Univerzitet u Zenici
Pedagoški fakultet
odsjek "Matematika i informatika"

INFORMATIKA III
predavanja

mr. Samir Lemeš, Džavid Dautbegović

Cilj	Stručno osposobljavanje nastavnika – studenata za primjenu programskih jezika u rješavanju problema i zadataka u odgojno-obrazovnomn procesu osnovnih škola.
Zadaci	Student treba: - da nauči nekoliko programskih jezika koji se izučavaju u odgojno-obrazovnom sistemu - da se obuči u primjeni programskih jezika i kao predmeta i kao sredstva obuke u obrazovnom procesu - da se pripremi za kreiranje aplikacija uz pomoć programskih jezika - da se osposobi za rješavanje problema za potrebe nastavnih predmeta
Pregled programskih sadržaja	- Programiranje i programski jezici - Tehnike, metodika i stil programiranja - Algoritamske strukture - Rješavanje problema uz pomoć računara - Numeričke metode u rješavanju problema - Programski jezik QBASIC - Programski jezik C++ - Programski jezik PASCAL - Vizuelno programiranje

akad. godina 2004/2005

Sadržaj

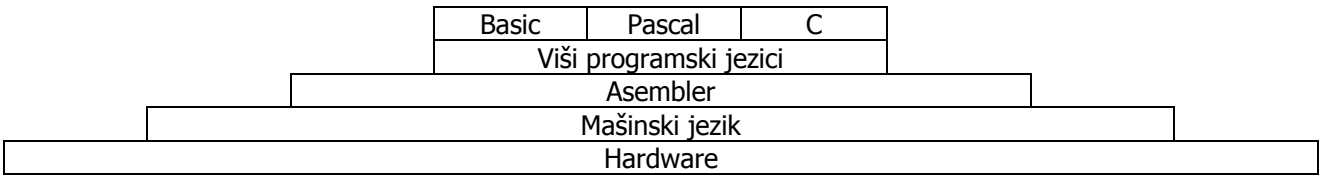
1. Programiranje i programski jezici

1.1. Pojam programa, pojam instrukcije, način izvršenja programa

Program je niz instrukcija napisanih određenim redoslijedom, tako da kao cjelina izvršavaju neki zadatak pretvaranja ulaznih u izlazne podatke. Programiranje u užem smislu riječi predstavlja pisanje programa nekim programskim jezikom. Programski jezik je set instrukcija koje računar razumije i koje može da interpretira.
Instrukcije se dijele na aritmetičke (sa fiksnim i pokretnim zarezom), logičke (poredbene), organizacijske (ulazno/izlazne i transportne), kontrolne (potprogrami, IF-THEN).

1.2. Generacije programskih jezika, vrste prevodilaca

Programski jezici se dijele na niže i na više programske jezike. Niži programski jezik je mašinski jezik, dok u više programske jezike spadaju BASIC, FORTRAN, PASCAL, COBOL, C, C++, itd. Niži programski jezici su platformski, okrenuti računaru (instrukcije se opisuju simbolički), dok su viši programski jezici problemski i bliži su korisniku (instrukcije su obično izvedene iz riječi engleskog jezika). Mašinski jezik se naziva i programski jezik prve generacije (1GL, 1950-1954). Asembler i makro asembler su jezici druge generacije (2GL, 1955-1959) i nalaze se između nižih i viših jezika. Jezici treće generacije (3GL) su pomenuti viši programski jezici, koji su proceduralni, dok u jezike četvrte generacije (4GL) spadaju SQL, HTML, PHP, ASP, tj. neproceduralni jezici sa usko specijaliziranom namjenom: opisni (služe opisivanju dokumenata - PostScript, HTML), upitni (generisanje podskupova iz baza podataka - SQL), grafički (LabView, G – jezici za programiranje virtualnih instrumenata).
Računar može da izvršava samo programe napisane mašinskim jezikom (set instrukcija koje razumije mikroprocesor), tako da je sve programe napisane nekim od viših programskih jezika potrebno prevesti u mašinski jezik. Izvorni kod predstavlja niz instrukcija napisanih višim programskim jezikom, dok izvršni kod predstavlja datoteku koja sadrži instrukcije mašinskog jezika.
Prema vremenu prevođenja instrukcija u mašinski jezik, prevodioci se dijele na kompajlere (compiler) i interpretere. Interpreter prevodi instrukciju neposredno nakon njenog izdavanja, tj. u RAM memoriji računara se nalazi izvorni kod programa, koji se mora prevoditi svaki put kad se program pokrene. Kompajler prevodi kompletan izvorni kod i snima ga u datoteku, tako da se u RAM memoriji računara ne nalazi izvorni kod nego samo mašinske instrukcije.



1.3. Postupak prevođenja programa – kompajliranje, linkovanje, testiranje

Prevođenje programa iz izvornog u izvršni (binarni) kod se sastoji iz dvije faze:

- Kompajliranjem (compiling) se dobija objektni kod, koji je potrebno povezati sa gotovim bibliotekama funkcija. Tokom kompajliranja provjerava se i sintaksa instrukcija, tj. da li su sva imena instrukcija pravilno napisana i da li su svi argumenti uneseni na dozvoljeni način.
- Povezivanje (linking) objektnog koda sa bibliotekama gotovih funkcija je potrebno da bi računar mogao izvršiti instrukcije i funkcije koje sadrži (naprimjer funkcije SIN, COS, LOG ili ulazno/izlazne instrukcije)

Nakon prevođenja treba izvršiti testiranje (debuging), kada se ispituje da li je program moguće izvršiti: da li će se javiti dijeljenje s nulom, korjenovanje negativnog broja ili slični slučajevi nerješivi za računar.

1.4. Tehnike, metodika i stil programiranja

Radi preglednosti programa, treba koristiti konvencije za pisanje izvornog koda.
Između setova instrukcija koje obavljaju pojedine dijelove programa (unos podataka, sortiranje, proračun, crtanje, deklaracija varijabli, itd.) poželjno je stavljati komentare. Komentari se u raznim programskim jezicima označavaju na razne načine (u QBASIC-u naredbom REM, u Visual Basic-u apostrofom ', u PHP-u sa dvije kose crte //, u FORTRAN-u slovom C, i sl.). Prilikom kompajliranja, linije u programu označene kao komentari se ne prevode.

Ciklične strukture (petlje) i blokove instrukcija (IF-THEN-ELSE) je poželjno uvlačiti, radi preglednosti, da bi se lakše provjerilo gdje počinje a gdje završava blok instrukcija.
Primjer:

Pregledan program	Ravni, nepregledan program

REM Deklaracija varijabli
 DIM A(100) as single, I as long
REM Unošenje podataka u niz
 FOR I = 1 TO 25
  A(I) = INPUTBOX(I)
 NEXT I
REM Sortiranje niza
 FOR I = 1 TO 25
  FOR J = I+1 TO 25
  IF A(I) < A(J) THEN
  T = A(I)
  A(I) = A(J)
  A(J) = T
  END IF
  NEXT J
 NEXT I
	REM Deklaracija varijabli
	DIM A(100) as single, I as long
	REM Unošenje podataka u niz
	FOR I = 1 TO 25
	A(I) = INPUTBOX(I)
	NEXT I
	REM Sortiranje niza
	FOR I = 1 TO 25
	FOR J = I+1 TO 25
	IF A(I) < A(J) THEN
	T = A(I)
	A(I) = A(J)
	A(J) = T
	END IF
	NEXT J
	NEXT I

U ovom primjeru treba obratiti pažnju na to da se instrukcija NEXT J nalazi tačno ispod instrukcije FOR J, instrukcija END IF tačno ispod IF i NEXT I tačno ispod FOR I. Program će raditi i bez ovoga, ali će izvorni kod biti nepregledan.
Struktuirano programiranje je specifičan pristup programiranju koji proizvodi programe koji su laki za čitanje, tj. u njima se lako prati tok programa. Struktuirani program uključuje tri konstruktora: sekvence, odluke i petlje. Nestruktuirani program sadrži dosta grananja (vidjeti objašnjenje naredbe GOTO).

1.5. Proceduralno, neproceduralno, modularno, objektno-orijentisano programiranje

Proceduralno programiranje se zasniva na programu kao nizu jednostavnih programskih odsječaka – procedura. Svaka procedura obavlja neki manji zadatak, a podaci su potpuno odvojeni od samih operacija u procedurama.
Kod modularnog programiranja, srodne procedure su grupisane u module. Svaki modul može imati vlastite podatke.
Razlika između proceduralnih i objektno-orijentisanih programskih jezika je u načinu na koji oni koriste resurse računara. Dok proceduralni program zauzme sve resurse (mikroprocesor, memorija) samo za sebe, tj. u trenutku kad je taj program pokrenut računar gubi komunikaciju s korisnikom, objektno-orijentisani program se zasniva na kombinacijama objekata i događaja. Objekte čine zajedno podaci i operacije. Elementi grafičkog prikaza programa na ekranu (prozor, naredbe u menijima, tipke i sl.) u svakom trenutku su na raspolaganju korisniku i mogu im se dešavati razni događaji (lijevi ili desni klik mišem, dvostruki klik mišem, pritisak na tipku ENTER ili ESC sa tastature, prelazak miša preko tog objekta, itd.). Svakoj kombinaciji objekta i događaja se zatim pridružuju određene procedure (dijelovi programa).
Postupak spajanja operacija i podataka se naziva enkapsulacija (eng. *encapsulation*). Pri tome su podaci privatni za svaki objekat, što znači da nisu na raspolaganju drugim dijelovima programa. Ta osobina se naziva skrivanje podataka (eng. *data hiding*). Dijelovi programa se mogu koristiti više puta, i ta osobina se naziva ponovna iskoristivost (eng. *reusability*). Prilikom ponovnog korištenja, novi objekti nasljeđuju osobine onoga iz kojeg su nastali, i ta se osobina zove nasljednost (eng. *inheritance*). Polimorfizam (eng. *polymorfism*) je osobina objektnog modela da se automatski prilagođava vanjskim uslovima, bez potrebe ponovnog redefinisanja objekta.
Najpoznatiji objektno orijentisani programski jezici su C++, Smalltalk, Java i vizuelni jezici.

1.6. Vizuelno programiranje

Iz potrebe za lakšim kreiranjem Windows aplikacija (programa) napravljena su programska rješenja koja se nazivaju vizualnim programskim jezicima (Visual Basic, Visual InterDev, Visual C++, Visual C#, ...). To su uglavnom objektno-orijentisani programski jezici koji u sebi imaju integrisane alate za brže kreiranje grafičkih objekata od kojih se programski interfejs sastoji.

2. Algoritamske strukture

Riječ "algoritam" je arapskog porijekla (prema imenu matematičara AlHorezmija iz IX vijeka) a predstavlja skup pravila koja vode rješenju nekog matematičkog problema. U širem smislu, algoritam služi za definisanje procedure kojom se od ulaznih podataka dobiju rezultati (izlazni podaci).
Karakteristike algoritma su: definisanost (redoslijeda izvođenja), determinisanost (proces pretvaranja ulaza u izlaz mora biti jednoznačno određen), masovnost (za svaki skup ulaznih elemenata mora biti definisano šta se smatra rezultatom, tj. mora se predvidjeti izlaz za svaki mogući skup ulaza), konačnost (mora postojati procedura završetka) i efikasnost (vrijeme potrebno za izvršenje algoritma).
Strukture algoritama mogu biti linijske, strukture grananja, cikličke i složene. U praksi se najčešće javljaju složene strukture, koje predstavljaju kombinaciju navedene tri vrste struktura.

2.1. Linijske strukture

Svaki korak algoritma se izvršava samo jedanput, tačno određenim redoslijedom, bez ponavljanja koraka i bez skokova ili grananja.

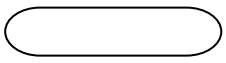
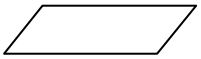
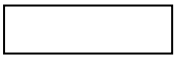
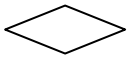
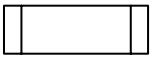
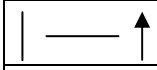
2.2. Strukture grananja

Podrazumijevaju postojanje provjere da li je neki uslov zadovoljen. Na osnovu rezultata provjere definiše se kako će se algoritam dalje odvijati za svaki slučaj rezultata provjere.

2.3. Cikličke strukture

Struktura s petljom podrazumijeva višestruko ponavljanje. Takve strukture mogu imati unaprijed definisan broj koraka, da se izlaz iz petlje ostvaruje kad je zadovoljen neki uslov ili da se petlja ponavlja sve dok je neki uslov zadovoljen.

Blok dijagrami predstavljaju grafički prikaz toka algoritma, odnosno programa. U izradi blok dijagrama koriste se sljedeći simboli:

Simbol	Naziv	Opis
	Početak, kraj ili prekid programa	
	Ulaz/izlaz	Čitanje ulaznih i prikazivanje izlaznih podataka
	Operacija u radnoj memoriji	Kalkulacije, konverzija podataka, transfer podataka unutar radne memorije
	Odluka (grananje)	Ispitivanje uslova, na osnovu čega se odlučuje dalji tok programa
	Program	Zaokružena programska cjelina
	Linije toka	
	Konektor	Koristi se kad se zbog preglednosti blok dijagram prekida da bi se nastavio u tački gdje postoji identičan simbol.

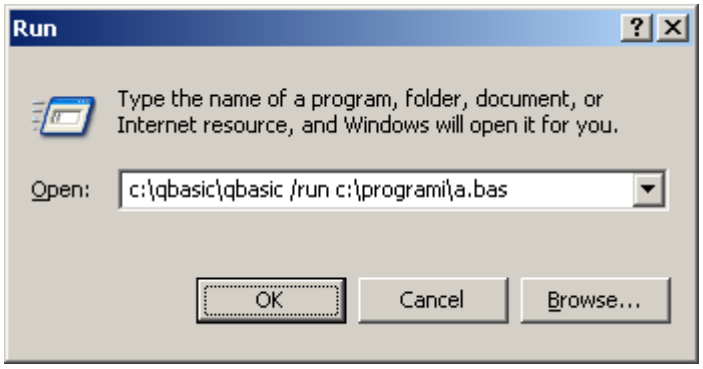
3. Programski jezik QBASIC

Interpreter za programski jezik BASIC je sastavni dio operativnog sistema MS-DOS. Do verzije 5.0, to je bio GWBasic, a od verzije 5.0, QBASIC.
Inače, programski jezik BASIC (Beginners All-purpose Symbolic Instruction Code) je razvijen u SAD oko 1970. godine, a postoji veliki broj verzija, različitih proizvođača (Borland, Microsoft,...). Prvi kućni računari su imali BASIC u ROM memoriji, dok još nije bilo operativnih sistema (za trajno pohranjivanje podataka nisu se koristili diskovi, nego audio kasete i magnetne trake). Razlike između raznih verzija BASIC-a su vrlo male, a svode se na različitu sintaksu pojedinih naredbi, te postojanje nekih instrukcija koje ne postoje u drugim verzijama. Starije varijante BASIC-a su koristile brojeve za označavanje naredbi, iz praktičnog razloga što

nije bilo editora za uređivanje teksta programa. Naredbe su se numerisale brojevima 10, 20, 30,... Ako bi trebalo dodati naredbu između 20 i 30, toj novoj naredbi dao bi se broj 25, i ona bi automatski bila smještena u memoriju na mjesto koje joj pripada po redoslijedu brojeva naredbi.
QBASIC ne mora koristiti numerisanje naredbi, jer se program uređuje pomoću solidnog editora.

3.1. Instalacija, pokretanje, način rada, izlazak, help

Za instalaciju QBASIC-a dovoljno je kopirati datoteku QBASIC.EXE na odredišni računar, jer se radi o 16-bitnoj, DOS aplikaciji.
Pokreće se iz komandne linije ("Start Menu/Programs/MS-DOS Prompt" u Windows 95/98, odnosno "Start Menu/Programs/Accessories/Command Prompt" u Windows 2000 i XP). Može se pokrenuti i pomoću "Start Menu/Run". Iza naziva programa mogu se navesti i parametri pokretanja:



QBASIC /RUN TEST.BAS	Pokreće se QBASIC interpreter, učitava se program snimljen u datoteku TEST.BAS, izvršava se taj program, te se nakon završetka programa zatvara i QBASIC interpreter
QBASIC /EDITOR A.BAS	Pokreće se QBASIC interpreter, učitava se program snimljen u datoteku A.BAS, koji se ne izvršava automatski
QBASIC	Pokreće se QBASIC interpreter, bez učitavanja datoteke

Interpreter može da radi u direktnom načinu (Immediate), odnosno da se naredbe izvršavaju odmah nakon unošenja. "Immediate" prozor se nalazi u donjem dijelu prozora QBASIC-a. Gornji dio prozora služi za programski način rada; naredbe se ne izvršavaju dok se ne zada naredba "Run" ili pritisne tipka F5. Tako unesen program se može pohraniti u datoteku naredbom "File, Save As". Tipka F6 služi za prelazak iz gornjeg u donji prozor i obrnuto. Meniji se otvaraju mišem ili tipkom ALT, naredbe se biraju strelicama na tastaturi, a izvršavaju se tipkom ENTER.Meni se može zatvoriti bez pokretanja naredbe tipkom ESC.
Tipkom SHIFT-F1 se pokreće jako dobar online-help sistem, gdje se mogu pronaći sve informacije o sintaksi naredbi, konvencijama jezika, vrstama podataka, te značenju naredbi iz menija. Za korištenje help sistema potrebno je da postoji datoteka qbasic.hlp.
U svakom redu se unosi po jedna naredba. Može se unijeti i više naredbi u jedan red, tako što se odvoje znakom ":". QBASIC interpreter se zatvara naredbom EXIT iz menija FILE.

3.2. Naredbe ulaza i izlaza (PRINT, INPUT, CLS, REM)

Za pisanje prvog programa "Hello world" dovoljna je jedna naredba. Naredbom PRINT ispisuje se tekst na ekranu. Iza naredbe PRINT mogu se unijeti:

- broj (primjer: PRINT 82.5)

- aritmetički izraz (primjer: PRINT 32-6*7)
- tekst naveden unutar navodnika (primjer: PRINT "Hello world!")

Ako se iza PRINT navede tekst bez korištenja navodnika, QBASIC će misliti da se radi o varijabli i ispisaće broj 0 – podrazumijevanu vrijednost za varijable kojima nije dodijeljena vrijednost. Ako se na kraju navede znak ";", kursor ostaje u istom redu, neposredno iza ispisanog teksta. Ako se na kraju navede znak ",", kursor ostaje u istom redu, na prvom sljedećem mjestu koje je djeljivo sa 14.
Naredba INPUT služi za unos podataka putem tastature. Jednom naredbom INPUT se može unijeti 255 karaktera.

Sintaksa:

```
INPUT [;] ["prompt"{; | ,}] varijable
```

"Prompt" je opcioni string koji se prikaže na ekranu prije nego što korisnik unese podatke. Znak ";" nakon prompta dodaje upitnik iza tog stringa. Imena varijabli se odvajaju zarezima. Znak ";" neposredno iza naredbe INPUT zadržava kursor u istom redu nakon unosa podataka.
Naredba CLS služi za brisanje ekrana. Naredba REM služi za unošenje komentara u program – sve što se navede iza REM, biće ignorisano prilikom pokretanja programa.

Primjer:

```
INPUT "Unesi broj godina" X%
INPU "Unesi dan, mjesec i godinu rođenja" D%,M%,G%
```

4. Operatori i funkcije (aritmetički, logički, komparativni)

Aritmetički operatori:

- "+" za sabiranje: 23+3=26
- "-" za oduzimanje: 34-2=32
- "*" za množenje: 4*3=12
- "/" za dijeljenje: 19/7=2.714286
- "^" za stepenovanje: 2^8=256
- "\" za cjelobrojno dijeljenje: 19\7=2
  Kad su operatori realni, vrši se implicitna konverzija, tj. zaokruživanje brojeva se vrši prije dijeljenja
  Realni brojevi koji završavaju sa .5 se zaokružuju na bliži paran broj: 3.5\2.5=4/2=2; 4.5\2.5=4/2=2; 5.5\2.5=6/2=3
- "MOD" za ostatak pri dijeljenju: 11 MOD 3 = 2
  Predznak ostatka je jednak predznaku djelitelja: -13 MOD 4 = -13 MOD –4 = -1

Aritmetičke funkcije:

- SGN(x) - daje vrijednost –1,0 ili 1 ako je x manje, jednako ili veće od nule
- ABS(x) - apsolutna vrijednost : ABS(-8)=8
- INT(x!) - najbliži manji ili jednak cijeli broj: INT(2.79)=2; INT(-2.79)=-3
- FIX(x!) - zanemaruje decimale i prikazuje cijeli dio broja: FIX(2.79)=2; FIX(-2.79)=-2
- SIN(x) - sinus ugla u radijanima
- COS(x) - kosinus ugla u radijanima
- TAN(x) - tangens ugla u radijanima
- ATN(x) - arkus tangens ugla u radijanima
- EXP(x) - e^x
- LOG(x) - prirodni logaritam (baza e)
- SQR(x) - kvadratni korijen
- STR$(x) - pretvara broj u tekstualni string: STR$(10.45)="10.45"
- VAL(tekst) – pretvara string u broj: VAL("23")=23

Logički operatori:

- NOT – ne
- AND – i
- OR – ili

Komparativni operatori:

- > - veće od
- < - manje od
- = - jednako
- >= - veće ili jednako
- <= - manje ili jednako
- <> - različito (nije jednako)

4.1. Varijable, imenovanje, sufiksi, deklaracija, dodjeljivanje vrijednosti

Imena varijabli mogu imati do 40 karaktera, a smiju se koristiti slova A-Z, cifre 0-9 i tačka. Svaka cifra zauzima po jedan karakter u imenu. Nema razlike između velikih i malih slova (varijable A22 i a22 su iste). Na kraju imena varijable se može dodati sufiks koji definiše tip podataka koji se može pohraniti u tu varijablu:

- a% : INTEGER, cijeli broj (16 bit)
  -32768...32767
- a& : LONG, veći cijeli broj (32 bit)
  -2147483648...2147483647
- a! : SINGLE, realni broj jednostruke preciznosti (32 Bit)
  +-2.802597 *10^-45...+-3.402823 *10^38
- a# : DOUBLE, realni broj dvostruke preciznosti (64 Bit)
  +-4.446590812571219 *10^-323...+-1.79769313486231 *10^308
- a$: STRING, tekst-string
  (max. 32767 karaktera)

Sufiks je sastavni dio imena, npr. varijable "A%" i "A" nisu iste, mogu imati dvije različite vrijednosti u istom programu.
Vrijednosti varijablama se dodjeljuju tako što se navede ime varijable, znak jednakosti i vrijednost. Ako je varijabla string, vrijednost koja joj se dodjeljuje mora biti navedena unutar navodnika.
Naredba SWAP služi da dvije varijable zamijene svoje vrijednosti:

T=A1: A1=A2: A2=T

je isto što i:

SWAP A1,A2

Naredba CLEAR briše sadržaj svih varijabli; sve numeričke dobiju vrijednost 0, a sve znakovne "".

5. IF-THEN-ELSE

Jedna od najčešće korištenih struktura u programiranju, uz petlje, je provjera ispunjenja nekog uslova, što za posljedicu ima promjenu toka programa. Naredba IF se u Qbasic-u može koristiti na više načina:
Sintaksa 1 (izvršavanje samo jedne naredbe kao rezultat provjere):

```
IF uslov THEN naredba-1 [ELSE naredba-2]
```

Ako je "uslov" izraz čiji je rezultat "tačno", onda se izvršava "naredba–1", a ako je rezultat izraza "uslov" "netačno", onda se izvršava "naredba–2".
Izraz "uslov" može biti bilo koji logički izraz čiji rezultat je "tačno" (true) ili "netačno" (false). Kao uslov se može koristiti i numerički izraz, čiji je rezultat brojčana vrijednost. U tom slučaju se rezultat različit od nule ponaša kao "tačno", a kad je rezultat izraza nula, ponaša se kao "netačno". Dio koda naveden unutar srednjih zagrada je opcija, tj. ne mora se navoditi uvijek nego samo kada je to potrebno.
Sintaksa 2 (izvršavanje blokova naredbi kao rezultat provjere):

```
IF uslov1 THEN
    [blok naredbi - 1]
 [ELSE
    [blok naredbi - 2]]
END IF
```

Ovdje se pod blokom naredbi smatra niz od jedne ili više naredbi.
Ako je "uslov1" izraz čiji je rezultat "tačno", onda se izvršava blok naredbi – 1, a ako je rezultat izraza "uslov1" "netačno", onda se izvršava blok naredbi – 2.
Sintaksa 3 (višestruke provjere uslova):

```
IF uslov1 THEN
    [blok naredbi - 1]
[ELSEIF uslov2 THEN
    [blok naredbi - 2]]...
[ELSE
    [blok naredbi - n]]
END IF
```

Ako je "uslov1" izraz čiji je rezultat "tačno", onda se izvršava blok naredbi – 1, a ako je rezultat izraza "uslov1" "netačno", onda se prvo provjerava "uslov2", te ako je i njegov rezultat "tačno" izvršava se blok naredbi – 2...

Primjer:

```
INPUT "1 ili 2? ", i%
IF i% = 1 OR i% = 2 THEN
 PRINT "OK"
ELSE
 PRINT "Uneseni broj nije 1 niti 2"
END IF
```

5.1. Izrazi

Izrazi koji se navode kao uslov u naredbi IF mogu biti:

a) logički – koji mogu biti prosti (poredbeni, u kojem se koriste operatori: >, >=, <, <=, =, <>) ili složeni, koji kombinuju dva ili više izraza logičkim operatorima (OR, AND, NOT)
b) numerički – rezultat izraza je brojčana vrijednost (konstanta, varijabla, izraz sa aritmetičkim operatorima)

Primjeri izraza:

```
a$ = "ime"
starost < 18
abs(x%) < 5 OR sqr(x%) mod 2 = 0
```

5.2. GOTO

Naredba GOTO se koristi kada je potrebno izvršiti "bezuslovni" skok na neki drugi dio programa. Iza naredbe se navodi ili broj naredbe na koju treba preusmjeriti tok (ako su naredbe numerisane) ili "labela" (LABEL). Labela je riječ koja se navodi unutar programa, iza koje slijedi dvotačka (:), i koja se koristi da označi početak bloka naredbi.

