UNIVERZITET U KRAGUJEVCU
PRIRODNO MATEMATIČKI FAKULTET
INSTITUT ZA FIZIKU

Dragoslav Nikezić

FORTRAN 90 I VISUAL FORTRAN

Kragujevac, 2004.

SADRŽAJ

I

U V O D

U poslednjih 30 do 40 godina razvijen je i korišćen veći broj različitih programskih jezika - mnogi od njih već su u potpunosti zaboravljeni. Istovremeno sa potiskivanjem starih, pojavljuju se i novi jezici za specijalne namene kao što je internet programiranje i sl. Medjutim, najstariji programski jezik slavne prošlosti – FORTRAN- još uvek živi, i to je i danas dobar izbor za naučne i tehničke primene.

Programski jezici trpe konkurenciju mnogih softverskih paketa namenjenih za specijalne svrhe; to su razni programi orijentisani ka matematičkim i numeričkim izračunavanjima, (kao na primer MATHEMATICA, MATHLAB ili MATCAD i dr), programi za obradu tabela i unakrsna množenja, programi za obradu baza podataka, editori teksta, grafike i dr. Medjutim, ako se želi sopstveni razvoj, ili se želi pravljenje originalnih softverskih paketa u naučnom ili inžinjersko - tehničkom pogledu, potrebno je znanje jednog od viših programskih jezika kakav jeste FORTRAN.

FORTRAN je jedan od nasjtarijih programskih jezika. Više od dvadeset godina, to je bio jedini programski jezik u širokoj upotrebi. U FORTRAN-u su razvijene čitave biblioteke, kakve ne postoje ni u jednom drugom programskom jeziku, i to je jedan, mada ne i jedini, od razloga vitalnosti ovog jezika. Istovremeno, ta karakteristika je vremenom postala otežavajuća okolnost u daljem razvoju jezika. Mnoge naredbe, koje su svojevremeno bile glavno obeležje FORTRAN jezika i po kojima se ovaj jezik prepoznavao, su sada zastarele. Verovatno najpoznati primer je vrlo kritikovana naredba GO TO *n* (predji na naredbu sa brojem *n*). Primena programa sa ovom i sličnim naredbama je omogućena i u novijim verzijama FORTRAN-a. Nove verzije FORTRAN-a prihvataju sve stare programe, ali su takve naredbe označene kao zastarele »*obsolescent*«. Takve naredbe su kandidati za eliminisanje u sledećim verzijama ovog jezika, i programeri ne treba da ih koriste u novo razvijenim programima.

I.1. ISTORIJAT FORTRAN -A

Prvi FORTRAN-i razvijeni su početkom pedesetih godina i bili su veliki napredak u odnosu na mašinski jezik u kome se do tada programiralo. Sama reč FORTRAN je akronim od dve engleske reči ¨**for**mula **tran**slation¨, formula za prevodjenje, i pod tim se podrazumevalo prevodjenje matematičkih formula na jezik razumljiv čoveku, a koji istovremeno može da ¨razume¨ i računar.

Krajem pedesetih i početkom šezdesetih godina prošlog veka, FORTRAN se znatno proširio i pojavilo se više raznih ¨dijalekata¨ ovog jezika. Programi pisani u raznim dijalektima nisu bili prenosivi, tj. radili su samo na računaru na kome je taj dijalekt FORTRAN-a instaliran, a ne i sa drugom verzijama jezika. Da bi se ovo izbeglo i postigla prenosivost programa usvojena je standardna verzija FORTRAN-a, takozvani FORTRAN IV. Standard je definisao Američki Nacionalni Institut za Standarde (American National Standards Institute) poznat kao ANSI. Ovo je u opšte bio prvi standard za programske jezike. Ova verzija FORTRAN-a je kasnije preimenovana u FORTRAN 66. Medjutim, FORTRAN 66 je imao znatnih ograničenja tako da su dijalekti FORTRAN-a nastavili da se razvijaju i dalje. Tek 1978 god. usvojen je novi standard nazvan FORTRAN 77 koji je ponudio mnoga poboljšanja. Medjutim, i ova verzija

FORTRAN-a je vremenom znatno zastarela i zaostala za drugim modernijim programskim jezicima, kao što su PASKAL, C, C^{++} i dr. Znatna unapredjenja FORTAN-a obavila je kompanija Microsoft početkom devedesetih godina nakon čega je usvojen novi standard, tzv. FORTRAN 90. Pored FORTRAN-a Microsoft je napravio takozvani RAZVOJNI STUDIO, Microsoft Developer Studio, o čemu će kasnije biti reči. Izmene učinjene u FORTRAN-u 90 su u znatnoj meri radikalne, tako da su ovaj jezik učinile modernijim i dovele ga u red savremenih viših programskih jezika strukturnog programiranja.

Dalji razvoj ovog jezika preuzela je prvo kompanija DIGITAL, a zatim i COMPAQ koja je razvila najnoviju verziju Visual FORTRAN . Takodje razvijen je i verzij 2005 VISUAL FORTRAN. Konačno, dalji razvoj ovog jezika preuzela je poznata firma INTEL, koja priprema nove verzije sa objektno orijentisanim programiranjem.

I.2. ZAŠTO FORTRAN ?

Često se čuje pitanje, zašto FORTRAN, zašto ne C, C^{++} ili neki drugi programski jezik. Autor ovog teksta nudi više odgovora na ovo pitanje. FORTRAN je *jednostavniji i lakši za učenje* od C jezika. Iz tog razloga treba početi sa FORTRAN-om, a kasnije, ako potrebe zahtevaju preći na druge jezike. U sintaksi FORTRAN-a nema nepotrebnih i zbunjujućih karaktera. Drugi razlog je što su kompajleri razvijeni za FORTRAN, takoreći dovedeni do perfekcije, *numerička izračunavanja se najbrže i najlakše* obavljaju u FORTRAN-u. FORTRAN je još uvek *najmoćnije sredstvo* za numerička računanja. Skoro svi značajniji svetski softwari sa numeričkom osnovom sa tradicijom dužom od desetak godina su pisani u FORTRAN-u. Pored toga, postoje *ogromne biblioteke* programa razvijene u FORTRAN-u za raznorazne svrhe. Ovakva podrška nije raspoloživa za druge programske jezike. Najpoznatije fortranske biblioteke namenjene numeričkim izračunavanjima su IMSL (International Mathematical and Statistical Libraries), NAG (Numerical Algorithmus) i Numerical Recepis. Ove biblioteke se sastoje od više stotina programa, (poneke i više hiljada) koje se pozivaju po potrebi. Iako su ove biblioteke razvijene u FORTRAN-u mogu se pozivati i iz drugih programskih jezika (ako je ostavljena takva mogućnost u tim programskim jezicima). Takodje, u FORTRAN-u 90 omogućeno je pozivanje nefortranskih procedura.

I.3.PREDSTAVLJANJE BROJEVA-PODATAKA U RAČUNARIMA I BROJČANI SISTEMI

I.3.1. Brojčani sistemi

Brojevi se u računarima predstavljaju pomoću jedinica i nula korišćenjem binarnog brojčanog sistema. Pored binarnog, u računarstvu se još koriste oktalni, decimalni i heksadecimalni brojčani sistemi.

Osnova brojčanog sistema je broj različitih simbola, (cifara) koje se koriste za predstavljanje brojeva. Binarni brojčani sistem koristi dve cifre za predstavljanje svih ostalih brojeva, i to nulu i jedinicu (0,1). Nula i jedinica u binarnom brojčanom sistemu se još mogu tretirati kao tačno ili netačno. Oktalni sistem koristi osam cifara, 0-7, a heksadecimalni 16 cifara, 0-9,A,B,C,D,E,F. U Tabeli 1 data je veza izmedju brojeva u binarnom, oktalnom heksadecimalnom i decimalnom sistemu.

Tabela 1. Brojčani sistemi

Decimalni	Binarni	Oktalni	Heksadecimalni
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	100	10

Ovi brojčani sistemi su pozicioni, tj. vrednost jedne cifre zavisi od položaja na kome se ta cifra nalazi. Radi jasnoće, osnovu u kojoj je neki broj napisan, pišemo kao desni donji indeks pored tog broja. Na primer broj 238_{10} označava broj 238 u decimalnom brojčanom sistemu. Vrednost broja 238 se može dobiti preko celobrojnog eksponenta broja 10 na sledeći način.

$$238_{10} = 2 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0$$

Slično ovome dobija se vrednost broja u binarnom sistemu, ali je osnova broj 2. Na primer, broj 101_2 je

$$101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 1 = 5_{10}$$

Prethodna jednačina daje primer konverzije broja iz binarnog u decimalni brojčani sistem. Brojevi u ostalim brojčanim sistemima se mogu predstaviti i konvertovati u decimalne na sličan način.

Konverzija binarnog u oktalni se bavlja tako što se binarne cifre grupišu u grupe po tri (počev s desna u levo). Dobijene grupe se direktno prevode u oktalne brojeve. Na primer prevodjenje broja 1110111001_2 u oktalni obavlja se na sledeći način:

$$1|110\|111\|001| = 1671_8 = 1 \cdot 8^3 + 6 \cdot 8^2 + 7 \cdot 8^1 + 1 \cdot 8^0 = 512 + 384 + 56 + 1 = 953_{10}$$

Prevodjenje iz binarnog u heksadecimalni se obavlja tako što se binarne cifre grupišu u grupe po 4 (počev s desna) a zatim se direktno prevode u heksadecimalne brojeve. Prethodni broj bi se preveo u heksadecimalni

$$11|1011\|1001| = 3B9_{16} = 3 \cdot 16^2 + 11 \cdot 16^1 + 9 \cdot 16^0 = 3 \cdot 256 + 176 + 9 = 953_{10}$$

Iz prethodne Tabele vidi se da je najveći broj koji se može predstaviti sa n binarnih cifara jednak 2^n-1. Na primer $11_2=3_{10}$; $111_2=7_{10}$; $1111_2=15_{10}$.

Aritmetičke operacije se mogu definisati u raznim brojčanim sistemima. Analogno sabiranju u decimalnom b.s., sabiranje u binarnom sistemu se obavlja prema pravilima

0+0=0

0+1=1+0=1

$1+1=10_2$

Oduzimanje se svodi na sabiranje sa negativnim brojem i nema potrebe za posebnim definicijama. Množenje u binarnom b.s., se obavlja prema pravilima

0*0=0

0*1=1*0=0

1*1=1.

Na primer proizvod brojeva 5 i 3 =15 u binarnom zapisu je 101 * 11 i računa se kao

$$101*11=\begin{array}{r}101\\101\\\hline 1111\end{array}$$

Deljenje se svodi na množenje recipročnom vrednošću drugog broja u proizvodu.
Sve ostale računske operacije se mogu izraziti preko sabiranja.

I.3.2. Predstavljanje brojeva u računarima.

Memorisanje brojeva u računarima je omogućeno postojanjem bistabilnih elemenata, tj. elementima koji imaju dva stanja. To može biti magnetni elemenat, koji je namagnetisan ili ne, ili poluprovodnički elementi na kojima postoji napon/struja ili ne (poznati kao flip-flop) i sl. Jedno stanje predstavlja jedinicu, a drugo nulu. Ovakav elemenat, koja može da pamti jedinicu ili nulu naziva se ***bit***. Grupacija od 8 bit-a čini jedan bajt (byte), B. Grupa od 1024 bajta je jedan kilobajt (kB). Ovo je verovatno jedini slučaj gde prefiks *kilo* ne predstavlja umnožak sa 1000. S druge strane MB (megabajt) takodje predstavlja 1024 kB. Gigabajt je 10^6 bajta. Memorije diskova se obično izražavaju u GB, a RAM memorije su još uvek na nivou nekoliko stotina MB (naravno ima izuzetaka).

Brojevi, ali i ostale potrebne informacije se prvo prevode u binarni brojčani sistem, a zatim se pamte u obliku nula i jedinica. Memorija računara se sastoji od memorijskih registara. Današnji računari koriste memorijske registre sa po 32 bita, tj, 4 bajta. To su *trideset dvobitni računari*. Do pre nekoliko godina korišćeni su 16 bitni računari, a ranije i osmobitni. Pojavljuju se računari sa 64 bita. Dalje izlaganje se odnosi na 32 bitne računare.

Prvi bit od ukupno 32 se koristi za znak broja; ako je prvi znak 0 onda je broj pozitivan (+), a jedinica predstavlja minus (-). Najveći *ceo* pozitivan broj koji se može smestiti u 32 bitni registar se piše sa 31 jedinicom te je zato on jednak

$$2^{31}-1=2147483647$$

Najnegativniji *ceo* broj je

$$-2^{31} = -2147483648$$

Iako se prethodni brojevi čine velikim, u računanju je relativno lako prevazići navedene vrednosti. U tom slučaju računar pojavljuje grešku.

