

Klikom na dugme ***OK*** konačno se potvrđuje, da je meni uređen i okvir za dijalog nestaje sa ekrana. U programu se tada ispod naslovne linije pojavljuje napravljeni menija. Lista menija dobija se klikom na njegov naslov na liniji menija. Klikom na dugme ***Cancel***, uvijek se može odustati od izrade menija.

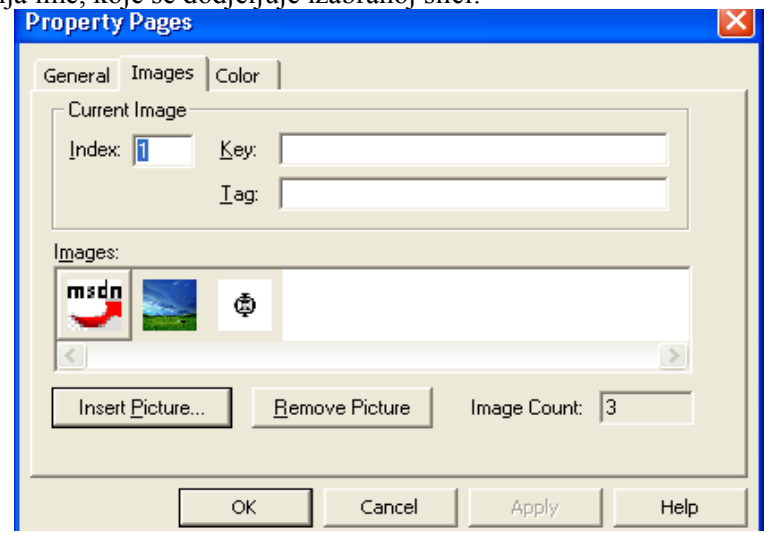
7.21. IZRADA SOPSTVENOG *TOOLBAR-A*

Kada se prave složeni programi često imamo potrebu da napravimo sopstveni *ToolBar* ili *StatusBar*. Ovi objekti postoje u *Visuel Basic*-u, ali se ne nalaze u standardnoj paleti objekata. Da bi dodali ove objekte u paletu objekata, potrebno je u meniju ***Project*** izabrati stavku ***Components.*** Zatim treba čekirati komponentu ***Microsoft Windows Common Control 6.0*** na listi u kartici ***Controls*** i klikom na dugme ***OK*** u paleti objekata pojavljuje se više novih objekata, koji se mogu kombinovati sa standardnim okvirom za dijalog. Ti dodatni objekti koji se pojavljuju u paleti alatki su: *TabStrip, Toolbar, Statusbar, Progressbar, Imagelist, Slider* i *ImageCombo*.
Da bi se u *Toolbar* mogle dodati slike novih ikonica, na formu je potrebno prvo dodati objekat *Imagelist*  . Ovaj objekat se ne vidi kada se program pokrene. U njega se dodaju sve one slike, koje se potencijalno žele prikazati u *Toolbar*-u. Poželjno je da se dodaju ikonice sa ekstenzijom *.ico*. Sve ikonice ne moraju biti iste veličine, pa ih je zato potrebo podesiti. Povešavanje ovih osobina se vrši, tako što se na objektu *Imagelist* desnim klikom miša dođe do osobine *Properties*. Pojavljuje se dijaloški prozor sa tri menija za podešavanje: *General, Images* i *Color*. Dijaloški prozor ***General*** prikazan je na slici 7.32. Koristi se za podešavanje veličine ikonice u pikselima.

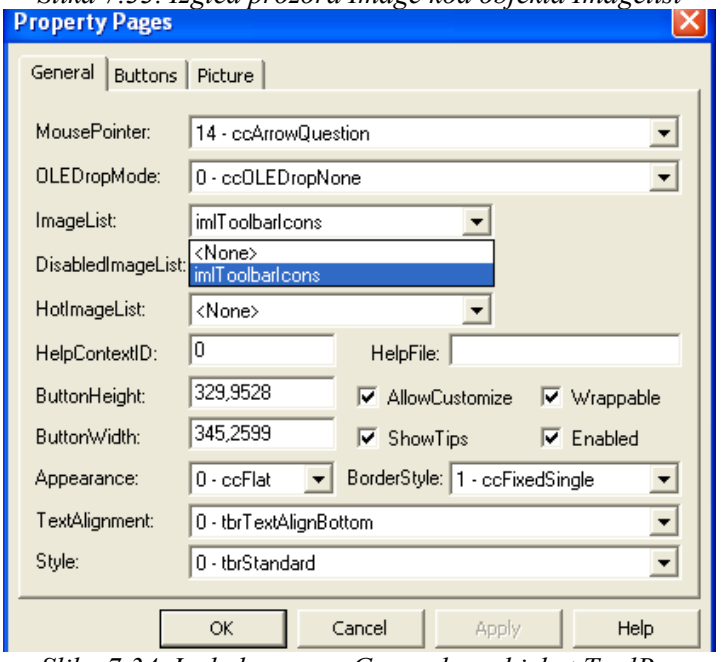


Slika 7.32. Izgled prozora Gener kod objekta Imagelist

Dijaloški prozor ***Images*** prikazan je na slici 7.33. i koristi se za dodavanje novih slika u ovaj objekat. Preko dugmeta ***InsertPicture*** ubacuje se željena slika iz željenog direktorijuma. Polje ***Index*** označava redni broj slike u listi, a ***Key*** predstavlja ime, koje se dodjeljuje izabranoj slici.



Slika 7.33. Izgled prozora Image kod objekta Imagelist



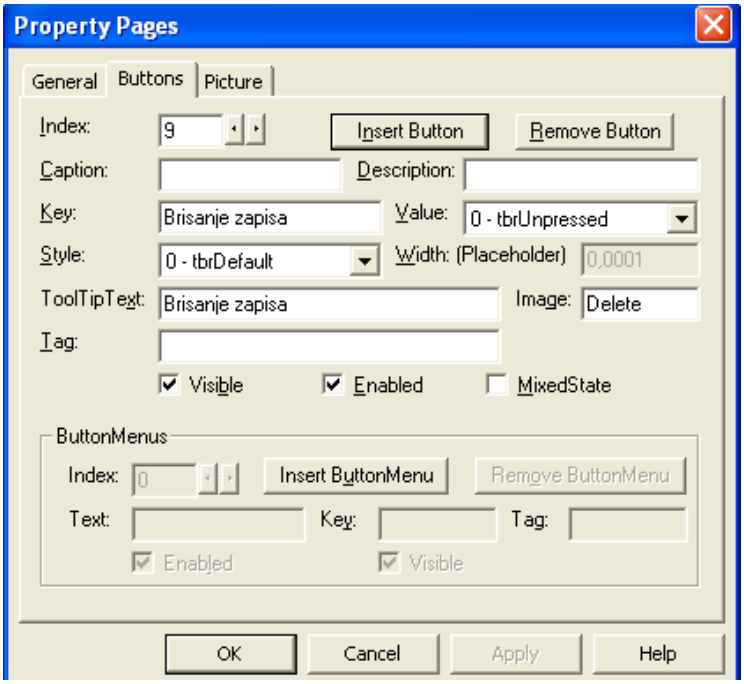
Slika 7.34. Izgled prozora General za objekat ToolBar

Dijaloški prozor ***Color*** se koristi za podešavanje boje maske i pozadine ikonica koje se dodaju u ovaj objekat.

Da bi se u *ToolBar*-u prikazale željene ikonice potrebno je prvo dodati *ToolBar* na željenu formu. Zatim desnim klikom miša na dodati *ToolBar*, izabrati opciju *Properties*, kako bi se moglo izvršiti povezivanje *ToolBar*-a sa nekim od objekata *ImageList*. Pojavljuje se dijaloški prozor sa tri menija za podešavanje: *General, Buttons* i *Picture*.

Dijaloški prozor ***General*** prikazan je na slici 7.34. Koristi se za povezivanje sa željenim objektom *ImageList* preko njegovog imena. To se vrši preko parametra ***Image List,*** koji je organizovan kao *ComboBox*. U *ComboBox*-su se pojavljuje lista svih objekata *ImageList*, koji postoje na trenutno aktivnoj formi. Ostali parameti na ovom prozoru se koriste za fino podešavanje ikonica.

Dijaloški prozor ***Buttons*** prikazan je na slici 7.35. Koristi se za izbor ikonica iz objekta *ImageList,* koje se žele prikazati u *ToolBar*-u. To se vrši preko parametra ***Index*** i pomoću dugmeta ***Insert Button.*** Preko parametra ***Image*** unosi se ime ikonice. Parametar ***Key*** se koristi za unos teksta, koji će se koristiti za definisanje akcije nad tom ikonicom u kodu. U osobinu ***ToolTipText*** se unosi tekst, koji će pisati iznad ikonice, kada se preko nje pređe mišem, prilikom pokretanja programa.



Slika 7.35. Izgled prozora Buttons za objekat ToolBar

7.22. *ACTIVEX* OBJEKATI

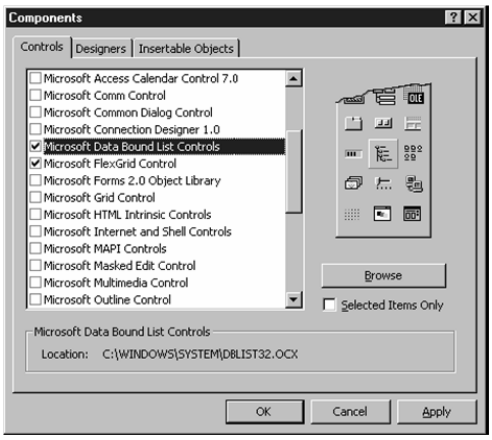
Komplet objekata dostupnih u paleti objekata, može se prilagoditi svakom projektu. Svaki objekat se mora nalaziti u paleti objekata, prije nego što je možete dodati na formu projekta. U ovom poglavlju biće predstavljeni objekti, koji se ne nalaze u standardnoj paleti objekata programskog jezika *Visual Basic*. Međutim, radi se o vrlo korisnim objektima, koji omogućuju pravljenje profesionalnih aplikacija. Pomoću ovih objekata se olakšava pristup bazama podataka, ali i svim drugim programima, koji su instalirani na računaru.

ActiveX objekte (kontrole) i dodatne objekate možete pripojiti svom projektu postavljajući ih u paletu objekata. Dodavanje ovih objekata u paletu objekata programa vrši se kroz sljedeće korake.

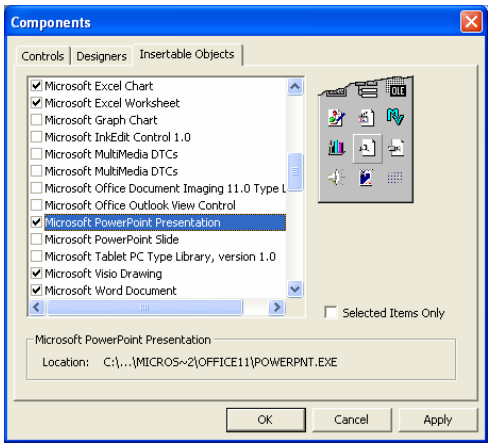
1) U meniju ***Project*** odaberite stavku ***Components***. Prikazaće se dijaloški okvir ***Components*** kao na slici 7.36. Stavke ispisane u ovom dijaloškom okviru uključuju sve registrirane *ActiveX* objekte, dodatne objekte i *ActiveX* kreatore.