Primjer:

```
INPUT "Unesite broj", x%
IF x% > 0 THEN GOTO veci ELSE GOTO 93
veci:
 PRINT "Broj je pozitivan"
 GOTO kraj
93 PRINT "Broj je negativan"
kraj:
```

Na ovom primjeru se može vidjeti kako se naredba GOTO koristi za preskakanje blokova naredbi. Isti program bez naredbe GOTO bi glasio:

```
INPUT "Unesite broj", x%
IF x% > 0 THEN
 PRINT "Broj je pozitivan"
ELSE
 PRINT "Broj je negativan"
END IF
```

Ovakav program (bez upotrebe naredbe GOTO) se naziva struktuiranim programom.

5.3. SELECT CASE

Naredba SELECT CASE se koristi kada je potrebno izvršiti višestruka upoređivanja vrijednosti izraza i na osnovu njih izvršiti blok naredbi. Ova naredba u nekim slučajevima može zamijeniti višestruke ugniježđene IF-THEN-ELSE strukture.

Sintaksa:

```
SELECT CASE izraz
CASE izraz1
  [blok naredbi-1]
[CASE izraz2
  [blok naredbi-2]]...
[CASE ELSE
  [blok naredbi-n]]
END SELECT
```

Umjesto "izraz" se može navesti bilo koji numerički ili string izraz, koji se zatim upoređuje sa datim mogućim vrijednostima "izraz1", "izraz2",... Ako se u tim izrazima koriste poredbeni operatori, ispred njih se mora navesti riječ IS. Argumenti izraza mogu imati jednu od sljedećih formi, ili njihovih kombinacija, odvojenih zarezima, gdje se zarez ponaša kao logičko OR:

```
izraz[,izraz]...
izraz TO izraz
IS komparacioni operator izraz
```

Primjer:

```
INPUT "Unesite ocjenu (1-5): ", ocjena
 SELECT CASE ocjena
 CASE 5
  PRINT "Odlično!"
 CASE 2 TO 4
  PRINT "Dovoljno, dobro ili vrlo dobro."
 CASE 1
  PRINT "Nedovoljno."
 CASE ELSE
  PRINT "Nemoguća ocjena!"
 END SELECT
```

Ovaj program se interpretira na sljedeći način: "Ako je unesena ocjena 5, na ekranu se ispiše 'Odlično', ako je od 2 do 4, na ekranu se ispiše 'Dovoljno, dobro ili vrlo dobro', a ako je ocjena 1, na ekranu se ispiše 'Nedovoljno'. Za sve ostale slučajeve ispisuje se 'Nemoguća ocjena'."
Naredba CASE ELSE se koristi za sve ostale slučajeve koji se mogu pojaviti, a da nisu obuhvaćeni postojećim CASE izrazima.

Primjer (CASE sa korištenjem dvotačke za niz naredbi u jednom redu i apostrofa za komentare u redu s naredbom):

```
SELECT CASE i%
 CASE IS < 23 : x=0    'i% < 23
 CASE 50      : x=1    'i%=50
 CASE IS > 127: x=3    'i% > 127
END SELECT
```

6. FOR-NEXT, EXIT FOR

Struktura koja se ponavlja se naziva petlja. U Qbasicu postoji više naredbi koje mogu realizovati petlju, a jedna od njih je FOR-NEXT petlja. Ona se koristi u slučaju kad se unaprijed zna koliko puta se petlja treba ponoviti.

Sintaksa:

```
FOR brojac = pocetak TO kraj [STEP korak]
    [blok naredbi]
NEXT [brojac [,brojac]...]
```

U ovoj petlji je "brojac" numerička varijabla koja se koristi kao brojač petlje. "pocetak" i "Kraj" su početna i krajnja vrijednost koja se dodjeljuje brojaču, dok je "korak" veličina za koju se brojač povećava u svakom ponavljanju petlje..

Primjer:

```
FOR i% = 1 TO 15
 PRINT i%
NEXT i%
FOR i% = 7 TO -6 STEP -3
 PRINT i%
NEXT i%
```

Prva petlja će odštampati brojeve od 1 do 15 (sa korakom 1), a druga brojeve od 7 do –6, unazad, sa korakom 3. Ako petlju treba ranije završiti, koristi se naredba EXIT FOR. Ako se ne navede korak, podrazumijeva se vrijednost 1. Ako je početna vrijednost veća od krajnje, onda korak treba biti negativan broj. Višestruke petlje se ne smiju ukrštati, što znači da se varijabla-brojač koja se prva upotrijebila kod naredbe FOR treba posljednja završiti naredbom NEXT.

Primjer1 (višestruka petlja):

```
FOR i% = 1 TO 15
 PRINT i%
 FOR j% = 7 TO -6 STEP -3
```

```
  PRINT j%;
 NEXT j%
NEXT i%
```

Primjer2 (višestruka petlja sa zajedničkom naredbom NEXT):

```
FOR i% = 1 TO 15
 PRINT i%
 FOR j% = 7 TO -6 STEP -3
  PRINT j%;
 NEXT j%, i%
```

Primjer3 (prekid petlje):

```
FOR a% = 1 TO 1000
  PRINT a%;
 IF a%^2 > 5*a% THEN
  EXIT FOR
 END IF
NEXT a%
```

6.1. DO-WHILE, DO-UNTIL, EXIT DO, WHILE-WEND

Ako broj koraka petlje nije moguće unaprijed odrediti, koristi se naredba DO za realizaciju petlje. Blok naredbi između naredbi DO i LOOP se ponavlja dok je navedeni uslov zadovoljen (WHILE), odnosno dok ne bude zadovoljen (UNTIL).

Sintaksa1:

```
DO [{WHILE | UNTIL} uslov]
    [blok naredbi]
LOOP
```

Sintaksa2:

```
DO
    [blok naredbi]
LOOP [{WHILE | UNTIL} uslov]
```

Uslov je numerički ili logički izraz čiji rezultat je "tačno" (true; različit od nule), "netačno" (false; nula).

Primjer1:

```
i% = 0
PRINT "Vrijednost i% na početku petlje je "; i%
DO WHILE i% < 10
 i% = i% + 1
LOOP
PRINT "Vrijednost i% na kraju petlje  "; i%
```

I kod DO petlje se ponavljanje koraka može prekinuti naredbom EXIT DO.

Primjer2 (EXIT DO):

```
i% = 0
DO WHILE i% < 10
 i% = i%^2 - 3*i% + 14
 IF SQR(i%) > 10 THEN EXIT DO
 PRINT i%
LOOP
```

Struktura WHILE-WEND je identična strukturi DO-WHILE-LOOP, ali kod nje ne postoji mogućnost prekida petlje poput EXIT DO:

Primjer3 (WHILE-WEND):

```
i% = 0
WHILE i% < 10
 i% = i%^2 - 3*i% + 14
 PRINT i%
WEND
```

Primjer4 (DO UNTIL):

```
i% = 0
DO
 i% = i%^2 - 3*i% + 14
 IF SQR(i%) > 10 THEN EXIT DO
 PRINT i%
LOOP UNTIL i% = 10
```

7. Nizovi, DIM, unošenje, štampanje i sortiranje niza

7.1. Nizovi

U nekim programima je nepraktično koristiti veliki broj imena varijabli. Naprimjer, zadatak "Unesi N brojeva, a zatim ih ispiši na ekran sortirane po veličini" nije moguće izvesti ako se N ne zna prije početka programa. Čak i u slučaju da je N poznato, takav program je teško napisati. Za takve zadatke koriste se nizovi (indeksirane varijable). Niz je skup varijabli čije se ime sastoji od zajedničkog dijela (naziv niza) i broja koji predstavlja relativni položaj te varijable u nizu (indeks niza, navodi se u zagradi). Svaki element niza se može koristiti kao posebna promjenljiva, određena svojim indeksom.

Primjer (program za unos i ispis 5 varijabli):

Bez korištenja niza	Sa korištenjem niza

`INPUT "A1" A1`
`INPUT "A2" A2`
`INPUT "A3" A3`
`INPUT "A4" A4`
`INPUT "A5" A5`
`PRINT A1`
`PRINT A2`
`PRINT A3`
`PRINT A4`
`PRINT A5`
	`FOR I=1 TO 5`
	`  INPUT A(I)`
	`NEXT I`
	`FOR I=1 TO 5`
	`  PRINT A(I)`
	`NEXT I`

7.2. DIM

Nizovi se u Qbasicu deklarišu naredbom DIM. Deklaracijom se rezerviše prostor u memoriji za određeni broj elemenata niza, i svim elementima niza se dodjeljuje podrazumijevana vrijednost (za numeričke 0, a za string ""). Ako se jednodimenzionalni niz ne deklariše, podrazumijeva se da indeks može biti najviše 10.

Sintaksa:

DIM ime[(granica)] [AS tip] [,ime[(granica)] [AS tip]]...

"Ime" je naziv niza; može ih se navesti više, odvojenih zarezima. "Granica" je maksimalni gornji indeks niza, koja se može navesti kao jedan broj (npr. DIM X(4) će deklarisati niz X sa 5 članova, sa indeksima 0,1,2,3 i 4), ili kao raspon brojeva (npr. DIM Y(3 TO 7) će deklarisati niz Y sa 5 članova, sa indeksima 3,4,5,6 i 7). Umjesto "tip" se navodi vrsta podataka koja će se pohranjivati u niz (INTEGER, LONG, SINGLE, DOUBLE, STRING). Podrazumijevani tip je LONG.

Primjer:

```
DIM LOTO(1 TO 39) AS INTEGER, BINGO(1 TO 99) AS INTEGER, BrojPogodaka(7)
```

Naredba OPTION BASE se može koristiti da se promijeni način indeksiranja niza, tj. da li će indeksi počinjati od 0 ili od 1.

Primjer (deklaracija niza DAN sa indeksima od 1 do 31 i niza SAT sa indeksima od 0 do 24):

```
OPTION BASE 1
DIM DAN(31) AS INTEGER
OPTION BASE 0
DIM SAT(24)
```

7.3. Unošenje, štampanje i sortiranje niza

U narednim primjerima prikazano je kako se deklarišu, unose i ispisuju (štampaju) nizovi za slučajeve kad jeste ili kad nije unaprijed poznat maksimalni broj članova niza. Treba obratiti pažnju da varijable koje se koriste kao indeksi imaju sufiks "%", što znači da su cjelobrojne.

Primjer1 (deklaracija i unošenje niza sa tastature ako NIJE unaprijed poznat maksimalan broj članova niza):

```
PRINT "Unesi broj članova niza"
INPUT N%
CLS
DIM X(1 TO N%)
PRINT "Unesi članove niza"
FOR I% = 1 TO N%
 INPUT X(I%)
NEXT I%
```

Primjer2 (deklaracija i unošenje niza sa tastature ako je poznat broj članova niza):

```
DIM X(5)
PRINT "Unesi članove niza"
FOR I% = 0 TO 5
 INPUT X(I%)
NEXT I%
```

Primjer3 (štampanje niza, članovi niza se ispisuju jedan ispod drugog):

```
DIM Y(1 TO 10)
CLS
PRINT "Članovi niza Y"
FOR I% = 1 TO 10
 PRINT "Y("; I%; "=";  Y(I%)
NEXT I%
```

Primjer4 (štampanje niza, članovi niza se ispisuju jedan pored drugog):

```
DIM Y(1 TO 10)
CLS
PRINT "Članovi niza Y"
FOR I% = 1 TO 10
 PRINT Y(I%);
NEXT I%
```

Primjer5 (sortiranje niza):

```
DIM A(6), B(6)
CLS
PRINT "Unesite članove niza A"
FOR I% = 0 TO 6
 INPUT A(I%)
NEXT I%
  REM Niz B ima iste članove kao niz A
FOR I% = 0 TO 6
 B(I%) = A(I%)
NEXT I%
  REM Sortiranje niza B od najmanjeg ka najvećem (rastući poredak)
FOR I% = 0 TO 5
  FOR J% = I%+1 TO 6
    IF B(J%) < B(I%) THEN SWAP B(I%), B(J%)
NEXT J%, I%
  REM Štampanje nizova A i B jedan pored drugog
FOR I% = 0 TO 6
  PRINT A(I%), B(I%)
NEXT I%
```

U primjeru 5 treba obratiti pažnju da, ako se želi zapamtiti niz A i u nesortiranom obliku, potrebno je formirati drugi niz sa različitim imenom a istim članovima (niz B). Sortiranje se vrši tako što se pokrenu dvije petlje, prva za indekse niza od 0 do 5 (od prvog do predzadnjeg člana), a druga za preostale veće indekse. Pomoću prve petlje se svi članovi niza (sa indeksom I%) porede sa preostalim članovima (koji imaju indekse J% od I%+1 do 6). Ako se nađe član sa indeksom J% koji je manji od člana sa indeksom I%, oni zamijene svoje vrijednosti naredbom SWAP.
Za sortiranje u opadajućem poretku (od najvećeg ka najmanjem), treba samo zamijeniti znak "<" u naredbi IF znakom ">".

8. Matrice, transponovanje, sabiranje, množenje, ekstrakcija nizova iz matrice

8.1. Matrice

Matrice su dvodimenzionalni nizovi. Kao i niz, i matrica ima naziv, ali za razliku od niza ima dva indeksa. Prvi indeks predstavlja oznaku horizontalnog reda, a drugi indeks vertikalne kolone. Sva pravila naredbe DIM koja važe za nizove, važe i za matrice.

$$Z_{3x4} = \begin{bmatrix} Z(1,1) & Z(1,2) & Z(1,3) & Z(1,4) \\ Z(2,1) & Z(2,2) & Z(2,3) & Z(2,4) \\ Z(3,1) & Z(3,2) & Z(3,3) & Z(3,4) \end{bmatrix}$$

Primjer1 (unošenje matrice sa 5 redova i 7 kolona red po red, tako da se svaki član unosi zasebno):

```
DIM Z(1 TO 5, 1 TO 7)
FOR I% = 1 TO 5
  FOR J% = 1 To 7
    INPUT Z(I%,J%)
NEXT J%, I%
```

Primjer2 (ispisivanje matrice sa 5 redova i 7 kolona):

```
FOR I% = 1 TO 5
  FOR J% = 1 To 7
    PRINT Z(I%,J%);
  NEXT J%
  PRINT
NEXT I%
```

U primeru 2 treba obratiti pažnju da iza naredbe PRINT Z(I%,J%) dolazi znak ";", što znači da petlja J% služi za ispis svih članova matrice u jednom redu. Naredba PRINT bez argumenata (između dva NEXT) služi za prelazak u novi red kada se promijeni indeks reda (I%).
Transponovanje matrice je postupak zamjene mjesta redova i kolona (matrica se rotira oko glavne dijagonale):

Primjer3 (transponovanje matrice $D_{6x8} = (C_{6x8})^T$):

```
FOR I% = 1 TO 6
  FOR J% = 1 To 8
    D(I%,J%) = C(J%,I%)
NEXT J%, I%
```

8.2. Sabiranje matrica

Matrice se sabiraju tako što im se zbroje odgovarajući članovi (član iz trećeg reda, druge kolone prve matrice sa članom iz trećeg reda i druge kolone druge matrice, itd):

$$A_{2x3} + B_{2x3} = \begin{bmatrix} A(1,1)+B(1,1) & A(1,2)+B(1,2) & A(1,3)+B(1,3) \\ A(2,1)+B(2,1) & A(2,2)+B(2,2) & A(2,3)+B(2,3) \end{bmatrix}$$

Primjer (sabiranje matrica X i Y sa po 5 redova i 7 kolona; Z=X+Y):

```
DIM X(1 TO 5, 1 TO 7), Y(1 TO 5, 1 TO 7), Z(1 TO 5, 1 TO 7)
FOR I% = 1 TO 5
  FOR J% = 1 To 7
    Z(I%,J%) = X(I%,J%) + Y(I%,J%)
NEXT J%, I%
```

8.3. Množenje matrica

Ovdje je prikazan primjer množenja dvije matrice sa po dva reda i dvije kolone. Vidi se da treba formirati zbir proizvoda odgovarajućih redova i kolona, čiji presjek daje indekse člana rezultujuće matrice.

$$A_{2x2} \times B_{2x2} = \begin{bmatrix} A(1,1)\times B(1,1) + A(1,2)\times B(2,1) & A(1,1)\times B(1,2) + A(1,2)\times B(2,2) \\ A(2,1)\times B(1,1) + A(2,2)\times B(2,1) & A(2,1)\times B(1,2) + A(2,2)\times B(2,2) \end{bmatrix}$$

Za matrice višeg reda, pojedini elementi se formiraju sabiranjem proizvoda prema:

$$Z_{MxN} = X_{MxL} \times Y_{LxN} \qquad Z_{i,j} = \sum_{k=1}^{l} X_{i,k} \times Y_{k,j}$$

Da bi se matrice mogle pomnožiti, moraju biti saglasne, tj. broj kolona prve matrice mora biti jednak broju redova druge. Rezultujuća matrica će imati broj redova kao kod prve, a broj kolona kao kod druge matrice.

Primjer (množenje matrica $Z_{4x8} = X_{4x6} \times Y_{6x8}$):

```
DIM X(1 TO 4, 1 TO 6), Y(1 TO 6, 1 TO 8), Z(1 TO 4, 1 TO 8)
FOR I% = 1 TO 4
  FOR J% = 1 To 8
    Z(I%,J%) = 0
    FOR K% = 1 TO 6
      Z(I%,J%) = Z(I%,J%) + X(I%,K%) * Y(K%,J%)
    NEXT K%
NEXT J%, I%
```

Prilikom množenja, treba formirati tri petlje, prva za redove rezultujuće matrice Z, druga za kolone rezultujuće matrice Z. Broj proizvoda u zbiru je jednak broju kolona prve, odnosno broju redova druge matrice (K% = 1 do 6).

8.4. Ekstrakcija nizova iz matrice

U primjeru 1 se od druge kolone matrice Z_{3x4} formira niz A, a od trećeg reda niz B. Treba obratiti pažnju da se I% može koristiti i za označavanje redova i kolona, samo što se u dosadašnjim primjerima radi preglednosti koristila oznaka I% za redove i J% za kolone.

Primjer1:

```
DIM Z(1 TO 3, 1 TO 4), A(1 TO 3), B(1 TO 4)
FOR I% = 1 TO 3
  A(I%) = Z(I%,2)
NEXT I%
FOR I% = 1 To 4
  B(I%) = Z(3,I%)
NEXT I%
```

$$Z_{3x4} = \begin{bmatrix} Z(1,1) & Z(1,2) & Z(1,3) & Z(1,4) \\ Z(2,1) & Z(2,2) & Z(2,3) & Z(2,4) \\ Z(3,1) & Z(3,2) & Z(3,3) & Z(3,4) \end{bmatrix} \quad A = \begin{bmatrix} Z(1,2) \\ Z(2,2) \\ Z(3,2) \end{bmatrix} \quad B = \begin{bmatrix} Z(3,1) \\ Z(3,2) \\ Z(3,3) \\ Z(3,4) \end{bmatrix}$$

U primjeru 2 se od glavne dijagonale matrice A_{6x4} formira niz X, a od sporedne niz Y. Iako se radi o matrici, dovoljno je koristiti samo jednu petlju, jer broj članova dijagonale je jednak broju redova ili broju kolona (zavisno od toga koji je od ta dva broja manji).

Primjer2:

```
DIM A(1 TO 6, 1 TO 4), X(1 TO 4), Y(1 TO 4)
FOR I% = 1 TO 4
  X(I%) = A(I%,I%)
  Y(I%) = A(I%,5-I%)
NEXT I%
```

$$A_{6x6} = \begin{bmatrix} A(1,1) & A(1,2) & A(1,3) & A(1,4) \\ A(2,1) & A(2,2) & A(2,3) & A(2,4) \\ A(3,1) & A(3,2) & A(3,3) & A(3,4) \\ A(4,1) & A(4,2) & A(4,3) & A(4,4) \\ A(5,1) & A(5,2) & A(5,3) & A(5,4) \\ A(6,1) & A(6,2) & A(6,3) & A(6,4) \end{bmatrix} \quad X = \begin{bmatrix} A(1,1) \\ A(2,2) \\ A(3,3) \\ A(4,4) \end{bmatrix} \quad Y = \begin{bmatrix} A(1,4) \\ A(2,3) \\ A(3,2) \\ A(4,1) \end{bmatrix}$$

U primjeru 3 se od parnih članova matrice A formira niz B. Kako broj parnih članova nije poznat na početku programa, pretpostavi se da je taj broj jednak nuli. Pomoću dvije petlje, jedne za redove i jedne za kolone, provjeri se parnost svakog člana matrice A. Kad se nađe član čiji je ostatak pri dijeljenju sa 2 nula (parni

član), broj parnih članova (K) se povećava za jedan. Niz B se delariše za maksimalni broj (proizvod broja redova i kolona matrice A).
U drugom dijelu primjera 3 se izračunava zbir članova niza B. Prvo se pretpostavi da je taj zbir jednak nuli, a zatim se taj zbir u svakom koraku petlje povećava za vrjednost člana niza B. Treba uočiti sličnost između operacija prebrojavanja (kada se zbir povećava za 1) i sabiranja (kada se zbir povećava za vrijednost člana niza ili matrice).

Primjer3:

```
DIM A(1 TO 6, 1 TO 4), B(1 TO 24)
K% = 0
FOR I% = 1 TO 6
  FOR J% = 1 TO 4
  IF A(I%,J%) MOD 2 = 0 THEN
    K% = K%+1
    B(K%) = A(I%,J%)
  END IF
NEXT J%,I%
ZBIR = 0
PRINT "NIZ B:"
FOR I% = 1 TO K%
  PRINT B(I%)
  ZBIR = ZBIR + B(I%)
NEXT I%
PRINT
PRINT "ZBIR ČLANOVA NIZA B ="; ZBIR
```

9. String funkcije

Stringovi su varijable i konstante koje se ponašaju kao tekst, tj. iako mogu sadržati i slova i cifre, nad njima se ne mogu vršiti numeričke operacije, kao što su sabiranje, množenje, stepenovanje,... Kada se string varijabli dodjeljuje vrijednost, ta vrijednost mora biti pod navodnicima

Primjer:

```
IME$ = "Alan"
PREZIME$ = "Ford"
```

I nad string varijablama se mogu vršiti operacije, pomoću string operatora i funkcija:

Funkcija / operator	**Opis**	**Primjer**	**Rezultat**
LEN (<String>)	Broj karaktera u stringu	LEN("Oliver")	6
LEFT$ (<String>,<N>)	Prvih "N" karaktera stringa	LEFT$("Bob",2)	"Bo"
RIGHT$ (<String>,<N>)	Zadnjih "N" karaktera stringa	RIGHT$("Grunf",3)	"unf"
MID$ (<String>,<P>,<K>)	Sredina stringa, od P-tog do K-tog karaktera	MID$("Grunf",2,4)	"run"
LCASE$ (<String>)	Pretvaranje stringa u mala slova	LCASE$("Bob")	"bob"
UCASE$ (<String>)	Pretvaranje stringa u velika slova	UCASE$("Bob")	"BOB"
LTRIM$ (<String>)	Uklanja prazna mjesta s lijeve strane stringa	LTRIM$(" BOB ")	"BOB "
RTRIM$ (<String>)	Uklanja prazna mjesta s desne strane stringa	RTRIM$(" BOB ")	" BOB"
SPACE$ (N)	Kreira string od "N" praznih mjesta	SPACE$(3)	" "
VAL (<string>)	Pretvara string u broj	VAL (" 3.1E12cm")	3.1E+12
STR$ (<broj>)	Pretvara broj u string	STR$ (32)	"32"
HEX$ (<broj>)	Pretvara decimalni broj u string jednak heksadecimalnoj vrijednost tog broja	HEX$(100)	"64"

Funkcija / operator	Opis	Primjer	Rezultat
OCT$ (<broj>)	Pretvara decimalni broj u string jednak oktalnoj vrijednost tog broja	OCT$(100)	"144"
INSTR ([N,]string1,string2)	Daje poziciju prvog pojavljivanja stringa2 u stringu1, počevši od pozicije N. Ako se N izostavi, podrazumijeva se N=1	INSTR("BobRock","Rock")	4
STRING$ (<n>, <Slovo>)	Kreira string od "n" slova	STRING$(5,"C")	"CCCCC"
<string> + <string>	Spaja dva stringa u jedan	"Broj "+"Jedan"	"Broj Jedan"
ASC (<String>)	Daje ASCII kod prvog karaktera u stringu	ASC("a")	97
CHR$ (<N>)	Daje karakter čiji je ASCII kod "N"	CHR$(97)	"a"

Primjer:

```
INPUT Text$
IF UCASE$(Text$) = "END" THEN END
PRINT "Decimalno 65 se u heksadecimalnom brojnom sistemu piše ";
PRINT "H" + HEX$(65)
```

10. Potprogrami - FUNCTION

Dijelovi koda koji se ponavljaju više puta se mogu izdvojiti kao posebna cjelina (potprogram) koja se zatim može više puta pozivati jednom naredbom. Potprogrami se dijele na potprograme-subroutine (koji mogu, a ne moraju vraćati vrijednost neke varijable kao rezultat), i na funkcije (koje obavezno vraćaju neku vrijednost u glavni program). Naredba DEF FN se koristi za definisanje lokalnog potprograma (lokalne funkcije).

Sintaksa:

```
DEF FNnaziv [(parametri)]
    [blok naredbi]
  FNnaziv = izraz
    [blok naredbi]
END DEF
```

Funkcija se definiše između naredbi DEF i END DEF, a njen naziv se koristi kao varijabla, čije ime mora početi sa FN, i čija vrijednost mora biti određena definicijom. Funkcije mogu imati i jedan ili više parametara, odvojenih zarezima. Sufiksi koji se dodaju na imena varijabli se mogu koristiti i kod imena funkcija. Naprimjer, ako se u programu više puta javlja funkcija $f(x,t)=3\cdot x^3-12.5\cdot t^2+7\cdot x-2\cdot t-0.5$, ta se funkcija može definisati sa vlastitim imenom, te se u programu poziva kao da je jedna od Qbasic funkcija.