Pored celih brojeva, naravno, koriste se i racionalni brojevi. Takvi brojevi imaju decimalni zarez. U engleskom pravopisu koristi se decimalna tačka umesto zareza, te je u celokupnom računarstvu u upotrebi decimalna tačka. Tako zapis, 1.2 predstavlja *jedan zarez dva*. Kod 32 bitnih računara decimalna tačka se nalazi izmedju 16^{te} i 17^{te} pozicije. Decimalni broj 5_{10} se u memorijskom registru predstavlja kao

$5.0_{10}=$ $00000000000001010000000000000000_2$

Za decimalnu tačku se ne ostavlja posebno mesti niti se ona unosi u registar na bilo koji način. Podrazumeva se da je ona izmedju 16 i 17 pozicije. Broj 5.5 se registruje kao

$5.5_{10}=$ $00000000000001011000000000000000_2$

Brojevi $_{10}$ i $_2$ u donjim indeksima u prethodnim zapisima označavaju osnovu brojčanog sistema.

Prethodni zapis se još naziva i *predstavljanje brojeva sa fisknim zarezom*, (fixed point). Ovakav zapis ima nekih nepovoljnosti kao što je relativno mali opseg brojeva. Pored fiksnog zareza moguće je brojeve predstaviti i u obliku »pokretnog zareza« (*floating point*). U tom slučaju se koristi eksponent broja 10. Na primer brojevi $0.456 \cdot 10^{21}$ i $0.301 \cdot 10^{-19}$ su brojevi zapisani u obliku »pokretnog zareza«. Normalizovni oblik ovih brojeva je kada je predeksponencijalni deo izmedju 0 i 1 (još se naziva i *mantisa*). Brojevi u pokretnom zarezu se pamte u memorijskom registru na sledeći način. 32 bitni memorijski registar se sastoji od 4 bajta. Prvi bajt se koristi za smeštaj izložioca broja deset, a u preostala tri bajta smešta se mantisa. Od rezervisanih osam bita, sedam se koristi za smeštaj izložioca, a osmi za znak. Tako je najveći mogući pozitivni izložilac dat kao $2^7 - 1 = 127$, a najnegativniji je $-2^7 = -128$.

Mantisa se smešta u preostala tri bajta što je ukupno 24 bita. Jedan bit se rezerviše za znak mantise.

Ukupan broj značajnih cifara (u decimalnom brojnom sistemu) koji se predstavlja u računaru u obliku pokretnog zareza je 6 (broj cifara iza zareza). Isto vači i za broj u obliku fiksnog zareza. Jasno je da je 6 značajnih cifara često nedovoljno, u mnogoborojnim primenama potrebna je znatno veća tačnost. Zbog toga je u FORTRAN-u ostavljena mogućnost da se brojevi pamte u dva memorijska registra. To se naziva dvostruka tačnost i broj značajnih cifara je 15. Dva memorijska registra imaju ukupno 8 bajta. Jedan bajt se koristi za smeštaj izložioca, a preostalih 7 za smeštaj mantise. Opseg brojeva u dvostrukoj tačnosti je znatno veći nego u običnoj tačnosti. Predstavljanje brojeva može biti različito u različitim računarima te se i opseg vrednosti brojeva menja od računara do računara čak i pri istoj dužini memorijskog registra.

1.4. RAD U MICROSOFT OVOM RAZVOJNOM STUDIJU

U ranijim verzijama FORTRAN-a osnovna jedinica je bio program. Sada je situacija u znatnoj meri promenjena i kao glavna jedinica ističe se projekat. Izvorni program, pripadajuće biblioteke i sve ostale potrebne fajle koje čine jednu celinu, kao i specifikacije koje prate dokumente, organizuju se u obliku projekta. Po startovanju FORTRAN-a 90 pojavljuje se Microsoft Developer Studio (u daljem tekstu koristiće se skraćenica MDS, a može se prevesti i kao Majkrosoftov Razvojni Studio), okruženje u kome se obavlja celokupan rad na razvoju, usavršavanju i ispitivanju programa. Izgled ekrana po startovanju MDS-a je prikazan na slici 1. Ekran je podeljen u tri dela, - okvira. Levi deo ekrana prikazuje projekat i fajlove koji mu pripadaju i koji su trenutno aktivne. Desni deo ekrana prikazuje sadržaj fajle koja se dvostrukim klikom miša bira u levom delu ekrana. U donjem delu ekrana izdaju se informacije pri kompiliranju i linkovanju programa. Moguće je neke fajlove učiniti neaktivnim, i one se onda ne prikazuju u levom delu ekrana. Po potrebi te fajle se mogu aktivirati. O ovome će biti više reči kasnije.

Sam MDS je u priličnoj meri kompleksan i za njegov detaljan opis bila bi potrebna cela jedna knjiga. Ovde će biti date samo osnovne instrukcije za rad u MDS-u.

Programe je potrebno prevesti sa FORTRAN-a, koji je razumljiv i prihvatljiv za čoveka, na mašinski jezik koji je sa druge strane razumljiv za računar. To prevodjenje se naziva ¨kompiliranje¨. Pri kompliranju, vrši se dijagnostika programa, tj. otkrivanje postojećih greški u programu. U ovoj fazi otkrivaju se samo greške (ERROR) koje su formalnog karaktera, kao što su nepravilno napisane naredbe, pogrešno zapisani matematički izrazi, skokovi na nepostojeće naredbe i sl. Poruke o greškama u programu štampaju se u donjem delu ekrana radnog prostora projekta. Daju se informacije o greški, tip greške, broj greške i red u kome je greška otkrivena. Dvostrukim klikom miša na informaciju o greški u donjem delu ekrana, doći će do pozicioniranja kursora na red u programu u kome se nalazi greška. Moguć je slučaj postojanja velikog broja greški u nekom programu. MDS prijavljuje najviše do 56 greški, a onda prekida dalje dijagnosticiranje programa. Da bi se program kompilirao i dalje linkovao i izvršavao nužno je otkloniti sve prijavljene greške.

Pored informacija o greškama MDS izdaje i upozorenja (WARNING). To su takodje greške, ali one nisu takvog karaktera da sprečavaju kompiliranje, linkovanje i izvršavanje programa. Program bi se izvršavao, ali najverovatnije rezultat ne bi bio korektan. Na primer upozorenje se izdaje ako se u nekom matematičkom izrazu pojavljuje veličina čija vrednost nije ranije bila definisana. Ako se ovo ne otkloni, onda se u računu uzima da je ta veličina jednaka nuli. Preporuka je da se otklone uzroci svih upozorenja.

Nakon uspešnog prevodjenja na mašinski jezik, potrebno je u jednu celinu povezati taj prevedeni mašinski kod i odgovarajuće rutine potrebne za izvršavanje programa. Ova procedura se naziva ¨linkovanje¨ što se prevodi kao povezivanje. Kao krajnji rezultat kreira se izvršna verzija programa sa ekstenzijom .EXE. Ove dve procedure, kompliliranje i linkovanje se mogu obaviti jednom operacijom ¨BUILD¨ što se prevodi kao ¨graditi¨. Tako, program se ¨izgradjuje¨ u FORTRAN- u 90.

1.4.1. Definisanje novog projekta

Otvaranje novog projekta obavlja se na sledeći način. Na vrhu ekrana nalazi se spisak standardnih opcija skoro svih WINDOWS aplikacija, File, Edit View, i dr.

1.U File meniju izaberi NEW.
Pojavi se *dijalog box* ¨New¨ sa ponudjenom listom od 8 mogućih stavki. Treba izabrati drugu stavku po redu, Project Workspace.

2.Pojavi se dijalog box ¨New Project Workspace¨. U tekst boxu koji se zove ¨Project Name¨ (ime projekta) treba ukucati ime novog projekta koji se definiše.

3. Sada treba izabrati tip projekta iz liste ¨Project types¨ . Na raspolaganju je ukupno 6 tipova projekta. Ako se obavlja samo račun onda treba izabrati Console application. Ukoliko se u programu pojavljuju i neke grafičke aplikacije treba izabrati Quick Win Application ili Standard Graphics Application.

4. Izabrati platformu projekta iz ponudjene liste. Za sada je omogućen rad jedino pod Win32 platformom, tako da je to već podešeno i korisnik nema potrebe da nešto ovde menja.

5. Moguće birati lokaciju na kojoj se smešta dati projekat. Po default-u to je C:\MSDEV\Projects\.

5. Kliknuti na dugme Create. Ovim se kreira radni prostor novog projekta. Istovremeno se u C:\MSDEV\Projects\ dodaje jedan poddirektorijum sa imenom novog projekta koje je korisnik izabrao. Sve fajle povezane sa datim projektom se smeštaju u taj poddirektorijum.

Ovim je novi projekat definisan. Sada je potrebno ovom projektu dodavati fajle i programe. Ako taj program nije već postojeći treba postupiti na sledeći način:

1. Kliknuti na INSERT (umetnuti).

Pojaviće se lista stavki koje se mogu umetati u projekat. Izaberi ¨Files into Poject¨. Nakon ovoga otvoriće se dijalog box pod imenom ¨Insert Files Into Project¨. Pojavi se box u kome treba ukucati ime fajle. Pri ovome treba voditi računa da ekstenzija fajle mora biti jedna od ekstenzija koje MDS prepoznaje (podržava)- programi koji će se kompilirati treba da imaju jednu od ekstenzija .FOR, .F90 ili .F. Uneti željeno ime sa odgovarajućom ekstenzijom i onda kliknuti na dugme Add. Ako fajla pod tim imenom ne postoji ili je MDS ne nalazi, postavlja se pitanje da li se želi dodati fajla pod tim imenom. Nakon odgovora ¨Yes¨ fajla pod datim imenom se dodaje projektu i njeno ime se pojavljuje u levom okviru (ukoliko se ona ne vidi kliknuti na znak + pored imena projekta u levom okviru). Medjutim fajla još uvek nije formirana. Ovom operacijom samo je dodata referenca u projekat da takva fajla postoji.

Dvostruki klik mišem na ime fajle otvara tu fajlu u desnom okviru ako je fajla postojeća. Ako fajla nije postojeća MDS pita da li se želi stvaranje takve fajle. Odgovorom ¨Yes¨ konačno se otvara fajla u desnom okviru. Ako je u pitanju nova fajla onda je ovaj desni okvir prazan i može se ukucavati novi program.

Ako se projektu dodaje već postojeća fajla onda treba prvo kliknuti na ¨Insert¨ Pojavi se spisak fajli koje se mogu dodati pdojektu. Izabere se željena fajla i klikne se na tipku ¨Add¨. Time je fajla dodata projektu.

Potrebno je napomenuti da je u jednom projektu moguće postojanje samo jednog glavnog fortranskog programa. Ako postoji više od jednog glavnog programa onda MSD ne zna koji program je glavni, a koji ne. U tom slučaju se pri kompiliranju javlja greška. Program se može skinuti sa projekta na sledeći način. Klikne se mišem na taj program i onda se pritisne tipka Delete. Ovim program nije obrisan već je samo deaktiviran. On se i dalje nalazi na disku računara u istom direktorijumu gde i pre i može se preko opcije Insert ponovo da se vrati natrag u projekat. Na ovaj način, mogu u jednom direktorijumu da se nalazi više fortranskih programa, ali samo jedan može da figuriše kao aktivan.

Ovo se ne odnosi na podprojekte, koji ako imaju ekstenziju .F90 mogu da figurišu u levom delu radnog prostora projekta.

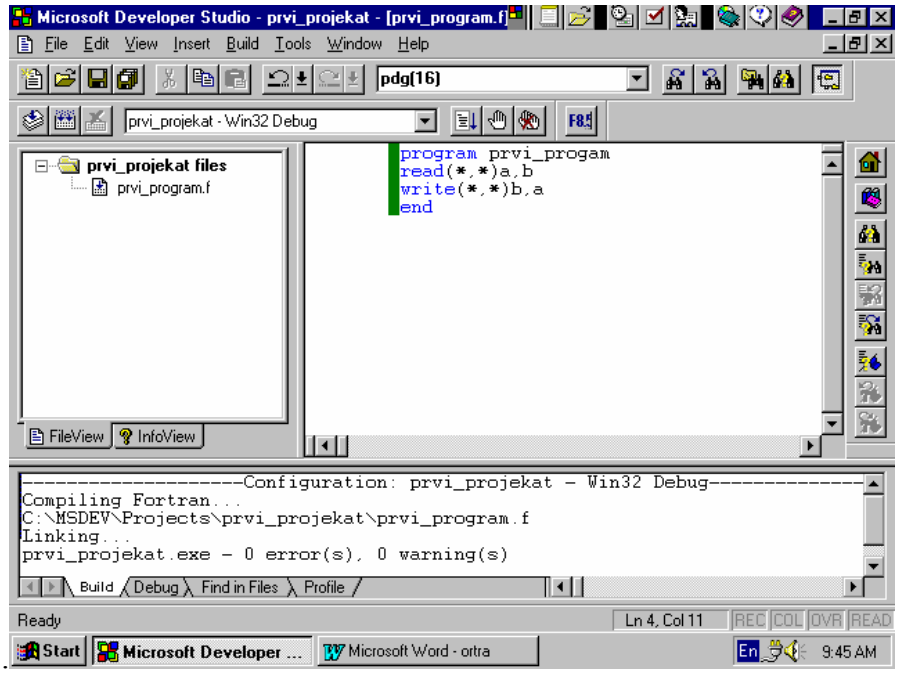
1.4.2. Otvaranje postojećeg projekta

Otvaranje postojećeg projekta se obavlja na sledeći način. Po startovanj MDS-a otvoriće se projekat na kome je poslednje radjeno. Ako se želi drugi projekat taj treba zatvoriti. Iz meni linije izabrati File, i onda Close Workspace. Ovim je taj projekat zatvoren. Ponovo aktivirati meni File, i onda Open Workspace. Pojavi se lista sa imenima postojećih projekata odakle se bira željeni. U dijalog boxu ¨Open Project Workspace¨ treba da se pojavi fajla sa ikonom  . Njena ekstenzija je .mdp (**M**icrosoft **d**evelo**p**er) i treba dva puta kliknuti na nju. Ovim je projekat otvoren.