2) Da bi dodali objekat (datoteke sa nastavkom **.ocx**) u paletu objekata, potvrdite ček-boks lijevo od imena objekta. Odaberite karticu ***Controls***, kako bi vidjeli kontrole sa nastavkom **.ocx** u imenu datoteke. Za dodavanje dodatnih objekata, kao što je *Microsoft Excel Chart*, odaberite karticu ***InsertableObjects*****,** kao na slici 7.37.

3) Odaberite dugme **OK** za zatvaranje dijaloškog okvira ***Components***. Svi *ActiveX* objekti koje ste odabrali, pojaviće se u paleti objekata.



Slika 7.36. Dijaloški okvir Components



Slika 7.37. Dodatni objekti

Za dodavanje *ActiveX* objekata u dijaloški okvir ***Components***, odaberite dugme ***Browse*** i pretražite druge direktorije, kako bi pronašli datoteke s nastavkom **.ocx**. Kad dodate *ActiveX* objekat na listu raspoloživih objekata, *Visual Basic* automatski potvrđuje kontrolnu kućicu. Svaki *ActiveX* objekat ima prateću datoteku sa

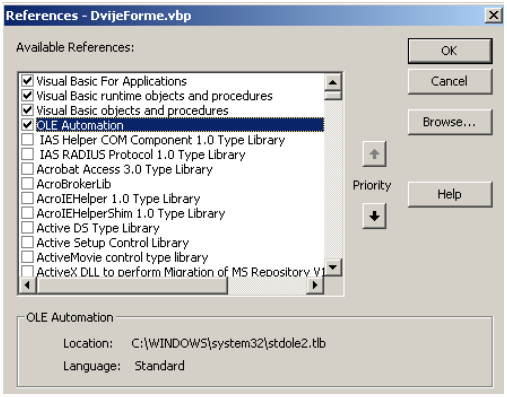
nastavkom **.oca**. Ta datoteka čuva podatke potrebne za komunikacikju *Visual Basic*-a sa svakim objektom. Datoteke sa nastavkom **.oca** obično su snimljene u istom direktoriju gdje su i *ActiveX* objekti i stvaraju se prema potrebi (veličina datoteke i datum stvaranja mogu se mijenjati).

7.22.1. Korištenje objekata drugih aplikacija

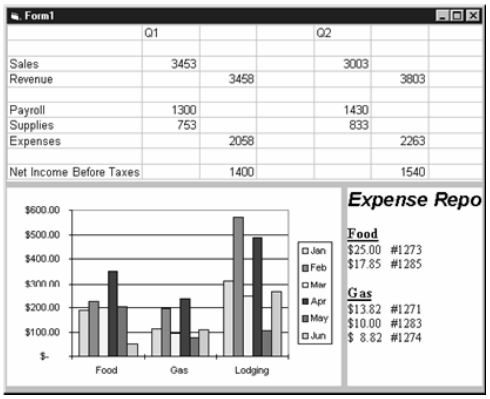
Možete također koristiti i objekate iz drugih aplikacija, kao što su oni uključeni u biblioteku objekata *Microsoft Excela*, kao objekte u paleti alatki ili kao programabilne objekate u programskom kodu. Da bi objekat druge aplikacije bio pristupačan vašem programskom kodu, ali ne kao objekat, potrebno je da postavite pokazivač na biblioteku objekata te aplikacije. Dodavanje pokazivača na biblioteke objekata vrši se kroz sljedeće korake.

1) U meniju ***Project*** odaberite stavku ***References***. Pojavljuje se dijaloški okvir ***References,*** kao na slici 7.38. Potvrdite ček-boks pored pokazivača, koji želite dodati svom programu. Za dodavanje pokazivača na aplikacije, koje nisu na listi dijaloškog okvira ***References***, odaberite dugme ***Browse***, pa odaberite aplikaciju.

2) Odaberite dugme **OK** za dodavanje odabranog pokazivača svom projektu.



Slika 7.38. Dijaloški okvir References



Slika 7.39. Dodatni objekti

Ako se ne koristite neki objekat u biblioteci pokazivača, potrebno je odznačiti kontrolnu kućicu za taj pokazivač, kako bi smanjili broj pokazivača na objekate koje *Visual Basic* mora riješiti. Na taj način ćete smanjiti vrijeme, koje će biti potrebno vašem programu za prevođenje. Jednom kad postavite pokazivače na biblioteke objekata koje želite, određeni objekat, njegove postupke i osobine možete pronaći u pretraživaču objekata izborom stavke *Object Bro*wser u meniju ***View***. U vašem programskom kodu možete iskoristiti svaki objekat, koji je izlistan u pretraživaču objekata.

Visual Basic pruža alate, koji vam omogućuju kombinovanje objekata iz različitih izvora. Sad možete graditi korisnička rješenja kombinujući najmoćnije osobine *Visual Basic*-a i aplikacija koje podržavaju automatizaciju (do sad znanu kao OLE automatizaciju). Automatizacija je osobina komponentnog objekatnog modela (*Component Object Model*, COM), industrijskog standarda koji koristi aplikacija za izlaganje objekata razvojnim alatima i drugim aplikacijama.

Aplikacije možete izgraditi povezivanjem ugrađenih objekata *Visual Basica*, a možete i koristiti objekate koje pružaju druge aplikacije. Razmislite o postavljanju sljedećih objekata na formu *Visual Basica*:

- Objekat grafikona *Microsoft Excela (Chart);*
- Objekat radnog lista *Microsoft Excela (Workshet);*
- Objekat dokumenta *Microsoft Worda.*

Ove objekate možete upotrijebiti za stvaranje aplikacije, koja će raditi kao knjiga računa, što je prikazano na slici 7.39. To će vam značajno uštediti vrijeme, jer ne morate pisti programski kod za obezbjeđenje fukcija, koje daju objekati *Microsoft Excel*-a i *Microsoft Word*-a.

7.22.2. Tipovi *ActiveX* sastavnih dijelova

Sastavni dijelovi tipa *ActiveX* daju vam snagu za sastavljanje usavršenih aplikacija, iz dijelova koji već postoje. Vaše *Visual Basic* aplikacije mogu uključivati nekoliko tipova *ActiveX* sastavnih dijelova:

- Aplikacije koje podržavaju *ActiveX* tehnologiju, kao *Microsoft Exc*el, *Microsoft Word* i *Microsoft Access*, pribavljaju objekate kojima možete programski upravljati iz svoje *Visual Basic* aplikacije. Na primjer, u vašoj aplikaciji možete upotrijebiti osobine, postupke i događaje tabele *Microsoft Excela*, dokumenta *Microsoft Worda* ili baze podataka *Microsoft Acc*essa.
- Sastavni dijelovi koda pružaju biblioteke programabilnih objekata. Na primjer, sastavni dio koda može uključivati biblioteku specijalizovanih financijskih funkcija za korisnike *Excel* tabele, ili elemente korisničkog okruženja, kao dijaloški okviri, koji su zajednički mnogim aplikacijama.

Za razliku od objekata u aplikaciji koja omogućuje ActiveX tehnologiju, objekat u sastavnom dijelu koda može raditi u istom procesu kao vaša aplikacija, omogućujući brži pristup objekatu. Možete dodati osobine bez potrebe da ih sami stvarate, korištenjem *ActiveX* objekata kao sastavnih dijelova. *ActiveX* objekti su dostupni od velikog broja proizvođača i pružaju puno specijalizovanih osobina, kao prikazivanje kalendara na formi ili čitanje podataka posebnog oblika.

Slijedi primjer programa koji na formi ima tri tekst-boksa za unost teksta (Text1, Text2 i Text3) i komandno dugme (Command1). U ovom programu je potrebno dodali pokazivač na biblioteku *Microsoft Excel 8.0 Object Library*. Nakon toga možete dodati kod potprogramu događaja *Command1_Click* komandnog dugmeta koji koristi postupak *Microsoft Excel Formula* za sabiranje dva broja upisana u tekst-boks Text1 i Text2, prikazujući rezultat u trećem tekst-boksu Text3.

```
Private Sub Command1_Click()
Rem Definisanje promjenjivih objekata za program Excel,
Rem radnu knjigu te aplikacije i objekate radnog lista.
Dim xlApp As Excel.Applicaton
Dim xlBook As Excel.Workbook
Dim xlShet As Excel.Workshet
  Rem Dodjela pokazivača objekata promjenjivim.
Set xlApp = New Excel.Application
  Rem Upotrijebite postupke Add za stvaranje
  Rem novih objekata radne knjige i radnog lista.
Set xlBook = xlApp.Workbooks.Add
Set xlShet = xlBook.Workshets.Add
  Rem Dodjela vrijednosti unesenih u tekst-boksove
  Rem ćelijama Microsoft Excela.
xlShet.Cells(1, 1).Value = Text1.Text
xlShet.Cells(2, 1).Value = Text2.Text
  Rem Upotreba postupka Formula za sabiranje
  Rem vrijednosti u Microsoft Excelu.
xlShet.Cells(3, 1).Formula = "=R1C1 + R2C1"
  Rem Prenos vrijednosti iz Excela u tekst-boks.
Text3.Text = xlShet.Cels(3, 1)
  Rem Snimanje radnog lista na određenu lokaciju.
xlShet.SaveAs "c:\Temp.xls"
  Rem Zatvaranje radne knjige.
xlBook.Close
  Rem Zatvaranje Microsoft Excela postupkom Quit.
xlApp.Quit
  Rem Otpuštanje objekata.
Set xlApp = Nothing
Set xlBook = Nothing
Set xlShet = Nothing
End Sub
```

7.22.3. Stvaranje pokazivača na objekat

Prije nego što u svom programu možete koristiti osobine, postupke i događaje objekata, prvo morate odrediti promjenjivu objekata, te zatim promjenjivoj dodijeliti pokazivač objekata. Dodijela pokazivača na objekte zavisi o dva uslova:

- **Isporučuje li *ActiveX* sastavni dio tipsku biblioteku**. Tipska biblioteka *ActiveX* sastavnog dijela sadrži definicije svih objekata, koje pruža sastavni dio, uključujući definicije svih dostupnih postupaka, osobina i događaja. Ako *ActiveX* sastavni dio pruža tipsku biblioteku, u svoj projekat *Visual Basic*-a trebate dodati pokazivač na tipsku biblioteku, prije nego što možete upotrebljavati biblioteke objekata.
- **Je li riječ o objekatu najvišeg nivoa**, objekatu stvorenom izvana (*externally creatableobject*) ili zavisnom objekatu (*dependent object*).