Primjer1 (Funkcija sa dva argumenta):

```
DEF FNf(s,v)
  FNf = 3*s^3-12.5*v^2+7*s-2*v-0.5
END DEF
T = 0.5
FOR X = 1 TO 10 STEP 0.2
 PRINT T, X, FNf(X,T)
NEXT X
```

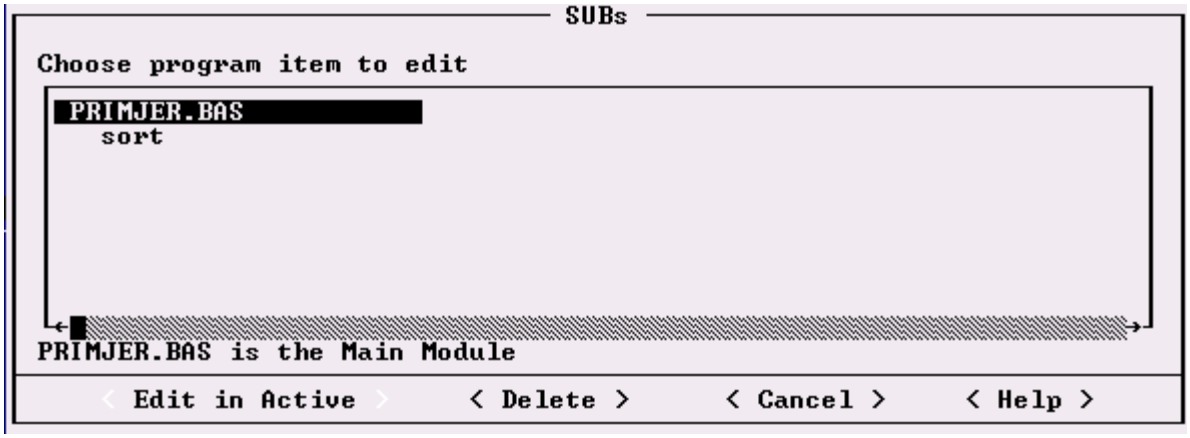
Drugi način definisanja funkcija je pomoću naredbe "DECLARE FUNCTION". To su takozvane "eksterne" funkcije. Unutar glavnog programa se deklariše funkcija naredbom "DECLARE FUNCTION", a zatim se naredbom FUNCTION definiše šta ta funkcija treba da radi.

Sintaksa:

```
FUNCTION naziv [(parametri)]
    [blok naredbi]
  naziv = izraz
    [blok naredbi]
END FUNCTION
```

Funkcija se definiše između naredbi FUNCTION i END FUNCTION, a njen naziv se koristi kao varijabla, čija vrijednost mora biti određena definicijom. Funkcije mogu imati i više parametara, odvojenih zarezima. Sufiksi koji se dodaju na imena varijabli se mogu koristiti i kod imena funkcija.

Tipka F2 ili naredba "SUBs..." iz menija "View" služi za izbor koja će se funkcija/potprogram ili glavni program prikazati na ekranu. Naredbom "Split" iz menija "View" se ekran može podijeliti po visini na dva dijela, a zatim se tipkom F2 bira u kojem dijelu će se prikazati koji potprogram/funkcija. Naredbom "Edit in Active" se označeni potprogram prikazuje u aktivnom prozoru.



U sljedećem primjeru su date funkcije koje pretvaraju stepene Celzijusa u Farenhajte, odnosno u stepene Kelvina. Pri tom eje jedna funkcija definisana kao eksterna (FUNCTION), a druga kao lokalna (DEF FN). Treba obratiti pažnju na deklarisanje tipa podataka unutar deklaracije funkcije.

Primjer2 (Pretvaranje temperature):

```
REM Glavni program
DECLARE FUNCTION farenhajt! (celz AS SINGLE)
DEF FNKelvin (celz AS SINGLE)
  FNKelvin = celz + 273.15
END DEF

INPUT "Unesi temperaturu u stepenima Celzijusa "; stepen
PRINT FNKelvin(stepen); "K"
PRINT farenhajt!(stepen); "F"
END

REM Eksterna funkcija
FUNCTION farenhajt! (celz AS SINGLE)
  farenhajt! = celz * 1.8 + 32
END FUNCTION
```

Naziv funkcije se može navesti samo sa lijeve strane znaka jednakosti unutar funkcije; zbog toga se u primjeru 3 koristi privremena varijabla "z%". Pojavljivanje naziva desno od znaka "=" Qbasic shvata kao poziv te funkcije (funkcija znači poziva samu sebe).

Primjer3 (Faktorijel):

```
REM Glavni program
DECLARE FUNCTION fakt%(x%)
INPUT "Unesite broj" K%
PRINT "Faktorijel tog broja je"; fakt%(K%)
PRINT "K!/(K-2)! ="; fakt%(K%)/fakt%(K%-2)

REM Eksterna funkcija
FUNCTION fakt%(x%)
  z% = 1
  FOR i% = 2 TO x%
    z% = z% * i%
  NEXT i%
  fakt% = z%
END FUNCTION
```

10.1. Potprogrami - SUB

Potprogrami-subroutine ne moraju vraćati neku vrijednost u glavni program. Kao i funkcije, moraju se deklarisati naredbom "DECLARE SUB", a zatim definisati naredbom "SUB". Tipka F2 ili naredba "SUBs..." iz menija "View" služi za izbor koja će se funkcija/potprogram ili glavni program prikazati na ekranu.

Sintaksa:

```
SUB naziv [(parametri)]
    [blok naredbi]
END FUNCTION
```

Potprogram se definiše između naredbi SUB i END SUB. Ako potprogram ima više parametara, odvajaju se zarezima. Sufiksi koji se dodaju na imena varijabli se ne mogu koristiti kod imena opštih potprograma (jer se to ime ne može koristiti kao varijabla). Iz glavnog programa se potprogrami pozivaju naredbom CALL.

Primjer1 (Potprogram koji štampa tekst):

```
REM Glavni program
DECLARE SUB Ispis()
CLS
CALL Ispis
END

REM Potprogram
SUB Ispis
  PRINT "Dobar dan"
END SUB
```

Kod potprograma se kao parametri mogu koristiti imena varijabli, konstante ili nizovi. Imena nizova se moraju pisati sa sufiksom "()". Ako se parametar navede unutar zagrada, potprogram može koristiti njegovu vrijednost, ali je ne može promijeniti (ne vraća promijenjenu vrijednost nazad u glavni program).

Primjer2 (Potprogram koji sortira niz):

```
REM Glavni program
DECLARE SUB sort(x(),n)
DIM NIZ(10)
FOR I = 1 TO 10
  INPUT NIZ(I)
NEXT I
PRINT "Nesortirani niz:"
FOR I = 1 TO 10
  PRINT NIZ(I)
NEXT I
CALL sort(NIZ(),10)
PRINT "Sortirani niz:"
FOR I = 1 TO 10
  PRINT NIZ(I)
NEXT I
END

REM Potprogram za sortiranje
SUB sort(x(),n)
  FOR I% = 1 TO n-1
    FOR J% = I%+1 TO n
      IF NIZ(J%) < NIZ(I%) THEN SWAP NIZ(I%), NIZ(J%)
  NEXT J%, I%
END SUB
```

U primjeru 3 su dati potprogrami za pretvaranje uglova datih stepenima, minutama i sekundama u sekunde i obrnuto, da bi se dva ugla mogla sabrati. Potprogrami su definisani bez parametara, ali su zato varijabla "sekunde" i niz "kut(x)" deklarisane kao globalne naredbom "DIM SHARED"; to znači da se njihova vrijednost prenosi kroz glavni program i sve potprograme.

Primjer3 (Sabiranje dva ugla data u stepenima, minutama i sekundama):

```
REM Glavni program
DECLARE SUB izsekundi ()
DECLARE SUB usekunde ()
DIM SHARED kut(3) AS INTEGER
DIM SHARED sekunde AS LONG
PRINT "Unesi prvi ugao (stepeni, minuta, sekundi)";
```

```
INPUT kut(1), kut(2), kut(3)
CALL usekunde
temp = sekunde
PRINT "Unesi drugi ugao (stepeni, minuta, sekundi)";
INPUT kut(1), kut(2), kut(3)
CALL usekunde
sekunde = sekunde + temp
CALL izsekundi
FOR I = 1 TO 3
  PRINT kut(I);
NEXT I
END

REM Potprogrami
SUB izsekundi
  kut(1) = sekunde \ 3600
  kut(2) = sekunde \ 60 MOD 60
  kut(3) = sekunde MOD 60
END SUB

SUB usekunde
  sekunde = 3600 * kut(1) + 60 * kut(2) + kut(3)
END SUB
```

11. Rad sa datotekama

Podaci se ne moraju uvijek unositi putem tastature, niti se rezultat mora prikazati na ekran. Umjesto toga, moguće je ulaz/izlaz preusmjeriti na datoteku. Prije korištenja datoteke, potrebno im je dodijeliti broj, naredbom OPEN:

Sintaksa:

```
OPEN file$ [FOR mode] AS [#]Broj%
```

Umjesto "file$" se navodi naziv datoteke (koji može uključivati i putanju sa nazivima foldera i/ili diskova). Umjesto "mode" se unosi način otvaranja datoteke: APPEND, INPUT, OUTPUT. "Broj%" je cijeli broj od 1 do 255 koji će se u programu koristiti kao oznaka te datoteke (to znači da se iz istog programa može otvoriti i koristiti više datoteka, svaka sa svojim brojem).
Načini otvaranja datoteke imaju sljedeće značenje: APPEND – podaci koji su ranije bili u datoteci ostaju u njoj, a novi podaci se dodaju na kraj te datoteke; INPUT – datoteka se otvara za sekvencijalno čitanje podataka iz nje, tj. u nju se ne mogu dodavati novi podaci; OUTPUT – podaci koji su ranije bili u datoteci se brišu, i datoteka se otvara za sekvencijalni unos podataka;
Nakon što se datoteka otvori, podaci se u nju unose naredbom PRINT #N (umjesto N se stavlja broj datoteke). Čitanje podataka iz datoteke otvorene sa brojm M se vrši naredbom INPUT #M. Po završetku rada sa datotekom, potrebno ju je zatvoriti naredbom CLOSE. Iza naredbe CLOSE se navode brojevi datoteka koje treba zatvoriti. Ako se izostavi broj, zatvaraju se sve otvorene datoteke.

Primjer1:

```
INPUT "Unesite ime datoteke: "; n$
OPEN n$ FOR OUTPUT AS #1
PRINT #1, "Ovo je snimljeno u datoteku."
CLOSE
OPEN n$ FOR INPUT AS #1
INPUT #1, a$
PRINT "Podaci iz datoteke: "; a$
CLOSE
```

Funkcija EOF se koristi za testiranje završetka datoteke. Funkcija EOF daje vrijednost "tačno" (true) kada se prilikom čitanja podataka iz datoteke dođe do kraja datoteke. Ta se funkcija koristi kada se iz datoteke uzimaju podaci za koje se ne zna unaprijed koliko će ih biti.

Sintaksa:

```
EOF(broj%)
```

Parametar "broj%" predstavlja broj pod kojim je ta datoteka otvorena naredbom OPEN.

Primjer2:

```
CLS
```

```
OPEN "TEST.DAT" FOR OUTPUT AS #1
  FOR i% = 1 TO 10
    PRINT #1, i%, 2 * i%, 5 * i%
  NEXT i%
CLOSE #1
OPEN "TEST.DAT" FOR INPUT AS #1
  DO
    LINE INPUT #1, a$
      PRINT a$
  LOOP UNTIL (EOF(1))
CLOSE #1
```

U ovom primjeru se otvara datoteka "TEST.DAT", u nju se unose brojevi 1,2,5; 2,4,10; 3,6,15;... 10,20,50, po tri broja u jednom redu, pomoću FOR-NEXT petlje. Zatim se ponovo otvara ista datoteka, ovaj put za čitanje podataka. Redovi iz datoteke se pomoću DO-LOOP petlje unose u string varijablu "a$", koja se zatim ispisuje na ekran. Iz petlje se izlazi kada funkcija EOF(1) da vrijednost "true", odnosno kada se dođe do kraja datoteke "TEST.DAT".
Naredba "LINE INPUT" se od naredbe "INPUT" razlikuje po tome što LINE INPUT čita podatke sve dok ne naiđe na kraj reda (red po red), dok naredba INPUT čita podatke dok ne naiđe na zarez. Dakle, naredba "LINE INPUT #1, a$" čita cijeli red i smješta ga u varijablu "a$", dok bi naredba "INPUT #1, a$" čitala svaki broj posebno, zato što su u datoteci "TEST.DAT" podaci upisani u obliku:

```
1,2,5
2,4,10
3,6,15
...
```

Primjer3 (Niz iz datoteke "podaci.txt" se sortira i upisuje u datoteku "rezultat.txt". Zatim se u datoteku "ekstremi.txt" koja se nalazi na disketi – A: upisuju najveći i najmanji član niza):

```
OPEN "podaci.txt" FOR INPUT AS #10
OPEN "rezultat.txt" FOR OUTPUT AS #20
OPEN "A:\ekstremi.txt" FOR OUTPUT AS #30
DIM NIZ(1 TO 10)
  REM Unošenje članova niza iz datoteke podaci.txt
FOR I% = 1 TO 10
 INPUT #10, NIZ(I%)
NEXT I%
REM Sortiranje niza
FOR I% = 1 TO 9
  FOR J% = I%+1 TO 10
    IF NIZ(J%) < NIZ(I%) THEN SWAP NIZ(I%), NIZ(J%)
NEXT J%, I%
  REM Zapisivanje sortiranog niza u datoteku rezultat.txt
FOR I% = 1 TO 10
  PRINT #20, NIZ(I%)
NEXT I%
  REM Pronalaženje najvećeg i najmanjeg člana
  REM Pretpostavlja se da je prvi član i najveći i najmanji
  REM Zatim se svi ostali članovi uporede s njima
NAJV = NIZ(1)
NAJM = NIZ(1)
INAJV% = 1
INAJM% = 1
FOR I% = 2 TO 10
  IF NIZ(I%) > NAJV THEN
    NAJV = NIZ(I%)
    INAJV% = I%
  ELSE
    IF NIZ(I%) < NAJM THEN
      NAJM = NIZ(I%)
      INAJM% = I%
    END IF
  END IF
NEXT I%
  REM Zapisivanje najvećeg i najmanjeg člana u datoteku rezultat.txt
```

```
  REM Funkcije LTRIM$ i STR$ se koriste da se izbjegnu prazna mjesta
  REM između zagrade i broja
PRINT #30, "Najveći član niza je: NIZ(" + LTRIM$(STR$(INAJV%)) + ")";
PRINT #30, "="; NAJV
PRINT #30, "Najmanji član niza je: NIZ(" + LTRIM$(STR$(INAJM%)) + ")";
PRINT #30, "="; NAJM
CLOSE
```

Rezultati u datoteci "ekstremi.txt" (ako su npr. najveći 3. član niza sa vrijednošću 25 a najmanji 6. član sa vrijednošću –3) bi izgledali ovako:

```
Najveći član niza je: NIZ(3) = 25
Najmanji član niza je: NIZ(6) = -3
```

Ako se ne bi koristile funkcije LTRIM$ i STR$, onda bi kraj programa izgledao ovako:

```
PRINT #30, "Najveći član niza je: NIZ("; INAJV%; ")"; "="; NAJV
PRINT #30, "Najmanji član niza je: NIZ("; INAJM%; ")"; "="; NAJM
```

Rezultati u datoteci "ekstremi.txt" za pomenuti primjer izgledali bi ovako:

```
Najveći član niza je: NIZ( 3 ) = 25
Najmanji član niza je: NIZ( 6 ) = -3
```

12. Rješavanje problema uz pomoć računara

Faze rješavanja problema primjenom računara su izvedene iz faza rješavanja matematičkih problema: razumijevanje zadatka, stvaranje plana rješavanja zadatka, realizacija plana rješavanja zadatka i osvrt na rješenje. U informatici te četiri faze su razrađene u sedam faza:

1. Definisanje problema
   U ovoj fazi treba odgovoriti na pitanja: **šta** treba da se uradi, **kada** treba da se uradi i **koliko puta** treba da se to uradi da bi se zadati problem riješio.
2. Definisanje izlaza
   Svrha svakog programa je dobijanje nekih izlaznih podataka. Da bi se znalo šta i kako program treba da uradi, na početku se mora definisati šta se očekuje kao rezultat tog programa.
3. Definisanje ulaza
   Pored izlaznih, treba pravilno definisati i ulazne podatke koje treba ili koje je moguće obezbijediti da bi se dobio željeni izlaz. U ovoj fazi se definiše i način obezbjeđivanja ulaznih podataka (ručni, automatski, poluatomatski, iz datoteke,...
4. Determinisanje procesa
   Nakon definisanja problema, ulaza i izlaza treba pristupiti determinisanja procesa kojim će se raspoloživi ulaz transformisati u željeni izlaz.
5. Dizajniranje, kodiranje programa
   U procesu kodirnja programa važna su tri koncepta: generalnost, modularnost i hijerarhija. Generalnost znači da će program raditi sa svim ulaznim podacima. Modularnost je proces raščlanjivanja (dekompozicije) procesa i podjelu na pojedinačne funkcije u kojima se rješavaju zasebni dijelovi problema. Svaka pojedinačna funkcija postaje modul ili potprogram koji se povezuje sa drugim potprogramima i sastavni je dio glavnog programa. Da bi se ostvarile veze među modulima, potrebno je ostvariti hijerarhijsku strukturu sistema, po kojoj će se znati koji modul ima nadređenu a koji podređenu vezu prema drugim modulima.
6. Testiranje programa
   Veoma je teško u jednom koraku doći do konačnog proizvoda. Veoma često softverske firme na tržište daju alfa i beta verzije softvera, kako bi ih sami korisnici testirali i ukazali na greške koje postoje u programu. Čak i nakon izlaska finalne verzije programa, često se putem Interneta ili na drugi način objavljuju "zakrpe" (patch, service pack, update)– programi koji popravljaju uočene greške u programu.
7. Vrednovanje programa
   Kvalitet programa može biti interni i eksterni. Interni kvalitet označava efikasnost programa sa stanovišta korištenja računarskih resursa. Ocjenjivanje eksternog kvaliteta programa vrše i programeri i sami korisnici. Postoje čak i međunarodni standardi za ocjenu kvaliteta softverskih rješenja.

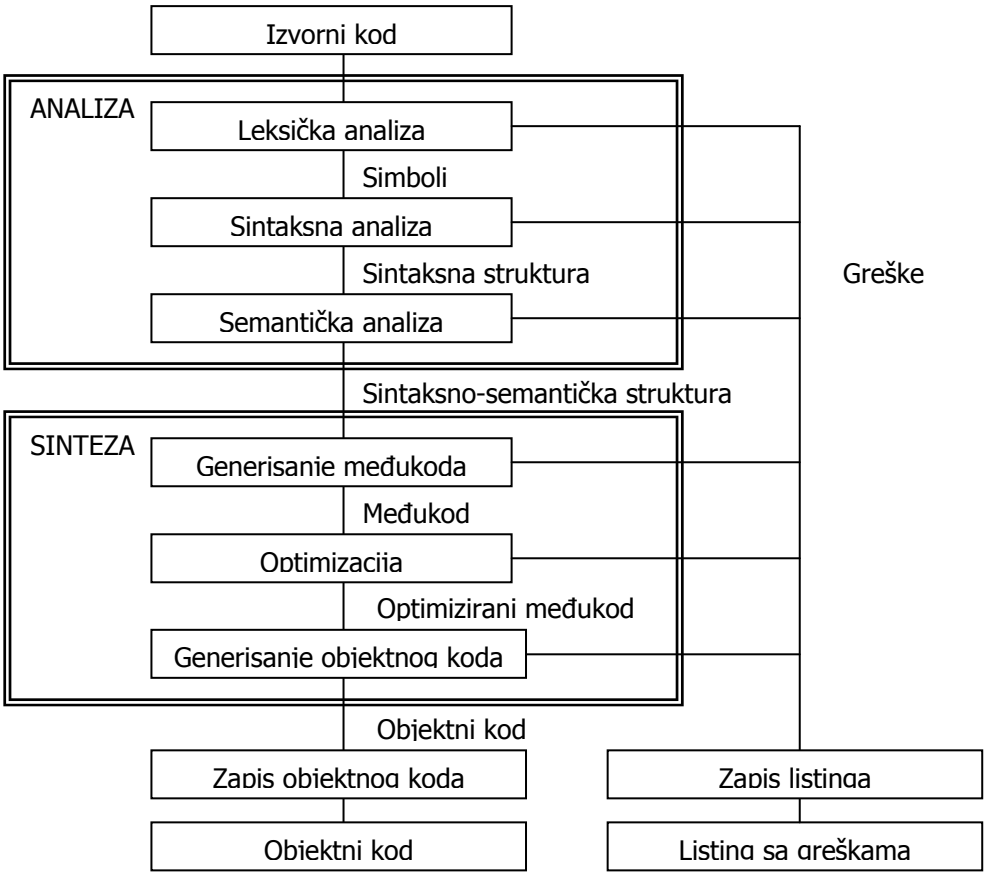
Faza dizajniranja (kodiranja) programa se sastoji iz sinteze matematičkog modela, definicije algoritma rješavanja, izbora programskog jezika i samog kodiranja (pisanja izvornog koda). Matematički model čini skup aritmetičkih i logičkih izraza kojima se dati problem opisuje na način koji je moguće riješiti računarom.

Definicija algoritma obuhvata predviđanje svih mogućih slučajeva do kojih može doći (dijeljenje s nulom, korijen negativnog broja, i sl.). Izbor programskog jezika zavisi od tri faktora:

- Kojim jezikom vlada programer?
- Da li je na raspolaganju prevodilac za taj jezik?
- Da li je taj jezik pogodan za rješavanje tog tipa problema?

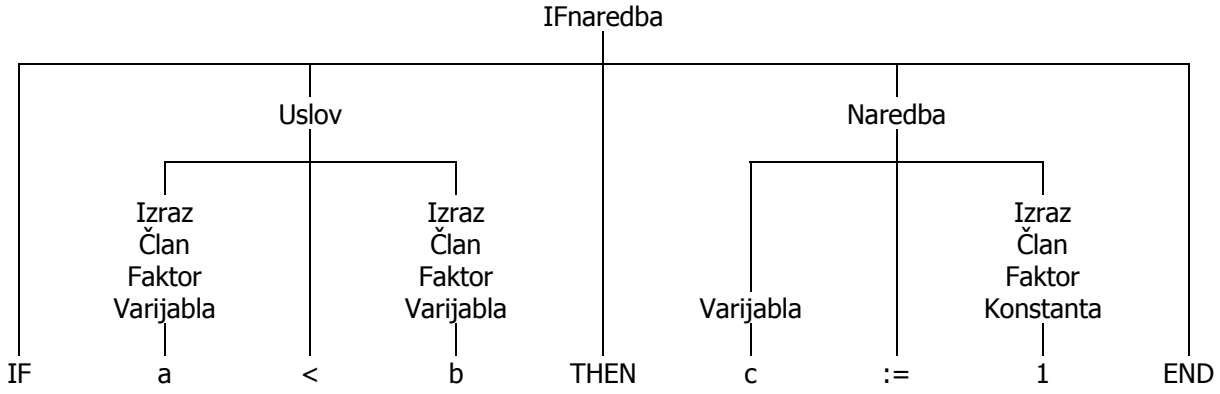
12.1. Struktura kompajlera i faze kompajliranja programa

Prevođenje programa iz izvornog u objektni kod je složen proces i stoga se razlaže na više logičkih koraka koji se nazivaju faze komapliranja. Na ulazu kompajlera je izvorni kod napisan nekim programskim jezikom. Na izlazu kompajlera se dobiju objektni kod (program u binarnom obliku) i listing (izvještaj o kompajliranju sa naznačenim greškama). Glavne faze kompajliranja se dijele na analitičke i sintetičke. Analitičke faze su leksička, sintaksna i semantička analiza, a sintetičke faze su generisanje međukoda, optimizacija međukoda i generisanje objektnog koda. Na slici 1. je prikazana struktura kompajlera sa fazama kompajliranja.



Slika 1. Struktura kompajlera sa fazama kompajliranja

Leksička analiza transformiše izvorni kod iz niza znakova u niz simbola. Znakovi se grupišu u logičke cjeline koje predstavljaju identifikatore (varijable), konstante (122.3, "rezultat=",...), rezervisane riječi (IF, WHILE,...), operatore (+, -, MOD, <=,...) i simbole interpunkcije (;, :,...).
Sintaksna analiza prihvata simbole i konstruiše sintaksnu strukturu izvornog koda, na osnovu sintaksnih pravila. Sintaksna struktura se naziva stablo prevođenja (derivaciono stablo). Primjer takvog stabla je prikazan na slici 2., za naredbu IF. Na krajevima stabla se nalaze simboli izvornog koda, dok se u unutrašnjosti nalaze sintaksni pojmovi (uslov, faktor, operator,...).
Semantička analiza se svakoj varijabli dodjeljuju atributi: naziv, tip, veza sa drugim varijablama itd. Naprimjer, provjerava se da li su varijable deklarisane i da li su kompatibilnog tipa, da li predstavljaju funkcije, varijable ili konstante, da li su cjelobrojne, realne, string itd.
Međukod je apstraktna forma objektnog koda iz koje je moguće generisati objektni kod različitih računara. Kako mašinski jezik nije isti za sve vrste računara, potrebno je koristiti međukod. Međukod je po formi sličan asembleru, ali je od njega jednostavniji, jer ne koristi koncept generalnih registara. Ako se međukod koristi kao konačan izlaz iz kompajlera (bez objektnog koda), on se izvršava posebnim programom – interpreterom.



Slika 2. Primjer derivacionog stabla za naredbu IF

13. Programiranje algoritama

Veliki broj problema koji se rješavaju programiranjem sadrže neke tipične algoritme. Isti algoritmi se koristi u svim programskim jezicima, samo što se razlikuje sintaksa (dodjeljivanje vrijednosti, tipovi podataka, pravila za imenovanje varijabli i sl.). Ovdje će biti prikazani neki najčešće korišteni algoritmi.

13.1. Brojači i akumulatori

Često je u programu potrebno izvršiti prebrojavanje. Ono se vrši tako što se nekoj varijabli dodijeli početna vrijednost, koja se zatim, najčešće unutar petlje, povećava za jedan svaki put kad se pojavi nova vrijednost. Kad se petlja završi, ta varijabla ima konačnu vrijednost petlje.