Opisana procedura se mora dosledno poštovati. Skretanje od te procedure izazvaće probleme pri kompiliranju i/ili izvršavanju programa. Slična procedura postoji i u Visual Fortranu.

1.4.3.Kompiliranje, linkovanje i izvršavanje programa

Kompiliranje se obavlja naredbom Compile iz menija Build. Ako nema formalnih greški u programu onda se može preći na linkovanje programa naredbom Build (ime fajle.exe) takodje u meniju Build. Pri linkovanju pokupe se sve potrebne rutine za izvršavanje programa i formira se izvršna fajla. MDS kreira novi poddirektoriju pod imenom DEBUG gde se smešta izvršna fajla. DEBUG je poddirektorijum u već otvorenom direktorijumu radnog prostora projekta.



Slika 1. Izgled ekrana po startovanju MDS-a i otvaranju nekog projekta

Izmedju levog i donjeg okna nalaze se dva TAB-a, FileView Tab, i , InfoView Tab. Klikom na InfoView Tab otvara se Help.

1.4.4. Rad u Visual Developer Studiju

Rad u Visual Fortranu je sličan kao i u MSDEV, ali postoje i neke razlike. Po startovanju Visual Fortrana pojavljuje se sličan ekran kao i kod MSDEV a. To je u suštini isti developer studio koji se koristi i kod C^{++}. Definisanje novog projekta se obavlja na sledeći način. Klikne se na FILE, a zatim na NEW. Otvori se dijalog box u kome se na vrhu nalazi niz tabova gde se izabere novi projekat (Project). Pored novog projekta može se birati nova file, novi radni prostor ili ostale vrste fajli kao što su Excel, Word dokumenti i sl. Ako se izabere novi projekat onda se u prozoru na desnoj strani ukucava ime projekta, a u prozoru levo bira se tip projekta. Za običan račun bira se konzolna aplikacija. Po definisanju projekta treba izabrati Empty Project ili Simple Project. Ako je izabran Simple Project onda se automatski kreira jedna izvorna fajla sa nazivom istim kao i naziv projekta i sa ekstenzijom .F90.

Dodavanje fajli projektu se obavlja preko padajućeg menija Project, gde se otvara mogućnost Add to Project. Kompiliranje, linkovanje i izvršavanje programa se obavlja takodje preko Build opcije padajućeg menija (kao i kod MSDEV).

II

ULAZAK U FORTRAN

II.1.PRAVILA UNOŠENJA PROGRAMA

Fortranski program se sastoji od niza naredbi. One se unose prema odredjenim pravilima. U jednom redu može se uneti do 132 karaktera- ovde se uračunavaju i prazna mesta, tzv.– blanko. U principu, prazno mesto nema značaja, ali ima dosta situacija kada ono nije dozvoljeno ili kada je ono obavezno.

U jednom redu može se uneti više od jedne naredbe; u tom slučaju one se odvajaju znakom tačka- zarez (;) , što je dato primerom:

A=B*C; C=D/A; X=X**2

U slučaju da jedna naredba ne može da stane u jednu liniju programa onda se ona može nastaviti u sledeću liniju programa. U tom slučaju se na kraj linije koja se nastavlja postavlja znak & (ampersend). Maksimalna dužina jedne naredbe u programu je 2640 karaktera (uključujući i prazna mesta). Najveći broj linija u kojima se može smestiti jedna naredba je 40.

Program je potrebno komentarisati da bi se poboljšala njegova čitljivost i razumljivost. Za komentarisanje programa koristi se znak uzvičnika (!). Sve karaktere iza znaka ! program tretira kao komentar i ne uzima ih u obzir pri kompiliranju. Komentarisane delove MDS oboji u zeleno. Ako se znak ! nadje na početnoj poziciji onda je ceo red komentar. Znak & na kraju komentara ne znači prenos komentara u novi red – u ovom slučaju ovaj znak je sastavni deo komentara. Ovakav način zapisivanja programa naziva se slobodni zapis ¨*free form*¨. Programi koji se unose po pravilima slobodne forme treba da imaju ekstenziju .F90.

Pored ovih, zaista jednostavnih pravila, postoje i druga, starija – istorijski značajna pravila za unošenje FORTRAN-skih programa u računar. Istorijski gledano to su pravila za bušenje kartica, iz vremena kada su se program i ostali podaci unosili u računar preko bušenih kartica. U modernoj terminologiji takav način zapisa FORTRAN-skog programa naziva se fiksna forma ¨*fixed form*¨. Programi pisani u fiksnoj formi moraju da imaju ekstenziju .FOR ili .F.

Pravila za fiksnu formu unošenja programa su sledeća:

- naredba programa se unosi počev od sedme, a završava se najviše sa sedamdesed drugom kolonom,
- ako se u šestoj koloni nadje neki znak, to onda znači da se prethodni red nastavlja. Ovo se koristi u slučaju kada je neka nareda previše dugačka i ne može da stane u jedan red.
- Ako se u prvoj koloni postavi znak C onda se taj red tretira kao komentar i ne prevodi se pri kompiliranju.
- U kolonama 2, 3, 4 i 6 unosi se broj naredbe (ako je potrebno).

Ovakva pravila unošenja programa nose oznaku OBOLESCENT- zastarelo. Medjutim, kako je veoma veliki broj programa napisan baš na ovakav način, onda je u FORTRAN-u 90 omogućen rad sa ovakvim programima (prihvatanjem ekstenzije .FOR). Moguće je u jednom istom programu koristiti oba načina. Na primer u jednom programu koji je napisan u fiksnoj formi postavi se nareba
$FREEFORM

deo programa u slobodnoj formi
$NOFREEFORM

Slobodna forma počinje od naredbe $FREEFORM, a završava se naredbom $NOFREEFORM. Znak $ treba da bude u prvoj koloni. Naredbe koje počinju znakom $ nazivaju se *metacommand,* metakomande. Naredba $NOFREEFORM, se može izostaviti i onda slobodna forma ostaje do kraja programa.

Prazno mesto (blanko) se ne može stavljati u imenima koje programer kreira, kao ni u rečima koje su sastavni deo FORTRAN jezika (službene reči FORTRAN-a). Na primer, službena reč FORTRAN-a je PROGRAM i to treba da je prva reč koja se piše u progamu. Zapis, PRO GRAM je pogrešan i MDS javlja grešku pri kompiliranju. S druge strane, prazno polje, je obavezno izmedju službene reči FORTRAN-a i broja naredbe (labele). Na primer zapisi, 100INTEGER ili 300CONTINUE su nekorektni. Treba zapisati 100 INTEGER i 300 CONTINUE sa razmakom iza 100 i 300 (ovo su oznake pojedinih naredbi, tzv. labele).

II.2.UNOŠENJE I IZDAVANJE PODATAKA

Često je potrebno uneti vrednosti veličina za koje će se obavljati računanja; takodje je potrebno izdati- dobiti vrednosti nekih izračunatih veličina. Na modernim računarima unošenje se obavlja preko tastature, a izdavanje na monitorima. Unošenje podataka se obavlja naredbom READ, a izdavanje naredbama WRITE i PRINT. Ove naredbe imaju veći broj opcija, ali će se ovde obraditi samo najosnovnije. Sledeći program, pod imenom primer1.for je demonstracija unošenja brojeva, računanja njihovog proizvoda i štampanja rezultata računanja.

```
PROGRAM primer1
      WRITE(*,*)'unesi dva broja'
C  ovo je komentar
!   i ovo je komentar
      READ(*,*)a,b
      c=a*b
      WRITE(*,*)a,b,c
      END
```

Objašnjenje programa primer1.for

U prvoj liniji programa postavi se službena reč FORTRAN-a ¸¨PROGRAM¨, a iza nje piše se ime programa koje korisnik definiše. Ime programa u ovom primeru je *primer1*. Na ovom mestu ne stavlja se ekstenzija; u suprotnom kompajler javlja grešku. Programi i fajle imaju svoja imena (kao i u drugim oblastima računarstva), koje definiše programer i ona su u ovo tekstu ukucana malim slovima. Nasuprot ovome, reči koje pripadaju samom FORTRAN-u, (službene reči) kao što su već pomenuti PROGRAM, WRITE, READ i END u primer1.for programu, označavaju se velikim slovima.

U drugom redu programa stoji reč WRITE(*,*). Ovo je izvršna naredba FORTRAN-a i znači ¨piši¨. Pisanje može biti po ekranu monitora računara. Ako se baš to želi onda se u zagradama na prvom mestu stavi zvezdica, tj. *. Ako se pak želi pisanje u fajlu onda se umesto prve zvezdice stavlja broj koji označava datu fajlu.

Druga zvezdica u naredbi WRITE znači ¨*piši bilo kako*¨, to jest računar će ispisati tekst ¨onako kako mu je zgodno¨ bez ikakvog uredjivanja teksta. Ako se pak, želi nekakvo uredjivanje teksta (formatiranje) onda se umesto druge zvezdice unosi broj koji označava FORMAT naredbu koja definiše način uredjivanja teksta. Ovo će detaljnije biti objašnjeno kasnije. Tekst iza naredbe WRITE izmedju apostrofa se pojavljuje kao tekst na ekranu monitoru (ako se štampanje obavlja na monitoru) ili će se upisati u označenu fajlu.

Sledi naredba READ(*,*)*a,b* što znači ¨čitati¨ vrednosti veličina *a* i *b* sa tastature. Zvezdice u naredbi READ imaju isto značenje kao i u naredbi WRITE. Prva zvezdica znači uneti vrednosti veličina a i b sa tastature; ako se želi unos iz neke fajle onda se umesto te zvezdice postavlja broj koji označava tu fajlu. Druga zvezdica u READ naredbi označava da se unos obavlja na bilo koji način. Ukoliko se želi da se unos obavi na odredjeni način, onda se umesto druge zvezdice stavlja broj FORMAT naredbe koja bliže odredjuje način unosa. Vrednosti veličina a i b, se mogu uneti u jednom redu i brojevi treba da su razdvojena praznim mestom ili zarezom. Na primer zapis 3.25, 5.54 je korektan.Unos se može obaviti i u dva reda, te je zapis

```
3.25
5.54
```

takodje korektan.

Svaka READ naredba znači jedan red na ulazu. Tako sledeća dva zapisa nisu sasvim jednaka.

```
      READ(*,*)a,b    !ovo je prvi zapis
i
      READ(*,*)a      !ovo je drugi zapis
      READ(*,*)b
```

Prvi zapis podrazumeva da se unos obavi u jednom redu. Ali kako je ovaj unos slobodan, (tj. nije formatiran), može se obaviti i u dva reda; nakon unosa veličine a, računar stoji i čeka unos veličine b i nije bitno da li je u istom ili sledećem redu. Drugi zapis podrazumeva unos u dva reda, u prvom je vrednost veličine *a*, a u drugom vrednost veličine *b*. Ako se pri drugom zapisu unos obavi u jednom redu vrednost veličine b neće biti definisana i računar neće krenuti sa daljim izvršavanjem programa. Drugim rečima, računar će pročitati vrednost veličine, *a*, koja je prva navedena u redu, a drugi podatak će ignorisati. Potreban je još jedan red sa vrednošću veličine *b*.

Veličine *a* i *b* se nazivaju promenljive. Pri kompiliranju programa svakoj promenljivoj se pridružuje odredjeni memorijski prostor (jedan ili više memorijskih registara, zavisno od vrste promenljive). Kada se izvrši naredba READ(*,*)a,b onda se brojčane vrednosti koje se unesu u računar, smeste u memorijske registre rezervisane za te veličine.

Sledeća naredba je obično množenje *c=a*b*. Množenje se u FORTRAN-u označava zvezdicom izmedju brojeva koji se množe. Ova naredba se može pročitati na sledeći način: uzeti vrednosti veličina *a* i *b* iz odgovarajućih memoriskih registara, zatim ih pomnožiti i dobijenu vrednost pridružiti veličini *c*, tj. upisati u memorijski registar rezervisan za veličinu *c*. U skladu sa ovakvim obavljanjem operacije množenja, a i

drugih operacija u FORTRAN-u korektan je i sledeći zapis: *a=a*+1 (iako je matematički apsurdan). Ova naredba znači, uzeti vrednost veličine *a* iz memorije, povećati tu vrednost za 1 i novi rezultat vratiti u isti memorijski registar. Znak + se koristi da se označi sabiranje. Tako korektan je i zapis *a=a*a* i td.