Pokazivač vanjski stvorenom objekatu možete dodijeliti direktno, dok se pokazivači na zavisne objekate dodjeljuju posredno.

Ako je objekat stvoren **izvana**, pokazivač objekata se može dodijeliti promjenjivoj korištenjem ključne riječi ***New***, te funkcija ***CreateObject*** ili ***GetObject*** u izrazu ***Set*** izvan sastavnog dijela.

Ako je objekat, **zavisan objekat**, pokazivač objekata se dodjeljuj korištenjem postupka iz objekata višeg nivoa u izrazu ***Set***.

Postupak stvaranja pokazivača na objekat definisan u tipskoj biblioteci odvija se kroz sljedećih 6 koraka.

1) U meniju ***Project*** odaberite ***References***.
2) U dijaloškom okviru ***References***, odaberite ime *ActiveX* sastavnog dijela, koji sadrži objekate koje želite upotrijebiti u svojoj aplikaciji.
3) Možete upotrijebiti dugme ***Browse***, za traženje datoteke tipske biblioteke, koja sadrži potreban objekat. Tipske biblioteke imaju datoteke sa nastavcima imena **.tlb** ili **.olb**. Izvršne datoteke (.exe) i dinamički povezive biblioteke (.dll), također mogu pružiti tipske biblioteke, pa možete potražiti i datoteke sa tim nastavcima imena. Ako niste sigurni podržava li program *ActiveX* tehnologiju i pruža li tipsku biblioteku, pokušajte dodati pokazivač na nju korištenjem dugmeta ***Browse***.
4) U meniju ***View***, odaberite ***Object Browser***, kako bi vidjeli pokazanu tipsku biblioteku. Odaberite odgovarajuću tipsku biblioteku iz liste ***Project/Library***. U svom programu možete upotrijebiti sve objekate, postupke i osobine ispisane u pretraživaču objekata.
5) Odredite promjenjivu objekata klase objekata. Na primjer, mogli bi odrediti promjenjivu klase *Excel.Chart*, za upućivanje na objekat *Microsoft Excel Chart*.

```
Dim xlChart As Excel.Chart
```

6) Dodijelite pokazivač objekata promjenjivoj korištenjem ključne riječi ***New***, funkcija ***CreateObject*** ili ***GetObject*** u izrazu ***Set***.

Dodjela pokazivača objekta promjenjivoj

Nakon što odredite promjenjivu objekta, morate dodijeliti pokazivač objekta promjenjivoj, prije nego što možete upotrebljavati osobine, postupke i događaje objekta. Novi pokazivač objekta možete dodijeliti na nekoliko načina:

- Ako odredite promjenjivu korištenjem ključne riječi ***New***, *Visual Basic* će automatski stvoriti novi pokazivač objekta, kada prvi put upotrijebite promjenjivu.
- Pokazivač na novi objekt možete dodijeliti u izrazu ***Set***, korištenjem ključne riječi ***New*** ili funkcije ***CreateObject***.
- Pokazivač na novi ili postojeći objekt možete dodijeliti u izrazu ***Set***, korištenjem funkcije ***GetObject***.

Dodjela pokazivača objekta korištenjem ključne riječi *New*

Ako odredite promjenjivu objekta sa ključnom riječi ***New***, *Visual Basic* će automatski stvoriti novi objekt, kada prvi put upotrijebite promjenjivu. U izrazu ***Set*** možete također upotrijebiti ključnu riječ ***New***, za dodjelu pokazivača novom objektu specifične klase.

U sljedećem primjeru prikazan je kod programa, koji pokazuje dodjelu pokazivača na novi objekt DAO tabele promjenjivoj **tdfDatum**, postavljajući osobinu *Name* tabele na "Datum":

```
Dim tdfDatum As DAO.TableDef
Set tdfDatum = New DAO.TableDef
tdfDatum.Name = "Datum"
```

Dodjela pokazivača objekta korištenjem funkcije *CreateObject*

Ova dodjela pokazivača objekta korišćenjem funkcije *CreateObject* odvija se preko sljedeće sintakse.

```
Set promjenjiva = CreateObject("progID", ["imekorisnika"])
```

Argument ***progID*** je obično potpuno označeno ime klase, iz koje je stvoren objekat (na primjer *Word.Document*). Međutim, argument *progID* može biti različit od imena klase. Na primjer, *progID* za objekat *Microsoft Excela* je "Sheet" prije nego "Worksheet".

Neobavezni argument ***imekorisnika*** može biti određen, za stvaranje objekta na udaljenom računaru putem mreže. On je dio imena računara u imenu dijeljenja. Na primjer, sa mrežnim dijeljenim imenom
\\ MojKorisnik \Javno, argument *imekorisnika* bio bi "MojKorisnik".

Sljedeći primjer programskog koda pokreće *Microsoft Excel* (ako *Microsoft Excel* već nije pokrenut) i uspostavlja promjenjivu xlApp za upućivanje na objekt

klase *Application*. Argument "Excel.Application" potpuno označava klasu *Application*, kao klasu koju određuje *Microsoft Excel*.

```
Dim xlApp As Excel.Application
Set xlApp = CreateObject("Excel.Application")
```

Dodjela pokazivača objekata korištenjem funkcije *GetObject*

Funkcija *GetObject* se najčešće koristi za dodjeljivanje pokazivača na postojeći objekat. Ali možete je također upotrijebiti i za dodjeljivanje pokazivača na novi objekat. Kako bi dodijelili pokazivač na postojeći objekat, upotrijebite sljedeću sintaksu.

```
Set promjenjivaobjekata = GetObject([imeputanje] [, progID])
```

Argument ***imeputanj**e* može biti putanja do postojeće datoteke, prazan string ili se može potpuno izostaviti. Ako je izostavljena, argument *progID* je obavezan. Određivanje putanje do postojeće datoteke uzrokovaće stvaranje objekata funkcijom *GetObject* uz korištenje informacija spremljenih u datoteci. Korištenje praznog stringa kao prvog argumenta, uzrokovaće djelovanje funkcije *GetObject* kao funkcije *CreateObject* i stvoriće novi objekat klase, čiji je programski identifikator jednak argumentu *progID*.

Sljedeći primjer koda opisuje rezultate korištenja funkcije *GetObject*.

1) Ako se izvodi *ActiveX* sastavni dio rezultat

```
Set X = GetObject( "MojPos.Application")
   Rem X upućuje na postojeći objekat Application.
Set X = GetObject("", "MojPos.Object")
   Rem X upućuje na novi, izvana stvoren objekat.
```

2) Ako se ne izvodi ActiveX sastavni dio rezultat

```
Set X = GetObject( "MojPos.Object")
   Rem Vraća se pogreška.
Set X = GetObject("", "MojPos.Object")
   Rem Pokreće se ActiveX sastavni dio (MojPos), te
X ukazuje na novi objekat.
```

Primjer koda kada promjenjiva *wdApp* ukazuje na aplikaciju *Microsoft Word*, koja se izvodi:

```
Dim wdApp As Word.Application
Set wdApp = GetObject("", "Word.Application")
```

Isto kao kod funkcije *CreateObject*, argument "Word.Application" je programski identifikator za klasu *Application*, određenu *Microsoft Word*-om. Ako

se izvodi više dokumenata *Microsoft Word*-a, ne možete predskazati na koji će dokument ukazivati promjenjiva *wdApp*.

Primjer koda za pokretanje *Excel* stranice, ako *Excel* već nije pokrenut, tako što će se otvoriti dokument "Proba.xls" na određenoj lokaciji hard diska.

```
Dim xlBook As Excel.Workbook
Set xlBook = GetObject("C:\Prvi\Proba.xls")
```

7.22.4. Otpuštanje *ActiveX* sastavnog dijela

Kad ste gotovi sa korištenjem objekata, očistite sve promjenjive, koje pokazuju na objekat, kako bi objekat mogao biti otpušten iz memorije. Kako bi očistili promjenjivu objekata, postavite varijablu na *Nothing*.

```
Dim acApp As Access.Application
Set acApp = New Access.Application
MsgBox acApp.SysCmd(acSysCmdAccessVer)
Set AcApp = Nothing
```

Ako želite da promjenjiva zadrži svoju vrijednost kroz potprograme, upotrijebite javne promjenjive ili promjenjive na nivou forme, ili stvorite potprograme, koji vraćaju objekat. Sljedeći kod pokazuje, kako bi mogli upotrijebiti javnu promjenjivu.

```
Public wdApp As Word.Application
' Stvaranje objekata Word i pokretanje Microsoft Worda.
Set wdApp = New Word.Application
' Microsoft Word se neće zatvoriti sve dok se
aplikacija ne završi ili dok pokazivač ne bude
postavljen na Nothing.
Set wdApp = Nothing
```

Također, pripazite da postavite sve pokazivače objekata na *Nothing*, kad završavate sa korišćenjem objekata, čak i za nezavisne objekate.

```
Dim xlApp As Excel.Application
Dim xlBook As Excel.Workbook
Set xlApp = New Excel.Application
Set xlBook = xlApp.Workbooks.Add
Set xlApp = Nothing
' Pripazite! xlBook bi još uvijek
' mogao sadržavati pokazivač objekata.
Set xlBook = Nothing
' Sad su očišćeni svi pokazivači.
```

7.22.5. Rukovanje greškama tokom rada u *ActiveX* sastavnim dijelovima

Programski kod za rukovanje greškama je posebno važan, kad radite sa *ActiveX* sastavnim dijelovima, jer se kod, u sastavnom dijelu, koristi iz vaše *Visual Basic* aplikacije. Gdje je to moguće, trebali bi uključiti programski kod za rukovanje greškama, koje može izazvati sastavni dio. Na primjer, dobra je praksa potražiti grešku, koja se pojavljuje kad korisnik neočekivano zatvori aplikaciju sastavnog dijela.

```
Function PokreniWord()
' Pokretanje Microsoft Worda.
On Error Goto ZamkaZaGresku
' Određivanje promjenjive tipa Application za Microsoft
Word, te cjelobrojne promjenjive za zamku pogreške.
Dim wdApp As Word.Application
Dim iPokusaja As Integer
' Dodjela pokazivača objekata.
Set wdApp = New Word.Application
' Otpuštanje varijable objekata.
Set wdApp = Nothing
Exit Function

ZamkaZaGresku:
' Zamka za grešku koja se pojavljuje ako
' Microsoft Word ne može biti pokrenut.
Select Case Err.Number
Case 440 ' Greška automatizacije.
iPokusaja = iPokusaja + 1
' Dozvoljavanje najviše 6 pokušaja pokretanja Worda.
If iPokusaja < 5 Then
  Set wdApp = New Word.Application
  Resume
  'kontinualno izvršenje suspendovanog programa
Else
  Err.Raise Number=VBObjectError + 28765, _
  Description= "Ne mogu pokrenuti Word"
End If
Case Else
Err.Raise Number= Err.Number
End Select
End Function
```

Ako se u prethodnom primjeru pojavi bilo koja osim greške 440, potprogram prikazuje grešku i oživljava grešku. Aplikacija koja pruža objekat može nazad proslijediti svoju vlastitu grešku. U nekim slučajevima, aplikacija može upotrijebiti isti kod greške, koji *Visual Basic* koristi za drugu grešku. U takvim slučajevima, trebate upotrijebiti izraz ***On Error Resume Next***. Zatim provjeriti greške odmah nakon svake linije, koja može uzrokovati grešku. Takav tip provjeravanja greške naziva se **ugrađeno rukovanje greškom** (*inline error-handling*).