Sintaksa:

```
broj = broj + 1
```

Primjer (Prebrojavanje brojeva od 1 do 1000 djeljivih sa 5 i sa 7):

```
REM Početna, nulta vrijednost broja N
N = 0
REM Petlja za provjeravanje djeljivosti
FOR X = 1 TO 1000
  IF X MOD 5 = 0 AND X MOD 7 = 0 THEN
    N = N + 1
  END IF
NEXT X
PRINT "Od 1 do 1000 ima "; N; "brojeva djeljivih sa 5 i sa 7."
```

U ovom primjeru se varijabla N naziva "brojač", jer se povećava svaki put kad je ispunjen traženi uslov (djeljivost sa 5 i sa 7).

Primjer upotrebe brojača je korištenje DO petlje umjesto FOR petlje:

Primjer (FOR petlja):

```
FOR K = 1 TO 100
  PRINT K
NEXT K
```

Isti zadatak se upotrebom DO petlje i brojača može riješiti ovako:

Primjer (DO petlja s brojačem):

```
K = 0
DO WHILE K <= 100
  K = K + 1
  PRINT K
LOOP
```

Akumulatori se koriste na sličan način, samo što se varijabla ne povećava za jedan, nego za neku drugu vrijednost. Na taj način se može izračunati zbir više vrijednosti.

Sintaksa:

```
suma = suma + broj
```

Primjer (Izračunavanje zbira negativnih članova niza):

```
DIM NIZ(1 TO 200)
REM Početna, nulta vrijednost sume S
S = 0
REM Petlja za provjeravanje predznaka članova niza
FOR I = 1 TO 200
  IF NIZ(I) < 0 THEN
    S = S + NIZ(I)
  END IF
NEXT I
PRINT "Zbir negativnih članova niza S = "; S
```

Treba napomenuti da je u ovom primjeru izostavljeno unošenje članova niza (podrazumijeva se da niz već ima neke vrijednosti).

Za izračunavanje proizvoda se koristi sličan algoritam:

Sintaksa:

```
proizvod = proizvod * broj
```

Početna vrijednost proizvoda mora biti jedan (ako bi bila nula kao kod sume, rezultat bi uvijek bio jednak nuli). Unutar petlje se zatim taj proizvod svaki puta množi nekim faktorom. Na kraju petlje, rezučltat predstavlja proizvod svih tih brojeva.

Primjer (Izračunavanje proizvoda članova niza):

```
DIM NIZ(1 TO 10)
DIM P AS DOUBLE
REM Početna vrijednost proizvoda P
P = 1
REM Petlja za množenje vrijednosti članova niza
FOR I = 1 TO 10
  P = P * NIZ(I)
NEXT I
PRINT "Proizvod vrijednosti svih članova niza P = "; P
```

13.2. Maksimum, minimum

Često se iz nekog skupa brojeva treba pronaći onaj sa najvećom ili najmanjom vrijednošću. Rješenje tog problema je u tome da se pretpostavi da je jedan (obično prvi) broj najveći (najmanji), a zatim se svi ostali uspoređuju s njim. Ako se radi o većoj količini brojeva koje treba uporediti, pogodno je koristiti nizove ili matrice, jer to omogućuje upotrebu petlji.

Primjer (Traženje maksimalnog člana u trećem redu matrice):

```
DIM MAT(1 TO 10,1 TO 10)
MAKS = MAT(3,1)
JM = 1
FOR J = 1 TO 10
  IF MAT(3,J) > MAKS THEN
    MAKS = MAT(3,J)
    JM = J
  END IF
NEXT J
PRINT "Najveći član trećeg reda matrice je: "; MAKS
PRINT "Nalazi se u "; JM; ". stupcu matrice"
```

U ovom primjeru određen je i indeks navećeg člana treće reda matrice (JM). Ako zadatak ne traži određivanje tog indeksa, naredbe "JM = 1" i "JM = J" se mogu izostaviti.

Primjer (Traženje minimalnog neparnog člana matrice):

```
DIM MAT(1 TO 10,1 TO 10) AS INTEGER
MIN = MAT(1,1)
IMIN = 1
JMIN = 1
FOR I = 1 TO 10
  FOR J = 1 TO 10
    IF MAT(I,J) < MIN AND MAT(I,J) MOD 2 <> 0 THEN
      MIN = MAT(I,J)
      IMIN = I
      JMIN = J
    END IF
NEXT J,I
PRINT "Najmanji neparni član matrice je: "; MIN
PRINT "Nalazi se u "; IMIN"; ". redu i "; JMIN; ". stupcu matrice"
```

Ovdje je naredba IF proširena provjeravanjem parnosti operatorom MOD. Ako je ostatak pri dijeljenju sa 2 različit od nule, onda je taj broj neparan. I ovdje se memorišu indeksi reda i kolone najmanjeg neparnog člana, kao varijable IMIN i JMIN. Na početku programa, prije petlje, pretpostavi se da su njihove vrijednosti jednake 1.

Ako iz bilo kojeg razloga nije moguće inicijalizirati varijablu za maksimum ili minimum, onda se toj varijabli dodijeli neka "dovoljno velika" ili "dovoljno mala" vrijednost, kao u sljedećem primjeru.

Primjer (Traženje maksimalnog člana niza):

```
XMAX = -1000000
XMIN = 1000000
FOR I = 1 TO 10
  IF X(I) > XMAX THEN XMAX = X(I)
  IF X(I) < XMIN THEN XMIN = X(I)
NEXT I
PRINT "Najveći član niza je: "; XMAX
PRINT "Najmanji član niza je: "; XMIN
```

13.3. Prosti brojevi

Broj je prost ako je djeljiv bez ostatka samo sa jedinicom i sa samim sobom. Sljedeći primjer pokazuje kako se provjerava da li je broj prost. Broj koji se provjerava se deklariše kao cijeli broj (integer). Varijabla PR se koristi kao kontrolna varijabla koja ima vrijednost 1 ako broj jeste prost, odnosno vrijednost 0 ako je broj djeljiv sa bilo kojim drugim brojem. Potrebno je koristiti petlju u kojoj se broj X mijenja od najmanje moguće vrijednosti s kojom N može biti djeljiv (2) do polovine broja N, jer N ne može biti djeljivo sa brojem koji je veći od njegove polovine. Naredba EXIT FOR se koristi da se provjeravanje prekine čim se pronađe prvi X s kojim je broj N djeljiv bez ostatka.

Primjer (Provjeravanje da li je broj N prost):

```
DIM N AS INTEGER
PR = 1
PRINT "Unesite neki broj: ", N
FOR X = 2 TO N / 2
  IF N MOD X = 0 THEN
    PR = 0
    EXIT FOR
  END IF
NEXT X
IF PR = 1 THEN
  PRINT N; "je prost"
ELSE
  PRINT N; "nije prost"
END IF
```

14. Numeričke metode u rješavanju problema

U fazi definisanja algoritma se često koriste numeričke metode, čije efikasno korištenje podrazumijeva upotrebu računara. Jedan od osnovnih pojmova numeričke matematike, koji se sreće u većini numeričkih metoda je **iteracija**. U opštem slučaju to predstavlja ponavljanje određenog niza radnji ili postupaka s ciljem poboljšanja prethodnog rezultata.
U matematici postoji veliki broj problema koje nije moguće tačno riješiti. Za praktičnu upotrebu dovoljno je izračunati neko približno rješenje čija je greška dovoljno mala da se može zanemariti. Naprimjer, jednačina "x^2-4=0" ima tačno rješenje "x=2". Jednačina "x^2-2=0" takođe ima tačno rješenje $x = \sqrt{2}$, ali je vrijednost korijena broj sa beskonačno mnogo decimala, tako da i to, uslovno rečeno, tačno rješenje u sebi sadrži grešku zaokruživanja na određeni broj decimala. Jedna numerička metoda za izračunavanje vrijednosti korijena je sljedeća:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{2}{x_n} \right)$$

Na početku se pretpostavi vrijednost za x, naprimjer x_0 = 1.5. Primjenom navedene rekurzivne formule se dobiju vrijednosti:

x_1 = 1,4167
x_2 = 1,414216
x_1 = 1,4142135 62...

Broj tačnih cifara (podvučene cifre) stalno raste, ali se proračun ipak treba jednog trenutka zaustaviti na nekom broju cifara, koje nam daju rezultat dovoljno tačan za praktičnu upotrebu.
Neke jednačine koje se ne mogu tačno riješiti, nego je potrebno koristiti približne metode su:

$$\sin x = x + 2$$

$$\frac{dy}{dx} = x^2 + y^2$$

Postoji niz problema koji se mogu riješiti numeričkim metodama. Primjer takvih problema, a koji je već obrađen u predmetu "Informatika 2" su određivanje nula funkcije metodom polovljenja intervala ili metodom regula falsi.

14.1. Metoda proste iteracije

Prosta iteracija (metoda sukcesivnih aproksimacija) se može ilustrovati primjerom programa za rješavanje proizvoljne jednačine. Jednačina "f(x) = 0" se zamijeni ekvivalentnom jednačinom "x = g(x)". Naprimjer, jednačina "$f(x) = x^2 - x - 2 = 0$" se može zamijeniti jednačinama:

$a)\quad x = x^2 - 2$

$b)\quad x = \pm\sqrt{x+2}$

$c)\quad x = 1 + \frac{2}{x}$

$d)\quad x = x + \frac{x^2 - x - 2}{a}$

Ako je $x=x_0$ približna vrijednost korijena date jednačine, takozvana prva aproksimacija, tada se može izračunati sljedeći niz vrijednosti:

$x_1 = g(x_0)$
$x_2 = g(x_1)$
$x_3 = g(x_2)$
...
$x_{n+1} = g(x_n)$

Ako taj niz konvergira, on ima graničnu vrijednost y, koja zadovoljava jednačinu

$y = g(y)$

Iterativni postupak se u praksi zaustavlja (u slučaju kad konvergira) onda kada dvije uzastopne vrijednosti x zadovoljavaju nejednakost:

$$\left|x_{n+1} - x_n\right| < \varepsilon$$

gdje je ε neka unaprijed zadata mala vrijednost.

Pojam konvergencije rezultata se može pokazati na primjeru jednačine "$f(x) = x^2 - x - 2 = 0$" napisane u obliku a):

$x = x^2 - 2 = g(x)$

Ako se krene od početne vrijednosti "$x_0 = 3$", iteracijama se dobije:

$x_1 = g(x_0) = 3^2 - 2 = 7$
$x_2 = g(x_1) = 7^2 - 2 = 47$
$x_3 = g(x_2) = 47^2 - 2 = 2207$
....

Proces očigledno divergira, jer je svaka sljedeća vrijednost veća od prethodne. Ako se koristi oblik b):

$$x = \pm\sqrt{x+2} = g(x)$$

tada se za istu početnu vrijednost "$x_0 = 3$" dobije:

$$x_1 = g(x_0) = \sqrt{3+2} = 2.236$$
$$x_2 = g(x_1) = \sqrt{2.236+2} = 2,058$$
$$x_3 = g(x_2) = \sqrt{2.058+2} = 2.014$$
$$x_4 = g(x_3) = \sqrt{2.014+2} = 2.004$$

Ovaj proces konvergira ka konačnom rezultatu "y = 2".

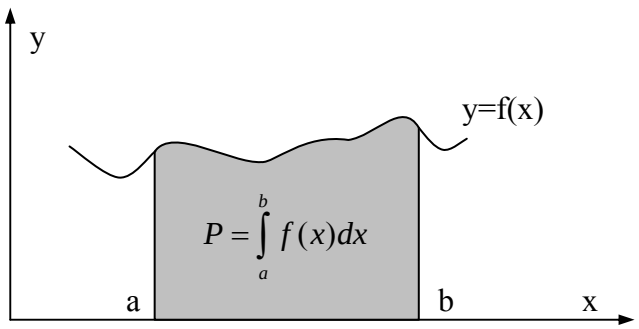
Primjer (Metoda proste iteracije):

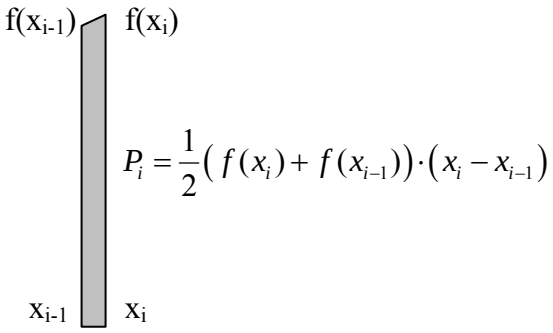
```
DECLARE FUNCTION G (X)
REM Učitavanje početne aproksimacije X0, zadane tačnosti EPS,
REM i maksimalnog broja iteracija NMAX
INPUT "Unesite početnu aproksimaciju X0", X0
INPUT "Unesite željenu tačnost EPS", EPS
INPUT "Unesite maksimalni dozvoljeni broj iteracija", NMAX
REM Varijabla N predstavlja broj iteracija
N = 0
XS = X0
X = G(XS)
DO WHILE ABS(X-XS) > EPS AND N < NMAX
  N = N + 1
  XS = X
  X = G(XS)
  PRINT N, "X ="; X
LOOP
IF ABS(X - XS) <= EPS THEN
  PRINT "Korijen X = "; X; " je određen nakon "; N; " iteracija"
ELSE
  PRINT "Postupak nije konvergirao nakon "; N; " iteracija"
END IF
END

FUNCTION G (X)
  G = SQRT(X + 2)
END FUNCTION
```

15. Numeričko izračunavanje određenog integrala

Integral predstavlja površinu ograničenu granicama intervala sa lijeve i desne strane, osom x sa donje i funkcijom f(x) sa gornje strane.





Površina integrala se izračunava približno tako što se podijeli na segmente male širine, čije se površine izračunavaju pojedinačno, a zatim se sabiraju. Ako se interval podijeli na N segmenata, širina svakog segmenta biće jednaka:

$$h = \frac{(b-a)}{N} = \left(x_i - x_{i-1}\right)$$

Površina segmenta jednaka je proizvodu širine h i polovine zbira visina (vrijednosti funkcije f(x) na krajevima segmenta). Približna vrijednost integrala (suma površina svih segmenata) biće:

$$P = \int_a^b f(x)dx = \sum_{i=1}^{N-1} P_i = \sum_{i=1}^{N-1} \frac{1}{2}\left(f(x_i) + f(x_{i-1})\right) \cdot \left(x_i - x_{i-1}\right)$$

gdje je $x_1 = a$ i $x_2 = b$.

Primjer (Približno izračunavanje određenog integrala):

```
DECLARE FUNCTION F (X)
INPUT "Unesite početak intervala A", A
INPUT "Unesite kraj intervala B", B
INPUT "Unesite broj segmenata N", N
REM Širina jednog segmenta je H
H = (B - A) / N
P = 0
FOR X = A TO B - H STEP H
  P = P + H * (F(X) + F(X+H)) / 2
NEXT X
PRINT "Vrijednost integrala je P = "; P
END

FUNCTION F (X)
  F = EXP(-X^2)
END FUNCTION
```

16. Rješavanje sistema jednačina

Sistem linearnih algebarskih jednačina se obično rješava Kramerovim pravilom, dijeljenjem determinanti sistema jednačina. Ta metoda zahtijeva jako mnogo aritmetičkih operacija za sisteme sa velikim brojem jednačina/nepoznatih, tako da nije pogodna za rješavanje pomoću računara. Za tu svrhu se koriste direktne i iterativne metode. Direktne metode su one koje bi u odsustvu greške zaokruživanja dovele do tačnog rješenja poslije konačnog broja aritmetičkih operacija. Iterativne metode počinju od neke aproksimacije i upotrebljavajući odgovarajući algoritam postepeno poboljšavaju tu aproksimaciju. One eventualno dovode do tačnog rješenja samo poslije beskonačno mnogo iteracija. To je u praksi neizvedivo pa se postupak iteracije prekida kad se postigne određena tačnost. Osnovni problem iterativnih metoda je problem konvergencije (asimptotskog približavanja tačnom rješenju). Njihova prednost je uniformnost operacija koje treba izvršiti, što ih čini pogodnim za programiranje na računaru.

16.1. Gausova metoda eliminacije – direktna metoda

Dat je sistem jednačina:

$$\begin{aligned} x_1 &+ x_2 + x_3 = 4 \\ 2x_1 &+ 3x_2 + x_3 = 9 \\ x_1 &- x_2 - x_3 = -2 \end{aligned}$$

Suština Gausove metode eliminacije je da se ovaj sistem prevede u trougaoni oblik, iz kojeg se supstitucijom određuju nepoznate. Prevođenje datog sistema u trougaoni oblik vrši se množenjem cijele jednačine i sabiranjem redom sa ostalim jednačinama postigne da svi koeficijenti uz nepoznatu x_1 u preostalim jednačinama budu jednaki nuli (koeficijent uz x_1 u prvoj jednačini naziva se pivot).
Prvi korak je množenje druge jednačine sa (-2) i treće sa (-1):

$$\begin{aligned} x_1 &+ x_2 + x_3 = 4 \\ 2x_1 &+ 3x_2 + x_3 = 9 \quad \left|\cdot(-2)\right. \\ x_1 &- x_2 - x_3 = -2 \quad \left|\cdot(-1)\right. \end{aligned}$$

Trougaona forma (nakon svih množenja) izgleda:

$$\begin{aligned} x_1 &+ x_2 + x_3 = 4 \\ &x_2 - x_3 = 1 \\ &- 4x_3 = -4 \end{aligned}$$

Nakon toga se vrši "zamjena unazad", tj. polazeći od posljednje jednačine izračunavaju se nepoznate. Da bi se izbjeglo množenje velikim brojem i eventualno oduzimanje dva broja različitog reda veličine, što unosi veliku grešku u rezultat, koristi se Gausova metoda s parcijalnim pivotiranjem, kod koje pivot u svakoj fazi mora po apsolutnoj vrijednosti biti veći od svih koeficijenata koje treba eliminisati. Tako svi faktori kojima se množe jednačine postaju po apsolutnoj vrijednosti manji od jedan, čime se eliminiše pomenuti problem.

Primjer:

```
DIM SHARED A(1 TO 20), B(1 TO 20), X(1 TO 20), IND(1 TO 20) AS INTEGER
DIM SHARED KP1 AS INTEGER, N AS INTEGER, K AS INTEGER
DECLARE SUB eliminacija ()
REM Učitavanje broja jednačina N, koeficijenata uz nepoznate A(I,J)
REM i slobodnih članova B(I)
INPUT "Unesite broj jednačina: "; N
FOR I = 1 TO N
  FOR J = 1 TO N
    INPUT "Unesite koeficijent uz nepoznatu ", A(I,J)
NEXT J,I
FOR I = 1 TO N
  INPUT "Unesite slobodni koeficijent ", B(I)
NEXT I
REM Parcijalno pivotiranje
REM Inicijaliziranje tabele indeksa jednačina
FOR I = 1 TO N
  IND(I) = I
NEXT I
REM Nalaženje najvećeg elementa u koloni
NM1 = N - 1
FOR K = 1 TO NM1
  AMAXIM = ABS(A(K,K))
  L = K
  KP1 = K + 1
  FOR I = KP1 TO N
    IF ABS(A(I,K)) <= AMAXIM THEN EXIT FOR
    AMAXIM = ABS(A(I,K))
    L = I
  NEXT I
  IF AMAXIM = 0 THEN
    PRINT "Dati sistem jednačina nema jedinstveno rješenje"
    END
  END IF
  IF L = K THEN
    CALL eliminacija
  ELSE
    REM Zamjena vrsta
    FOR J = 1 TO N
      SWAP A(K,J), A(L,J)
    NEXT J
    SWAP B(K), B(L)
    SWAP IND(K), IND(L)
```

```
      CALL eliminacija
    END IF
  NEXT K
  REM Zamjena unazad: nalaženje rješenja x(i)
  X(N) = B(N) / A(N,N)
  FOR II = 1,NM1
    I = N - II
    X(I) = B(I)
    IP1 = I + 1
    FOR K = IP1 TO N
      X(I) = X(I) - A(I,K) * X(K)
    NEXT K
    X(I) = X(I) / A(I,I)
  NEXT II
  REM Ispisivanje rezuktata
  PRINT:PRINT "Rješenja datog sistema su:"
  FOR K = 1 TO N
    I = IND(K)
    PRINT "X(" + LTRIM$(STR$(I)) + ") ="; X(I)
  NEXT K
  END

  SUB eliminacija
    FOR I = KP1 TO N
      FOR J = KP1 TO N
        A(I,J) = AM(I,J) - A(I,K) * A(K,J) / A(K,K)
      NEXT J
      B(I) = B(I) - A(I,K) * B(K) / A(K,K)
    NEXT I
  END SUB
```

16.2. Gaus-Seidelova metoda – iterativna metoda

Ova metoda se zasniva na Jacobijevoj metodi, koja se zasniva na transformaciji sistema jednačina tako da se svaka jednačina riješi po jednoj nepoznatoj:

$$\begin{aligned} 10x_1 + x_2 + x_3 &= 12 \\ x_1 + 10x_2 + x_3 &= 12 \\ x_1 + x_2 + 10x_3 &= 12 \end{aligned} \quad \Rightarrow \quad \begin{aligned} x_1 &= \frac{1}{10}\left(12 - x_2 - x_3\right) \\ x_2 &= \frac{1}{10}\left(12 - x_1 - x_3\right) \\ x_3 &= \frac{1}{10}\left(12 - x_1 - x_2\right) \end{aligned}$$

Nakon toga se, polazeći od proizvoljne početne aproksimacije $x_1 = x_1^{(0)}, x_2 = x_2^{(0)}, x_3 = x_3^{(0)}$ računa niz uzastopnih aproksimacija prema sljedećim iterativnim formulama:

$$x_1^{(n+1)} = \frac{1}{10}\left(12 - x_2^{(n)} - x_3^{(n)}\right)$$
$$x_2^{(n+1)} = \frac{1}{10}\left(12 - x_1^{(n)} - x_3^{(n)}\right) \qquad \left(n = 0,1,2,\ldots\right)$$
$$x_3^{(n+1)} = \frac{1}{10}\left(12 - x_1^{(n)} - x_2^{(n)}\right)$$

sve dok se ne postigne odgovarajuća tačnost, tj. dok ne bude:

$$\max_{1 \le i \le 3} \left| x_i^{(n+1)} - x_i^{(n)} \right| < \varepsilon$$

gdje je ε unarijed zadana mala vrijednost (veličina dozvoljene greške).
Gaus-Seidelova metoda podrazumijeva da se nove (tačnije) vrijednosti koriste čim se izračunaju, tako da se koriste sljedeće iterativne formule:

$$x_1^{(n+1)} = \frac{1}{10}\left(12 - x_2^{(n)} - x_3^{(n)}\right)$$
$$x_2^{(n+1)} = \frac{1}{10}\left(12 - x_1^{(n+1)} - x_3^{(n)}\right) \qquad (n = 0,1,2,...)$$
$$x_3^{(n+1)} = \frac{1}{10}\left(12 - x_1^{(n+1)} - x_2^{(n+1)}\right)$$

Koristeći tu modifikaciju, u većini slučajeva se ubrzava konvergencija rješenja.
Treba napomenuti da ova metoda rješavanja sistema jednačina ima ograničenu primjenu i da ne konvergira uvijek, jer se može koristiti samo na sistemima sa dijagonalno dominantnim matricama koeficijenata. To su sistemi čiji su dijagonalni koeficijenti (koeficijent uz x_1 u prvoj, uz x_2 u drugoj jednačini, itd.) po apsolutnoj vrijednosti veći od sume apsolutnih vrijednosti svih ostalih elemenata u toj vrsti.

Primjer:

```
DIM SHARED A(1 TO 50,1 TO 50), B(1 TO 50), XS(1 TO 50), XN(1 TO 50)
DECLARE SUB iteracija ()
REM Učitavanje broja jednačina N, koeficijenata uz nepoznate A(I,J),
REM slobodnih članova B(I), dozvoljene greške EPS i broja
REM iteracija MAXIT
INPUT "Unesite broj jednačina: "; N
INPUT "Unesite veličinu dozvoljene greške: "; EPS
INPUT "Unesite najveći dozvoljeni broj iteracija: "; MAXIT
FOR I = 1 TO N
  FOR J = 1 TO N
    INPUT "Unesite koeficijent uz nepoznatu ", A(I,J)
NEXT J,I
FOR I = 1 TO N
  INPUT "Unesite slobodni koeficijent ", B(I)
NEXT I
REM Provjera dijagonalne dominantnosti sistema
FOR I = 1 TO N
  S = 0
  FOR J = 1 TO N
    IF I <> J THEN S = S + ABS(A(I,J))
  NEXT J
  IF ABS(A(I,J)) > S THEN
    EXIT FOR
  ELSE
    PRINT "Dati sistem nije dijagonalno dominantan"
  END IF
NEXT I
REM Početna aproksimacija
FOR I = 1 TO N
  XS(I) = 0
NEXT I
REM Iterativno određivanje korijena
NIT = 0
DO
  NIT = NIT + 1
  FOR I = TO N
    IM1 = MI - 1
    S1 = 0
    FOR J = 1 TO IM1
      S1 = S1 + A(I,J) * XN(J)
    NEXT J
    IP1 = I + 1
```

```
    S2 = 0
    FOR J = IP1 TO N
      S2 = S2 + A(I,J) * XS(J)
    NEXT J
    XN(I) = (B(I) - S1 - S2) / A(I,J)
  NEXT I
  REM Greška i kontrola konvergencije
  GR = 0
  FOR I = 1 TO N
    G = ABS(XN(I) - XS(I))
    IF G > GR THEN GR = G
  NEXT I
  FOR I = 1 TO N
    XS(I) = XN(I)
  NEXT I
  IF NIT > MAXIT THEN
    PRINT "Zadana tačnost EPS ="; EPS; "nije postignuta nakon";
    PRINT MAXIT; "iteracija"
    EXIT DO
    END
  END IF
LOOP WHILE GR >= EPS
  PRINT "Rješenja datog sistema jednačina su:"
  FOR I = 1 TO N
    PRINT "X(" + LTRIM$(STR$(I)) + ") ="; XN(I)
  NEXT I
END
```

17. Programski jezik C++

Programski jezik C je 1972. kreirao Dennis Ritchie iz firme Bell Labs, s namjerom da napravi programski jezik za kreiranje operativnog sistema UNIX. Od 1983. do 1988. godine razvijen je standardni C jezik, "ANSI C". C++ je kreirao Bjarne Sroustrup od 1983 do 1985, kao proširenje jezika C klasama i objektno-orijentisanim komponentama. C++ se ponaša kao prošireni C, tako da se programi napisani u C jeziku mogu prevoditi prevodiocima za C++.