Sledeća naredba WRITE(*,*)a,b,c štampa vrednosti veličina a,b i c na monitoru računara.

Naredba END (kraj) se stavlja da označi fizički kraj programa. To je poslednja naredba programa. Ako se ova naredba ne stavi, kompajler javlja grešku.
Kraj objašnjenja programa primer1.for

Veličine a i b u prethodnom primeru su brojčane veličine. Pored brojčanih veličina u FORTRAN-u postoje i druge vrste veličina koji će se kasnije obradjivati. U FORTRAN-u 90 omogućeno je da korisnik sam definiše svoje sopstvene tipove veličina.

Pored množenja, nad brojčanim veličinama se mogu obavljati i ostale računske operacije, kao što su sabiranje (+), oduzimanje (-), deljenje (označava se sa /) , stepenovanje **, i dr.
Naredbe READ(*,*) se može uprostiti; najprostija forma ove naredbe je READ *,. Naredba WRITE (*,*) se ne može uprostiti na sličan način, ali se može zameniti naredbom PRINT *, kojom se veličine štampaju na ekranu monitora.

II.3. IMENA PROMENLJIVIH

Iz prethodnog teksta je jasno da promenljive imaju imena, tj., programer zadaje imena promenljivima po svojoj volji. Ovakav koncept potiče od VonNeumana. Postoje jednostavna pravila pri zadavanju imena promenljivih. Ime promenljive mora da sadrži samo brojke i slova (alfanumeričke karaktere) i može da sadrži znak podvlačenja ¨_¨ (underscore). Prvi karakter mora da bude slovo. Broj karaktera u imenu promenljive je limitiran na 31. U ranijim verzijama FORTRAN a , na primer u FORTRAN –u 77 ime promenljive je moglo da sardži najviše do 6 simbola. Povećanje dužine imena može da rezultuje u poboljšanju čitljivosti programa. Na primer lakše je razmeti ime *broj_radnika* nego *br*.

Prazno mesto, tj. blanko, ne može biti u imenu promenljive. Takodje, znaci interpukcije i znaci aritmetičkih operacija ne mogu biti u imenu promenljive.

Potrebno je još istaći da FORTRAN90 ne pravi razliku izmedju velikih i malih slova; tako je ime ¨*ocena*¨ u potpunosti jednako sa imenom ¨*OCENA*¨. Ovim je FORTRAN90 zadržao ovu (rekao bih dobru) karakteristiku ranijih verzija FORTRAN-a. Ako se program kompilira na računaru pod Unix operativnim sistemom, onda u ovom pitanju može doći važnog izuzetka. U slučaju da se u programu koristi neka fajla van osnovnog programa, onda je značajno kakvim je slovima napisano ime te fajle, jer Unix operativni sistem pravi razliku izmedju velikih i malih slova.

Ne preporučuje se da se za imena promenljivih koriste službene reči FORTRAN-a kao što su READ, WRITE, PRINT, PROGRAM, END i sl. Dato je nekoliko ispravnih i neispravno napisanih imena promenljivih.

Ispravno ime	Neispravno ime i razlog neispravnosti
X	1x (počinje brojem)
XZ	X Z (sadži prazno mesto)
XY	X-Y (sadrži znak minus)
ENDOFDAY	X*Z (sadrži znak množenja)

II.4. UNOS PODATAKA IZ FAJLE I UPISIVANJE U FAJLU

Iako se unos podataka sa tastature možda čini najpogodnijim, nije uvek tako. Na primer, česte su situacije u kojima su ulazni podaci glomazni i njihovo ukucavanje bi izazvalo gubitak u vremenu i greške pri ukucavanju. Može se desiti slučaj da potreban unos i više hiljada podataka, a onda je to teško moguće sa tastature. Svaka greška bi značila ponovno ukucavanje celog seta podataka od početka. U ovakvim situacijama formira se fajla sa odredjenim imenom i u nju se smeste potrebni podaci, a zatim se oni pročitaju iz programa. Otvaranje fajle obavlja se naredbom OPEN. Sintaksa ove naredbe je OPEN(n,FILE=’imefajle’,STATUS=’status’).

U ovoj naredbi broj n je broj koji označava datu fajlu. Korisnik definiše ime fajle. STATUS može biti: OLD ako fajla već postoji, NEW ako treba da se formira nova, ili UNKNOWN ako je nepoznato. STATUS se može izostaviti. Fajla koja se otvara sa OPEN mora se komandom INSERT dodati u radni prostor projekta. Čitanje podataka iz jedne fajle i upis u drugu fajlu je dat u sledećim programom primer2.f90.

```
PROGRAM primer2
      OPEN(10,FILE='ulaz.dat',STATUS='OLD')
      OPEN(20,FILE='izlaz.dat',STATUS='NEW')
      READ(10,*)aa,bb
   cc= aa*bb
      WRITE(20,*) cc
END
```

Naredba OPEN(10,FILE='ulaz.dat',STATUS='OLD') ima sledeći smisao; otvoriti fajlu pod imenom ulaz.dat, i pridružiti joj broj 10. Naredba OPEN omogućuje pristup i korišćenje ove fajle. STATUS='OLD' znači da je ova fajla već postojeća i da je već ranije naredbom INSERT kreirana u radnom prostoru istog projekta gde se nalazi i program2.f90. Naredbom OPEN(20,FILE='izlaz.dat',STATUS='NEW') kreira se nova fajla pod imenom izlaz.dat i pridružuje joj se broj 20. Ova fajla ne treba da postoji u radnom prostoru projekta pre izvršavanja programa. Naredbom READ(10,*)aa,bb obavlja se čitanje vrednosti veličina *aa* i *bb* iz fajle koja je otvorena pod brojem 10, a to je ulaz.dat. Sledi računanje njihovog proizvoda cc, i na kraju upis rezultata naredbom WRITE(20,*)*cc* u fajlu izlaz.dat otvorenu pod brojem 20. Ekstenzija .DAT znači da se radi o fajli sa podacima (DATA podaci). Naredba OPEN ima veći broj raznih opcija od kojih će neke biti obradjivane kasnije.

II.5.TIPOVI VELIČINA I PODATAKA U FORTRANU

Postoji više tipova podataka i veličina u FORTRAN-u. Pored toga, moguće je definisati sopstvene tipove podataka. U ovom poglavlju biće obradjena dva osnovna tipa podataka, celobrojne i realne. To su brojčane (numeričke) veličine.

U FORTRAN –u postoji znatna razlika izmedju celobrojnih i realnih veličina. Celi brojevi su oni koji u svom zapisu ne sadrže decimalnu tačku, a mogu da sadrže znak

minus (-) za označavanje negativne vrednosti. Na primer, zapis ¨5¨ ili ¨-2¨ označavaju cele brojeve. Realni brojevi u svom zapisu sadrže decimalnu tačku; na primer sledeći zapisi, 5.5; 2.2; -0.1, su realni brojevi. Ako je decimalni deo nula može se pisati bez nule kao na primer, 2.: takodje ako je ceo deo broja nula umesto 0.2 može se pisati samo .2. Razlika izmedju celih i realnih brojeva potiče od različitog predstavlja ovih brojeva u memorijskim registrima računara.

Jedna od osobina koje su takoreći bile obeležje ranijih verzija FORTRAN-a je bila takozvana unutrašnja koncencia o tipu veličina. Ona glasi: veličine čija imena počinju sa jednim od slova I,J,K,L,M, ili N su celobrojne veličine. Veličine koje ne počinju sa jednim od ovih slova su po unutrašnjoj konvenciji FORTRAN-a, su realne. Ova unutrašnja konvencija se smatra zastarelom- prevazidjenom. Zato je nova verzija FORTRAN-a omogućila njeno isključivanje naredbom IMPLICIT NONE. Ova naredba se stavlja na početku programa. Postojanje ove naredbe, sa druge strane znači obavezno deklarisanje tipova svih veličina koje se pojavljuju u programu.

Pored numeričkih, postoje i nenumeričke veličine.

II.5.1. Deklarisanje tipa promenljivih

Pored unutrašnje konvencije FORTRANA-a, tip promenljive se može definisati eksplicitnom ili implicitnom deklaracijom. Eksplicitna deklaracija definiše tip promenljive prema njenom imenu.

Eksplicitna deklaracija celobrojne promenljive obavlja se naredbom

INTEGER lista imena promenljivih

Sve promenljive navedene u listi iza naredbe integer su celobrojne. Imena promenljivih se u listi odvajaju zarezima. Na primer

INTEGER *a, broj_radnika, broj_dana*

Promenljive *a, broj_radnika, broj_dana* su celobrojne.

Eksplicitna deklaracija realne promenljive se obavlja naredbom

REAL lista imena promenljivih

Sve promenljive navedene u listi iza naredbe REAl su realnog tipa.
Na primer

REAL *interes, kamata*

Promenljive *interes* i *kamata* su realne.
Eksplicitna deklaracija ima prioritet u odnosu na unutrašnju konvenciju FORTRAN-a.
Implicitna deklaracija tipa promenljive definiše tip promenljive prema prvom slovu imena promenljive. Oblik implicitne deklaracije je

IMPLICIT TIP prva slova imena

Tip može biti INTEGER ili REAL
Tako na primer

IMPLICIT INTEGER A,B,C

Definiše promenljive koje počinju slovima *a,b* i *c* kao celobrojne, a

IMPLICIT REAL D-N, T

definiše kao realne sve promenljive koje počinju bilo kojim slovom od *d* do *n* i promenljive koje počinju slovom *t*.

Ustanovljen je prioritet deklaracija tipa promenljivih. Najviši prioritet ima eksplicitna deklaracija, zatim implicitna a poslednja po prioritetu je unutrašnja konvencija FORTRAN-a.

Postoje naredbe koje prevode brojeve iz jednog tipa u drugi. Prevodjenje realnog broja u ceo broj obavlja se naredbom

INT(*a*)

gde je *a* neki realni broj. Kao rezultat dobija se neki ceo broj, tj odbacuje se decimalna tačka i decimalni deo. Na primer INT(2.245) daće broj 2 kao rezultat. Prevodjenje celog broja u realni obavlja se naredbom REAL(*a*), gde je *a* neki broj (ne mora biti ceo broj-kompajler neće da prijavi grešku). Slična je i naredba FLOAT(*i*) koja prevodi ceo broj, *i* u realni, ali argument *i* mora biti ceo broj. Ako se umesto argumenta stavi realni broj kompajler prijavljuje grešku u fazi kompiliranja.
Naredbe FLOAT, INT i REAL su unutrašnje funkcije FORTRAN-a. Ima veliki broj takvih funkcija i kasnije će biti više reči o njima.

II.6. FORMATIRANI UNOS I ŠTAMPANJE CELOBROJNIH I REALNIH VELIČINA

U ranijem tekstu objašnjen je slobodni unos i štampanje veličina. Medjutim pojavljuje se potreba definisanog unosa i uredjenog štampanja izlaznih veličina. Da bi se to realizovalo koristi se naredba FORMAT. To je opisna naredba FORTRAN-a i sadrži opise polja u kojima se unose (ili štampaju) vrednosti veličina.

Celobrojne veličine (In opis)
Za opis celobrojnih veličina koristi se opis In; n je broj pozicija (karaktera) koje zauzima polje u kome se unosi dati broj. Razmortrimo sledeći primer:

```
      PROGRAM primer3
      WRITE(*,*)'unesi dva trocifrena cela broja po formatu I4'
      READ(*,10)i,j
10    FORMAT(i4,i4)
      WRITE(*,*)i,j
      PRINT *, ‘'unesi dva realna broja po formatu F6.4'
      READ(*,20)a,b
20    FORMAT(F6.4,F6.4)
```

```
      WRITE(*,30) a,b
30    FORMAT(' ',2F6.4)
      END
```

Tekst koji se nalazi izmedju apostrofa u prvoj WRITE naredbi pojaviće se na ekranu. Naredba READ(*,10) omogućava unošenje veličina *i* i *j* (celobrojne prema unutrašnjoj konvenciji). Prva zvezdica znači unos sa tastature, a broj 10 je obeležje FORMAT naredbe prema kojoj se vrši unos podataka. Unos se obavlja prema opisu I4. To znači da se veličine *i* i *j* unose u polju sa četiri pozicija. Ako je broj trocifren, onda prvu poziciju treba ostaviti praznu. Unošenje drugog broja treba početi od pete pozicije u redu. Ako unošenje drugog broja ne počne od pete, već od četvrte pozicije, onda unos nije korektan i doći će do pogrešnog interpretiranja rezultata. Ako se na primer ukuca 123456789 onda će se kao izlaz dobiti 1234 5678 što znači da je poslednja cifra (9) jednostavno zanemarena. Ako su brojevi negativni onda se ispred stavlja znak minus (-) i pozicija za taj znak se takodje uračunava u I formatu. Ako se na primer ukuca –123–321 (bez razmaka) onda se dobija na izlazu –123 –321.