7.23. NIZ OBJEKATA

Na jednoj formi može biti više objekata istog tipa. Svakom novom objektu koji se dodaje na formu *Visual Basic* sam dodjeljuje ime, počev od 1 pa nadalje, prema redoslijedu njihovog postavljanja na formu. Prilikom davanja imena objekata nikada se nemože desiti da dva objekta imaju isto ime. Budući da jedan projekat može da sadrži više formi potrebno je jednoznačno imenovati i pojedine forme jer forme međusobno, takođe čine jedan niz, čijim članovima se pridružuju brojevi počev od 1 pa nadalje, prema redoslijedu njihovog implementiranja u projekat.

Niz objekata je grupa objekata, koji dijele isto ime i tip. Oni također dijele i iste potprograme događaja. Niz objekata ima barem jedan element i može se povećavati do onoliko elemenata koliko dopuštaju vaši sistemski izvori i memorija. Najveći indeks koji možete koristiti u nizu objekata je 32767. Elementi istog niza objekata imaju vlastite vrijednosti osobina. Uobičajena upotreba niza objekata uključuje grupiranje objekata za unos i prikaz podataka, kao i objekata za izbor.

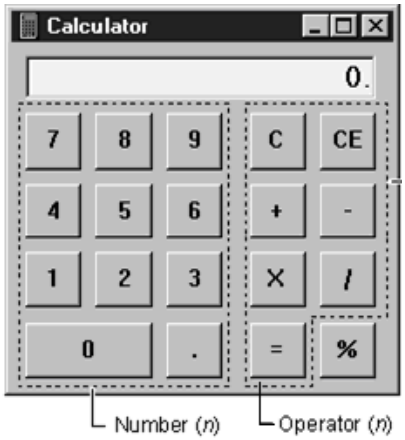
Niz objekata je koristan, ako želite da više objekata dijeli isti programski kod. Na primjer, ako su tri komandna dugmeta kreirana kao niz objekata, izvršiće se isti programski kod bez obzira na koje je dugme kliknuo korisnik. Sa nizom objekata, svaki novi element nasljeđuje zajedničke potprograme događaje tog niza. Korištenjem mehanizma niza objekata, svaki novi objekat nasljeđuje zajedničke potprograme događaja, koji su već napisani za niz. Na primjer, ako vaša forma ima nekoliko *TextBox* objekata, gdje svaki od njih može primiti vrijednost u obliku datuma, niz objekata može biti podešena tako da svi *TextBox* objekati dijele isti programski kod za provjeru ispravnosti unešenih podataka.

Primjer programa *Calculator* (koji se nalazi u *Help*-u na direktorijumu *Samples*) prikazan na slici 7.40. Sadrži dva niza objekata kontrolno dugme - dugme sa brojevima i dugme sa operatorima. Niz za brojeve ima 11 elemenata od *Number(0)* do *Number(10)*, a niz operatora ima 7 elemenata od *Operator (0)* do *Operator(6)*.

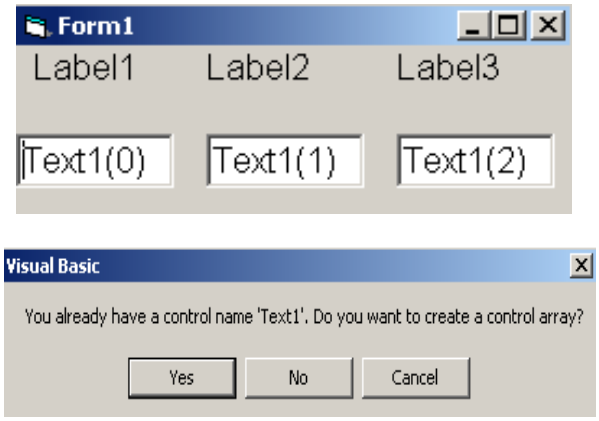
Svaki objekat se poziva sintaksom *objekat(indeks)*. Indeks objekta određuje redoslijed kada je stvaren. Zapravo, određivanje indeksa objekta tokom izrade čini taj objekat dijelom niza. Osobina *Index* razlikuje jedan element niza objekata od drugog. Kad jedan od objekata u nizu prepozna događaj, *Visual Basic* poziva

zajednički potprogram događaja i prosljeđuje argument (vrijednost osobine *Index*) za identifikaciju, koji je objekat niza zapravo prepoznao događaj. Na primjer, prva linija potprograma događaja *Number_Click* je:

```
Private Sub Number_Click(Index As Integer)
```



Slika 7.40. Dva niza objekata



Slika 7.41. Pravljenje niza objekata

Ako objekat Number(0) prepozna događaj, *Visual Basic* prosljeđuje 0 kao argument *Index*. Ako događaj prepozna objekat Number(1), *Visual Basic* će proslijediti 1 kao argument *Index*. Osim vrijednosti indeksa, ostatak programskog koda potprograma *Number_Click*, koji će se izvršiti, jednak je za sve objekte od Number(0) do Number(9).

Postoje tri načina stvaranja niza objekata tokom izrade aplikacije:

1) Dodijelite isto ime za više od jednog objekta.
2) Kopirajte postojeći objekat i zatim ga nalijepite na formu.
3) Postavite vrijednost osobine *Index* objekta na vrijednost koja nije *Null* (0).

Najlakši način pravljenja niza objekata je kopiranjem postojećeg objekta, a to se odvija kroz sljedeća tri koraka.

1) Kreirajte objekat na formi.
2) Dok objekat ima fokus, odaberite ***Copy*** u meniju ***Edit***.
3) U meniju ***Edit***, odaberite ***Paste***. *Visual Basic* će prikazati dijaloški prozor tražeći od vas potvrdu da želite stvoriti niz objekata (**You already have a control name 'Text1'. Do you want to create a control array?**). Odaberite ***Yes*** za potvrdu akcije (primjer kopiranja objekta **Text1** na slici 7.41.).

Taj novi objekat koji ste napravili dobija vrijednost indeksa 1. Objekat koji se prvi kreira ima vrijednost indeksa 0. Vrijednost indeksa svakog novog elementa niza odgovara redoslijedu, po kojem je element dodan u niz objekata. Kad su objekti dodati na ovaj način, većina vidljivih osobina, kao visina, širina i boja, kopiraju se od prvog objekta u niz objekata novim objektima.

Ako ste prilikom kopiranja objekta izabrali dugme ***No,*** na formi će se pojaviti identičan objekat, ali pod drugim imenom (primjer kopiranja objekta **Label1** na slici 7.41.).

Osnovna prednost uređivanja objekata u niz, jeste njihova grupna upotreba i obrada. Pristup svim objektima jednog niza vrši se pomoću petlje *For ... Next,* od najmlađeg do najstarijeg člana. Obično se pomoću razmatrane petlje mijenjaju vrijednosti osobinama objekata, na primjer:

```
For Each Control in Formi.Controls
   'Sekvencija naredbi obrade objekata
Next Control
```

Struktura ima značenje da se obradi svaki objekat (*Control)* na i-toj formi od posljednjeg, do objekta koji je postavljen prvi. U slučaju da je upotrijebljena samo jedna forma, tada se ime forme Formi može izostaviti. Zapaža se da je uloga posmatrane petlje slična ulozi ranije proučavane petlje *For ... Next.* Obično se pomoću razmatrane petlje mijenjaju vrijednosti osobina objekata, na primjer:

```
Vidljivo=Text1.Text     ' broj vidljivih objekata Text3
For petlja1 = 0 To Vidljivo
    Text3(petlja1).Visible = True    ' Vidljivi objekti
    Visina="h"& petlja1       ' imena tabela u bazi
    Text3(petlja1).Text = Rezervoar(Visina)
     Rem punjenje niza objekata Text3 iz baze
Next
For petlja2 = Vidljivo To 98 'ukupno 99 objekta u nizu
   Text3(petlja2).Visible = False  ' Nevidljivi objekti
Next
```

7.24. 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.

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* 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.

Slika 7.42. Tabulatorni reda

Tabela 7.4. Promjena TabIndexa

Objekat	TabIndex prije zamjene	TabIndex poslije zamjene
Text1	0	1
Text2	1	2
Command1	2	0

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* tu vrijednost pretvara u broj za jedan manji od broja kontrola.

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.

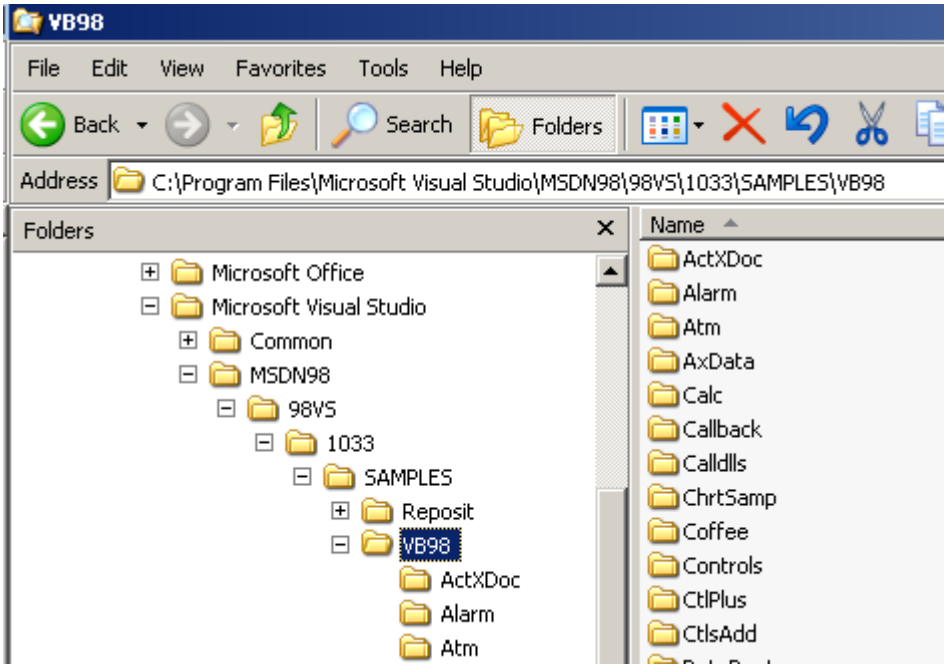
Grupa dugmadi izbora ima jedinstven izbor tipkom TAB. Potvrđeno dugme (ono čije je osobina *Value* postavljeno na *True*) automatski ima osobina *TabStop* postavljeno na *True*, a ostala dugmad u grupi imaju osobinu *TabStop* postavljenu na *False*.