17.1. Instalacija, pokretanje, način rada, MAIN identifikator, konvencije pisanja naredbi

Danas postoji veliki broj prevodilaca za C i C++, pa čak i integrisanih razvojnih okruženja, koja se sastoje od editora, debuger-a, help-a i samog prevodioca. Jedno od tih okruženja, koje je besplatno (pod GPL licencom) je Dev-C++, firme Bloodshed Software (www.bloodshed.net). Na navedenoj adresi se može naći instalacijska datoteka, čijim se pokretanjem instalira razvojno okruženje za C++. Nakon instalacije, potrebno je samo izabrati jezik menija i može se odmah početi sa radom.
U Dev-C++ mogu se kreirati složeniji projekti (File/New/Project) ili jednostavne datoteke sa izvornim kodom (File/New/Source File). Projekti mogu biti Windows aplikacije, konzolne (DOS) aplikacije, statičke i dinamičke (DLL) biblioteke. U ovom kursu, radi jednostavnosti izvornog koda, ograničićemo se na konzolne aplikacije.
Najjednostavniji program, koji štampa tekstualnu poruku na ekran je sljedeći:

Primjer:

```
#include <iostream>
using namespace std;
main()
{
cout << "Hello world\n";
}
```

Za ovakav program ne treba kreirati projekat, nego se može zapisati u jednoj datoteci izvornog koda (File/New/Source File). Nakon unošenja koda u program, isti se prevodi i pokreće naredbom "Compile & Run" iz menija "Execute".
U ovom programu se može vidjeti struktura C++ programa. Program se sastoji iz preprocesorske direktive (#Include <iostream>), naredbe "using namespace std", zaglavlja funkcije (main()) i tijela funkcije (sve naredbe između "{" i "}").
Preprocesorska direktiva iza naredbe "include" ima naziv datoteke u kojoj se nalaze biblioteke funkcija. Za prikazani primjer, dovoljno je u program uključiti biblioteku "iostream", koja sadrži funkcije za ulaz i izlaz podataka. Ako se naziv datoteke navede unutar znakova "<" i ">", to znači da prevodilac treba da traži tu datoteku u sistemskom direktoriju prevodioca. Ako se pak naziv datoteke unese pod navodnim znacima, to znači da se ta biblioteka nalazi u radnom direktoriju (onaj direktorij u kojem se snima datoteka sa izvornim kodom). Sistemske biblioteke se uvijek navode između znakova "<" i ">".
Naredba "using namespace" aktivira područje imena funkcija "std", u kojem se nalaze sve standardne funkcije. Korištenje područja imena (eng. *namespace*) omogućuje da ne dođe do kolizije između funkcija istog imena koje dolaze iz različitih biblioteka.
Treba obratiti pažnju da svaka naredba u C++ jeziku (osim preprocesorske direktive) mora završavati znakom tačka-zarez ";". Razlog za to je što sva prazna mjesta, tabulatore i prelaske u novi red prevodilac ignoriše, te se na taj način naredbe međusobno razdvajaju. Jedini obavezni razmak je između naredbe i naziva varijable, a jedini obavezan prelazak u novi red je nakon preprocesorske direktive.
Osim direktive, C++ program se sastoji od jedne ili više funkcija. Funkcije se opet sastoje od zaglavlja i tijela funkcije. Svaki program mora imati funkciju koja se zove "main()". U ovom slučaju zaglavlje funkcije samo sadrži njen naziv, unutar zagrada nema nikakvih argumenata, jer ih ova funkcija i nema, a funkcija nije ni deklarisana (nije navedeno koji tip podataka koristi).
Između vitičastih zagrada se nalazi tijelo funkcije. U ovom slučaju sadrži samo jednu naredbu koja koristi objekat "cout". Objekat "cout" iz biblioteke "iostream" je u stvari izlazni tok podataka, a koristi se za ispisivanje teksta na konzolu. Operatorom "<<" se podatak koji slijedi (u ovom primjeru tekst pod navodnicima) upućuje na izlazni tok – konzolu.

Primjer (funkcija napisana u jednom redu):

```
#include <iostream>
using namespace std; main(){cout << "Hello world\n";}
```

Nakon prevođenja i pokretanja ovog programa, može se steći lažni utisak da se ništa ne dešava, zato što neke verzije Windows operativnog sistema po završetku DOS (konzolnog) programa automatski zatvaraju prozor u kojem se program izvršio. Da bi se ipak vidio rezultat programa, može se dodati naredba:

```
system("PAUSE");
```

Ta naredba izvršava MS DOS naredbu "PAUSE" koja zaustavlja izvršenje programa, sa porukom "Press Any Key To Continue..." i čeka da se pritisne neka tipka na tastaturi za nastavak rada.
Drugi način privremenog zaustavljanja programa je korištenje objekta ulaznog toka (cin) koji zaustavlja program očekujući unos nekog karaktera sa tastature u varijablu "x":

```
char x;
cin >> x;
```

17.2. Deklaracija varijabli, tipovi podataka, dodjeljivanje vrijednosti

Imena varijabli mogu da sadrže samo 25 slova engleskog alfabeta (bez č,ć,š,đ,ž), brojeve i znak "_" (underscore). Moraju počinjati slovom ili znakom "_", i broj karaktera u imenu koji se uzima u obzir je 31. Imena varijabli ne smiju sadržati prazna mjesta. Pravilo je da se sva imena lokalnih varijabli pišu malim slovima. Inače, C++ pravi razliku između velikih i malih slova, tako da svaka od varijabli "Ime", "IME" i "ime" u istom programu može imati različitu vrijednost.
Pet osnovnih tipova podataka u C++ jeziku su:

int	- integer: cijeli broj između –2 147 483 648 i +2 147 483 647
float	- floating point: realni broj između 1.E-36 i 1.E+36, sa preciznošću od 7 cifara
double	- double-precision floating point: realni broj dvostruke preciznosti između 1.E-303 i 1.E+303, sa 13 cifara preciznosti
char	- character: jedan karakter teksta
void	- tip podataka za posebne potrebe, bez dodijeljene vrijednosti.

Granične vrijednosti za pojedine tipove treba uzeti uslovno, jer se oni razlikuju od jednog do drugog kompajlera. Deklaracija varijabli se može izvršiti na više načina, kao što je navedeno u primjeru:

Primjer (deklaracija varijabli):

```
#include <iostream>
using namespace std;
main()
{
int a,b,c;
float x = 4.32;
}
```

Varijable a, b i c su deklarisane kao cjelobrojne, jednom naredbom "int". Varijabla x je deklarisana kao realan broj, i prilikom deklaracije joj je istovremeno i dodijeljena vrijednost. Sve varijable koje se koriste u programu **moraju biti deklarisane**!
Dodjeljivanje vrijednosti se vrši korištenjem znaka jednakosti, tako što se s lijeve strane unosi naziv varijable, a sa desne njena vrijednost. Za razliku od drugih programskih jezika, moguće je istovremeno dodijeliti istu vrijednost više varijabli, kao u sljedećem primjeru:

Primjer (dodjeljivanje vrijednosti varijablama):

```
#include <iostream>
using namespace std;
main()
{
int a,b,c;
char ime;
a = 4;
b = c = 12;
ime = 'C';
}
```

Ovim programom će varijablama b i c biti dodijeljena vrijednost 12. Varijabli "ime" se dodjeljuje vrijednost 'C'. Treba obratiti pažnju da varijabla tipa "char" ne može sadržati više od jednog karaktera, i prilikom dodjeljivanja se taj karakter navodi između dva apostrofa.
Osim pet navedenih tipova varijabli, koriste se još i:

short	- short integer: zauzima 2 bajta; cijeli broj od –32 768 do +32 767
unsigned short	- zauzima 2 bajta; cijeli broj od 0 do 65 535
unsigned int	- zauzima 4 bajta; cijeli broj od 0 do 4 294 967 295
long	- long integer: zauzima 4 bajta; isto kao i tip int

unsigned long - isto kao i tip unsigned int
long double - zauzima 10 bajta; ±3.4E-4932 do ±3.4E+4932
bool - logički podaci: "true"-tačno ili "false"-netačno

Prilikom dodjeljivanja vrijednosti, treba znati da tip podatka koji se dodjeljuje ne mora odgovarati tipu kojim je deklarisana varijabla, nego se vrši konverzija. Naprimjer, ako je varijabla deklarisana kao cjelobrojna, a dodijeli joj se realna vrijednost, C++ će ignorisati sve što se nalazi iza decimalne tačke.

Primjer (dodjeljivanje vrijednosti varijablama neodgovarajućeg tipa):

```
#include <iostream>
using namespace std;
main()
{
short a,b;
a = 50.67;
b = 32770;
cout << a << "\n";
cout << b << "\n";
}
```

Kako su varijable "a" i "b" deklarisane kao cjelobrojne, varijabla "a" će prihvatiti samo broj "50", bez decimala, a varijabla "b" će dobiti vrijednost -32 766, jer dolazi do prekoračenja (*overflow*). Najveća vrijednost koju tip "short" može primiti je 32 767, i ako se takvoj varijabli dodijeli broj veći od toga, on će sve što prelazi tu vrijednost dodati na najmanju moguću vrijednost (-32 768) i tako prikazati tu varijablu.

18. Naredbe ulaza i izlaza

U C++ programskom jeziku, za izlaz podataka najčešće se koristi objekat izlaznog toka podataka "cout". Iza tog objekta se koriste usmjerivači toka "<<", u kombinaciji sa imenima varijabli, konstantama i stringovima.

Sintaksa:

```
cout << [podaci];
```

Primjer (ispisivanje teksta i vrijednosti varijable pomoću objekta cout):

```
#include <iostream>
using namespace std;
main()
{
short a = 50;
cout << "Vrijednost varijable a je " << a << "\n";
}
```

Usmjerivač toka "<<" se stavlja između pojedinih dijelova izlaza. String "\n" se naziva *escape*-sekvenca. Znak "\" unutar stringa označava *escape*-sekvencu, koja se ne ispisuje kao obični tekst, nego ima specijalnu upotrebu. Najčešće korištene *escape*-sekvence su:

"\n" - newline: prelazak kursora u novi red
"\t" - tab: pomjeranje kursora do sljedećeg tabulatora
"\\" - backslash: ispisuje znak "\"
"\'" - single quote: ispisuje apostrof '
"\"" - double quote: ispsuje navodnik "

Za unošenje vrijednosti u program se najčešće koristi objekat ulaznog toka podataka "cin". Kao i objekat "cout", i on je definisan u standardnoj biblioteci <iostream>, koju je potrebno uključiti u program preprocesorskom direktivom "include".

Sintaksa:

```
cin >> [varijabla];
```

Kad program naiđe na objekat "cin", zaustavlja se izvršenje programa i čeka se na unošenje nekog podatka sa tastature. Unos se završava tipkom ENTER.

Primjer1:

```
#include <iostream>
using namespace std;
main()
{
int x;
cin >> x;
cout << "Vrijednost varijable x je " << x << "\n";
}
```

Usmjerivač toka koji se koristi uz objekat "cin" je okrenut u smjeru u kojem teku podaci. Kod ispisivanja objektom "cout", tok podataka je od podatka (varijable, stringa) prema objektu "cout", a kod objekta "cin" podaci teku od objekta "cin" prema varijabli.
Program iz primjera 1 ima nedostatak, jer korisnik ne zna šta se od njega očekuje kada se zaustavi na objektu "cin". Bolje je isti program napisati na sljedeći način:

Primjer2:

```
#include <iostream>
using namespace std;
main()
{
int x;
cout << "Unesite vrijednost varijable x:";
cin >> x;
cout << "Vrijednost varijable x je " << x << "\n";
}
```

Ovako će korisnik prije unosa podatka dobiti poruku na ekranu kojom se objašnjava kakav podatak treba unijeti.
Slično kao što objekat "cout" može prikazati više vrijednosti koristeći više usmjerivača toka, i objektom "cin" se može unijeti više podataka, koji se međusobno odvajaju praznim mjestom.

Primjer3:

```
#include <iostream>
using namespace std;
main()
{
int a,b,c;
cout << "Unesite vrijednosti varijabli a, b i c";
cout << "Međusobno ih razdvojite praznim mjestima\n";
cin >> a >> b >> c;
cout << "a = " << a << "\n";
cout << "b = " << b << "\n";
cout << "c = " << c << endl;
}
```

Prilikom unošenja podataka u ovom primjeru, osim praznog mjesta podaci se mogu razdvojiti i pritiskom na tipku ENTER, važno je da se unesu tri broja koja će se pohraniti u varijable a, b i c.
Kod varijable "c" je kao primjer upotrijebljen objekat "endl" umjesto *escape*-sekvence "\n". Sasvim je svejedno koji će se objekat koristiti, jer imaju isti rezultat – prelazak u novi red.

18.1. Aritmetički operatori i funkcije

Aritmetički operatori su prikazani u sljedećoj tabeli:

Operator	Namjena	Primjer	Rezultat
+	Sabiranje	5+2	7
-	Oduzimanje	5-2	3
*	Množenje	5*2	10
/	Dijeljenje	5/2	2
%	Ostatak pri dijeljenju	5%2	1

Svi ovi operatori se nazivaju binarni, jer podrazumijevaju po dva operanda (dva sabirka, dijeljenik i djelitelj i sl.). Osim binarnih postoje i unarni operatori (naprimjer prefiks za negativne brojeve) i ternarni operatori.
Aritmetički operatori se mogu kombinovati sa operatorom dodjeljivanja vrijednosti "=", na sljedeći način:

Izraz	Kombinovani izraz
a = a + 2;	a += 2;
a = a - 2;	a -= 2;
a = a * 2;	a *= 2;
a = a / 2;	a /= 2;
a = a % 2;	a %= 2;

Kombinovani izrazi u desnoj koloni tabele daju isti rezultat kao i izrazi u lijevoj koloni tabele.
Redoslijed izvođenja operacija je kao i u matematici: prvo se vrši unarna negacija, zatim množenje, dijeljenje i ostatak, i na kraju sabiranje i oduzimanje. Redoslijed se može mijenjati korištenjem zagrada. Ako se izvodi više operacija istog značaja, računanje se vrši slijeva nadesno (npr. 8/2*4 će dati rezultat 16, jer dijeljenje prethodi množenju).

Posebno kod dijeljenja, treba obratiti pažnju na tip podataka, zato što rezultat zavisi od tipa operatora. Ako su oba operatora cijeli brojevi, i rezultat će biti cijeli broj. To može dovesti do pogrešnog rezultata, jer cjelobrojno dijeljenje ignoriše sve što se nalazi iza decimalne tačke.

Primjer (cjelobrojno dijeljenje):

```
#include <iostream>
using namespace std;
main()
{
int a;
float x;
a = 5/2;
x = 5/2;
cout << "a = " << a << endl;
cout << "x = " << x << endl;
}
```

U oba slučaja rezultat će biti "2", jer su operatori u dijeljenju cijeli brojevi. Ako je jedan operator realan broj, prilikom dijeljenja se vrši konverzija drugog operatora, a tek onda dijeljenje.

Primjer (dijeljenje):

```
#include <iostream>
using namespace std;
main()
{
int a;
float x;
a = 5/2.;
x = 5/2.;
cout << "a = " << a << endl;
cout << "x = " << x << endl;
}
```

Varijabla "a" će dobiti vrijednost "2", jer je deklarisana kao cjelobrojna, pa se konverzija realnog rezultata "2.5" vrši prilikom dodjeljivanja vrijednosti. Varijabla "x" će dobiti vrijednost "2.5", jer je deklarisana kao realni broj. Ako se umjesto brojeva kao operatori koriste varijable, moguće je izvršiti konverziju na jedan od sljedećih načina:

```
float rezultat = (float) x / y;
float rezultat = float (x) / y;
float rezultat = x / (float) y;
float rezultat = x / float (y);
```

Varijable "x" i "y" su prethodno delarisane kao cjelobrojne, ali se prilikom računanja varijable "rezultat" istovremeno vrši i konverzija tipa u "float", bilo stavljanjem varijable u zagradu i dodavanjem prefiksa "float", bilo navođenjem prefiksa "loat" u zagradu ispred naziva varijable.

C++ nema operator za stepenovanje, nego se koristi funkcija "pow", koja je definisana u biblioteci <cmath>.

Primjer (stepenovanje):

```
#include <iostream>
#include <cmath>
using namespace std;
main()
{
double a,b,c;
a = 2;
b = 16;
cout << "c = " << pow(a, b) << endl;
}
```

Ako je potrebno neku vrijednost povećati za 1, to se može uraditi na sljedeći način:

```
int x = 5;
x = x + 1;
```

Nakon ove dvije naredbe, x će imati vrijednost 6. Isti rezultat će se dobiti i sa:

```
int x = 5;
x += 1;
```

C++ ima unarni operator za inkrementiranje "++", koji povećava vrijednost varijable iza koje stoji za 1:

```
int x = 5;
```

```
x++;
```

Osim inkrementa, postoji i unarni operator dekrementa, koji varijablu umanjuje za 1:

```
int x = 5;
x--;
```

Operatori za inkrement i dekrement se mogu pisati kao prefiks ili postfiks operatori. Prefiks operator (++a) prvo izvrši inkrement (povećanje za 1), a zatim koristi vrijednost varijable, a postfiks operator (a++) prvo koristi vrijednost varijable, a zatim izvrši inkrementiranje (povećanje za 1). Ista analogija važi i za dekrement (--a i a--).

Primjer (inkrement i dekrement):

```
#include <iostream>
using namespace std;
main()
{
int a = 2,b = 2;
cout << a << b << endl;
cout << ++a << b++ << endl;
cout << a << b << endl;
}
```

Rezultat koji se dobije u prvom slučaju je 22 ("a" i "b" prije inkrementa), u drugom 32 ("a" je inkrementiran pa zatim prikazan, "b" je prikazan pa onda inkrementiran), a u trećem 33 (i "a" i "b" su već inkrementirani). Osim varijabli, u C++ se mogu koristiti i konstante. To su posebne vrste varijabli kojima se vrijednost može dodijeliti samo jedanput (takozvane "read-only" varijable). Ispred njihove deklaracije piše se kvalifikator "const".

Primjer (konstanta):

```
#include <iostream>
using namespace std;
main()
{
float r, p;
const float pi=3.14159265359;
cout << "Unesite radijus kružnice" << endl;
cin >> r;
p = r*r*pi;
cout << "Površina je: " << p << endl;
}
```

Matematičke funkcije koje se najčešće koriste su definisane u biblioteci <cmath>.

Primjer (matematička funkcija):

```
#include <iostream>
#include <cmath>
using namespace std;
main()
{
float e = exp(1.);
cout << "e = " << e << endl;
}
```

Funkcija	Primjer	Značenje	Rezultat
abs(x)	abs(-9)	apsolutna vrijednost	9
pow(x,y)	pow(2,3)	stepenovanje x na y	8
sqrt(x)	sqrt(9)	kvadratni korijen	3
exp(x)	exp(1.)	e^x	2.71828
log(x)	log(2.71828)	ln(x)	1
cos(x)	cos(3.14159)	kosinus ugla u radijanima	-1
sin(x)	sin(3.14159)	sinus ugla u radijanima	0
acos(x)	acos(-1.)	arkus kosinus ugla u radijanima	3.14159
asin(x)	asin(0.)	arkus sinus ugla u radijanima	0
atan(x)	atan(1.)	arkus tangens ugla u radijanima	3.14159/4

19. Logički i komparativni operatori, IF, logički izrazi

Za logičke podatke postoje tri operatora: !x (logička negacija), x && y (logičko i) i x || y (logičko ili). Logička negacija je unarni operator koji mijenja logičku vrijednost varijable. Tablica logičkih operacija glasi:

x	y	!x	!y	x && y	x || y
true	true	false	false	true	true
true	false	false	false	false	true
false	true	true	true	false	true
false	false	true	true	false	false

U logičkim izrazima (izrazi koji koriste logičke operatore), ne moraju se koristiti vrijednosti istog tipa podataka. Prilikom ispisa logičkih tipova, "true" se pretvara u cjelobrojno 1, a "false" u cjelobrojno 0. Ako se logičkoj varijabli dodijeli aritmetički podatak, 0 se pretvara u "false", a svi ostali brojevi različiti od 0 u "true".
Komparativni (relacioni) operatori su:

- `<` - manje od
- `<=` - manje ili jednako
- `>` - veće od
- `>=` - veće ili jednako
- `==` - jednako
- `!=` - različito

Za uslovno grananje programa koristi se naredba "if".

Sintaksa:

```
if (uslov) naredba;
```

U zagradi iza "if" se navodi logički izraz. Naredba navedena iza zagrade se izvršava ako je vrijednost tog izraza "true". Treba paziti da se umjesto komparacionog operatora "==" ne koristi operator "=" za dodjeljivanje vrijednosti, jer to ne predstavlja grešku sintakse i moći će se prevesti, ali neće dati traženi rezultat (izraz "x=0" će uvijek dati vrijednost "false", a izraz "x==0" će biti "false" samo ako je x različito od nule).

Primjer (prosta naredba if):

```
#include <iostream>
using namespace std;
main()
{
int x;
cout << "Unesite neki broj";
cin >> x;
if (x % 2 == 0) cout << "Broj je paran" << endl;
}
```

Ako je potrebno da se izvrši više naredbi u slučaju zadovoljenog uslova, koriste se vitičaste zagrade da obuhvate sve te naredbe.

Primjer (if sa više naredbi):

```
#include <iostream>
using namespace std;
main()
{
int x;
cout << "Unesite neki broj";
cin >> x;
if (x % 2 == 0)
{
 cout << "Broj je paran" << endl;
 cout << "Broj nije neparan" << endl;
 cout << "Uneseni broj je x = " << x << endl;
}
}
```

Treba obratiti pažnju da se iza uslova ne piše tačka-zarez (;), jer to nije završetak naredbe "if".
Naredba "else" se koristi ako je potrebno izvršiti neke naredbe i kad uslov nije zadovoljen.

Sintaksa:

```
if (uslov) naredba1;
else naredba2;
```

Naredba "else" se ne može koristiti ako prije nje nema naredbe "if". Analogno naredbi "if", više naredbi se može navesti između vitičastih zagrada.

Primjer (else):

```
#include <iostream>
using namespace std;
main()
{
int x;
cout << "Unesite neki broj";
cin >> x;
if (x % 2 == 0)
 cout << "Broj je paran" << endl;
else
 cout << "Broj je neparan" << endl;
}
```

Primjer (višestruko if/else):

```
#include <iostream>
using namespace std;
main()
{
 int BB;
 cout << "Unesite broj bodova: ";
 cin >> BB;
 if (BB >= 90) cout << "Ocjena je 5" << endl;
 else if (BB >= 80) cout << "Ocjena je 4" << endl;
 else if (BB >= 70) cout << "Ocjena je 3" << endl;
 else if (BB >= 60) cout << "Ocjena je 2" << endl;
 else
  cout << "Ocjena je 1" << endl;
}
```

Ako je potrebno izvršiti višestruku provjeru, odnosno izvršiti neke naredbe za više od dva slučaja cjelobrojnih vrijednosti testirane varijable (kao kod if/else), koristi se naredba SWITCH. Ta naredba je analogna naredbi CASE iz Basic-a. Testirana varijabla mora biti cjelobrojna (ili char).

Primjer (switch):

```
#include <iostream>
using namespace std;
main()
{
 int ocj;
 cout << "Unesite ocjenu: ";
 cin >> ocj;
 switch (ocj)
  {
   case 5:
    cout << "Imate 90 - 100 bodova" << endl;
    break;
   case 4:
    cout << "Imate 80 - 89 bodova" << endl;
    break;
   case 3:
    cout << "Imate 70 - 79 bodova" << endl;
    break;
   case 2:
    cout << "Imate 60 - 69 bodova" << endl;
    break;
   default:
    cout << "Imate ispod 60 bodova" << endl;
  }
}
```

Naredba "default" ima ulogu kao "else" kod naredbe "if", da obuhvati sve slučajeve koji nisu pokriveni odgovarajućom "case" naredbom.

Naredba "break" služi za izlazak iz niza naredbi iza naredbe "switch". Kada se ne bi koristila naredba "break", za slučaj kad je unesena ocjena 3 (u navedenom primjeru), izvršile bi se naredbe "case" i za 3 i za 2 i za "default" (sve ostale), jer su navedene ispod naredbe "case 3".

Sintaksa:

```
switch (izraz)
 {
  case vrijednost1: naredba 1; break;
  case vrijednost2: naredba 2; break;
  case vrijednost3: naredba 3; break;
  ...
  default: naredba n;
 }
```

Izraz u zagradi može biti bilo koji cjelobrojni izraz.

Kako C++ razlikuje velika i mala slova, Varijable sa vrijednošću "A" i "a" neće imati istu vrijednost (C++ vidi samo njihov ASCII kod, koji nije isti).

Karakter	0	9	A	Z	a	z
ASCII kod	48	57	65	90	97	122

Sljedeći primjer pokazuje kako se naredba "switch" može koristiti i za velika i za mala slova.

Primjer (velika i mala slova):

```
#include <iostream>
using namespace std;
main()
{
 char odgovor;
 cout << "Želite li nastaviti (d/n): ";
 cin >> odgovor;
 switch (odgovor)
  {
   case 'D':
   case 'd':
    cout << "Odgovor je potvrdan" << endl;
    break;
   case 'N':
   case 'n':
    cout << "Odgovor nije potvrdan" << endl;
    break;
   default:
    cout << "Unijeli ste pogrešno slovo" << endl;
  }
}
```

U slučaju da se unese D ili d, ispisaće se potvrdan odgovor, za N ili n negativan, a za sve ostale odgovore ispisaće se poruka o grešci.

If-Else struktura se može realizovati i pomoću ternarnog operatora (?:).