Druga WRITE naredba štampa vrednosti veličina *i* i *j* na ekranu i služi da se proveri da li je unos obavljen korektno.

Formatirani ulaz ima smisla kada se unos podataka obavlja iz fajli; ako se unos obavlja sa tastature onda je formatiranje nepotrebno i često otežava rad.

Realne veličine (F opis)

Realne veličine se unose u poljima opisanim F formatom. Sintaksa F fromata je Fk.d . Ovde je k ukupan broj pozicija koja zauzima broj, a d je broj decimalnih mesta. Jedno mesto se ostavlja za decimalnu tački, a jedno mesto za znak broja (obavezno se unosi znak minus (–) ako je broj negativan). Znak + ne mora da se unosi, te ne mora da se rezerviše mesto u opisu za njega. U opisu F6.3 mogu se unositi sledeći brojevi, -3.123; 12.123; -0.987. Primer korišćenja F opisa je dat u prethodnom programu primer3.
U FORMAT naredbama broj 10 i 20, isti opis se ponavlja dva puta. Umesto I4,I4, može se zapisati 2I4. Takodje umesto F6.4,F6.4 može se pisati 2F6.4. Tako kompletna sintaksa I i F opisa je nIk, i nFk.d, gde broj n pokazuje koliko puta se ponavlja opis sa k pozicija.

Realne veličine (E opis)

Realni brojevi mogu biti vrlo veliki, ili vrlo mali, tako da je unošenje prema F opisu nepraktično. Na primer naelektrisanje elektrona je $e=1.602\cdot10^{-19}$ C, i bilo bi teško unositi broj sa 15 ili 16 nula. Trebalo bi da se koristi format F25.22. U takvim slučajevima se koristi E opis, koji ima sintaksu nEk.d; k je broj pozicija koje se rezervišu za unošenje nekog broja, a d je broj decimalnih mesta. Ovaj zapis podrazumeva korišćenje eksponenta broja 10 u zapisu. Umesto broja 10 koristi se slovo E. Tako naelektrisanje elektrona bi se napisalo kao 1.602E-19. Pri korišćenju E opisa mora da se ostavi jedno mesto za znak broja (ako je negativan), jedno mesto za decimalnu tačku, jedno mesto za slovo E, jedno mesto za znak eksponenta (ako je negativan) i dva mesta za izložilac. Tako, rezervisano je ukupno 6 mesta. Ako je znak izložioca pozitivan onda on može da se izostavi te su ispravni i sledeći zapisi:

-1.602E-19 (naelektrisanje elektrona u Kulonima, C),
9.1E-31 masa elektrona u kg, (izostavljen pozitivan znak broja),

2.99792458E8 (brzina svetlosti u m/s) izostavljen znak i napisana jedna cifra izložioca,
.1E2 (izostavljeno: znak ispred broja, nula ispred tačke, znak eksponenta, i jedna cifra u eksponentu).
Pri slobodnom neformatiranom zapisu brojevi se mogu unositi na ovaj način.
Pored I, F i E opisa, postoji i čitav niz drugih, koji se odnose na razne tipove promenljivih koje postoje u FORTRAN-u. Postoji i takozvani generalisani opis, G – opis. Može se koristiti za bilo koji tip promenljive koja je definisana u FORTRAN-u, ali zapis Gk.d, kako se inače navodi generalisani opis, ima različit smisao zavisno od toga koji tip promenljive predstavlja.

Pored formatiranog ulaza, koristi se, i od većeg je značaja formatirani izlaz. Često je potrebno urediti izlazne podatke na željeni način, bilo radi preglednosti, estetike ili rada lakšeg zahvatanja tih podataka od strane drugih programa. Pri izdavanju celobrojnih veličina koristi se opis In. Za štampanje realnih veličina koriste se opisi nFk.d ili nEk.d, sa istim smislom kao i kod ulaza. Razlika je što se u FORMAT naredbama koje su vezane za izlazne WRITE naredbe na početku stavlja prazno mesto izmedju apostrofa, tj. piše se
FORMAT(' ', opisi)
Razlog za ovo je istorijske prirode. Računari iz 60 tih i 70 tih godina prošlog veka su imali izlaz na štampačima. Pri štampanju bilo je potrebno pomeriti papir te je ovaj prvi simbol služio za kontrolu pomeranja papira. Ako stoji samo prazno mesto izmedju apostrofa ' ' onda se papir pomera za jedan red; sada to znači da se štampa obavlja u novom redu. Ako se ne navede ' ' onda će se za pomeranje papira za jedan red upotrebiti prvi sledeći znak i može da se desi da je ispis na ekranu ili u fajli nekorektan. U programu primer3 dat je primer korišćenja FORMAT naredbe u štampi. Ako se na ulazu za a i b unese sledeće, a=1.2345 i b=6.5432 onda će se na izlazu dobiti 123456.5432 što je vrlo nepregledan zapis i ne zna se gde se jedan broj završava a gde počinje drugi. Da bi se ovo izbeglo koriste se prazna polja. Opis praznog polja je nX; n je broj pozicija koje su prazne- tj. širina praznog polja. Tako ako se u prethodnom primeru stavi
30 FORMAT(' ', F6.4, 2X, F6.4) onda će se za isti ulaz dobiti na izlazu 1.2345 6.5432.
Formatirani ispis podataka se može primeniti i bez korišćenja FORMAT naredbe. U tom slučaju se u samoj WRITE naredbi umesto broja FORMAT naredbe postavljaju odgovarajući opisi, kao na primer u naredbi
WRITE(*,'(1x,F6.3,2x,F8.3)')a,b,c

Format se stavlja izmedju apostrofa i u zagradama. Prvo polje 1x služi za 'pomeranje papira', tj za vertikalno uredjenje izlaza. Broj opisa ne mora da bude jednak broju veličina koje se štampaju. U prethodnoj naredbi štampaju se tri veličine, *a,b* i *c*, a ima dva opisa. Pri radu, iskoriste se svi opisi koji su navedeni, a onda se ponovo krene od početka. Veličine *a* i *b* će se štampati u jednom redu po formatima F6.3 i F8.3, a veličina *c* će se štampati u drugom redu po formatu F6.3.

Treba još napomenuti da ako je neki broj veći nego što je predvidjeno opisom u FORMAT naredbi onda se u tom polju štampaju zvezdice.

II.7. ZADAVANJE POČETNIH VREDNOSTI PROMENLJIVIH

Pre korišćenja neke veličine potrebno je da ona ima odredjenu vrednost. Dodeljivanje vrednosti promenljivima se može obaviti naredbom READ kada se to čini sa tastature ili iz neke fajle. Medjutim, dodeljivanje vrednosti veličinama se može obaviti i na veći broj drugih načina. Ovo se još naziva i inicijalizacija. Ukoliko vrednost veličine nije definisana, pri kompiliranju se pojavljuje upozorenje da se ta veličina koristi a vrednost nije odredjena. Nedefinisane veličine imaju vrednost nula (0). Na nekim računarskim sistemima nedefinisane veličine uzimaju vrednost koja se slučajno našla u memorijskom registru koji je rezervisan za tu promenljivu. Ovakva haotična situacija sigurno izaziva grešku i netačan račun.

Eksplicitna deklaracija tipa promenljive se može iskoristiti za inicijalizaciju. Sintaksa naredbe je

REAL :: ime=vrednost

Promenljivoj čije je ime navedeno se pridružuje odgovarajuća vrednost. Obratiti pažnju da se koristi zapis od dva puta po dve tačke. Naredbama

```
REAL ::A=1.
INTEGER:: S1=25
```

se realnoj promenljivoj A dodeljuje vrednost 1., a celobrojnoj promenljivoj S1 se dodeljuje 25.

Početna vrednost se može zadati posebnom naredbom DATA. Sintaksa ove naredbe je

```
DATA lista_imena/vrednosti/
```

Promenljivima koje su date u listi_imena se dodeljuju redom vrednosti. U listi se može navesti veći broj imena promenljivih koja se odvajaju zarezima. Početne vrednosti se unose izmedju kosih crta i takodje se odvajaju zarezima.Korektni su sledeći zapisi:

```
DATA ss,s1 /10.,20./
DATA x,y,z,j /3*100.,1/
```

Promenljiva *ss* u prvoj naredbi dobija početnu vrednost 10. , a promenljiva *s1* vrednost 20. U drugoj DATA naredbi veličine *x,y,z* dobijaju početne vrednosti 100, a veličina *j*, vrednost 1. Obratiti pažnju da se broj 100. ne ukucava tri puta, već se koristi zapis 3*100. (Jasno je da množilac 3 u prethodnom primeru ne može biti negativan).

Navedene vrednosti se moraju slagati sa imenima navedenih veličina po redu tipu i broju. Pokušaj da se veličini j (u drugoj DATA naredbi) pridruži realna vrednost, 1., neće izazvati grešku pri kompiliranju, jer će kompajler sam odbaciti decimalnu tačku. Medjutim naredba DATA x,y,z,j /3*100./ neće proći pri kompiliranju jer veličini *j* nije pridružena vrednost.

Može se postaviti pitanje, ¨zašto se u programu jednostavno ne napiše nareba j=1, tj. zašto je uopšte potrebna DATA naredba¨. Zadavanje početnih vrednosti preko DATA naredbi omogućuje istovremeno iniciranje većeg broja veličina, čime je omogućen kraći i jednostavniji zapis u programu. Umesto jedne naredbe, kakva jeste prethodna, bilo bi potrebno ukucavanje 4 naredbi u programu.

Često je potrebno inicirati neku veličinu, a pri tome njena vrednost ne treba da bude se menja u toku rada. To su konstante. Na primer broj π ne bi trebalo da se menja u toku rada. Da bi se jedna veličina definisala kao nepromenljiva konstanta koristi se naredba

PARAMETER (constanta=vrednost)

Naredba PARAMETER se stavlja na početku programa i definiše vrednost konstante. Istovremeno se može definisati i tip promenljive, pri čemu se tip stavlja pre reči PARAMETER. Na primer korektni su sledeći zapisi

```
REAL, PARAMETER :: pi=3.141592
INTEGER, PARAMETER :: GODINA=2004
```

(obratiti pažnju na zarez izmedju REAL i PARAMETER i na dve tačke ispred imena konstanti.) Vrednosti pridružene veličinama *pi* i *godina* su fiksne u programu. Pokušaj menjanja vrednosti ovih veličina izaziva grešku, koja se pojavljuje u toku kompiliranja.

II.8.JEDNOSTAVAN RAČUN U FORTRANU

II.8.1. Aritmetičke operacije

Osnovne aritmetičke operacije, sabiranje, oduzimanje, množenje i deljenje se u FORTRAN-u unose uobičajenim znacima, +, -, * i / . Stepenovanje se označava sa dve zvezdice **. Tako, zapis a**b, predstavlja a^b. Aritmetičke operacije imaju odredjeni prioritet. Najviši prioritet ima stepenovanje, (koji označavamo sa 1), množenje i deljenje imaju niži prioritet (br. 2) i na kraju sabiranje i oduzimanje imaju najniži prioritet jednak 3. Računske operacije se obavljaju sa leva u desno, poštujući prioritet. Zapis *A+B/C**2* se izvršava na sledeći način: prvo se *C* kvadrira, pa se *B* deli sa rezultatom kvadriranja i na kraju se sve to sabere sa *A*. Zapis *A/B/C* se izvršava tako što se *A* prvo podeli sa *B* a zatim se dobijeni rezultat deli sa *C*. Ovo je isto što i A/(B*C), ali je različito od zapisa A/B*C gde se prvo A deli sa B pa se rezultat množi sa C. Promena redosleda izvršavanja računanja se može obaviti zagradama.

Aritmetičke operacije izmedju raznih tipova podataka

Aritmetička operacija izmedju celih brojeva kao rezultat daje ceo broj. Ovo se odnosi i na deljenje, celih brojeva (ranije pomenuto)č deljenje celih brojeva daje ceo broj, tj dolazi do odbacivanja decimalnog dela. Sledeći program ilustruje deljenje celih brojeva.

```
PROGRAM deljenje_celih
IMPLICT NONE
INTEGER a,b,c
PRINT *,'uneti dva cela broja'
READ*, a,b
c=a/b
PRINT *, c
END
```

Ako je *a*= 3, a *b*= 2, onda se dobija *c*=1; jasno je da je 3/2=1.5 ali ovde dolazi do odbacivanja decimalnog dela. Slično, za *a*=2 i za *b*=3 , dobija se *c*=0. O ovome treba voditi računa pri radu sa celim brojevima.