8. PRIMJERI *VISUAL BASIC* PROGRAMA

U ovom poglavlju biće predstavljeni primjeri programa napisanih u programskom jeziku *Visual Basic*. Prvo će biti predstavljeni primjeri programa, koji se mogu naći u *Help*-u programskog jezika *Visual Basic*. Zatim će biti predstavljeni praktični primjeri kodova programa, koji pokazuju kako mogu da se koriste osnovni objekti, koji postoje u *Visual Basicu*. Na kraju će biti predstavljeno par programa, koji se mogu koristiti za komunikaciju sa *Microsoft Excel*-om i *Microsoft Access*-om.

8.1. PRIMJERI IZ *HELP*-A *VISUAL BASIC* PROGRAMA

U *Help*-u programskog jezika *Visual Basic* postoji dosta gotovih primjera programa, koji mogu poslužiti kao dobra osnova, za učenje ovog programskog jezika. Svaki primjer se nalazi na posebnom direktorijumu i ima već urađene gotove programske blokove, koji se mogu iskopirati u novi program, koji se pravi. Do ovih primjera se najlakše može doći preko explorera na direktorijumu "**C:\Program Files\MicrosoftVisual Studio\MSDN98\98VS\1033\SAMPLES\VB98**", kao na sljedećoj slici 8.1.



Slika 8.1. Prozor sa direktorijumima gdje se nalaze gotovi primjeri programa

Na direktorijumu **ActXDoc** nalazi se primjer programa, koji omogućuje pozivanje nove forme i vezu prema nekom sajtu.

Na direktorijumu **Alarm** nalazi se primjer programa, koji prikazuje korišćenje objekta *Timer*.

Na direktorijumu **Atm** nalazi se primjer programa, koji simulira korišćenje bankomata. U ovom programu je prikazano korišćenje *Currency* tipa podataka.

Na direktorijumu **Calc** nalazi se primjer programa, u kome je napravljen digitron sa osnovnim matematičkim operacijama.

Na direktorijumu **Calldlls** nalazi se primjer programa, koji prikazuje slučajno kretanje objekta na formi u raznim smjerovima.

Na direktorijumu **ChrtSamp** nalazi se primjer programa, koji omogućuje crtanje dijagrama. U ovom programu je prikazano korišćenje *CommonDialog* objekta, za pravljenje vlastitog padajućeg menija.

Na direktorijumu **Control** nalazi se primjer programa, koji prikazuje pravljenje vlastitog menija i korišćenje raznih objekata. Prikazan je način korišćenja objekata: *CheckBox, Option Bootm, Frame*, rad sa slikama i promjena mjesta slika, kroz primjer mješanja karti.

Na direktorijumu **CtlsAdd** nalazi se primjer programa, koji prikazuje korišćenje objekta *Data grid*. Ovaj objekat se koristi za prikaz podataka u tabeli.

Na direktorijumu **DataRept** nalazi se primjer programa, koji prikazuje korišćenje objekta *Toolbar,* za generisanje raznih izvještaja.

Na direktorijumu **Datatree** nalazi se primjer programa, koji prikazuje pravljenje strukture eksplorera korišćenjem objekata: *Treeview, Listview, Imagelist, CommonDialog*.

Na direktorijumu **DataAware** nalazi se primjer programa, koji prikazuje otvaranje i čitanje podataka iz baze podataka.

Na direktorijumu **Dialer** nalazi se primjer programa, koji prikazuje simulaciju pozivanja telefonskog broja.

Na direktorijumu **Errors** nalazi se primjer programa, koji prikazuje obrađivanje grešaka, koje mogu nastati u toku rada sa programom.

Na direktorijumu **Filects** nalazi se primjer programa, koji prikazuje pretraživanje dokumenata po diskovima i direktorijumima, korišćenjem objekata: *DriveListBox, FileListBox, ListBox* i *PictureBox*.

Na direktorijumu **Firstapp** nalazi se primjer programa, koji prikazuje korišćenje objekata *Data* i *MsFlexGrid* za pristup bazi podataka.

Na direktorijumu **Geofacts** nalazi se primjer programa, koji prikazuje pozivanje *Excel* dokumenta iz *Visual Basic*-a.

Na direktorijumu **Listcmbo** nalazi se primjer programa, koji prikazuje povezivanje sa *Access* bazom podataka, koja se bira preko *Open* menija.

Na direktorijumu **MCI** nalazi se primjer programa, koji prikazuje simulaciju muzičke linije.

Na direktorijumu **MDI** nalazi se primjer programa, koji prikazuje pravljenje padajućeg menija i menija sa vlastitim ikonicama.

Na direktorijumu **Msflexgd** nalazi se primjer programa, koji prikazuje povezivanje sa bazom podataka i prikaz podataka u *MsFlexGrid* tabeli.

Na direktorijumu **Olecnt** nalazi se primjer programa, koji prikazuje korišćenje OLE kontrola za otvaranje raznih dokumenata.

Na direktorijumu **Optimize** nalazi se primjer programa, koji prikazuje korišćenje objekta *Timer,* da bi stvorili privid da se slike kreću.

Na direktorijumu **Palmode** nalazi se primjer programa, koji prikazuje neprestanu promjenu slika na ekranu, uz pomoć objekta *Timer*.

Na direktorijumu **Picclip** nalazi se primjer programa, koji prikazuje simulaciju leta leptira i okretanja čigre, uz pomoć objekta *Timer*.

Na direktorijumu **Vcr** nalazi se primjer programa, koji prikazuje simulaciju leta leptira. Let se prikazuje na simulaciji video plejera, uz pomoć objekta *Timer*.

8.2. *ADO* PRISTUP BAZAMA PODATAKA

U ovom poglavlju biće opisan *ADO (ActiveX Data Objects)* pristup *Access* bazama podataka iz programa *Visual Basic*. Nakon pokretanja svakog novog projekta *(File—> New Project—> StandardExe),* u glavnom meniju *Visual Basic*-a*,* potrebno je izabrati *Project —> References,* a zatim čekirati opciju

Microsoft DAO Object 3.51. *ili*

Microsoft DAO 3.6 Object Library.

Ovim se omogućava rad sa bazama podataka preko *DAO* modela u otvorenom projektu. Poslije toga otpočinje pisanje koda, tako što se prvo definišu objekti, promjenljive i konstante.

```
Dim dbFpe As Database
```

Ovim se samo definiše objekat **dbFpe** kao *Database,* što znači da taj objekt može da prihvati rad samo sa bazom podataka. Da bi se objektu dbFpe dodijelila neka konkretna vrijednost (baza podataka) treba napisati sljedeće:

```
Set dbFpe  = Opendatabase (strPutBaze)
```

gdje je **strPutBaze** string promjenljiva, koja označava putanju baze podataka na disku računara (na primjer strPutBaz = "D:/Mirko/Fpe.mdb"). Naravno, prethodno je potrebno definisati **strPutBaz** kao string promjenljivu

```
Dim strPutBaz As String
```

ili se jednostavno može napisati (direktno definisanje putanje):

```
Set dbFpe = OpenDatabase ("D:/Mirko/Fpe.mdb")
```

bez uvođenja string promjenljive. Na taj način objekat **dbFpe** sada, u stvari, predstavlja bazu podataka **Fpe.mdb**.

Umjesto direktnog načina definisanja putanje može se uraditi i apsolutno definisanje putanje. Tada je potrebno da baza podataka i program u kome se radi sa tom bazom budu na istom direktorijumu.

```
Set dbFpe = OpenDatabase (App.Path & "\Fpe.mdb") ili
Set dbFpe = DBEngine.Workspaces(0).OpenDatabase _
(App.Path & "\Fpe.mdb")
```

Pošto se baza podataka sastoji iz više tabela, treba definisati objekat, kojem će biti dodijeljena neka od tabela. Na primjer:

```
Dim rsRadnik As Recordset
```

Ovim se **rsRadnik** definiše kao objekat *Recordset* i ovom objektu se dodjeljuje neka konkretna vrijednost, tj. neka od tabela iz baze **Fpe.mdb** na sljedeći način:

```
Set rsRadnik = dbBanke.OpenRecordset("Radnik", dbOpenDynaset)
```

Objekat **rsRadnik** sada predstavlja tabelu **Radnik** iz baze podataka **Fpe.mdb**. U ovaj objekat, odnosno tabelu moguće je kako upisivati tako i čitati podatke. To je određeno prilikom kreiranja objekta **rsRadnik** sa osobinom *dbOpenDynaset*, koja omogućava upis i čitanje podataka iz tabele.

Pored ove osobine postoji još opcija *dbOpenSnapshot*, koja omogućava čitanje podataka iz tabele, ali ne i njihovo ažuriranje, kao i neke druge manje bitne osobine.

Upisivanje podataka u tabelu **Radnik** vrši se preko objekta **rsRadnik**, jer je on definisan tako da predstavlja tu tabelu, na sljedeći način:

```
rsRadnik. AddNew
rsRadnik.lme = "Stefan"
rsRadnik.Prezime = "Covic"
rsRadnik.GodinaR = 1989
rsRadnik.Update
```

ili kraće

```
With rsRadnik
   .AddNew
   .Ime = "Stefan"
   .Prezime = "Covic"
   .GodinaR = 1989
   .Update
End With
```

Naravno podrazumijeva se da se u tabeli **Radnik** nalaze kolone **Ime**, **Prezime** i **GodinaR**, tj. da su prethodno kreirane u nekom od programa za pravljenje baza podataka i da su definisane kao *string*. Pomoću naredbe **.AddNew** u tabeli **Radnik** dodaje se novi red, a zatim se u kolone tabele **Ime**, **Prezime** i **GodinaR**, unose

vrijednosti. Da bi novoupisani podaci u tabelu bili snimljeni (spašeni), primjenjuje se naredba **.Update**.

Ukoliko želimo da izmijenimo podatke, koji već postoje u bazi podataka, koristi se naredba **Edit**. Na primjer:

```
With rsRadnik.Edit
.Ime = "Mihailo"
.GodinaR = 1971
.Update
End With
```

Ako je potrebno izbrisati određeni red u tabeli **Radnik**, prvo se sa naredbom **MoveFirst** pomjeri pokazivač u tabeli na prvi red. Zatim se naredbom **FindFirst** pronađe željeni red. Nakon toga se naredbom **Delete** vrši brisanje reda. Na primjer:

```
With rsRadnik
.MoveFirst
.FindFirst "GodinaR = '1938'"
.Delete
End With
```

ili

```
rsRadnik.MoveFirst
rsRadnik.FindFirst "GodinaR = '1938' "
rsRadnik.Delete
```

Pretraživanje unutar tabele može biti

- ***FindFirst*** - pronalaženje prvog podatka po kome se radi pretraga,
- ***FindLast*** - pronalaženje zadnjeg podatka po kome se radi pretraga,
- ***FindNext*** - pronalaženje narednog podatka po kome se radi pretraga,
- ***FindPrevious*** - pronalaženje prethodnog podatka po kome se radi pretraga,
- ***EOF***- pretraživanje do zadnjeg reda u tabeli.