Sintaksa:

```
(uslov ? vrijednost_ako_je_true : vrijednost_ako_je_false)
```

Primjer:

```
#include <iostream>
using namespace std;
main()
{
 int bodovi;
 cin >> bodovi;
 cout << ( bodovi >= 60 ? "Pozitivna ocjena" : "Negativna ocjena" );
}
```

Ako je uslov zadovoljen, tj. ako je u primjeru vrijednost varijable "bodovi" veća ili jednaka 60, ispisaće se poruka "Pozitivna ocjena", odnosno vrijednost navedena iza znaka "?", a ako nije, ispisuje se poruka "Negativna ocjena", tj. vrijednost navedena iza znaka ":".

20. FOR, DO, WHILE

Za realizaciju petlji može se koristiti naredba "for".

Sintaksa:

```
for (brojac=start; brojac<=kraj; ++brojac)
 naredba;
```

Primjer (for petlja):

```
#include <iostream>
using namespace std;
main()
{
 int x;
 for (x=1; x<=10; ++x)
  cout << x << endl;
}
```

U ovom primjeru ulogu brojača ima varijabla x (koja prethodno mora biti deklarisana). U zagradi iza naredbe "for" se navode tri naredbe: prva definiše koja će se naredba izvršiti prvi put (u ovom slučaju kolika će biti početna vrijednost brojača), druga uslov koji mora biti zadovoljen da bi se petlja ponavljala, a u trećoj se definiše naredba koja će se izvršiti u svakom koraku. Naredbe se odvajaju znakom ";", a iza treće naredbe nema znaka ";". Najčešće se brojač inkrementira za 1 pa se stoga i koristi inkrement ++. Kod definisanja granice treba obratiti pažnju da se ne formira beskonačna petlja (ako je početna vrijednost veća od krajnje, koristi se operator ">=", i korak se dekrementira.

Ako se unutar petlje treba izvršiti više od jedne naredbe, iste se navode između vitičastih zagrada.

Primjer (for petlja sa više naredbi i sa negativnim korakom):

```
#include <iostream>
using namespace std;
main()
{
for (int x=0; x>=-10; x-=2)
  {
   cout << x << endl;
   cout << endl;
  }
}
```

Tri naredbe iza naredbe "for" se ne moraju navoditi u zagradi; mogu biti definisane i na drugom mjestu, ali znakovi ";" koji ih odvajaju se moraju navesti, da se zna da li su naredbe u zagradi početna vrijednost, granica ili način inkrementiranja.

Primjer (for petlja sa drugim načinom navođenja brojača):

```
#include <iostream>
using namespace std;
main()
{
 int i=1;
 for (; i<=15;)
   cout << i++ << endl;
}
```

U prikazanom primjeru ispisaće se brojevi od 1 do 15. Ako bi se umjesto inkrementa "i++" koristilo "++i", ispisali bi se brojevi od 2 do 16, jer se inkrementiranje tada vrši prije ispisivanja na ekran.

Primjer (izračunavanje faktorijela):

```
#include <iostream>
using namespace std;
main()
{
 int f=1, n;
  cout << "Unesite broj";
  cin >> n;
   cout << "Faktorijel broja " << n << " = ";
 for (i=1; i<=n; i++)
  f*=i;
 cout << f << endl;
}
```

"For" petlje se mogu navoditi i jedna unutar druge, kao u sljedećem primjeru, koji ispisuje tekst:
AAAAA
BBBB
CCC
DD
E

Primjer (ugniježđene for petlje):

```
#include <iostream>
using namespace std;
main()
{
char s='A';
for (int i=5; i>=1; i--)
 {
  for (int j=1; j<=i; j++)
   cout << s;
 cout << endl;
 s++;
 }
}
```

Iako je varijabla "s" tipa "char", program koristi njenu ASCII vrijednost, tako da inkrement s++ povećava ASCII kod za 1 i na taj način mijenja vrijednost varijable s u B, C, D i E. Vanjska petlja, sa brojačem "i", koji se mijenja od 5 do 1, izvršava sve naredbe između vitičastih zagrada koje joj slijede (unutrašnju petlju, prelazak u novi red i inkrement varijable s). Unutrašnja petlja ispisuje "i" puta varijablu "s", u istom redu. Drugi način za realizaciju petlje je naredba "while".

Sintaksa:

```
while (uslov)
 naredba;
```

Naredba navedena iza naredbe "while" se ponavlja sve dok je uslov zadovoljen, tj. sve dok ima vrijednost "true". Više naredbi se može ponavljati tako što se navedu između vitičastih zagrada "{ i "}".

Primjer (while petlja):

```
#include <iostream>
using namespace std;
main()
{
 int x=1;
 while (x<=10)
  cout << x++ << endl;
}
```

U prikazanom primjeru se ispisuju brojevi od 1 do 10, jedan ispod drugog.

Primjer (ugniježđene while petlje):

```
#include <iostream>
using namespace std;
main()
{
char s='A';     // Varijabla za ispis teksta
int i=5;        // Varijabla za broj redova
while (i>=1)    // Petlja se prekida kad "i" dobije vrijednost 1
 {
  int j=1;      // Početna vrijednost varijable za broj slova u jednom redu
  // Unutrašnja petlja koja ispisuje slova u jednom redu
  while (j<=i) // U jednom redu ima "i" slova
   {
    cout << s; // Ispiši slovo
    j++;       // Povećaj "j" za 1
   }
 cout << endl; // Pređi u novi red
 s++;          // Promijeni vrijednost "s" na sljedeće slovo
 i--;          // Umanji broj slova u redu za 1
 }
}
```

U ovom primjeru, koji daje isti rezultat kao i prikazani primjer sa dvije "for" petlje, upotrijebljene su oznake komentara (dvije kose crte "//"). Vidi se da se komentari mogu unositi i na početku reda i iza naredbi. Drugi način pisanja komentara je pomoću oznaka "/*" i "*/", koji se može unijeti i unutar jedne naredbe:

Primjer (komentar u sredini naredbe):

```
cout << /* Komentar */ endl;
```

Još jedan način realizacije petlji je pomoću naredbe "do".

Sintaksa:

```
do {
   naredbe;
} while (uslov);
```

Naredbe navedene između vitičastih zagrada se ponavljaju sve dok je uslov zadovoljen, tj. dok ima vrijednost "true".

Primjer (do petlja):

```
#include <iostream>
using namespace std;
main()
{
 int x=1;
 do {
  cout << x++ << endl;
 } while (x<=10);
}
```

I ovaj primjer ispisuje brojeve od 1 do 10.

Uslovi za prekid, odnosno nastavak petlje mogu biti i složeni logički izrazi koji se formiraju koristeći logičke operatore && (i), || (ili) i ! (ne).

Primjer (složeni uslov):

```
#include <iostream>
using namespace std;
main()
{
 int x;
 char izbor;
 do {
  cout << "Unesite broj" << endl;
  cin >> x;
  cout << "Želite li nastaviti (d/n)?" << endl;
  cin >> izbor;
 } while (x>=0 && izbor!='n');
}
```

Petlja koja sadrži naredbe za unošenje jednog broja i jednog slova će se ponavljati sve dok se unose pozitivni brojevi (x>=0) i dok je varijabla "izbor" različita od 'n'.

21. String funkcije

Varijable koje su deklarisane kao tip "char" mogu sadržati samo jedan karakter, i one u stvari pamte ASCII kod tog karaktera. Međutim, često je potrebno da varijabla sadrži više od jednog karaktera. Jedan način da se to realizuje je da se koristi niz karaktera, takozvani "C-string". U jeziku C++ uveden je novi tip podataka, koji se naziva "C++ string class".

Primjer:

```
#include <iostream>
using namespace std;
main()
{
 char ch;
 do  {
  cout << "Pritisnite K ili k za kraj, a bilo koju tipku za nastavak \n";
  cin >> ch;
  if (ch != 'K' && ch != 'k')
   cout << "Zelite nastaviti?\n";
  else
   cout << "Kraj programa";
 } while (ch != 'K' && ch != 'k');
}
```

Ovaj primjer dobro funkcioniše jer završava program ako se unese "k" ili "K", odnosno program se nastavlja ako se unese bilo koji drugi karakter. Problem nastaje ako korisnik programa pritisne samo tipku ENTER. U tom slučaju objekat "cin" očekuje da se unese neka vrijednost, pa tek onda pritisne ENTER.

U C++ programskom jeziku neki objekti imaju vlastite funkcije, koje se ne mogu pozvati samostalno (kao npr. funkcija "pow"), nego samo tako da ih poziva objekat kojem pripadaju. Takve funkcije se nazivaju "member" funkcije, o navode se iza imena objekta kojem pripadaju, odvojeno tačkom. Funkcija "get" koja pripada objektu "cin" služi za prihvatanje bilo kojeg unosa sa tastature, pa tako i pritiska na tipku ENTER.

Sintaksa:

```
cin.get(varijabla)
```

Primjer:

```
cin.get(ch);
```

U prethodnom primjeru treba liniju "cin >> ch;" zamijeniti linijom "cin.get(ch);". Kad se prikazani primjer prevede i pokrene, treba obratiti pažnju na to da se poruka "Pritisnite K ili k..." ispisuje onoliko puta više, koliko je karaktera uneseno prije tipke ENTER. Ako se pritisne samo ENTER (bez unosa karaktera), poruka se ispisuje samo jedanput.

"Input buffer" je dio memorije koji služi za pohranjivanje podataka od trenutka kada se program prekine objektom "cin" do trenutka kada se ti podaci dodijele varijabli. Kada se umjesto jednog karaktera unese više karaktera, input buffer se prazni tako što sa početka unesenog niza karaktera uzima prvi, dodjeljuje ga varijabli i nastavlja program. Kako se ovdje radi o petlji, sve naredbe iz petlje se ponavljaju sve dok se input buffer ne isprazni.

Funkcija "ignore" objekta "cin" se koristi za pražnjenje input buffer-a, tako da se u kombinaciji sa "cin.get" ili "cin >>" može koristiti za prevazilaženje prikazanog problema. U principu, treba se držati dva pravila:

Pravilo 1:

```
char ch;
cin >> ch;
cin.ignore();
```

Kad god se upotrijebi "cin >>" za dodjeljivanje vrijednosti varijabli, treba da slijedi naredba "cin.ignore();", koja će isprazniti input buffer, zato što kod naredbe "cin >>" u input buffer-u nakon dodjeljivanja vrijednosti varijabli ostaje '\n' karakter.

Pravilo 2:

```
char ch;
cin.get(ch);
if (ch != '\n')
 cin.ignore();
```

Kad god se upotrijebi "cin.get" sa jednim argumentom, treba isprazniti input buffer naredbom "cin.ignore();", ali samo ako je u input buffer-u karakter '\n'. Taj slučaj se dešava kad se umjesto unošenja bilo kojeg karaktera samo pritisne ENTER.

Biblioteka funkcija <cctype> sadrži nekoliko funkcija koje služe za rad sa karakterima.
Funkcija "toupper" služi za pretvaranje karaktera u velika slova, dok funkcija "tolower" služi za pretvaranje karaktera u mala slova. Ako argument funkcije nije slovo, ove funkcije vraćaju nepromijenjenu vrijednost argumenta.

Primjer:

```
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
 char ch;
 do {
  cout << "Pritisnite K ili k za kraj, a bilo koju tipku za nastavak \n";
  cin.get(ch);
  ch = toupper(ch);
  if (ch != '\n')
   cin.ignore();
  if (ch != 'K')
   cout << "Nastavak programa?\n";
  else
   cout << "Kraj programa";
 } while (ch != 'K');
}
```

Bez obzira da li se unese "k" ili "K", funkcija "toupper" pretvara uneseni karakter u veliko slovo. Treba obratiti pažnju da osim biblioteke <iostream> treba uključiti i biblioteku <cctype>, jer funkcija "toupper" nije definisana u <iostream>.
Funkcija "toupper" sama po sebi ne mijenja vrijednost varijable u zagradi, nego se promijenjena vrijednost dodjeljuje varijabli znakom "=".
Funkcije iz biblioteke <cctype> koje služe za provjeru vrijednosti karaktera su date u tabeli:

Funkcija	Ima vrijednost "true" kad je argument:
isalpha	slovo iz alfabeta
isalnum	slovo iz alfabeta ili cifra
isdigit	cifra (0..9)
islower	malo slovo iz alfabeta
isupper	veliko slovo iz alfabeta
isprint	printabilni karakter
ispunct	printabilni karakter osim slova i cifara
isspace	prazan prostor (space, tabulator '\t', '\n')

Primjer:

```
char x;
cin >> x;
if (isupper(x) || ispunct(x)) cout << x;
```

Ako je varijabla "x" bilo koje veliko slovo, prazan prostor (space), tabulator ili prelazak u novi red (ENTER), ispisaće se vrijednost varijable x.
Tip podataka "string" nije standardni tip podataka u jeziku C++, nego je definisan u biblioteci <string>. Za razliku od tipa "char", tip "string" može imati vrijednost koja sadrži više od jednog karaktera. Ta vrijednost se ne navodi između apostrofa, nego između navodnika.
U biblioteci <string> se nalaze još neke funkcije za rad sa stringovima.
Funkcije "length()" i "size()"daje dužinu stringa, odnosno broj karaktera koje string sadrži.

Primjer:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
 string x = "Del Boy";
 int y = x.length();
 cout << y;
}
```

Funkcija "length()" se navodi uz varijablu pomoću tačke. U ovom primjeru y=7, jer string ima 7 karaktera.

Operator "+" se koristi za spajanje dva stringa.

Primjer (spajanje dva stringa):

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
 string x = "Del";
 string y = "Boy";
 cout << x+y;
}
```

Rezultat ovog programa je ispisivanje stringa "DelBoy".

Kao i kod numeričkih vrijednosti, i ovdje se mogu koristiti skraćeni operatori "+=" i "-="

Primjer (spajanje stringova operatorom +=):

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
 string x = "Del";
 string y = "Boy";
 x += y;
 cout << x;
}
```

U prethodnom primjeru naredba "x += y" odgovara naredbi "x = x + y".

Za poređenje stringova koriste se komparacioni operatori.

Primjer (poređenje stringova):

```
string s1 = "Del";
string s2 = "del";
string s3 = "Delboy";
s1 < s2  // true jer "j" ima veću ASCII vrijednost od "J"
s3 > s1  // true jer "b" ima veću ASCII vrijednost od null-karaktera
s2 > s3  // true jer "d" ima veću ASCII vrijednost od "D"
```

22. Funkcije

Funkcija je skup naredbi koje zajedno izvršavaju neki zadatak. Svaki C++ program mora imati jednu funkciju sa imenom "main". Osim te glavne funkcije, u programima se koriste i funkcije iz standardnih biblioteka kao što je funkcija "pow" iz biblioteke <cmath>. Osim tih funkcija, mogu se definisati i druge korisničke funkcije, pomoću kojih se glavni program podijeli na više manjih potprograma, čime se dobija na razumljivosti programa. Drugi razlog za korištenje funkcija je mogućnost njihovog višestrukog pozivanja.

Implementacija korisničke funkcije podrazumijeva dva koraka:

1. Definisanje funkcije
2. Pozivanje funkcije

Da bi se razumjela terminologija funkcija, treba pogledati program sa jednom funkcijom (main()).

Primjer (program s jednom funkcijom "main"):

```
#include <iostream>
using namespace std;
int main();
{
 cout << "Hello World!";
 return 0;
}
```

Prva linija, "int main()" se naziva **zaglavlje funkcije** (*function header*). Za razliku od drugih naredbi, na kraju zaglavlja funkcije ne dolazi znak ";". Zaglavlje funkcije se sastoji od:

- tipa podatka,
- naziva funkcije i
- liste argumenata.

U ovom slučaju tip podataka je integer, funkcija se zove "main" i nema argumenata.

Iza zaglavlja funkcije slijedi **tijelo funkcije**, između vitičastih zagrada "{" i "}". Unutar tijela prikazane funkcije navedene su dvije naredbe. Naredba "return 0;" (*return statement*) definiše koju će vrijednost funkcija vratiti. Svaka funkcija mora sadržati tu naredbu, osim kad je tip funkcije "void"; u tom slučaju naredba "return" je opciona.
Zaglavlje i tijelo funkcije zajedno čine **definiciju funkcije**. Kad je definisana, funkcija se izvršava kada se pozove. Sve funkcije se pozivaju iz programa, osim funkcije "main", koja se automatski izvršava kad se program pokrene.
U sljedećem primjeru je prikazana funkcija podijeljena na tri funkcije: "main", "poruka", "novi_red".
Primjer (program s funkcijom "main" i dvije korisničke funkcije):

```
#include <iostream>          //  1
using namespace std;         //  2
void poruka(void)            //  6
{                            //  7
 cout << "Hello World!";     //  8
}                            //  9

void novi_red(void)          // 11
{                            // 12
 cout << endl;               // 13
}                            // 14
int main();                  //  3
{                            //  4
 poruka();                   //  5
 novi_red();                 // 10
 return 0;                   // 15
}                            // 16
```

Sve funkcije moraju biti definisane prije pozivanja; u ovom primjeru prvo su definisane funkcije "poruka" i "novi_red", pa tek onda funkcija "main". Tok izvođenja naredbi u ovom programu naveden je u komentarima (iza //). Bez obzira na tok izvođenja, funkcija "main" se definiše posljednja.
Tip podataka "void" kod funkcije "poruka" znači da ta funkcija ne vraća nikakvu vrijednost glavnom programu.
Funkcija "poruka" ima kao argument "void" što znači da nema argumenata, kao ni funkcija "main". Navođenje riječi "void" nije obavezno, što znači da se može pisati i "funkcija(void)" i "funkcija()". Funkcija "poruka" ne sadrži naredbu "return;", zato što ne vraća nikakvu vrijednost.

22.1. Lokalne, globalne i statičke lokalne varijable

Varijable koje se koriste samo unutar jedne funkcije se nazivaju lokalnim.
Primjer (neprevodiv program sa lokalnom varijablom "broj"):

```
#include <iostream>
using namespace std;
void poruka(void)
{
 broj++;
 cout << "Hello World! " << broj << endl;;
}
int main()
{
int broj=0;
for (int i=1; i<=5; i++)
 {
  poruka();
 }
 return 0;
}
```

U ovom primjeru se varijabla "broj" koristi i unutar funkcije "main" i unutar funkcije "poruka", što nije ispravno, jer je upotreba te varijable ograničena na funkciju "main". Ako bi se ta varijabla deklarisala i unutar funkcije "poruka", te dvije varijable ne bi imale nikakve veze – unutar jednog programa postojale bi dvije varijable istog imena, svaka u svojoj funkciji, i mogle bi imati različite vrijednosti. Umjesto da funkcija "poruka" svaki put poveća vrijednost varijable "broj" za 1 (naredbom "broj++;"), funkcija "poruka" uopšte ne zna za varijablu "broj".

Globalne varijable se mogu koristiti u cijelom programu, tj. u svim funkcijama od kojih se program sastoji. Te varijable se deklarišu na početku programa, prije zaglavlja svih funkcija u programu. Prethodni primjer, koji se ne može prevesti jer se u funkciji "poruka" koristi varijabla "broj" koja nije deklarisana u toj funkciji, može se ispraviti tako što varijabla "broj" postane globalna, tako što se deklariše na početku programa.

Primjer (program sa globalnom varijablom "broj"):

```
#include <iostream>
using namespace std;
int broj;
void poruka(void)
{
 broj++;
 cout << "Hello World! " << broj << endl;;
}
int main()
{
 broj=0;
 for (int i=1; i<=5; i++)
 {
  poruka();
 }
}
```

Nije uvijek dobro koristiti globalne varijable. Nekad je praktičnije koristiti statičke lokalne varijable. Te varijable se deklarišu kao lokalne, samo što se prije tipa navodi riječ "static".

Primjer (program sa statičkom lokalnom varijablom "broj"):

```
#include <iostream>
using namespace std;
void poruka(void)
{
 static int broj=0;
 broj++;
 cout << "Hello World! " << broj << endl;;
}
int main()
{
 for (int i=1; i<=5; i++)
 {
  poruka();
 }
}
```

Riječ "static" ispred tipa varijable omogućuje da se inicijalna vrijednost (u ovom slučaju "0") toj varijabli dodjeljuje samo prvi put, dok svaki put kad se vrijednost te varijable promijeni ona pamti tu novu vrijednost.

22.2. Slanje informacije u funkciju

Vrijednost varijable se može prenositi iz jedne u drugu funkciju ako je ta varijabla deklarisana kao globalna. Drugi način je da varijabla bude argument funkcije.

Primjer (varijabla kao argument funkcije):

```
#include <iostream>
#include <string>
using namespace std;
void poruka(string s)
{
 cout << "Upisali ste " << s << endl;;
}
int main()
{
 string str;
 cout << "Unesite string: ";
 cin >> str;
 poruka(str);
}
```

U prikazanom primjeru se vrijednost varijable "str" prosljeđuje funkciji "poruka" i dodjeljuje argumentu "s" te funkcije. Ispred argumenta treba navesti tip (u ovom slučaju "string").
Funkcije mogu imati i više argumenata, koji se onda odvajaju zarezima.

Primjer (varijabla kao argument funkcije):

```
void poruka(string ime, string prezime, int godina)
```

Prilikom prosljeđivanja vrijednosti takvim funkcijama treba strogo voditi računa o redoslijedu navođenja argumenata – redoslijed mora biti isti i kod definisanaj funkcije i kod njenog pozivanja.
Ovakav način prosljeđivanja varijabli funkciji se naziva "*by value*" – po vrijednosti. To znači da funkcija kojoj se vrijednost proslijedi ne može mijenjati vrijednost te varijable. Tačnije, može je promijeniti ali je ne može vratiti funkciji koja je pozvala.

Primjer (prosljeđivanje po vrijednosti):

```
#include <iostream>
using namespace std;
void uduplaj(int x)
{
   cout << "Broj koji treba uduplati " << x << endl;
   x *= 2;
   cout << "Broj uduplan u funkciji uduplaj je " << x << endl;
}
int main()
{
   int num;
   cout << "Unesi broj: ";
   cin >> num;
   uduplaj(num);
   cout << "Broj uduplan u funkciji main je " << num << endl;
}
```

Rezultat programa:

```
Unesi broj: 4
Broj koji treba uduplati 4
Broj uduplan u funkciji uduplaj je 8
Broj uduplan u funkciji main je 4
```

Iz prikazanog primjera se vidi da se broj unesen u funkciji "main" (4) proslijedi funkciji "uduplaj", tamo se ispiše njegova vrijednost, pomnoži se sa 2, ispiše se uvećana vrijednost, ali se ta vrijednost ne vraća funkciji "main", nego tamo varijabla "x" i dalje ima vrijednost "4". Ako je potrebno da se vrijednost promijenjena u funkciji vrati u funkciju iz koje je pozvana, koristi se prosljeđivanje po referenci (*by reference*). To se postiže tako što se na oznaku tipa varijable (argumenta funkcije) dodaje sufiks "&".

Primjer (prosljeđivanje po referenci):

```
#include <iostream>
using namespace std;
void uduplaj(int& x)     // Znak "&" iza tipa varijable
{
   cout << "Broj koji treba uduplati " << x << endl;
   x *= 2;
   cout << "Broj uduplan u funkciji uduplaj je " << x << endl;
}
int main()
{
   int num;
   cout << "Unesi broj: ";
   cin >> num;
   uduplaj(num);
   cout << "Broj uduplan u funkciji main je " << num << endl;
}
```

Rezultat programa:

```
Unesi broj: 9
Broj koji treba uduplati 9
Broj uduplan u funkciji uduplaj je 18
Broj uduplan u funkciji main je 18
```

22.3. Vraćanje informacije iz funkcije

Argumenti se koriste da bi se vrijednosti poslale u funkciju. Naredba "return" u funkciji služi za vraćanje vrijednosti pozvane funkcije.

Primjer:

```
#include <iostream>
using namespace std;
int zbir (int x, int y)
{
 return x+y;
}
int main ()
{
 int x, y;
 cout << "Unesi brojeve x i y: ";
 cin >> x;
 cin >> y;
 cout << "x + y = " << zbir(x,y) << endl;
}
```

Rezultat programa:

```
Unesi brojeve x i y: 5 6
x + y = 11
```

U ovom slučaju funkcija "zbir" ne može biti tipa "void", jer treba vratiti neku vrijednost. U prikazanom primjeru funkcija "zbir" je tipa "int". U ovom primjeru također treba obratiti pažnju da se funkcija poziva direktno iza objekta "cout", što znači da se vrijednost koja je vraćena iz funkcije "zbir" nigdje ne pamti. Ako bi dalje u programu bilo potrebno koristiti tu vrijednost, korisno je pohraniti je u varijablu, jer svaki poziv funkcije troši resurse računara i usporava izvođenje programa. To u ovom jednostavnom primjeru nije primjetno, ali kod funkcija koje imaju puno naredbi, ta je razlika jako uočljiva.

Primjer1:

```
cout << "x + y = " << zbir(x,y) << endl;
cout << "x + y = " << zbir(x,y) << endl;
```

Primjer2:

```
z = zbir(x,y);
cout << "x + y = " << z << endl;
cout << "x + y = " << z << endl;
```

Primjer 2 bi se brže izvršio, jer funkciju "zbir" i sve naredbe iz nje poziva samo jedanput, zapamti vraćenu vrijednost u varijabli "z" i koristi tu varijablu za ispis. Primjer 1 bi dva puta pozvao funkciju "zbir" i sve njene naredbe.

23. Polja

Pod "poljima" se u programiranju podrazumijevaju indeksirane varijable (nizovi, matrice, tabele). Kao i druge varijable, i polja se moraju deklarisati prije upotrebe. Sintaksa deklaracije je skoro identična sintaksi deklaracije varijable.

Sintaksa:

```
tip naziv[broj];
```

Tipovi podataka su isti kao i kod drugih varijabli, i svi članovi polja moraju biti istog tipa (u jednom polju ne može biti podataka različitog tipa). Broj naveden između uglastih zagrada iza naziva polja se naziva deklarator veličine (*size declarator*) i on definiše maksimalni mogući broj članova polja.

Primjer:

```
int x[10];
```

U prikazanom primjeru se rezerviše prostor u memoriji za polje od 10 članova tipa integer. Ako varijabla tipa integer zauzima 4 bajta, to znači da ova deklaracija rezerviše 4x10=40 bajta memorije za smještaj ovog polja.

Deklarator veličine ne može biti varijabla, nego samo konstanta ili literal (broj koji se ne može mijenjati)!

Primjer:

```
#include <iostream>
using namespace std;
int main()
{
 const int br = 5;
 int x[br];
}
```

Svi članovi polja imaju isti naziv, a svaki član ima svoj indeks, koji pokazuje položaj člana u polju. Indeksi u C++ jeziku uvijek počinju od vrijednosti 0.

Deklaracija polja osim rezervisanja memorije za smještaj polja takođe dodjeljuje i inicijalne vrijednosti članovima polja. Koje će to vrijednosti biti, to zavisi od mjesta deklaracije. Ako se polje deklariše kao globalna varijabla (prije definicije funkcija), inicijalne vrijednosti će biti nula za numeričke, odnosno null karakter ('\0') za polja tipa karakter. Ako se polje deklariše lokalno, inicijalne vrijednosti neće biti jednake nuli.

Primjer (deklaracija polja kao lokalne varijable):

```
#include <iostream>
using namespace std;
int main ()
{
  int x[4];
  int i=0;
  for (i = 0; i < 4; i++)
    {
       cout << i << '\t' << x[i] << endl;
    }
}
```

Rezultat programa:

```
0       2293600
1       2009198341
2       2293664
3       4198592
```

Dodjeljivanje inicijalnih vrijednosti članovima polja se naziva inicijalizacija, koja može biti implicitna i eksplicitna.

Primjer (eksplicitna inicijalizacija polja):

```
int x[4] = {33, 23, 44, 55};
float y[3] = {33.2, 2.3, -0.44};
char azb[3] = {'A', 'B', 'V'};
string gr_alf[4] = {"alfa", "beta", "gama", "delta"};
```

Kod eksplicitne inicijalizacije se u uglastim zagradama navodi broj članova polja, a zatim se navodi niz vrijednosti u vitičastim zagradama, koje se dodjeljuju članovima polja. Tih vrijednosti ne smije biti više nego što polje ima članova, a ako ih ima manje, preostali članovi polja dobiju vrijednost 0, 0.0 ili '\0', ovisno o kojem se tipu podataka radi.

Kod implicitne inicijalizacije, broj članova polja se ne navodi, nego se odredi iz broja vrijednosti nabrojanih u vitičastoj zagradi iza deklaracije polja.

Primjer (implicitna inicijalizacija polja):

```
int x[] = {33, 23, 44, 55};
float y[] = {33.2, 2.3, -0.44};
char azb[] = {'A', 'B', 'V'};
string gr_alf[] = {"alfa", "beta", "gama", "delta"};
```

Polja tipa karakter se mogu inicijalizirati i na drugi način, kako je prikazano u sljedećem primjeru, gdje je polje "x" inicijalizirano implicitno pomoću niza karaktera, a polje "y" pomoću stringa. Bez obzira što je "y" deklarisano kao tip karakter, zbog uglastih zagrada se ponaša kao polje, a ne kao obična varijabla.

Primjer (implicitna deklaracija karakter polja):

```
#include <iostream>
using namespace std;
int main ()
{
  char x[] = {'u','p','s'};
  char y[] = "ups";
  int i=0;
  for (i = 0; i < 3; i++)
    {
      cout << y[i];
    }
  cout << endl << x << endl;
}
```

Rezultat programa:

```
ups
ups
```

Iako se na prvi pogled čini da je C++ nedosljedan u pravilima, jer se polje ispisuje i sa i bez uglastih zagrada, zna se tačno kada se trebaju a kada ne koristiti uglaste zagrade. Kod polja "y" u datom primjeru se koriste uglaste zagrade jer se u njima kao indeks koristi varijabla "i". Kod polja "x", kako uglastih zagrada nema, ispisaće se redom svi članovi polja jedan za drugim.

Kao pravilo se može uzeti da se uglaste zagrade koriste onda kad se ne koristi cijelo polje, pa treba koristiti varijablu za indeks. Ako uglastih zagrada nema, koriste se svi članovi polja, jedan za drugim.

Ako se objekat "cin" koristi za unos vrijednosti polja tipa karakter, uneseni niz znakova ne može sadržati prazno mjesto, jer "cin" to interpretira kao završetak unosa. To se može prevazići korištenjem funkcije "get", kao u sljedećem primjeru.

Primjer (korištenje funkcije cin.get):

```
#include <iostream>
using namespace std;
int main ()
{
  char y[5];
  cout <<"Unesite riječ do 5 slova" << endl;
  cin.get(y,6,'\n');
  cout << y << endl;
}
```

Treći argument funkcije "get" je ENTER, odnosno vrijednost koja završava unos podataka. Drugi način unosa vrijednosti u polje tipa karakter, je pomoću funkcije "getline".

Primjer (korištenje funkcije cin.getline):

```
#include <iostream>
using namespace std;
int main ()
{
  char y[5];
  cout <<"Unesite riječ do 5 slova" << endl;
  cin.getline(y,6);
  cout << y << endl;
}
```

I kod funkcije "get" i kod funkcije "getline" drugi argument treba biti za 1 veći od broja članova polja, jer je potreban prostor za posljednji karakter koji se unosi ENTER ('\n').

Primjer (program za unošenje, sortiranje i štampanje polja):

```
#include <iostream>
using namespace std;
int main ()
{
  // DEKLARACIJA
  int x[10];
  int y[10];
  int i, j, n;
  // UNOSENJE
  cout << "Unesite broj članova polja: ";
  cin >> n;
  for (i = 0; i < n; i++)
    {
      cout << "Unesite clan br. " << i << ": ";
      cin >> x[i];
      y[i] = x[i];
    }
  // SORTIRANJE
  for (i = 0; i < n-1; i++)
    {
      for (j = i+1; j < n; j++)
        {
          if (y[i] < y[j]) swap(y[i],y[j]);
        }
    }
  // STAMPANJE
  cout << "x:" << '\t' << "y:" << endl;
  for (i = 0; i < n; i++)
    {
      cout << x[i] << '\t' << y[i] << endl;
    }
}
```

Rezultat programa:

```
Unesite broj clanova polja: 4
Unesite clan br. 0: 3
Unesite clan br. 1: -2
Unesite clan br. 2: 4
Unesite clan br. 3: 0
x:      y:
3       4
-2      3
4       0
0       -2
```

Matrica, kao dvodimenzionalno polje, se koristi tako što se svaki indeks navodi u posebnim zagradama.

Primjer (program za unošenje matrice x_{5x5}):

```
#include <iostream>
using namespace std;
int main ()
{
  int x[5][5];
  int i, j;
  for (i = 0; i < 5; i++)
    {
      for (j = 0; j < 5; j++)
        {
          cout << "Unesite clan " << i << "," << j << ": ";
          cin >> x[i][j];
        }
    }
}
```

24. Datoteke

Datoteka je skup podataka snimljenih na disk (ili drugu formu trajne memorije). Datoteke mogu biti tekstualne ili binarne. Tekstualne datoteke sadrže samo tekst, tj. koleciju karaktera, pri čemu se svaki karakter u datoteci predstavlja njegovim ASCII kodom. Primjer tekstualne datoteke su dokumenti kreirani pomoću programa Windows Notepad. Binarne datoteke, osim teksta, mogu da sadrže i sve druge vrste podataka. Naprimjer, MS Word dokument sadrži osim teksta i podatke o fontu, veličini i boji slova, marginama, tabelama, slikama, itd.
Za unošenje podataka sa tastature i prikazivanje podataka na ekranu koriste se objekti "cin" i "cout" iz biblioteke <iostream>. Za rad sa datotekama koristi se biblioteka <fstream> (slovo "f" u imenu predstavlja riječ *file* – datoteka).
Biblioteka <fstream> uvodi tri nova tipa podataka:

- ofstream – za kreiranje datoteka i zapisivanje u datoteku (*output*)
- ifstream – za čitanje podataka iz datoteka (*input*)
- fstream – za čitanje podataka iz datoteka, kreiranje datoteka i zapisivanje u datoteku

Primjer (zapisivanje u datoteku):

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  int x;
  ofstream izlaz;
  izlaz.open("c:\\podaci\\podaci.txt");
  cout << "Unesite neki broj" << endl;
  cin >> x;
  izlaz << x;
  izlaz.close();
}
```

Na početku se deklarišu cjelobrojna varijabla "x" i datoteka "izlaz". Datoteka koja se u programu koristi pod nazivom "izlaz" je u stvari datoteka "podaci.txt" iz foldera "c:\podaci\". U apsolutnom imenu datoteke koriste se po dva znaka "\", jer bi se jedan znak "\" interpretirao kao escape-sekvenca. Broj koji se unese sa tastature se prvo pohrani u varijablu "x" a zatim se zapiše u datoteku "izlaz". Na kraju se datoteka mora zatvoriti member funkcijom "close()".
Ako folder "c:\podaci" ne postoji, program neće javiti grešku, niti će kreirati taj folder. Zato treba paziti da se kod navođenja relativnih naziva datoteka vodi računa o folderima.
Ako datoteka ne postoji, program će je kreirati funkcijom "open". Ako već postoji datoteka, program će je obrisati, kreirati novu pod istim imenom, a zatim u nju upisati podatke.

Primjer (zapisivanje u datoteku):

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  int x;
  ifstream ulaz;
  ulaz.open("c:\\podaci\\podaci.txt");
  ulaz >> x;
  cout << "Broj iz datoteke " << x << endl;
  ulaz.close();
}
```

Uz funkciju "open" se može navesti i drugi argument, koji definiše način otvaranja datoteke.

Argument	Opis
ios::app	*Append* – dodavanje na kraj datoteke (prethodni sadržaj datoteke se ne mijenja, nego se na kraj dodaju novi podaci
ios::binary	*Binary* – zapisivanje u datoteku u binarnom obliku

Primjer (zapisivanje na kraj datoteke):

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  int x;
  ofstream izlaz;
  izlaz.open("c:\\podaci\\podaci.txt", ios::app);
  cout << "Unesite neki broj" << endl;
  cin >> x;
  izlaz << x << endl;
  izlaz.close();
}
```

U ovom slučaju će prethodni sadržaj datoteke "podaci.txt" biti sačuvan, a novi podaci će se dodavati na kraj datoteke, jedan ispod drugog, jer iza izraza "izlaz << x" slijedi oznaka novo reda "endl".
Unošenje teksta u datoteku zahtijeva korištenje stringova. U sljedećem primjeru pokazana je petlja za unos teksta u datoteku, koja se završava kad se unese "***".

Primjer (zapisivanje teksta u datoteku):

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main ()
{
  string x;
  ofstream izlaz;
  izlaz.open("c:\\podaci\\podaci.txt", ios::app);
  while (x != "***")
    {
      cout << "Unesite neki tekst (za kraj unesite ***):" << endl;
      cin >> x;
      izlaz << x << endl;
    }
  izlaz.close();
}
```

Kod čitanja iz datoteke, koristi se funkcija "fail" za detekciju kraja datoteke.

Primjer (štampanje teksta iz datoteke):

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main ()
{
  string x;
  ifstream ulaz;
  ulaz.open("c:\\podaci\\podaci.txt");
  while (!ulaz.fail())
    {
      ulaz >> x;
      cout << x << endl;
    }
  ulaz.close();
}
```

Vrijednost funkcije "fail()" objekta "ulaz" je jednaka jedinici kada se dođe do kraja datoteke. Sve dok ima podataka, objekat "ulaz.fail()" ima vrijednost nula, odnosno njegova negacija "!ulaz.fail()" ima vrijednost jedan, tako da se petlja nastavlja. Na kraju datoteke, "ulaz.fail()" ima vrijednost 1, odnosno "!ulaz.fail()" ima vrijednost 0, što odgovara logičkom "false", te se petlja tada prekida.

25. Objektno-orijentisano programiranje - strukture

Objektno-orijentisano programiranje se zasniva na objektima (*objects*). Grupe objekata se nazivaju strukture (*structures*) i klase (*classses*), koje prenose svoje osobine na objekte koje im pripadaju – osobina nasljeđivanja (*inheritance*). Između objekata postoje relacije (*relations*). Jedna od najvažnijih osobina objektno-orijentisanih programskih jezika je postojanje apstraktnih (*abstract*) tipova podataka.
Varijabla može da sadrži samo jednu vrijednost. Polje može da sadrži više vrijednosti, ali sve moraju biti istog tipa. C++ omogućuje da se više varijabli smjesti u strukturu (*structure*) koja tako može da sadrži vrijednosti različitih tipova podataka. Struktura se u programu ponaša kao tip podataka, ali ne kao standardni tip koji je ugrađen u programski jezik ili biblioteku, nego kao korisnički tip podataka (*user defined data type*). Kao i svaki drugi tip podataka, i struktura zahtijeva deklaraciju.

Primjer (deklaracija strukture):

```
struct osoba {
  string ime;
  int starost;
};
```

Deklaracija strukture se vrši naredbom "struct". Za razliku od funkcija, iza vitičaste zagrade koja zatvara deklaraciju strukture treba navesti znak ";".
Varijable koje pripadaju strukturi se nazivaju varijable članovi (*member*). Jedna varijabla može istovremeno biti član više struktura.
Deklaracija strukture se može izvršiti u bilo kojem dijelu programa, ali se obično deklarišu neposredno prije "main" funkcije. To ne znači da varijable deklarisane u strukturi automatski postaju globalne; deklaracijom strukture deklariše se novi tip podataka, a ne varijable koje ga čine. Da bi se taj tip podataka mogao koristiti globalno, mora tako biti i deklarisan.

Primjer (deklaracija strukture i korištenje varijabli):

```
#include <iostream>
#include <string>
using namespace std;
struct osoba {
  string ime;
  string prezime;
  int starost;
};
int main()
{
  osoba o1;
  osoba o2, o3;
  osoba face[100];
  o1.ime = "Rodney";
  o1.prezime = "Trotter";
  o1.starost = 25;
  cout << o1.ime << " " << o1.prezime << endl;
}
```

U ovom primjeru je deklarisana struktura tipa "osoba", koja sadrži tri varijable: ime, prezime i starost. Instanca strukture se naziva objekat. U funkciji "main" su zatim deklarisani objekti (varijable) o1, o2 i o3, te objekat (polje) face[100] koje može da sadrži 100 različitih vrijednosti tipa "osoba". Samo objektu "o1" su dodijeljene vrijednosti, dok su ostali objekti navedene samo da bi se vidjelo kako je deklaracija struktura ista kao i deklaracija varijabli i polja. Kako struktura tipa "osoba" sadrži tri vrijednosti različitog tipa, svaka vrijednost se dodjeljuje posebno, koristeći tačku kao separator. U narednoj tabeli prikazana je upotreba varijable, polja i strukture za smještaj tri vrijednosti.

	Zasebne varijable	Polje	Struktura

deklaracija	string ime;
	string prezime;
	int starost;
		string osoba[3];	struct x {
			  string ime;
			  string prezime;
			  int starost;
			};
			x osoba;

1. vrijednost	ime	osoba[1]	osoba.ime
2. vrijednost	prezime	osoba[2]	osoba.prezime
3. vrijednost	starost	osoba[3]	osoba.starost

Vrijednosti pohranjene u odvojene varijable mogu biti različitog tipa, ali se ne mogu koristiti u kombinaciji s petljama. Vrijednosti pohranjene u polje moraju biti istog tipa (u ovom primjeru je i starost pohranjena kao string, iako je to numerička vrijednost). Struktura predstavlja novi tip podataka, korisnički definisan, koji u sebi sadrži tri varijable, koje mogu biti različitog tipa. Nakon deklaracije strukture, u programu se deklariše novi objekat tipa "x", odnosno takav da sadrži tri varijable-članice.
Kao i polja, i strukture se mogu inicijalizirati.

Primjer (inicijalizacija strukture):

```
#include <iostream>
#include <string>
using namespace std;
struct osoba {
  string ime;
  string prezime;
  int starost;
};
int main()
{
  osoba o1 = {"Rodney", "Trotter", 25};
  cout << o1.prezime << " " << o1.ime << endl;
}
```

Za razliku od inicijalizacije polja, članovima strukture se mogu dodjeljivati vrijednosti različitog tipa (u ovom slučaju dva stringa i jedan broj).
Strukture se mogu deklarisati i kao ugniježđene (*nested*), odnosno tako da se deklaracija jedne strukture koristi unutar deklaracije druge strukture. Pri tome treba voditi računa da struktura mora biti deklarisana prije nego što se upotrijebi.

Primjer (ugniježđene strukture):

```
#include <iostream>
#include <string>
using namespace std;
struct datum {
  int dan;
  int mjesec;
  int godina;
};
struct osoba {
  string ime;
  string prezime;
  datum rodjen;
};
int main()
{
  osoba o1 = {"Rodney", "Trotter"};
  o1.rodjen.dan = 12;
  o1.rodjen.mjesec = 3;
  o1.rodjen.godina = 1966;
  cout << o1.ime << " " << o1.prezime << endl;
  cout << o1.rodjen.dan << "." << o1.rodjen.mjesec << "."
       << o1.rodjen.godina << endl;
}
```

Rezultat programa:

```
Rodney Trotter
12.3.1966
```

Prikazani primjer ilustruje relaciju "ima" ("*has a*"). U ovom slučaju struktura "osoba" ima strukturu "datum". Svi objekti tipa "osoba" (u ovom slučaju objekat "o1") imaju datum rođenja (u ovom slučaju objekat "rodjen").
Kad se kreira instanca strukture (objekat zasnovan na strukturi), automatski se izvršava funkcija koja se naziva konstruktor (*constructor*). Ta funkcija se ne mora navoditi, ali je poželjno koristiti ih, barem za definisanje podrazumijevanih (*default*) vrijednosti varijabli. Konstruktor se definiše kao funkcija istog naziva kao i struktura, sa ili bez argumenata.

Primjer (strukture bez konstruktora):

```
#include <iostream>
#include <string>
using namespace std;
struct datum {
  int dan;
  int mjesec;
  int godina;
};
struct osoba {
  string ime;
  string prezime;
  datum rodjen;
};
int main()
{
  osoba o1;
  cout << o1.ime << " " << o1.prezime << endl;
  cout << o1.rodjen.dan << "." << o1.rodjen.mjesec << "."
       << o1.rodjen.godina << endl;
}
```

Rezultat programa:

```
35.2.2293600
```

Iz prikazanog primjera se vidi da su podrazumijevane vrijednosti za string varijable prazni stringovi, dok numeričke varijable dobiju vrijednosti koje nemaju smisla. Za definisanje podrazumijevanih vrijednosti koristi se konstruktor, kao funkcija istog imena kao i struktura.

Primjer (konstruktor bez argumenata):

```
#include <iostream>
#include <string>
using namespace std;
struct datum {
  int dan;
  int mjesec;
  int godina;
  datum()
    {
      dan = mjesec = godina = 0;
    }
};
struct osoba {
  string ime;
  string prezime;
  datum rodjen;
};
int main()
{
  osoba o1;
  cout << o1.ime << " " << o1.prezime << endl;
  cout << o1.rodjen.dan << "."
       << o1.rodjen.mjesec << "."
       << o1.rodjen.godina << endl;
}
```

Rezultat programa:

```
0.0.0
```

Ako je potrebno da se inicijalne vrijednosti dodijele kasnije, iz programa, koristi se konstruktor sa argumentima.

Primjer (konstruktor sa argumentima):

```
#include <iostream>
#include <string>
using namespace std;
struct datum {
  public:
  int dan;
  int mjesec;
  datum(int d, int m)
  {
   dan = d;
   mjesec = m;
  }
};
int main()
{
  int d, m;
  cout << "Unesite podrazumijevani dan i mjesec rodjenja" << endl;
  cin >> d >> m;
  datum d1(d, m);
  cout << d1.dan << "." << d1.mjesec << "." << endl;
  int x;
  cout << "Unesite dan i mjesec rodjenja" << endl;
  cin >> x;
  if (x <=31) d1.dan = x;
  cin >> x;
  if (x <=12) d1.mjesec = x;
  cout << d1.dan << "." << d1.mjesec << "." << endl;
}
```

Rezultat programa:

```
Unesite podrazumijevani dan i mjesec rodjenja
1 1
1.1.
Unesite dan i mjesec rodjenja
32 11
1.11.
```

Ako se unese dan koji nije manji ili jednak od 31, vrijednost varijable "x" se neće proslijediti objektu "d1.dan", a ako se unese mjesec veći od 12, "x" se neće dodijeliti objektu "d1.mjesec". U oba slučaja, ako se ne dodijeli vrijednost, ostaje podrazumijevana vrijednost unesena u prethodnom koraku, naredbom "datum d1(d, m);".

26. Klase

Primjer (struktura):

```
#include <iostream>
#include <string>
using namespace std;
struct datum {
  int dan;
  int mjesec;
  int godina;
};
int main()
{
  datum d1;
  cout << "Unesi dan mjesec i godinu rodjenja" << endl;
  cin >> d1.dan >> d1.mjesec >> d1.godina;
  cout << d1.dan << "." << d1.mjesec << "." << d1.godina << endl;
}
```

Deklaracija klase (*class*) je istovjetna deklaraciji strukture, samo što se umjesto naredbe "struct" koristi naredba "class". Strukture su zadržane u jeziku C++ radi kompatibilnosti sa jezikom C.

Klasa, kao i struktura, pored atributa (varijabli) može sadržati i metode (funkcije). Razlika između strukture i klase je u tome što su kod klase sve varijable privatne, a kod strukture javne.
U prethodnom primjeru prikazana je struktura za tip podataka "datum", koji sadrži tri cjelobrojne varijable. Ako se naredba "struct datum {" zamijeni naredbom "class datum {", taj program se neće moći prevesti, jer su varijable "dan", "mjesec" i "godina" deklarisane unutar klase kao privatne (lokalne) varijable i ne mogu se koristiti u funkciji "main", niti u jednoj drugoj funkciji u programu. Varijable u klasi se mogu deklarisati kao privatne (*private*) i javne (*public*), koristeći odgovarajuće labele u programu.
U narednom primjeru je prethodna struktura pretvorena u klasu tako što je naredba "struct" zamijenjena naredbom "class" i ispred deklaracije varijabli u klasi dodata je labela "public:" kako bi se varijable deklarisale kao javne.

Primjer (klasa):

```
#include <iostream>
#include <string>
using namespace std;
class datum {
  public:
  int dan;
  int mjesec;
  int godina;
};
int main()
{
  datum d1;
  cout << "Unesi dan mjesec i godinu rodjenja" << endl;
  cin >> d1.dan >> d1.mjesec >> d1.godina;
  cout << d1.dan << "." << d1.mjesec << "." << d1.godina << endl;
}
```

Labele "public:" za javne varijable i "private:" za privatne varijable se mogu koristiti i kod klasa i kod struktura.
Funkcije koje pripadaju klasi mogu vršiti provjeru podataka prije nego što se unesena vrijednost dodijeli varijabli – članu klase. To se naziva enkapsulacija (*encapsulation*), odnosno sakrivanje podataka. Varijablama koje su deklarisane kao privatne mogu pristupiti samo funkcije koje su članice te klase.
Klasa može da sadrži i funkcije, kao u sljedećem primjeru.

Primjer (klasa sa funkcijom):

```
#include <iostream>
#include <string>
using namespace std;
class datum {
  public:
  int dan;
  int mjesec;
  int godina;
  void unesi_datum(datum& dat)
    {
      cout << "Unesi dan mjesec i godinu rodjenja" << endl;
      cin >> dat.dan >> dat.mjesec >> dat.godina;
     return;
    }
};

int main()
{
  datum d1;
  d1.unesi_datum(d1);
  cout << d1.dan << "." << d1.mjesec << "." << d1.godina << endl;
}
```

Funkcija "unesi_datum" je mogla biti deklarisana i izvan klase, ali je ovdje definisana unutar klase "datum", pa se ne poziva kao samostalna funkcija, nego kao funkcija koja pripada objektu "d1" tipa "datum". Tako je funkcija "unesi_datum" postala metoda klase "datum".
Ovdje se vidi i da se objekat "dat" u definiciji funkcije "unesi_datum" prosljeđuje po referenci, a ne po vrijednosti, tako što je tipu podataka "datum" dodat sufiks "&".

27. Programski jezik PASCAL

Pascal je programski jezik sličan jeziku C/C++, ali sa strožijim pravilima. Upravo iz tog razloga, Pascal je jedan od programskih jezika koji se najčešće (ali ne i isključivo) koristi za učenje tehnika programiranja. Prvi međunarodni standard koji definiše programski jezik Pascal je napisan 1983. godine, (ISO 7185). Ta verzija je poznata kao *unextended Pascal* ili "standardni Pascal". Proširena verzija standardnog Pascala (*extended Pascal*) je opisana u standardu ISO 10206, 1990. godine. Postoji i verzija Pascala koja ima osobine objektno-orijentisanog programskog jezika pod nazivom *Object Pascal*, a integrisano razvojno okruženje ovog jezika je *Delphi*.
Komercijalne verzije Pascala su *Borland Pascal* (poznatiji kao *Turbo Pascal*), *Delphi*, *Compaq Pascal*, *THINK Pascal*, i *CodeWarrior Pascal*. Postoje i *freeware* verzije kao što su *FreePascal* i *GNU Pascal*.

27.1. Instalacija, pokretanje, način rada

Ovdje će biti opisano *open source* integrisano razvojno okruženje *GNU Pascal*. Instalacijska datoteka aktuelne verzije se može naći na web stranici firme firme Bloodshed Software (www.bloodshed.net).
U GNU Pascalu mogu se kreirati Windows aplikacije, konzolne (DOS) aplikacije, statičke i dinamičke (DLL) biblioteke. U ovom kursu, radi jednostavnosti izvornog koda, ograničićemo se na konzolne aplikacije. Naredbom "New project..." iz menija "File" treba kreirati projekat tipa "Console Application", dodijeliti mu ime, te odrediti u kojem folderu će se pohraniti datoteka projekta (sa ekstenzijom .dp) i datoteka izvornog Pascal koda (sa ekstenzijom .pas). Preporučuje se koristiti nazive foldera i datoteka bez praznih mjesta (kao u MS-DOS-u).
Najjednostavniji program, koji štampa tekstualnu poruku na ekran je sljedeći:

Primjer:

```
program Hello;
begin
   writeln ('Hello, world.')
end.
```

Rezultat programa:

```
Hello, world.
```

Program se prevodi i pokreće naredbom "Compile and run" iz menija "Execute". Tom prilikom se dodjeljuje i ime izvornoj datoteci, jer se ista mora snimiti prije prevođenja. Nakon pokretanja ovog programa, u većini novijih verzija Windows operativnog sistema se konzolni prozor automatski zatvara nakon završetka programa, tako da nije moguće vidjeti rezultat. Dodavanjem naredbe "readln" u program, isti će se privremeno zaustaviti i omogućiti da se vidi rezultat. Pritiskom na tipku ENTER program se nastavlja (u ovom slučaju i završava).