Operacija izmedju celog broja i realnog broja daje za rezultat realan broj. Pri operaciji ceo broj se prvo prevede u realni, pa se tek onda obavlja tražena operacija. Tako pogodniji je zapis a/2. (dva sa tačkom) nego zapis a/2 (dva bez tačke) ako je a realan broj. U drugom zapisu, broj dva će se prvo prevesti u realan, pa će se tek onda obaviti deljenje. Tako, obavlja se jedna operacija više, čime se gubi na vremenu, odnosno brzini računanja. Zbog toga, pri pisanju realnih konstanti treba stavljati tačku iza broja i u onim slučajevima kada je decimalni deo nula; tj. treba pisati 2. ; 3. i sl.

Operacija izmedju realnih brojeva daje realan broj.

Zadatak: prevesti sledeće formule u fortranski zapis

$$c = \frac{a}{b} x^2$$

$$e = \frac{\sqrt{x^2 + y^2}}{\sqrt{x^2 - y^2}} + 1$$

$$a = (p_1 + \frac{R}{100})^n$$

Fortranski zapis prethodnih formula je

```
C=A/B*X**2
E=SQRT(X**2+Y**2)/SQRT(X**2-Y**2)+1.
A=(P1+R/100.)**N
```

Obratiti pažnju da je zapis kvadratnog korena velićine x, dat sa SQRT(x). Korišćenje x**(1/2) je pogrešno, jer je 1/2 deljenje celih brojeva i daje nulu. Zapis x**(1./2.) dovodi do korektnog računanja korena, ali je ovde nekoliko koraka računanja više. Za računanje kvadratnog korena najpogodnije je koristiti funkciju FORTRAN-a SQRT.

II.8.2.UNUTRAŠNJE FUNKCIJE U FORTRANU

Pored gore pomenutih funkcija, SQRT, REAL, FLOAT, postoji i veći broj unutrašnjih funkcija u FORTRAN-u 90, koje su namenjene za računanje elementarnih matematičkih funkcija. Ove funkcije se nazivaju intrinsic function, ili unutrašnje funkcije. One su sastavni deo FORTRAN90 jezika. Unutrašnje funkcije su posebni podprogrami i zasnovani su na primer, na razvoju funkcije u stepeni red ili nečem sličnom. Tako, računanje je približno, tj. sa odredjenom greškom.Na primer, da bi se izračunao sinus nekog ugla x, dovoljno je u FORTRAN-u napisati SIN(x). Pri tome, ugao se unosi u radijanima, a ne u stepenima; ovo važi i za ostale trigonometrijske funkcije.

Postoji više kategorija unutrašnjih funkcija. Mogu se podeliti na sledeći način:

(a) elementarne numeričke funkcije- primenjuju se za elementarna računanja;
(b) funkcije obaveštavanja (pitanja)- daju neku karakteristiku argumenta;
(c) funkcije transformacije- obavljaju odredjene transformacije argumenata i

(d) ne elementarne rutine.

Neke funkcije podrazumevaju da je argument tačno zadatog tipa, (kao na primer FLOAT(i), gde *i* mora biti celobrojno), dok su neke funkcije manje osetljive na tip argumenta, i mogu da prihvataju argumente različitog tipa. Kao rezultat računanja dobija se veličina odredjenog tipa. U sledećoj tabeli date su neke od elementarnih funkcija FORTRAN-a i odgovarajući komentari.

Tabela. Neke unutrašnje funkcije FORTRAN90 jezika.

Matematički zapis	Svrha	Fortran zapis	Tip argumenta	Tip rezulta
$\sqrt{x}$	Kvadratni koren	SQRT(X)	x≥0, realno	realan
e^x	Eksponent broja e	EXP(X)	x realno	realan
ln x	Prirodni logaritam	LOG(x), ALOG(X)	x realno	realan
log x	Dekadni logaritam	LOG10(x), ALOG10(x)	x realno	realan
sin x	Sinus ugla	SIN(x)	x realno u radijanima	realan
sin x	Sinus ugla	SIND(x)	x realno u stepenima	realan
cos x	Cosinus ugla	COS(X)	x realno u radijanima	realan
cos x	Cosinus ugla	COSD(X)	x realno u stepenima	realan
tan x	Tangens ugla	TAN(X)	x realno u radijanima	realan
		TAND(X)	x realno u stepenima	realan
cotg x	Cotangens ugla	COTAN(X)	x realno u radijanima	realan
		COTAND(x)	x realno u stepenima	realan
arcsin x	Arcus sinus	ASIN(x)	$\lvert x\rvert \le 1$, xrealno	Real ugao u radijanima izmedju -π/2 i π/2
arcsin x	Arcus sinus	ASIND(x)		Real ugao u stepenima izmedju -90^0 i 90^0 .
arc cos x	Arcus kosinus	ACOS(x)	$\lvert x\rvert \le 1$, xrealno	Real ugao u radijanima izmedju 0 i π
arc cos x	Arcus kosinus	ACOSD(x)		Real ugao u stepenima izmedju 0^0 i 180^0.
arc tan x	Arcus tangens	ATAN(x)	x realan	Real ugao u raijanima izmedju -π/2 i π/2
sinh x cosh x tanh x	Sinus hiperbolični Cosinus hiperbolični Tangens hiperbolični	SINH(x) COSH(x) TANH(x)	x realan	realan
	Najveća vrednost brojeva	MAX(a1,a2,a_i)	a_i realni ili celobrojni	istog tipa kao i a_i
	Najmanja vrednost brojeva	MIN(a1,a2,a_i)	a_i realni ili celobrojni	istog tipa kao i a_i
$\lvert x\rvert$	Apsolutna vrednost	ABS(x)	x realno	realno

U prethodnoj tabeli dat je mali deo unutrašnjih funkcija koje su na raspolaganju u FORTRAN90 jeziku. Pojedine funkcije su nešto različite u odnosu na starije verzije FORTRAN a. Na primer, u fortran-u 77, funkcija za računanje prirodnog logaritma je bila ALOG(x); simbol A ispred LOG je stavljan jer je prema internoj konvenciji fortrana, veličina koja počinje sa L bila celobrojna. U FORTRAN90 omogućeno je korišćenje funkcije LOG(x) za računanje prirodnog logaritma. Slična je situacija i sa dekadnim logaritmom gde se u FORTRAN90 koristi LOG10, a ranije je to bilo ALOG10(x).

Takodje, funkcije za nalaženja maksimalnog i minimalnog medju navedenim brojevima su u ranijim verzijama pisane kao AMAX1 ili AMIN1 a sada je MAX ili MIN, resp.

Obratiti pažnju na dualitet trigonometrijskih funkcija. U tabeli postoje SIN(x) i SIND(x); u prvom slučaju ugao se unosi u radijanima, a u drugom u stepenima. Slična je situacija i sa ostalim trigonometrijskim funkcijama. U FORTRAN-u 77 nije postojala mogućnost unošenja uglova u stepenima.

U kasnijem tekstu biće više reči o unutrašnjim funkcijama FORTRAN-a 90.

III
ODLUČIVANJE U FORTRANU

Ranije je napomenuto da je jedina računska operacija koja se zaista odigrava u procesoru sabiranje brojeva. Sve ostale operacije se obavljaju preko sabiranja. Pored sabiranja računar je u stanju da uporedi dve veličine, tj. da odredi koja je veća, a koja manja. Na osnovu poredjenja veličina mogu se donositi odluke i obaviti grananje programa.

Naredba bezuslovnog prelaska GO TO n

Naredba GOTO *n* znači preći direktno na naredbu u programu koja ima obeležje (broj) *n*. Ova naredba je vrlo mnogo kritikovana i smatra se da ona utiče vrlo negativno na strukturno modularno programiranje i da je sa tog stanovišta vrlo štetna. Iz istorijskih razloga ova naredba je morala da ostane i u novijim verzijama FORTRAN-a, kao što je to uostalom slučaj i sa svim ostalim naredbama starijih FORTRAN-a. Preporuka je da se ova nareba uopšte ne koristi pri razvoju programa. U novijim verzijama biće u potpunosti izbačena.

III.1. **IF *NAREDBE***

U programima i primerima datim ranije, račun se odvijao tako što su se naredbe izvršašvale jedna za drugom. Medjutim, često je potrebno izmeniti redosled izvršavanja naredbi, zavisno od nekog uslova. Ima više načina na koji se može ovo realizovati u FORTRAN-u. Jedna od tih mogućnosti je korišćenje IF naredbi. Engleska reč IF znači '*ako*'. Ova naredba ima veći broj varijanti. Najprostija varijanta je zapis u jednom redu sa sledećom sintaksom:

IF(*logički izraz poredjenja*)*naredba*

Ovakav zapis IF naredbe ima sledeći smisao. Ako je izraz poredjenja tačan, onda se izvršava naredba zapisana posle zagrade. Ako izraz poredjenja nije tačan, onda se prelazi na sledeću naredbu.

Izrazi koji mogu da budu tačni ili netačni se nazivaju logički izrazi. Uporedjivanje se obavlja pomoću uobičajenih matematičkih simbola nejednačina (i jednačina) koji su donekle prilagodjeni programiranju. Ti simboli i njihov smisao je :

< označava manje

<= manje ili jednako
> veće
>= veće ili jednako
== jednako

Dati su neki elementarni zapisi i njihov smisao;

Zapis	smisao
IF(a>b)c=a+b	Ako je a veće od b, onda izračunati c kao sumu a+b.
IF(a<b)c=a-b	Ako je a manje od b, onda je c razlika a-b.
IF(a==b)c=a*b	Ako je a jednako b, onda je c proizvod a*b.

i sl.

Primer: Napraviti program za obračun poreza po kliznoj skali. Ako je plata manja od 6000. onda se ne naplaćuje nikakav porez; za platu izmedju 6000 i 12000 naplačuje se 6 % , od 12000 do 24000 naplaćuje se 8 % i za plate veće od 24000. porez je 10 %.
Program je sledeći:

```
PROGRAM OBRACUN_POREZA
IMPLICIT NONE
REAL plata, stopa, porez
READ *, plata
IF(plata<=6000.) stopa=0.
IF(plata>6000..AND.plata<=12000.) stopa=0.06
IF(plata>12000..AND.plata<=24000.) stopa=0.08
IF(plata>24000.) stopa=0.10
PRINT *, 'stopa poreza', stopa
porez=plata*stopa
PRINT *, 'porez je', porez
END
```

Novi elemenat u prethodnom programu je korišćenje službene reči . AND. (što znači *i*). Igra ulogu "*konjukcije*", tj. izraz *a.and.b* je tačan ako su oba izraza tačna. Ovo je jedna od operacija logičkih poredjenja. Naredba IF(plata>6000..AND.plata<=12000.) stopa=0.06 znači sledeće; ako je istovremeno, plata veća od 6000 i manja ili jednaka 12000, onda je stopa jednaka 0.06.

U ranijim verzijama FORTRAN-a korišćeni su takozvani znaci poredjenja; to su bili akronimi engleskih reči, veće, manje i jednako.

.GT. (greater than) veće od; ovo je u FORTRAN90 zamenjeno sa >.
.GE. (greater than or equal to) veće ili jednako; zamenjeno sa >=.
.EQ. (equal) jednako; zamenjeno sa ==.
.LE. (less than or equal to) manje ili jednako; zamenjeno sa <.
.LE. (less than) manje od; zamenjeno sa <=.

Tako, prethodni primeri su se u FORTRAN77 pisali na sledeći način:

IF(a.GT.b)c=a+b	Ako je a veće od b, onda izračunati c kao sumu a+b.
IF(a.LT.b)c=a	Ako je a manje od b, onda je c jednako sa a.
IF(a.EQ.b)c=a*b	Ako je a jednako b, onda je c proizvod a*b.

III.1.1. IF-THEN struktura

U prethodnoj sintaksi IF naredbe, naredba koja je trebala da se izvrši ako je uslov zadovoljen je pisana u jednom redu. Medjutim, često nije moguće obaviti račun samo jednom naredbom. U takvim slučajevima koristi se IF THEN konstrukcija. Sintaksa je sledeća:

IF (logički izraz poredjenja)THEN
blok naredbi
END IF

Ako je izraz poredjenja tačan onda se izvršava blok naredbi izmedju redova u kome je IF i END IF. Ako izraz poredjenja nije tačan, onda se prelazi na naredbu koja je prva iza END IF. Kao ilustracija IF-THEN bloka dat je sledeći primer računanja kvadratne jednačine.

```
PROGRAM KVJED
IMPLICIT NONE
REAL A,B,C
REAL DISKR,X1,X2, KOREN
OPEN(10,FILE='ULAZKVJED.DAT') !FAJLA SA KONSTANTAMA JEDNACINE
READ (10,*) A,B,C
DISKR=B**2-4.*A*C
PRINT *, 'DISKRIMINANTA JEDNACINE JE', DISKR
IF(DISKR<0) THEN
 PRINT *, 'DISKRIMINANTA MANJA OD NULE'
 PRINT *, 'NEMA  REALNIH KORENA JEDNACINE'
END IF

IF(DISKR==0.) THEN
 PRINT * ,'JEDNACINA IMA JEDNO DVOSTRUKO RESENJE'
  X1=-B/2./A
 PRINT *, 'X1=', X1
END IF

IF (DISKR>0.)THEN
 PRINT *, ' JEDNACINA IMA DVA REALNA RESENJA'
   KOREN=SQRT(DISKR)
    X1=(-B+KOREN)/2./A
    X2=(-B-KOREN)/2./A
   PRINT *, 'RESENJA SU'
  PRINT *, X1
 PRINT *, X2
END IF
END
```

Koeficijenti kvadratne jednačine $ax^2+bx+c=0$, se unose iz fajle ULAZKVJED.DAT bez formatiranja. Računa se diskriminanta $diskr=b^2-4ac$ i zatim se na osnovu vrednosti diskriminante obavljaju računanja.