Kriterijum pretrage može biti jednostavan

```
Dim Uslov1, Uslov2 As String
Uslov1 = "Ime = 'Stefan'"   ' postavi uslov.
rsRadnik.FindFirst Uslov1
```

ili složen uz upotrebu operatora: =, >, <, *and* i *or*

```
Uslov2 = "[Ime] = 'Stefan' And [GodinaR] >= 1965"
rsRadnik.FindFirst Uslov2
```

Slijedi primjer koda programa koji iz *Access* baze podataka **Fpe.mdb** preuzima podatke iz tabele **Radnik,** za radnika sa registarskim brojem 1938. Registarski broj radnika se pretražuje po koloni **Regbroj**. Rezultati se upisuju u dva *TextBox*-a.

```
Public Sub IzbazeM()
Set dbBaza =OpenDatabase (App.Path & "\Fpe.mdb")
Set rsRadnik = baza.OpenRecordset("Radnik, dbOpenDynaset")
imazapis = False
rsRadnik.MoveFirst
Do While rsRadnik.EOF = False
If rsRadnik("Regbroj") = 1938 Then
  imazapis = True  Rem pronadjen trazeni radnik u bazi
End If
rsRadnik.MoveNext
Loop

If imazapis = True Then
  rsRadnik.MoveFirst
  Do While rsRadnik("Regbroj") <> 1938
    Rezervoar.MoveNext
  Loop
  Text4.Text = rsRadnik("Ime") Rem citanje iz baze
  Text7.Text = rsRadnik ("Prezime")
Else
    poruka = "PORUKA u bazi nema tog radnika"
    Stil = vbOK
    Naslov = "Baza radnici"
    odgovor = MsgBox(poruka, Stil, Naslov)
End If
rsRadnik.Close
dbBaza.Close
End Sub
```

8.3. *SQL* UPIT U *VISUAL BASIC*-U

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
```

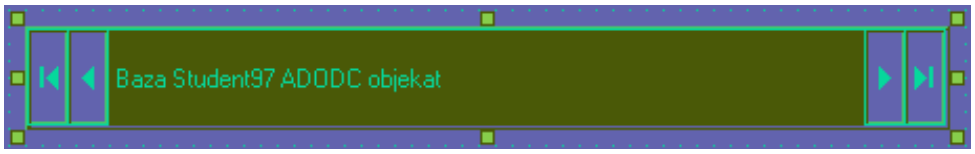
Lista-kolona je spisak svih kolona za koje pravimo upit iz tabele pod imenom ***ime-tabele***. ***Logički-izraz*** uključuje poređenja vrijednosti kolona, konstanti i izraza (odgovarajućeg tipa) u kojima kolone i konstante učestvuju. Pripadne relacijske operacije su: =, <>, <, <=, >, >=, *IN, BETWEEN, 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]]

Visual Basic omogućava razne vidove komunikacije sa relacionim bazama podataka, pa i da se iz njega prave *SQL* upiti. *SQL* upite iz *Visual Basica*-a možemo najlakše praviti na sljedeća dva načina:

- preko *ADODC* objekta;
- preko *aktivX* kontrole.



Slika 8.2. ADODC objekat

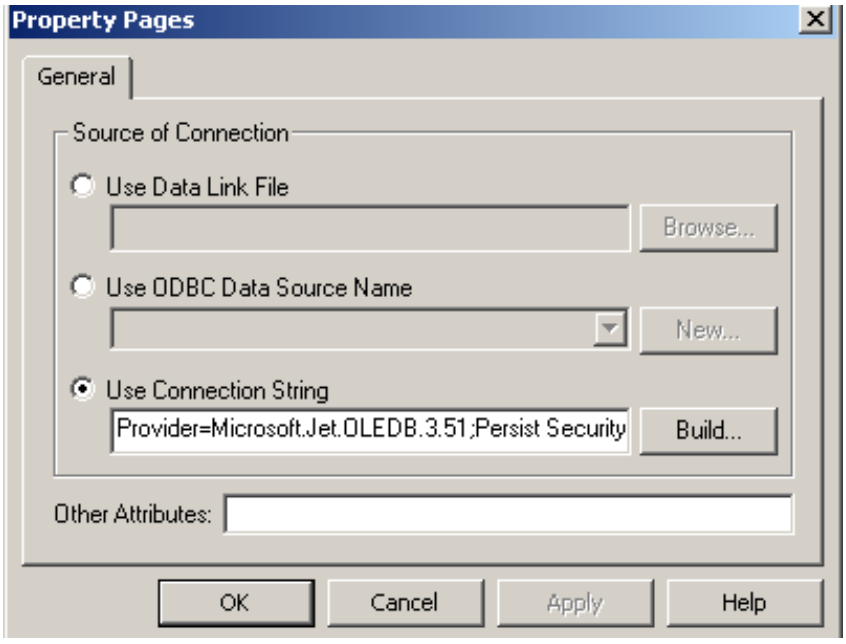
ADODC objekat (slika 8.2.) koristi se za automatsko povezivanje sa tabelom baze podataka kao ***Microsoft ActiveX Data Objects*** (*ADO*), ili za pravljenje *SQL* upita nad bazom. Pomoću ovog ***ADODC*** objekta rezultati pretrage baze podataka se prikazuju u sljedećim objektima: ***MSHFlexGrid***, ***DataGrid***, ***DataCombo, DataList*** i ***DataCombo***. Svi ovi objekti se ne nalaze u standardnoj paleti objekata. Dodavanje ovih objekata u paletu objekata vrši se kroz sljedeće korake. Prvo se klikom miša na meni ***Project*** (na liniji menija) otvara se list opcija, od kojih treba odabrati opciju ***Components***. Nakon toga, pojavljuje se okvir za dijalog sa spiskom svih komponenti (objekata). Potrebno je čekirati sljedeće komponente:

- ***Microsoft Ado Data Control 6.0 (ADODC)***
- ***Microsoft Hierarchical FlexGrid 6.0 (MSHFlexGrid)***
- ***Microsoft DataGrid Control 6.0 (DataGrid)***
- ***Microsoft DataList Control 6.0 (DataList)***

Objekat *ADODC* na prvi pogled liči na objekat *Data* iz poglavlja 7.15, jer se takođe može povezati samo sa verzijom ***Access 97*** bazom podataka. Međutim ova dva objekta se pomoću različitih osobina povezuju na bazu podataka. Dvije najbitnije osobine objekta *ADODC* su:

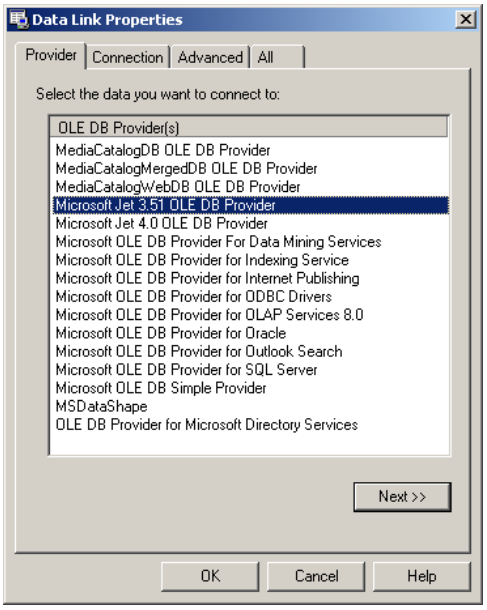
ConnectionString - za izbor baze sa kojom se vrši povezivanje.

RecordSource - za pravljenje SQL upita.

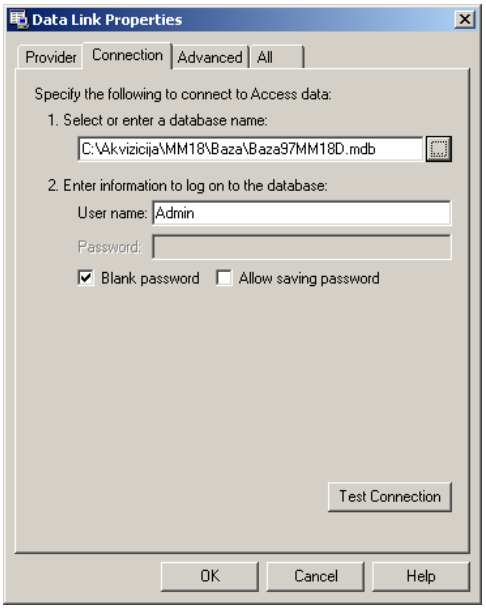


Slika 8.3. Osobina ConnectionString ADODC objekta

Kada se na objektu *ADODC* želi podesiti osobina *ConnectionString*, potrebno je na ovoj osobini izabrati ..., nakon čega se pojavljuje prozor kao na slici 8.3. Na ovom prozuru se bira opcija ***Use Connection String***, a zatim je potrebno kliknuti na dugme ***Build***. Nakon toga se otvara dijaloški prozor ***Data link Properties*** kao na slici 8.4. Na ovom prozoru prvo se izabere paleta ***Provider***. Na njoj se iz liste bira tip *OLE DB* provajdera. Ako se ovaj objekat želi povezati sa *Access* bazom podataka, onda je iz liste potrebno izabrati ***Microsoft Jet 3.51 OLE DB Provider***, a zatim i dugme ***Next***. Nakon toga se prelazi na paletu ***Connection*** kao na slici 8.5. Prvo je potreno izabrati ime baze podaka, sa kojom se želi izvršiti povezivanje. Kada se izabere ime baze podataka, u prostoru 1. (*Select or enter database name:*) se automatski prikaže kompletna putanja sa svim direktorijumima i imenom baze podataka. U ovoj paleti se može dodatno izabrati i ime korisnika, koji samo može da pristupa bazi pomoću ovog objekta. Kada se izabere ime baze podataka, klikom miša na dugme ***Test Connection*** se može izvršiti probni pristup bazi podataka. Ako se uspješno uspostavi veza sa izabranom bazom podataka pojaviće se poruka ***Test connection succeeded***. Ako se uspješno ne uspostavi veza sa izabranom bazom podataka pojaviće se poruka ***Test connection faled***. Najćešći razlog za nastanak greške pri uspostavljanju veze ovog objekta sa bazom podataka, je da *Access* baza nije spašena u verziji *Access* 97.



Slika 8.4. Izbor OLE DB provajdera



Slika 8.5. Izbor željene baze

Kada se izvrši povezivanje sa bazom podata preko ***ADODC,*** sadržaj tabela baze podataka se može prikazivati u sljedećim objektima: ***MSHFlexGrid, DataGrid, DataList*** i ***DataCombo***. Ovi objekti vizuelno dosta liče na objekte, koje smo ranije spominjali: ***MSFlexGrid, ListBox*** i ***ComboBox.*** Glavna razlika između ovih objekata leži u tome što se objekti: *MSHFlexGrid, DataGrid, DataList* i *DataCombo* mogu povezati sa bazom podataka samo preko objekta ADODC, dok se objekti: *MSFlexGrid, ListBox* i *ComboBox* sa bazom podataka mogu povezati samo preko objekta *Data*.

Microsoft Hierarchical FlexGrid 6.0 (MSHFlexGrid) objekat koristi se za automatsko povezivanje sa elementima jedne ili više kolona u tabeli baze podataka, ili rezultatom nekog *SQL* upita, koji je napravljen preko objekta *ADODC*.

Najbitnija osobina ovog objekta je ***DataSource,*** koja služi za izbor imena jednog *ADODC* objekta, koji je trenutno postavljen na formu, a preko koga se želimo povezati sa bazom podataka.

Ostale bitne osobine ovog objekta su:

- ***Name*** za definisanje imena objekta.
- ***DataSource*** za izabor imena predhodno postavljenog *ADDC* objekta, preko koga se želimo povezati sa bazom podataka.
- ***Font*** za izbor fonta, kojim se prikazuje tekst u tabeli.
- ***FontFixed*** za izbor fonta, kojim se prikazuje tekst zaglavlja u tabeli.
- ***FormatString*** za početni prikaz naziva kolona. Imena kolona se razdvajaju sa simbolom "|", kao u sljedećem primjeru "Indeks|Ime|Prezime|Smer".

DataGrid objekat koristi se za automatsko povezivanje sa elementima jedne ili više kolona u tabeli baze podataka, ili rezultatom nekog *SQL* upita, koji je napravljen preko objekta *ADODC*. Takođe je moguće da se preko ovog objekta vrši unos podataka u bazu podataka, a što nije moguće ostvariti pomoću objekata *MSFlexGrid* i *MSHFlexGrid*. Čim se promjene podaci u tabeli *DataGrid* na formi, automatski se mijenja sadržaj podataka u bazi.

Kretanje kroz redove u tabeli *DataGrid* vrši se klikom na strlice objekta *ADODC,* sa kojim je tabela *DataGrid* povezana. Tada dolazi i do automatskog pomjeranja i strelice koja se nalazi u prvoj fiksnoj koloni. Na određeni red se može izvršiti pozicioniranje i klikom miša. Red na koji se pozicionira strelica, može se trajno brisati sa forme, ali i iz baze, komandom *Delete* na tastaturi.

	Indeks	Ime	Prezime	Smer
▶	1	Mirko	Rosic	B2
	2	Mira	Skrba	B1
	3	Stefan	Covic	B2
	4	Mihailo	Rosic	A2
	5	Teodora	Damjanovic	B2

Slika 8.6. Objekat DataGrid

Najbitnija osobina ovog objekta je ***DataSource,*** koja služi za izbor imena jednog *ADODC* objekta, koji je trenutno postavljen na formu, a preko koga se želimo povezati sa bazom podataka. Ostale bitne osobine ovog objekta su:
- ***Name*** za definisanje imena objekta.
- ***AllowDelete*** za dozvoljavanje (True) ili ne dozvoljavanje (False) brisanja reda.
- ***AllowUpdate*** za dozvoljavanje (True) ili ne dozvoljavanje (False) izmjene sadržaja u tabeli, a samim tim i u bazi.
- ***DataSource*** za izabor imena predhodno postavljenog *ADDC* objekta, preko koga se želimo povezati sa bazom podataka.
- ***Font*** za izbor fonta, kojim se prikazuje tekst u tabeli.

DataList i ***DataCombo*** objekti koriste se za automatsko povezivanje sa elementima jedne kolone u tabeli baze podataka, ili rezultatom nekog *SQL* upita, koji je napravljen preko objekta *ADODC*. Najbitnije osobine ovih objekata su:
- ***Name*** za definisanje imena objekta.
- ***DataSource*** za izabor imena predhodno postavljenog *ADDC* objekta, preko koga se želimo povezati sa bazom podataka.
- ***DataField*** za izabor imena kolone predhodno postavljenog *ADDC* objekta, preko koga se želimo povezati sa kolonom u tabeli baze podataka.
- ***RowSorce*** za izabor imena predhodno postavljenog *ADDC* objekta, preko koga se želimo povezati sa bazom podataka.

U *Visual Basic*-u *SQL* upit se može pisati i preko *Activex* kontrole. U ovom slučaju se upit može raditi nad svim verzijama *Access* baze podataka, a ne samo sa verzijom 97, kao kod objekta *ADODC*. Kao i za svaku *Activex* kontrolu potrebno je prvo deklarisati promjenjivu. Promjenjiva koja se koristi za pravljenje *SQL* upita mora deklarisana kao ***QueryDef*** tip promjenjive.

Da bi *SQL* upit mogao da radi, potrebno je da se prvo deklariše promjenjiva za povezivanje sa bazom podataka (*As Database*), a zatim i promjenjive koje se koriste za povezivanje sa tabelama te baze (*As Recordset*). Da di se otpočeo rad preko *ActiveX* kontrole, potrebno je izvršiti povezivanje promjenjivih sa bazom, tabelama i *SQL* upitom preko komande SET.

Preuzimanje podataka iz *SQL* upita u neki objekat vrši se preko komande koja ima sljedeću sintaksu:

```
Ime_objekta.Osobina = Ime_promjenjive!Ime_kolone
```

gdje je:

Ime_objekta - ime objekta u koji se želi upisati neki podatak.

Osobina - osobina izabranog objekta preko koje se vrši upis podataka (*Caption, Text, TextMatrix*(0, 1), ...)

Ime_promjenjive - ime promjenjive koja se koristi za povezivanje sa tabelama te baze (*As Recordset*).

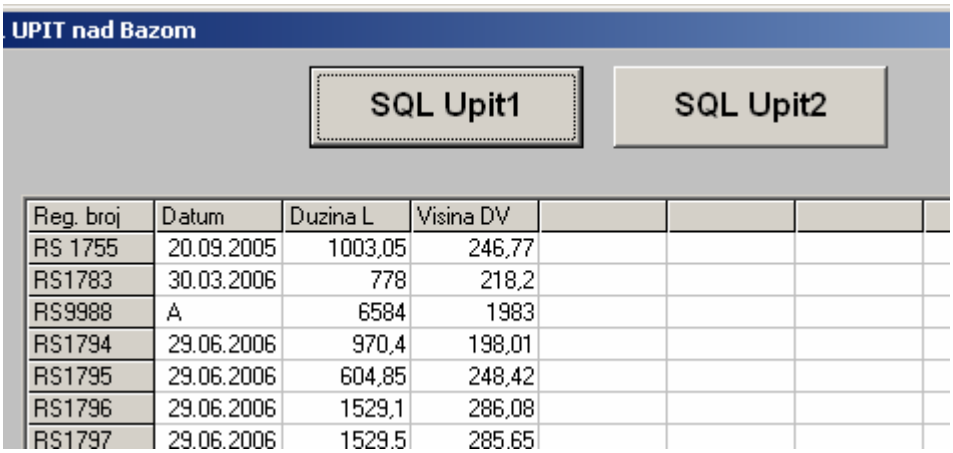
! - simbol koji naglašava preuzimanje podataka iz *SQL* upita, a ne tabele.

Ime_kolone - ime kolone u tabeli koja je rezultat *SQL* upita.

Prilikom korišćenja ove naredbe za preuzimanje podataka, koji su rezultat *SQL* upita, treba naglasiti da se može preuzeti samo podatak iz jednog reda izabrane kolone. Da bi se preuzeli podaci iz više redova neke kolone neophodno je da se koristi *Do While* petlja.

Nakon završetka rada sa bazom podataka potrebno je prvo zatvoriti sve otvorene tabele, a zatim zatvoriti i kompletnu bazu podataka komandom *Close*.

SQL upit koji se pravi preko *Activex* kontrole može se aktivirati preko komandnog dugmeta, a rezultati upita se mogu direktno upisivati u *MSFlexGred* tabelu (kao na slici 8.7.), *ListBox* ili *ComboBox*.



Slika 8.7. Pokretanje SQL upita preko komandnog dugmeta

Slijedi primjer koda programa za prikaz rezultata *SQL* upita u *FlexsGrid* tabeli.

```
Dim dbFpe As Database
  Rem definisanje promjenjive za rad sa bazom podataka
Dim rsRadnik As Recordset
  Rem definisanje promjenjive za rad sa tabelom baze
Dim SqUpit As QueryDef
  Rem definisanje promjenjive za rad sa SQL upitom

Set dbFpe = OpenDatabase("Fpe.mdb")
  Rem povezivanje sa bazom koja se nalazi na istom
direktorijumu kao i program koji pravimo
Set SqUpit = dbFpe.CreateQueryDef("")
  Rem postaviti da početni upit bude prazan
```

```
  Rem kod za pravljenje upita nad bazom "Fpe.mdb", i to
nad njenom tabelom "Radnik"
With SqUpit
   .Connect = ""
      Rem koristi se za povezivanje sa bazom
   .SQL = "SELECT * from Radnik"
      Rem koristi se za pisanje teksta upita
   Set rsRadnik = .OpenRecordset()
      Rem koristi se za otvaranje tabele u koju je
    upisan rezultat SQL upita
   rsRadnik.MoveFirst
      Rem pozicioniranje na prvi red tabele
End With

  Rem Prikaz trenutne vrijednosti zapisa različitih
kolona SQL upita preko MesgBoxa
MsgBox rsRadnik!Ime
MsgBox rsRadnik!Prezime
MsgBox rsRadnik!Redbroj

  Rem Prikaz trenutne vrijednosti zapisa različitih
kolona SQL upita u FlexGrid Tabeli
i = 1
rsRadnik.MoveFirst
Rem upis naziva kolona
Tabela1.TextMatrix(0, 0) = " Reg. broj"
Tabela1.TextMatrix(0, 1) = "Datum"

Do While rsRadnik.EOF <> True
      Rem upis rezultata upita u tabelu
    Tabela1.TextMatrix(i, 0) = rsRadnik!Regbroj
    Tabela1.TextMatrix(i, 1) = rsRadnik!Datum
    Tabela1.AddItem ""
    i = i + 1
    rsRadnik.MoveNext
Loop
rsRadnik.Close
  Rem zatvaranje tabele
dbFpe.Close
  Rem zatvaranje kompletne baze podataka
```

```
Rem Kod za pravljenje upita nad otvorenom bazom
"Fpe.mdb" i to nad njenom tabelom "Radnik" za izdva-
janje samo kolene "Datum" bez ponavljanja podataka.