Primjer:

```
program Hello;
begin
   writeln ('Hello, world.');
   readln
end.
```

Treba obratiti pažnju da iza naredbe "writeln" treba dodati i znak ";".

27.2. Struktura programa, konvencije pisanja naredbi

Osnovna struktura Pascal programa je sljedeća:

```
PROGRAM NazivPrograma (ListaDatoteka);
CONST
 (* Deklaracije konstanti *)
TYPE
 (* Deklaracije tipova podataka *)
VAR
 (* Deklaracije varijabli *)
 (* Definicije potprograma *)
BEGIN
 (* Naredbe samog programa *)
END.
```

Nabrojani elementi programa moraju biti navedeni tačno ovim redoslijedom, a neki od njih mogu biti izostavljeni ako nema potrebe za njima. Naredbe "program", "begin" i "end." su obavezne.

Komentari u programu se navode između oznaka "(*" i "*)". Neke verzije Pascala podržavaju i komentare između vitičastih zagrada "{" i "}".
Identifikatori su nazivi pomoću kojih se pozivaju pohranjene vrijednosti, kao što su varijable i konstante. Pravila za imenovanje identifikatora su ista kao i u C++ jeziku: mogu sadržati slova engleskog alfabeta, cifre i znak "_", a prvi znak mora biti slovo. Pascal **ne razlikuje velika i mala slova!** To znači da imena "Ime", "ime" i IME" označavaju isti identifikator.
Zaglavlje programa počinje naredbom "program", iza koje slijedi naziv programa. Na kraju zaglavlja nalazi se znak ";". Većina prevodilaca dozvoljava i da se zaglavlje programa izostavi.
Nakon zaglavlja slijedi deklaracijski dio programa. Prvo se deklarišu konstante, zatim tipovi podataka i na kraju varijable.
Konstante se deklarišu nakon naredbe "const", gdje im se dodjeljuje naziv i vrijednost koja se više ne može mijenjati.
Primjer:

```
program Konstante;
const
   ime = 'Rodney';
   prezime = 'Trotter';
   pi = 3.1415926535897932;
begin
end.
```

U standardnom Pascalu se stringovi navode isključivo između apostrofa, dok u nekim verzijama za označavanje stringova mogu koristiti i navodnici.
Osim naziva i vrijednosti, prilikom deklaracije se može definisati i tip podatka za konstantu.

```
const
   x : real = 22;
```

U prikazanom primjeru konstanta "x" će biti realna, tj. neće imati cjelobrojnu vrijednost "22", nego realni broj "22.0".
Varijable, za razliku od konstanti, mogu mijenjati svoju vrijednost unutar programa. Prilikom deklaracije varijabli, mora se odrediti i tip podataka koji će se pohranjivati u varijabli. Ako je potrebno deklarisati više varijabli istog tipa, one se mogu navesti kao lista varijabli, međusobno odvojene zarezima.
Primjer:

```
program Varijable;
var
   ime, prezime : string;
   x : integer;
   y : real;
   a : char;
   status : Boolean;
begin
end.
```

U prikazanom primjeru su varijable "ime" i "prezime" deklarisane kao tip "string", koji ne postoji u standardnom Pascalu, ali u većini novijih prevodilaca taj tip postoji. Ostali nabrojani tipovi imaju isto značenje kao i u jeziku C++, i to su četiri standardna tipa podataka u Pascalu.
Tip **integer** može da sadrži cijele brojeve od –32768 do 32767, i zauzima 16 bita. Postoji i 32-bitni tip podataka "longint". Tip podataka **real** može da sadrži realne brojeve od $3.4x10^{-38}$ to $3.4x10^{38}$, i od $-3.4x10^{38}$ to $-3.4x10^{-38}$. Tip **char** sadrži jedan 8-bitni karakter. Tip **Boolean** može imati dvije vrijednosti: TRUE i FALSE.
Osim dodjeljivanja tipa, prilikom deklaracije se varijabli može dodijeliti i početna (inicijalna) vrijednost.
Izvršni dio programa se nalazi između naredbi "begin" i "end.". Inače, te dvije naredbe se koriste da označe početak i kraj takozvanog "bloka naredbi", tako da i izvršni dio programa čini jedan blok naredbi. Iza naredbe "end" obavezno se stavlja tačka, kao oznaka kraja programa. Sve naredbe u Pascalu obavezno završavaju znakom ";", a taj znak se može izostaviti jedino kod naredbe koja se nalazi neposredno prije oznake kraja bloka "end". Suvišna prazna mjesta i prelaske u novi red prevodilac za Pascal ignoriše, i samo znak ";" služi za označavanje kraja naredbe.
Dodjeljivanje vrijednosti se vrši znakom ":=". Izraz dodjeljivanja vrijednosti može da sadrži jednu vrijednost, ili izraz čijim se izračunavanjem dobije rezultat (čija se vrijednost onda dodjeljuje identifikatoru). Izrazi se u Pascalu kreiraju pomoću konstanti, identifikatora, operatora i funkcija.

27.3. Operatori i funkcije (aritmetički, logički, komparativni)

Aritmetički operatori u Pascalu su:

Operator	Operacija	Operandi	Rezultat
+	sabiranje	real, integer	real, integer
-	oduzimanje	real, integer	real, integer
*	množenje	real, integer	real, integer
/	realno dijeljenje	real, integer	real
div	cjelobrojno dijeljenje	integer	integer
mod	ostatak	integer	integer

Osim za oduzimanje, operator "-" se koristi i kao unarni operator za negativnu vrijednost.
Standardne matematičke funkcije u Pascalu su:

Funkcija	Opis	Tip argumenta	Tip rezultata
abs	apsolutna vrijednost	real, integer	isto kao argument
arctan	arkus tangens u radijanima	real, integer	real
cos	kosinus ugla u radijanima	real, integer	real
exp	e^x	real, integer	real
ln	prirodni logaritam	real, integer	real
round	zaokruživanje na najbliži cijeli broj	real	integer
sin	sinus ugla u radijanima	real, integer	real
sqr	kvadrat (square); x^2	real, integer	isto kao argument
sqrt	kvadratni korijen (square root)	real, integer	real
trunc	zaokruživanje na najbliži manji cijeli broj	real, integer	integer

Relacioni operatori u Pascalu su:

Operator	Značenje
>	veće od
<	manje od
=	jednako
>=	veće ili jednako
<=	manje ili jednako
<>	različito

Bulovi operatori su "not", "and", "or" i "xor":

A	B	not A	A and B	A or B	A xor B
		negacija (~) NE	konjukcija (^) I	disjunkcija (v) ILI	ekskluzivno ILI
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

28. Naredbe ulaza i izlaza

Za unos podataka u program koristi se naredba "read", odnosno "readln". U sljedećoj tabeli je prikazana razlika između naredbi "read" i "readln".

Naredbe	Uneseni podaci	a	b	c	d
read (a); read (b);	45 97 3 1 2 3	45	97		
readln (a); read (b);	45 97 3 1 2 3	45	1		
read (a, b, c, d);	45 97 3 1 2 3	45	97	3	1
readln (a, b); readln (c, d);	45 97 3 1 2 3	45	97	1	2

Sintaksa:

```
readln (lista);
```

Argument naredbi "read" i "readln" je lista varijabli, razdvojenih zarezima. Konstante ne mogu biti argumenti tih naredbi, jer se njihova vrijednost ne može mijenjati.

Primjer:

```
program Unos;
var
  x, y : real;
begin
  readln (x, y);
end.
```

Za prikazivanje podataka na ekran koriste se naredbe "write" i "writeln". Razlika između njih je u tome što nakon naredbe "write" kursor ostaje u istom redu, a kod naredbe "writeln" prelazi u novi red.

Sintaksa:

```
writeln (lista);
```

Argument naredbe "writeln" je lista stringova, identifikatora i/ili izraza, razdvojenih zarezima.

Primjer:

```
program Ispis;
var
  x, y : real;
begin
  x := 23.45;
  y := 1.5;
  writeln ('x + y =', x + y);
end.
```

Rezultat:

```
x + y = 2.495000000000000e+01
```

Kako je rezultat sabiranja vrijednosti varijabli x i y realni broj, on se ispisuje eksponencijalnim načinom ispisivanja. Ispis se može formatirati, korištenjem operatora ":".

Sintaksa:

```
vrijednost : ukupna_širina
realni_broj : ukupna_širina : broj_decimalnih_mjesta
```

Primjer:

```
program formatirani_ispis;
var
  x, y : real;
begin
  x := 23.45;
  y := 1.5;
  writeln ('x =', x : 11 : 3);
  writeln ('x + y = ', x + y : 6 : 2);
end.
```

Rezultat:

```
x =     23.450
x + y =  24.95
```

U prikazanom primjeru je za varijablu "x" rezervisano ukupno 7 mjesta, od kojih su posljednja 3 rezervisana za decimale. Za rezultat "x+y" je rezervisano ukupno 6 mjesta, od kojih su posljednja 2 rezervisana za decimale.

Ista sintaksa važi i za stringove; nakon znaka ":" se navodi broj karaktera koje treba prikazati. Ako je broj karaktera u stringu kraći veći od broja "n", prikazuje se prvih "n" karaktera iz stringa. Ako je broj karaktera u stringu manji od broja "n", string se ispisuje poravnat po desnom kraju.

Primjer:

```
program formatirani_ispis;
begin
  writeln ('MAAJKA' : 3);
  writeln ('MAAJKA' : 8);
end.
```

Rezultat:

```
MAA
  MAAJKA
```

28.1. IF-THEN-ELSE, CASE

Sintaksa 1:

```
if izraz then
  naredba_ako_je_izraz_true;
```

Prikazana sintaksa se koristi ako se u slučaju ispunjenja uslova treba izvršiti samo jedna naredba. Treba obratiti pažnju na to da poslije "then" nema znaka ";", jer to nije završetak naredbe.
Ako je potrebno izvršiti više naredbi u slučaju ispunjenja uslova, koristi se blok naredbi, definisan naredbama "begin" i "end".

Sintaksa 2:

```
if izraz then
  begin
    naredba_1;
    naredba_2
  end;
```

Za slučaj kad treba izvršiti jednu naredbu kad je uslov "true", a drugu kad je uslov "false", koristi se:

Sintaksa 3:

```
if izraz then
  naredba_ako_je_izraz_true
else
  naredba_ako_je_izraz_false;
```

Prije naredbe "else" se također ne stavlja znak ";", jer bi u suprotnom to značilo da se radi o prostom "if" iskazu (kao što je prikazano u sintaksi 1).

Primjer:

```
Program primjer_za_if;
var
  x : integer;
begin
  writeln ('Unesite neki broj');
  readln (x);
  if x mod 2 = 0 then
    writeln ('Broj ', x, ' je paran')
  else
    begin
      write ('Broj ', x);
      writeln (' je neparan');
    end;
  readln;
end.
```

Ako je potrebno izvršiti višestruki izbor, on se može realizovati pomoću naredbe "if":

Primjer (višestruki izbor sa "if"):

```
if (b = 1) or (b = 7) or (b = 2037) or (b = 5) then
      naredba1
   else
      naredba2;
```

Isti primjer se može uraditi koristeći naredbu "case":

Primjer (višestruki izbor sa "case"):

```
case b of
   1,7,2037,5: naredba1;
   otherwise   naredba2
end;
```

Opšta forma iskaza "case" je:

Sintaksa:

```
case selektor of
   vrijednost1 : naredba1;
   vrijednost2 : naredba2;
   ...
   vrijednostn : naredban;
   othervise      naredba
end;
```

Iskaz "othervise" se ne mora navoditi, a u većini prevodilaca se u istu svrhu može koristiti i iskaz "else".

29. FOR, DO, REPEAT-UNTIL, WHILE

Za petlje kod kojih se unaprijed zna broj ponavljanja, koristi se naredba "for".

Sintaksa:

```
for indeks := pocetak to kraj do
   naredba;
```

Varijabla "indeks" se mora prethodno deklarisati, i njena vrijednost se može koristiti unutar petlje (ali se unutar petlje ne smije mijenjati njena vrijednost).

Primjer:

```
Program primjer_za_for;
var
  i : integer;
begin
  for i := 1 to 5 do
   writeln (i);
  readln;
end.
```

Početna vrijednost indeksa **mora biti manja** od krajnje, inače se petlja neće izvršiti. Ako se vrijednost indeksa treba smanjivati, umjesto riječi "to" se koristi riječ "downto".

Primjer:

```
Program odbrojavanje;
var
  i : integer;
begin
  for i := 10 downto 1 do
   write (i:3);
  readln;
end.
```

Rezultat:

```
 10  9  8  7  6  5  4  3  2  1
```

Ovdje treba obratiti pažnju i na korištenje naredbi "write", kada se vrijednosti ispisuju u jednom redu i naredbe "writeln", koja ispisuje vrijednosti jednu ispod druge.

Umjesto jedne naredbe, može se koristiti blok naredbi ograničen naredbama "begin" i "end".

Sintaksa:

```
for indeks := pocetak to kraj do
  begin
    naredba1;
    naredba2;
  end;
```

Primjer:

```
Program prosjek;
var
  i : integer;
  x, p : real;
begin
  p := 0;
  for i := 1 to 5 do
    begin
      writeln ('Unesite broj');
      readln (x);
      p := p + x;
    end;
  writeln (p/5 : 8 : 2);
  readln;
end.
```

U Pascalu "for" petlja može imati korak samo "1" (kad se koristi "to") ili "-1" (kad se koristi "downto").

Za petlje koje se ponavljaju sve dok je neki uslov zadovoljen, koristi se naredba "while".

Sintaksa:

```
while uslov do
  naredba;
```

Umjesto jedne naredbe, može se koristiti i blok naredbi ograničen naredbama "begin" i "end".

Sintaksa:

```
begin
  naredba1;
  naredba2;
end;
```

Primjer:

```
Program petlja_while;
var
  x : integer;
begin
  x := 1;
  while x < 6 do
    begin
      write (x : 2);
      x := x + 1;
    end;
  readln;
end.
```

Rezultat:

```
 1 2 3 4 5
```

Petlja "while" ispitivanje istinitosti uslova vrši prije nego što izvrši korak petlje. To znači da se naredbe iz petlje nikad neće izvršiti ako uslov nikad ne bude zadovoljen. Ako je potrebno testiranje izvršiti nakon izvršenja koraka, koristi se naredba "repeat". To se koristi u slučaju da naredbe iz petlje treba izvršiti barem jedanput, čak i ako uslov nikad ne bude ispunjen.

Sintaksa:

```
repeat
  naredba1;
  naredba2
until uslov;
```

U naredbu "repeat" ugrađene su oznake početka i kraja bloku naredbi, tako da nije potrebno koristiti "begin" i "end" za označavanje početka i kraja bloka.
U narednom primjeru prikazan je zadatak za pronalaženje prvih 10 članova Fibonaccijevog niza (svaki član niza je zbir dva prethodna).

Primjer (Fibonaccijev niz):

```
program Fibonacci;
var
   Fibo1, Fibo2 : integer;
   temp : integer;
   n : integer;
begin
   writeln ('Prvih 10 Fibonaccijevih brojeva su:');
   n := 0;
   Fibo1 := 0;
   Fibo2 := 1;
   repeat
      write (Fibo2:5);
      temp := Fibo2;
      Fibo2 := Fibo1 + Fibo2;
      Fibo1 := Temp;
      n := n + 1
   until n = 10;
   writeln;
   readln
end.
```

Rezultat:

```
Prvih 10 Fibonaccijevih brojeva su:
    1    1    2    3    5    8   13   21   34   55
```

29.1. Polja

Jednodimenzionalna polja (nizovi) se deklarišu u dva koraka, jer su to korisnički definisani tipovi podataka.

Sintaksa:

```
type
  ImeTipa = array [OpsegIndeksa] of TipElementa;
var
  ImeNiza : ImeTipa;
```

"ImeTipa" je naziv korisnički definisanog tipa podataka, a "OpsegIndeksa" predstavlja najmanji i najveći indeks, odvojene sa dvije tačke "..". "TipElementa" je tip podataka koji će moći imati pojedini elementi niza. U narednom primjeru se definiše korisnički tip podataka pod nazivom "niz".

Primjer:

```
program Niz;
  type
    niz = array [1..10] of integer;
  var
    m: niz;
    i : integer;
  begin
    for i := 1 to 10 do
      begin
        writeln ('Unesite broj');
        readln (m[i]);
      end;
    for i := 1 to 10 do
      begin
        write (m[i]: 3);
      end;
      readln
  end.
```

Drugi način deklaracije niza je bez navođenja naziva korisničkog tipa podatka.

Sintaksa:

```
ImeNiza : array [OpsegIndeksa] of TipElementa;
```

Primjer (kraća deklaracija niza "m" od 10 elemenata tipa integer):

```
var
  m: array [1..10] of integer;
```

Dvodimenzionalna polja (matrice) se delarišu na sličan način, tako što se opseg indeksa redova i kolona odvoje zarezom.

Sintaksa:

```
ImeMatrice : array [Redovi, Kolone] of TipElementa;
```

Primjer (realna matrice "x" sa 3 reda i 4 kolone):

```
program Matrica;
var
  x: array [1..3, 1..4] of real;
  i, j : integer;
begin
  for i := 1 to 3 do
    for j := 1 to 4 do
      begin
        write ('Unesite broj x(',i:1,',',j:1,'): ');
        readln (x[i,j]);
      end;
  for i := 1 to 3 do
    begin
      for j := 1 to 4 do
        begin
          write (x[i,j]:6:2);
        end;
      writeln;
    end;
  readln
end.
```

30. Potprogrami, funkcije

Dijelovi programa koji se često ponavljaju se mogu izdvojiti u poseban potprogram (proceduru) pomoću naredbe "procedure". Struktura potprograma je ista kao kod glavnog programa:

Struktura potprograma:

```
procedure Naziv_potprograma;
   const
      (* Konstante *)
   var
      (* Varijable *)
   begin
      (* Tijelo potprograma *)
   end;
```

Potprogrami se pozivaju navođenjem njihovog imena.
Ako potprogram koristi parametre (argumente), isti se navode u zagradi iza naziva potprograma.

Sintaksa:

```
procedure Naziv (lista_argumenata);
```

Lista argumenata se sastoji od jednog ili više identifikatora, odvojenih zarezima (ako su istog tipa), odnosno znakom ";" ako su različitog tipa.
Parametri se mogu prosljeđivati potprogramu po vrijednosti (*call by value*) ili po referenci (*call by reference*). Argumentima koji se prosljeđuju po referenci prethodi riječ VAR. Prosljeđivanje po vrijednosti odgovara pravljenju kopije varijable, a zatim prosljeđivanje kopije potprogramu. Potprogram koristi kopiju i na kraju je zanemaruje. Tako originalna varijabla ostaje nepromijenjena. Poziv po referenci ne pravi kopiju, nego potprogram može da promijeni vrijednost proslijeđene varijable i da je tako promijenjenu vrati programu koji je pozvao potprogram.

Primjer:

```
program Primjer_sa_potprogramom;
  var
    a, b, c : integer; pr : real;
  procedure prosjek (VAR p : real; x, y, z : integer);
    begin
      p := (x + y + z)/3;
    end;
  begin
    writeln ('Unesite 3 broja');
    readln (a, b, c);
    prosjek (pr, a, b, c);
    writeln (pr:7:3);
  end.
```

Osim procedura, mogu se koristiti i funkcije, koje u program koji ih je pozvao uvijek vraćaju samo jednu vrijednost, preko svog imena.

Sintaksa:

```
function Naziv (lista_argumenata) : tip_podataka;
```

Primjer:

```
program Primjer;
  var
    m : integer;
  function faktorijel (n : longint) : longint;
    var
      x, i : integer;
    begin
      x := 1;
      for i := 1 to n do
        x := x * i;
      faktorijel := x;
    end;
  begin
    writeln ('Unesite broj');
    readln (m);
    writeln (faktorijel(m));
  end.
```

U prikazanom primjeru se koristi varijabla "x" jer se naziv funkcije (u ovom slučaju "faktorijel") ne smije naći desno od operatora dodjeljivanja vrijednosti ":=".

30.1. Rad sa datotekama

Ako se podaci ne unose sa tastature, nego se čitaju iz tekst datoteke, prvo treba deklarisati varijablu koja će predstavljati tu datoteku.

Sintaksa:

```
var
  ImeVarijableDatoteke : text;
```

Kada je ta varijabla deklarisana, datoteka se otvara za čitanje naredbom "reset".

Sintaksa:

```
reset (ImeVarijableDatoteke, 'naziv_datoteke.ekstenzija');
```

Nakon toga, naredbama "read" ili "readln" se podaci iz datoteke čitaju i dodjeljuju se varijablama navedenim kao argumenti naredbi "read", odnosno "readln".

Sintaksa:

```
read (ImeVarijableDatoteke, lista_varijabli);
readln (ImeVarijableDatoteke, lista_varijabli);
```

Naredbom "close" se datoteka zatvara.

Sintaksa:

```
close (ImeVarijableDatoteke);
```

Primjer:

```
program Datoteka;
var
  podaci : text;
  x : integer;
begin
  reset (podaci, 'c:\ulaz.txt');
  read (podaci, x);
  writeln ('x=',x:5);
  close (podaci);
  readln
end.
```

U prikazanom primjeru iz datoteke se čita samo prvi broj koji je u nju zapisan i pohranjuje se u varijablu "x". Funkcija "eof" – *end of file* (kraj datoteke) se koristi za testiranje da li se prilikom čitanja došlo do kraja datoteke.

Primjer:

```
program Datoteke;
var
  podaci : text;
  x : integer;
begin
  reset (podaci, 'c:\ulaz.txt');
  while not eof (podaci) do
   begin
     read (podaci, x);
     writeln ('x=',x:5);
   end;
  close (podaci);
  readln
end.
```

Za zapisivanje u datoteku, procedura je slična, samo što se umjesto naredbe "reset" koristi naredba "rewrite", koja kreira novu datoteku i otvara je za zapisivanje. Ako datoteka sa istim imenom već postoji, svi podaci iz nje se brišu.

Sintaksa:

```
rewrite (ImeVarijableDatoteke, 'naziv_datoteke.ekstenzija');
```

Za zapisivanje se koriste naredbe "write" i "writeln", analogno kao kod ispisivanja na ekran, ali sa prvim argumentom koji predstavlja varijablu deklarisanu kao datoteka.

Sintaksa:

```
write (ImeVarijableDatoteke, lista_varijabli);
writeln (ImeVarijableDatoteke, lista_varijabli);
```

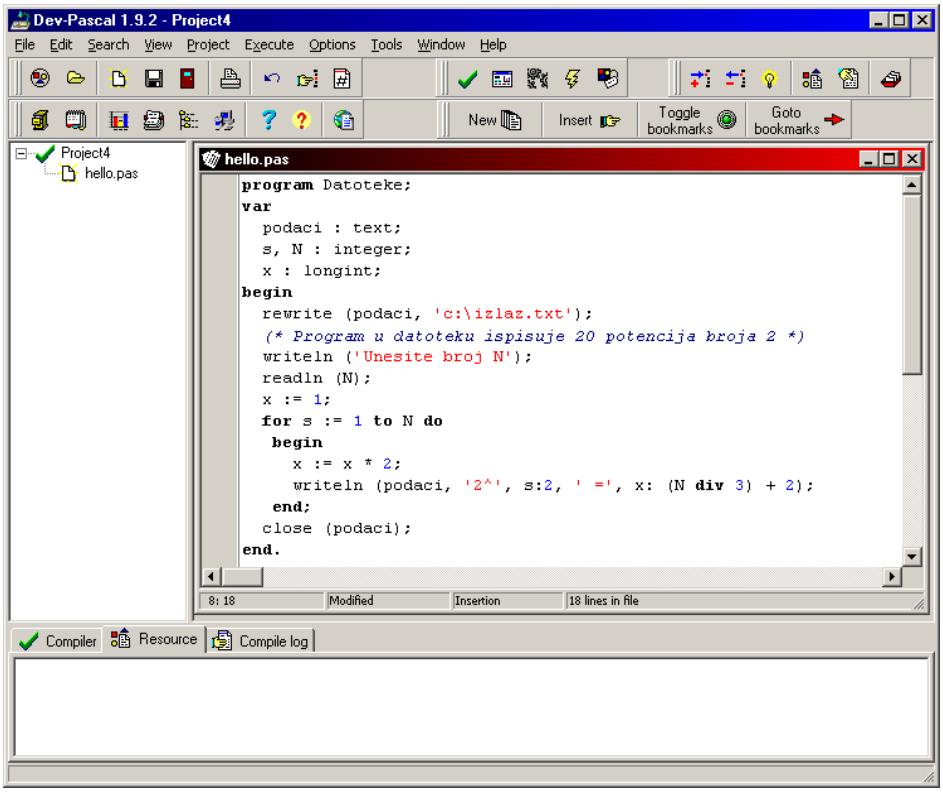
Primjer:

```
program Datoteke;
var
  podaci : text;
  s, N : integer;
  x : longint;
begin
  rewrite (podaci, 'c:\izlaz.txt');
  (* Program u datoteku ispisuje N potencija broja 2 *)
  writeln ('Unesite broj N');
  readln (N);
  x := 1;
  for s := 1 to N do
   begin
     x := x * 2;
     writeln (podaci, '2^', s:2, ' =', x: (N div 3) + 2));
   end;
  close (podaci);
end.
```

Rezultat (sadržaj datoteke "c:\izlaz.txt" ako se unese N=11):

```
2^ 1 =    2
2^ 2 =    4
2^ 3 =    8
2^ 4 =   16
2^ 5 =   32
2^ 6 =   64
2^ 7 =  128
2^ 8 =  256
2^ 9 =  512
2^10 = 1024
2^11 = 2048
```

U ovom primjeru se vidi da se broj cifara varijable "x" koje se zapisuju u datoteku može unijeti i kao izraz (u ovom slučaju N/3+2), tako da za N=11, prikazuje se 11/3+2 = 3+2 = 5 cifara.



Integrisano razvojno okruženje "GNU Pascal"