Pri pisanju je često korisno grupisati naredbe koje pripadaju jednom bloku. Ovim se poboljšava čitljivost programa i omogućuje lakša kontrola.

Problem je rešen pomoću tri IF-THEN bloka. Prvi blok se odnosi na slučaj kada je diskriminanta manja od nule i tada se jedino štampa odgovarajuća poruka. Drugi blok tretira slučaj kada je diskriminanta jednaka nuli i onda se računa jedno dvostruko rešenje

jednačine. Konačno u trećem bloku, kada je diskrimininanta veća od nule, računaju se i štampaju oba realna rešenja. Obratiti pažnju da se deljenje sa *(2·a)* obavlja sa */2./a* .

III.1.2. IF–THEN–ELSE struktura

Ova struktura se sastoji iz jedne IF naredbe kojom se ispituje neki logički uslov i iz dva bloka koja su odvojena rečju ELSE. To bi se moglo zapisati kao

```
IF(logički uslov) THEN
 blok 1
  ELSE
 blok 2
END IF
```

Ako je logički uslov tačan, onda se izvršava blok 1 i prelazi se na prvu naredbu iza END IF. Ako je uslov netačan onda se izvršava blok 2 i opet se prelazi na prvu naredbu iza END IF.
Moguće je u okviru jedne IF-THEN-ELSE-ENDIF strukture koristiti više ELSE naredbi. U ovom slučaju sintaksa je sledeća

```
IF(logički uslov1) THEN
  blok 1

  ELSE IF(logički uslov 2)
  blok 2

 ELSE IF(logički uslov 3)
  blok3
..
..
..
 ELSE
 blok n
END IF
```

Kao primer korišćenja IF-THEN-ELSE-ENDIF strukture sa više ELSE naredbi, ponovo se daje program KVJED1 za rešavanje kvadratne jednačine:

```
PROGRAM KVJED1
IMPLICIT NONE
REAL A,B,C
REAL DISKR,X1,X2, KOREN
OPEN(10,FILE='ULAZKVJED.DAT')                    !FAJLA SA KONSTANTAMA JEDNACINE
READ (10,*) A,B,C
DISKR=B**2-4.*A*C
PRINT *, 'DISKRIMINANTA JEDNACINE JE', DISKR
IF(DISKR<0) THEN
 PRINT *, 'DISKRIMINANTA MANJA OD NULE'
```

```
 PRINT *, 'NEMA  REALNIH KORENA JEDNACINE'
ELSE IF(DISKR==0.) THEN
 PRINT * ,'JEDNACINA IMA JEDNO DVOSTRUKO RESENJE'
  X1=-B/2./A
 PRINT *, 'X1=', X1
ELSE  IF (DISKR>0.)THEN
 PRINT *, ' JEDNACINA IMA DVA REALNA RESENJA'
   KOREN=SQRT(DISKR)
    X1=(-B+KOREN)/2./A
    X2=(-B-KOREN)/2./A
   PRINT *, 'RESENJA SU'
  PRINT *, X1
 PRINT *, X2
END IF
END
```

Kada se nadje na tačan logički izraz onda se obavlja blok iza tog izlaza.

III.1.3. Aritmetička IF naredba

Sintaksa aritmetičke IF naredbe je sledeća
IF(aritmetički izraz)n1,n2,n3

Ova naredba se izvršava na sledeći način. Prvo se izračuna vrednost aritmetičkog izraza. Ako je ona manja od nule odlazi se na naredbu sa brojem n1. Ako je izraz jednak nuli izvršava se naredba n2 i konačno ako je izraz pozitivan, prelazi se na naredbu n3.
Tako, rešavanja kvadratne jednačine pomoću aritmetičke IF naredbe se može rešiti na sledeći način u programu KVJED2. Program je delimično napisan u FORTRAN77 stilu koji treba izbegavati.

```
PROGRAM KVJED2
IMPLICIT NONE
REAL A,B,C
REAL DISKR,X1,X2, KOREN
OPEN(10,FILE='ULAZKVJED.DAT')!FAJLA SA KONSTANTAMA JEDNACINE
READ (10,*) A,B,C
DISKR=B**2-4.*A*C
IF(DISKR)10,20,30
10 CONTINUE
 PRINT *, 'DISKRIMINANTA MANJA OD NULE'
 PRINT *, 'NEMA  REALNIH KORENA JEDNACINE'
GO TO 50
20 CONTINUE
 PRINT * ,'JEDNACINA IMA JEDNO DVOSTRUKO RESENJE'
  X1=-B/2./A
 PRINT *, 'X1=', X1
GO TO 50
30 CONTINUE
 PRINT *, ' JEDNACINA IMA DVA REALNA RESENJA'
   KOREN=SQRT(DISKR)
    X1=(-B+KOREN)/2./A
    X2=(-B-KOREN)/2./A
   PRINT *, 'RESENJA SU'
  PRINT *, X1
```

```
  PRINT *, X2
50 CONTINUE
END
```

Novi element je naredba CONTINUE (znači nastaviti). Ova naredba nema dejstva, pogodna je da se na nju "skoči" i da se predje na sledeći red u programu. Kao i u prethodnim slučajevima prvo se izračuna diskriminanta sistema. Zatim, se u aritmetičkoj IF naredbi ispituje znak diskriminante. Ako je on negativan odlazi se na naredbu broj 10, gde se štampa da je diskriminanta negativna i da nema realnih rešenja. Sada sledi naredba GO TO 50, da bi se preskočili ostali elementi programa. Da nema ove naredbe GO TO 50 izvršavanje programa bi krenulo na naredbu broj 20 i to bi bila greška u programu. Ovakve greške su logičke greške i njih čini programer. Nekada je vrlo teško otkriti ovakve logičke greške koje se ponekada nazivaju *bagovi* u programu.

Iz izloženog se vidi da se isti problem može rešiti na više različitih načina i sa više različitih programa. Bolji je onaj program koji je kraći, koji brže radi i koji se lakše može uključiti u druge veće i komplikovanije programe. U slučaju ovako prostih programa, kao što su gore navedeni, brzina rada i dužina računanja nisu previše važni, ali je treći zahtev veoma bitan. U programu KVJED2.90 postoji naredba GO TO i ona onemogućava uključivanje ovog programa u drugi veći program. Razlog je što naredba 50, na koju se prelazi može da bude bilo gde u tom većem programu, te se mora menjati ili taj veći program ili ovaj koji je ovde dat. Suprotno ovome, prva dva programa nemaju nikakvih skokova i mogu se bez problema uključiti u neki drugi program.

Aritmetička IF naredba je takodje stavljena u spisak zastarelih naredbi (obsolescent) fortana i treba je izbegavati pri pisanju novih programa.

III.2. STRUKTURA SELECT CASE

Ova struktura omogućuje lakše grananje programa. Može se imenovati, tj. ispred reči SELECT može se staviti ime po želji, a može se i izostaviti. Njena sintaksa je sledeća:

```
ime SELECT CASE (izraz)
 CASE (selector 1)
blok 1
 CASE (selector 2)
blok 2
.
.
CASE default
blok n
END SELECT CASE
```

Izraz je celobrojni (moguć je i drugi tip izraza, ali ne realni) - ponekada se naziva indeks select case strukture. Izvršavanje programa se prenosi na odgovarajući blok zavisno od vrednosti selectora. Prvo se izračuna vrednost celobrojnog izraza. Selektori odredjuju

intervale vrednosti, pri čemu se koriste dve tačke (:) za definisanje intervala vrednosti, tj. za razdvajanje donje i gornje vrednosti. Na primer ako se zapiše CASE (4:6) znači da ako indeks strukture ima vrednosti 4,5 ili 6 obaviće se blok koji sledi iza ove CASE naredbe. Naredba CASE (:3) obuhvata indekse strukture koji su manji od 3 (uključujući i 3). Nareba CASE (7:) obuhvata sve slučajeve kada je indeks veći ili jednak 7.

Ako indeks strukture nije ni u jednom intervalu definisanom selektorima, onda se obavlja blok koji sledi iza CASE DEFAULT.

Primer: uneti ceo broj K, a zatim izračunati vrednost K^2+K-1. Uprogramirati tri slučaja, ako je vrednost izraza veća od 100, štampati, *rezultat veći od 100*. Ako je rezultat manji od 50, štampati, *rezultat manji od 50*, i ako je izmedju 50 i 100 štampati, *rezultat je izmedju 50 i 100*. U drugom primeru rešen je zadatak obračuna poreza korišćenjem SELECT CASE strukture.

```
PROGRAM SLUCAJ
 READ*,K
SELECT CASE (K**2+K-1)
 CASE (100:)
 WRITE(*,*)'REZULTAT VECI OD 100'
 CASE (50:99)
 WRITE(*,*)'REZULTAT IZMEDJU 50 I 100'
 CASE DEFAULT
 WRITE(*,*)'REZULTAT MANJI OD 50'
 END SELECT
 END
```

```
PROGRAM OBRACUN_POREZA
IMPLICIT NONE
REAL plata, stopa, porez
INTEGER IPLATA

READ *, plata
IPLATA=INT(PLATA)

SELECT CASE (IPLATA)
CASE (:6000)    !slucaj kada je plata manja ili jednak 6000.
STOPA=0.
CASE (6001:12000)         !slucaj kada je plata veca od 6000 i manja od 12000
STOPA=0.06
CASE (12001:24000)        !plata izmedju 12001  i 24000
STOPA=0.08
CASE DEFAULT
STOPA=0.10
END SELECT
PRINT *,'STOPA POREZ', stopa
porez=plata*stopa
PRINT *, 'porez je', porez
END
```

IV CIKLIČNE STRUKTURE

IV.1. DETERMINISTIČKE DO PETLJE

Često je potrebno neko izračunavanje ponoviti veći broj puta. Ovo se u FORTRAN-u može obaviti na više različitih načina, a jedan od njih je korišćenje takozvanih DO petlji (DO znači činiti-raditi). DO petlje imaju više različitih sintaksi. Jedna od najčešće korišćenih sintaksi je

```
DO   [broj naredbe] do promenljiva=donja granica, gornja granica,[porast]
  BLOK
[broj naredbe] END DO ili CONTINUE
```

DO petlja počinje sa DO naredbom. Broj naredbe je je opcioni parametar, može se staviti, a ne mora. Ako se ne stavi broj naredbe, onda se petlja mora završiti sa END DO. Ako se stavi broj naredbe, onda se zadnja naredba petlje obeležava tim brojem. Zadnja naredba je END DO ili CONTINUE.
"do promenljiva" je takodje opciona i može da se koristi ali ne mora. Ona se još naziva indeksna promenljiva ili indeks petlje i menja se od donje do gornje granice sa korakom koji definiše "porast". Ako se porast ne definiše uzima se po default-u da je jednak 1.
Na primer naredba DO 100 J=10,20,2 znači da promenljiva J uzima vrednosti počev od 10 do 20 sa korakom 2, tj. imaće u toku računa vrednosti 10, 12,14,16,18 i 20. Pri tome izvršavaće se naredbe u bloku izmedju DO i naredbe sa brojem 100. Kada se dospe do naredbe 100 ispituje se izlazni kriterijum. U ovom slučaju izlazni kriterijum je "da li je promenljiva J dostigla vrednost 20; ako nije, program se vraća i blok izmedju DO i naredbe 100 se izvršava opet; ako jeste izlazi se iz petlje.
Pokušaj promene indeksne promenljive u samom bloku računanja je greška u programu.
Na primer u segmentu programa

```
DO I=1,100
I=I+2
END DO
```

postoji naredba kojom se vrednost indeksne promenljive I menja u samoj petlji (I=I+2). Ovakva situacija nije dozvoljena jer remeti brojanje izvršavanja petlje.

Porast indeksne promenljive može da bude i negativan. U tom slučaju "gornja granica" treba da ima manju vrednost od "donje granice", a indeksna promenljiva se menja tako da je u svakom sledećem ciklusu računanja sve manja i manja. Tako u naredbi DO I=10,5,-1 promenljiva I uzima vrednosti 10,9,8,7,6 i 5.
Broj izvršavanja petlje se dobija preko izraza

(Gornja granica-donja granica+abs(porast))/abs(porast)

U prethodnom slučaju to je (10-5+1)/1=6.