With SqUpit
   .Connect = ""
   .SQL = "SELECT distinct Datum from Radnik"
  Set rsRadnik = .OpenRecordset()
  rsRadnik.MoveFirst
End With
Tabela1.TextMatrix(i, 1) = rs1!Datum

  Rem Kod za pravljenje upita nad otvorenom bazom
"Fpe.mdb", i to nad njenom tabelom "Radnik" uz WHERE
uslov

With SqUpit
   .Connect = ""
   .SQL = "SELECT Redbroj, Datum, Ime, Prezime from
Radnik where Redbroj>29"

Set rsRadnik = .OpenRecordset()
rsRadnik.MoveFirst
End With
 Tabela1.TextMatrix(i, 1) = rs1!Ime


  Rem Kod za pravljenje upita nad otvorenom bazom
"Fpe.mdb", i to nad njenom tabelom "Radnik" uz WHERE
uslov unesen preko Textboxa sa imenom TexRedBroj

Tabela1.Clear
Tekst1 = "SELECT Redbroj, Datum, Ime, Prezime from
Radnik where Redbroj>"
  Rem ovo predstavlja fiksni dio SQL upita
Tekst2 = TexRedBroj.Text
Rem Textbox preko koga se unosi brojni uslov
Tekst3 = Tekst1 + Tekst2
 Rem spajanje dva teksta u jedan, koji je sada SQL upit
With SqUpit
   .Connect = ""
   .SQL = Tekst2
   Set rsRadnik = .OpenRecordset()
```

```
  rsRadnik.MoveFirst
End With
```

8.4. PRISTUP *EXCEL* DOKUMENTU

U ovom poglavlju biće opisan primjer *ActiveX* pristupa *Excel* dokumentu sa više stranica iz programa *Visual Basic*. Nakon pokretanja svakog novog projekta *(File—> New Project—> StandardExe),* u glavnom meniju *Visual Basic*-a, potrebno je izabrati *Project —> References,* a zatim čekirati opciju

Microsoft Excel 5.0 Object Library *ili*

Microsoft Excel 11.0 Object Library.

Ovim se omogućava rad sa *Excel* dokumentima iz *Visual Basic*-a. Poslije toga otpočinje pisanje koda, tako što se prvo definišu objekti, promjenljive i konstante. Slijedi dio koda programa koji otvara *Excel* dokument pod nazivom "MokraM.xls", koji ima dvije stranice sa nazivima "Stranica1" i "Sranica2". Pokazaćemo kako se u *Excel* mogu upisivati podaci, odnosno iz *Excel*-a mogu preuzimati podaci. Prvo je predstavljen dio koda napisanog u *Modul*-u, gdje su deklarisane globalne (javne) promjenjive.

```
Public appMokraM As Excel.Application
  Rem promjenjiva za povezivanje sa Excel programom
Public wbMokraM As Excel.Workbook
 Rem promjenjiva za povezivanje sa Excel radnom knjigom
Public AA As Double
Public HHMAX As Integer
Public q(52) As Single
```

Slijedi kod koji predstavlja globalnu proceduru napisanu u modulu, pomoću koje se iz *Visual Basic*-a otvara *Excel* program.

```
Sub Setup()
  Set appMokraM = GetObject(, "Excel.Application")
      Rem Gledanje otvorenog Excel dokumenta
  If Err.Number <> 0 Then  'Ako Excel nije pokrenut
    Set appMokraM = CreateObject("Excel.Application")
       Rem Pokrenuti Excel
  End If
  Set wbMokraM = appMokraM.Workbooks.Open_
(App.Path & "\MokraM.xls")
End Sub
```

Slijedi kod koji predstavlja globalnu proceduru napisanu u modulu, pomoću koje se iz *Visual Basic*-a zatvara predhodno otvoreni *Excel* program.

```
Sub CleanUp()  'Zatvaranje Excel dokumenta
    wbMokraM.Application.Quit
    Set appMokraM = Nothing
    Set wbMokraM = Nothing
End Sub
```

Slijedi kod koji predstavlja lokalnu proceduru koja se aktivira klikom miša na komandno dugme *Command1*. U njoj je opisan postupak pristupa stranicama "Stranica1" i "Stranica2" *Excel* dokumenta "MokraM.xls".

```
Private Sub Command1_Click()
  Dim shtStranica1, shtStranica2 As Excel.Worksheet
    Rem dvije promjenjive za povezivanje sa stranicama
  Dim rngList1, rngList2 As Excel.Range
    Rem dvije promjenjive za pristup ćelijama na stranicama
  Setup ' Poziv globalne procedure za otvaranje Excela.
  Forma1.Visible = False 'Zatvaranje otvorene forme
  wbMokraM.Application.Visible = True
    Rem Da Excel program bude vidljiv
  Set shtStranica1 = wbMokraM.Sheets("Stranica1")
    Rem Definisanje Excel lista Stranica1
  Set shtStranica2 = wbMokraM.Sheets("Stranica2")
    Rem Definisanje Excel lista Stranica2
  Set rngList1 = shtStranica1.Rows(1)
    Rem Promjenjiva za list Stranica1
  Set rngList2 = shtStranica2.Rows(1)
    Rem Promjenjiva za list Stranica2

  rngList1.Cells(7, 2) = "Adresa"
  rngList1.Cells(9, 2) = "Datum"
    Rem upis podataka u list Stranica1
  AA=Text1.Text
  rngList2.Cells(10 + AA, 5) = VSTVARNO
    Rem upis podataka u list Stranica2
  BB=rngList1.Cells(10, 2)
    Rem upis u promjenjivu BB iz lista Stranica1

  Set shtStranica1 = Nothing
  Set shtStranica2 = Nothing
  Set rngList1 = Nothing
  Set rngList2 = Nothing
End Sub
```

Slijedi kod koji predstavlja lokalnu proceduru u kojoj je opisan postupak pristupa stranicama "Stranica1" i "Stranica2" *Excel* dokumenta "MokraM.xls".

```
Private Sub Command2_Click()  ' dugme izlaz
poruka = MsgBox(" Da li ste Zatvorili EXCEL dokument
MokraM?", vbYesNo, "Poruka")
If poruka = vbNo Then
    Rem Zatvaranje Excela iz Visual Basica, jer Excel
nije predhodno zatvaren sa X.
   CleanUp ' Poziv procedure za zatvaranje Excela.
End If

poruka = MsgBox("Da li želiš da spasiš ove podatke u
bazu podataka?", vbYesNo, "Poruka")

If poruka = vbYes Then
    GoTo Izlaz4
      Rem Onemogućiti izlaz iz forme Form1 sve dok se
    ne spase željeni podaci.
    Command1.Enabled = False
    Command3.Enabled = False
End If
Unload Forma1
Forma2.Show vbModal
Izlaz4: Rem labela na koju se skače sa naredbom GoTo
End Sub
```

8.5. ZADACI ZA SAMOSTALNI RAD

U ovom poglavlju biće dati problemi, koji se mogu riješiti pomoću programa napisanih u *Visual Basic*-u. Da bi se riješili ovi problemi, potrebno je prvo da se izaberu potrebni objekti i dodaju na radnu površinu. Tek zatim se piše odgovarajući kod programa.

8.5.1. Zadaci sa grananjem

Slijede 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 proizvoljan realan broj X i broj ZBI. Gdje je Gdje je ZBI zadnji broj indeksa studenta (ukoliko je nula, onda se uzima 3). Izračunati vrijednost funkcije Y

$$Y=\frac{X^3+2*X^2-5*X+8}{(X-ZBI)*(X-ZBI)}.$$

Rezultat prikazati u jednom *TextBox*-u.

Zadatak 4.

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

Ovi zadaci se rješavaju upotrebom funkcija, koje su ugrađene u *Visual Basic*.

Zadatak 5.

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 6.

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 7.

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 8.

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 9.

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 10.

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.

8.5.3. Zadaci sa petljama

Slijede zadaci koji u sebi sadrže razna ponavljanja. Svi oni se mogu riješiti uz upotrebu *For, Do While* ili *Do Until* petlje.

Zadatak 11.
Napraviti program koji se pokreće klikom na komandno dugme. 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 *FlexGrid* tabele.

Zadatak 12.
Napraviti program koji se pokreće klikom na komandno dugme. 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 maksimalni element niza BB (MaxB), a rezultat prikazati u drugom *TextBox*-u. Naći minimalni 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 *FlexGrid* tabele.

Zadatak 13.
Napraviti program koji se pokreće klikom na komandno dugme. Učitajte preko *InputBox*-a niz AA od N = (14 - 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 *FlexGrid* tabele.

Zadatak 14.
Napraviti program koji se pokreće klikom na komandno dugme. 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 minimalni i maksimalni 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 *FlexGrid* tabele.

Zadatak 14.
Napraviti program koji uz pomoć generatora slučajnih brojeva pounjava, realnim brojevima u rasponu od 1 do 5, *FleksGrid* 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.

8.5.4. Zadaci za pristup *Excel* dokumentu i *Access* bazi

Slijede zadaci gdje je potrebno iz *Visual Basic*-a pristupiti postojećem *Excel* dokumentu, a zatim i zadatak u kome je potrebno izvršiti povezivanje sa *Access* bazom podataka. Dat je i jedan zadatak sa primjenom *OLE* strukture.

Zadatak 15.
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 Basic*-u 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 16.
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 Basic*-u.
Pomoću komandnog dugmeta cijeli sadržaj tabele **Student** prikazati u jednoj *FlexGrid* 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 *FlexGrid* 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 *MSFlexGrid* tabeli ili u *MHSFlexGrid* tabeli.
Napraviti dugme za zatvaranje *Access* baze podataka i cijelog programa.

Zadatak 17.

Napraviti program koji omogućuje da se pomoću *OLE* strukture pristupi jednom *Word* dokumentu, jednom *Excel* dokumentu sa dijagramom i jednoj *Power Point* prezentaciji. Iskopirati dijagram iz *Excel*-a u *Word* i dinamički ga povezati, tako da se promjene podataka u *Excel*-u, automatski odražavaju na dijagram u *Word*-u.

L I T E R A T U R A

[1] Microsoft Visual Basic 6.0 Help, Microsoft, 1999.
[2] Dr Jozo J. Dujmović, *Programski jezici i metode programiranja*, Akademska misao, Beograd, 2003.
[3] Dr Tihomir Latinović, *Osnove programiranja (Visual Basic)*, Biblioteka Informacione tehnologije, Banja Luka, 2007.
[4] Dr Lazar Miličević, Mr Lazar Radovanović, *Programiranje (Visual Basic)*, Ekonomski fakultet, Brčko, 2005.
[5] Srđan Damjanović, Predrag Katanić, Borislav Drakul, *Zbirka zadataka iz poslovne informatike*, Fakultet spoljne trgovine, Bijeljina, 2008.
[6] Srđan Damjanović, Predrag Katanić, *Programski jezik VEE Pro*, Elektrotehnički Fakultet, Istočno Sarajevo, 2011.
[7] 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

004.432.2Visual Basic(075.8)(076)

ДАМЈАНОВИЋ, Срђан
Programski jezik Visual Basic : zbirka zadataka / Srđan Damjanović, Predrag Katanić. - Bijeljina : Fakultet poslovne ekonomije, 2014 (Bijeljina : Grafika Gole). - 227 str. : ilustr. ; 25 cm

Tiraž 200. - Napomene i bibliografske reference uz tekst. - Bibliografija: str. 226.

ISBN 978-99955-45-18-5
1. Катанић, Предраг [аутор]

COBISS.RS-ID 4324376