Mogući su slučajevi kada se petlja ne izvršava uopšte: to je slučaj kada je porast pozitivan, a gornja granica manja od donje. Najverovatnije radi se o greški programera. Takodje, moguće je slučaj kada se petlja izvršava beskonačan broj put. U tom slučaju napravljena je logička greška i računar ne može nikada da izadje iz petlje. Račun mora da se prekine u takvoj situaciji.

Sledeći primer ilustruje korišćenje DO petlje. Zadatak je izračunati sumu prvih N prirodnih brojeva. Jasno je da to može da se obavi naredbom SUMA=1+2+3+ . +N. Medjutim, ako se obavlja sabiranje velikog broja sabiraka onda bi ovakva naredba bila duga i nepraktična. Takodje, promenom broja N morala bi da se piše druga naredba. Korišćenje DO petlje olakšava ovo sabiranje.

Zadatak se rešava jednom DO petljom

```
DO I=1,N
SUMA=SUMA+I
END DO
```

Promenljiva I uzima sve vrednosti od 1 do N i sabira se sa prethodnom vrednošću sume, koja je celobrojna veličina u ovom programu. Kada promenljiva I dostigne vrednost N, petlja se izvrši i za I=N i izlazi se iz petlje. U ovom programu porast promenljive I je za 1, i zato se to posebno ne navodi u DO naredbi.
Ovakve petlje kod kojih postoji indeksna promenljiva nazivaju se brojački ciklusi.

Sledeći program računa sume parnih i neparnih brojeva u N prvih prirodnih brojeva. Ovo se obavlja u posebnim DO petljama u kojima je porast indeksne promenljive 2.

```
PROGRAM PRIMDO
INTEGER SUMA,SUMA1,SUMANP,SUMAP,SUMAUK,BROJNEPARNIH,BROJPARNIH
!PRIMER DO PETLJE
!RACUNA SE ZBIR PRVIH N PRIRODNIH BROJEVA
WRITE(*,*)' UNESITE KOLIKO PRIRODNIH BROJEVA (do 999)'
READ *, N
SUMA=0
DO  I=1,N
SUMA=SUMA+I
END DO
WRITE(*,20)N,SUMA
20  FORMAT(' ','SUMA',I3,2X,'PRIRODNIH BROJEVA JE =', I6,/)
WRITE(*,30)
30 FORMAT(' ','SUMA N PRVIH PRIRODNIH BROJEVA SE MOZE IZRACUNATI I PO FORMULI
SUMA=N(N+1)/2',/          )
SUMA1=N*(N+1)/2
WRITE(*,40)N,SUMA1
40 FORMAT(' ', 'SUMA PRVIH',I3,2X,'PRIRODNIH BROJEVA PO FOMULI JE=',  1I6,//)
WRITE(*,*) 'SADA SE RACUNA ZBIR NEPARNIH BROJEVA MEDJU PRVIH', N
BROJNEPARNIH=0
SUMANP=0
NEPARNI: DO  I=1,N,2
SUMANP=SUMANP+I
BROJNEPARNIH=BROJNEPARNIH+1
END DO NEPARNI
WRITE(*,50)BROJNEPARNIH,SUMANP
50  FORMAT(' ','SUMA ', I3,2X,' PRVIH NEPARNIH BROJEVA JE',I6,//)
```

```
WRITE(*,*) 'SADA SE RACUNA ZBIR PARNIH BROJEVA MEDJU PRVIH', N
BROJPARNIH=0
SUMAP=0
PARNI: DO J=2,N,2
SUMAP=SUMAP+J
BROJPARNIH=BROJPARNIH+1
END DO PARNI
WRITE(*,60)BROJPARNIH,SUMAP
60  FORMAT(' ','SUMA ', I3,2X,' PRVIH PARNIH BROJEVA JE',I6,//)
WRITE(*,*)'SUMA PARNIH I NEPARNIH TREBA DA JE JEDNAKA ZBIRU RVIH N BROJEVA'
SUMAUK=SUMAP+SUMANP
WRITE(*,*)' UKUPNA SUMA JE ',SUMAUK
END
```

Moguće je korišćenje imenovanih DO petlji. Izmedju imena petlje i DO naredbe stavljaju se dve tačke (:). U prethodnom programu imenovane DO petlje su NEPARNI: DO, i PARNI: DO u kojima se računaju sume parnih i neparnih brojeva (zapazi dve tačke). Ovakve petlje treba da se završe dodavanjem imena iza END DO, tako da stoji END DO PARNI i END DO NEPARNI (ovde se ne koriste dve tačke). Izostavljanje imena iza END DO imenovane petlje dovodi do javljanja greške pri kompiliranju programa. Imenovanje konstrukcija, uključujući i DO konstrukcije je nova mogućnost u FORTRAN90 jeziku.

U bloku koji se nalazi u DO petlji mogu se nalaziti i druge DO petlje, IF naredbe i dr. Medjutim, ne može se izaći iz petlje pre nego što se celokupna petlja ne izvrši, tj. dok indeksna promenljiva ne uzme sve zadate vrednosti. Takodje, dve DO petlje se ne mogu presecati.

IV.1.1. Naredba EXIT u do petlji

Engleska reč EXIT (što znači izlaz) se može naći u DO petlji. Pri dolasku na ovu naredbu odmah se izlazi iz petlje bez obzira na vrednost indeksne promenljive.

```
PROGRAM PRDO1
! PRIMER IZLASKA IZ DO PETLJE POMOCE EXIT NAREDBE
! RACUNANJE KVADRATA PRIRODNIH BROJEVA
PRINT *,'KOLIKO PRIRODNIH BROJEVA ZELITE, NAJVISE DO 99  '
READ *, N
PETLJA1: DO I=1,100
WRITE(*,*)I,I**2
IF(I.GE.N) EXIT PETLJA1
END DO PETLJA1
10  FORMAT(' ',9X,'BROJ',7X,'KVADRAT')
END
```

U ovom primeru dato je računanje kvadrata prirodnih brojeva (najviše do 99). Unosi se broj N- brojeva čije kvadrate treba izračunati. Računanje se obavlja u petlji PETLJA1. U ovoj petlji nalazi se naredba IF(I.GE.N)EXIT PETLJA1. Ona deluje na sledeći način: kada indeksna promenljiva dostigne vrednost N koja je zadata na ulazu, izlazi se iz petlje. Pri tome odlazi se na prvu naredbu iza END DO. Naredba EXIT omogućuje korišćenje DO petlji bez indeksne promenljive. Ovakve DO petlje se nazivaju iterativne. Petlje koje imaju indeksnu promenljivu su brojačke. Sintaksa iterativne DO petlje je

ime:DO
naredbe
IF(uslov) EXIT
naredbe
END DO ime

Ovakve petlje se izvršavaju sve dok se ne zadovolji logički uslov. Po zadovoljavanju uslova izlazi se na prvu naredbu iza END DO.

IV.1.2. Nareba CYCLE u DO petlji

Za razliku od naredbe EXIT koja omogućuje direktan izlazak iz DO petlje, naredba CYCLE omogućuje izvršavanje samo dela DO petlje. Pri nailasku na naredbu CYCLE odlazi se direktno na END DO gde se ispituje izlazni kriterijum. Na taj način deo DO petlje izmedju CYCLE i END DO se ne izvršava. Ako se zadovolji izlazni kriterijum izlazi se iz petlje, a ako ne onda se petlja izvršava od prve naredbe iza DO.

Zadatak: zadat je prirodan broj N, izračunati kvadrate brojeva koji su izmedju N/4 i N/2. Program je:

```
PROGRAM PRDO2
! PROGRAM ILUSTRUJE KORISCENJE NAREDBE CYCLE
! NAJVISE 99 PRIRODNIH BROJEVA
! PROGRAM DAJE KVADRATE PRIRODNIH BROJEVA IZMEDJU N/4 I N/2
WRITE(*,*)'KOLIKO PRIRODNIH BROJEVA'
READ *,N
DO I=2,N
IF(I<N/4.OR.I>N/2)CYCLE
WRITE(*,*)I,I**2
END DO
END
```

U programu se koristi naredba IF(I.LT.N/4.OR.I.GT.N/2)CYCLE. Ona deluje na sledeći način: ako je indeksna promenljiva I manja od N/4 ili (OR) veća od N/2 ne izvršava se deo petlje do ENDDO. U suprotnom, izvršava se pomenuti deo petlje i obavlja se računanje i štampanje brojeva i njihovih kvadrata.

4.1.3. Naredba DO WHILE

Ova naredba omogućuje drugačiji način kontrole izvršavanja DO petlje. U ovoj konstrukciji nema potrebe za indeksnom promenljivom koja se menja u toku izvršavanja programa. Reč WHILE bi se mogla prevesti kao ¨dok¨. Tako naredba DO WHILE ima smisao ¨raditi dok¨. Sintaksa ove strukture je

DO WHILE (logički izraz)
blok
END DO

Blok se izvršava dok je logički izraz zadovoljen. Kada logički izraz nije ispunjen, izlazi se iz petlje. Primer sledi

```
PROGRAM PRIMDO3
!PROGRAM DEMONSTRIRA KORISCENJE DO WHILE STRUKTURE
```

```
K=0
INDIKATOR=1
DO WHILE (INDIKATOR>0.)
WRITE(*,*) 'AKO ZELITE IZLAZ IZ PETLJE UKUCAJTE RAZLICITU VREDNOST MANJU OD 0'
READ(*,*)INDIKATOR
K=K+1
END DO
WRITE(*,*)'IMATE',K, '  UNETIH BROJEVA'
END PROGRAM PRIMDO3
```

U programu, promenljiva *indikator*, dobija početnu vrednost =1. Sve dok je ova promenljiva veća od 0 izvršava se blok izmedju DO WHILE i END DO, gde se brojač unetih brojeva K povećava za 1 i traži unos nove vrednosti indikatora. Kada se promenljivoj *indikator* da vrednost manja od 0, npr -1, još jednom se poveća vrednost brojača K i izlazi se iz petlje.

4.2. NEDETERMINISTIČKE PETLJE

FORTRAN90 omogućuje i petlje kod kojih nema indeksne promenljive koja bi se menjala u toku rada programa (ranije je ovo već napomenuto). Ovakva petlja ima sintaksu

DO
blok
END DO

Jasno je da mora da postoji naredba EXIT negde u petlji inače se nikada ne bi moglo izaći iz petlje; takva petlja bi bila beskonačna i ako se u radu udje u nju račun mora da se prekine koristeći kombinaciju na tastaturi CTRL+C. Pri tome dolazi do gubitka rezultata računanja. U nekim slučajevima ne pomaže ni CTRL+C kombinacija i računar bi morao da se resetuje.

U sledećem primeru računar zadaje jedan slučajan broj izmedju 1 i 100 a čovek (koji ne zna broj) pokušava da ga pogodi. Za kreiranje slučajnog broja koristi se generator pseudoslučajnih brojeva. Ovaj generator predstavlja kratak program koji daje jedan slučajan broj sa vrednošću izmedju 0 i 1. Slučajni brojevi su ravnomerno rasporedjeni izmedju 0 i 1 i treba da budu medjusobno nezavisni. Da bi generator slučajnih brojeva radio potrebno je ¨seme¨, tj, treba zadati prvi slučajni broj, što se još naziva inicijalizacija generatora. To se obavlja naredbom CALL RANDOM_SEED(). Ako se seme ne specificira, uzima se da je ono jednako 1 i onda pri svakom računu generator daje jedan isti niz slučajnih brojeva. Tako, moguće je niz reprodukovati, te dobijeni brojevi nisu u potpunosti nezavisni i zato se nazivaju pseudoslučajni. Pozivanje generatora brojeva se obavlja naredbom CALL RANDOM_NUMBER(rand). Slučajni broj je rand i on je $0 \le rand < 1$.

Da bi se dobio slučajni broj izmedju 1 i 100 koristi se naredba *broj*=INT(100**rand*+1). Slučajni broj se prvo množi sa 100 i dodaje se 1, a zatim se odseca decimalni deo korišćenjem funkcije INT. Sada se preko tastature unosi broj *mojbroj* i ulazi se u DO petlju. Obavlja se ispitivanje da li je izabrani broj jednak broju koji je zadao kompjuter naredbom IF(*mojbroj==broj*) EXIT, i ako je jednakost zadovoljena izlazi se iz petlje. Obratiti pažnju da se za poredjenje jednakosti (*mojbroj* **jednak** *broj*) koriste dva znaka jednakosti ==. U starijim verzijama FORTRAN-a korišćena je logička operacija .EQ. i

ovaj uslov bi bio napisan kao IF(*mojbroj*.EQ.*broj*)exit. Takodje, znak za veće je >, dok je to ranije bilo .GT. Ako korisnik izabere broj koji je veći od broja zadatog računarem onda se štampa ¨previsoko¨, a u obrnutom slučaju ¨prenisko¨. Tek kad se pogodi dati broj izlazi se iz petlje. Program broji broj pokušaja i ako je on veći od 7 malo se i šali.

```
    PROGRAM POGADJANJE
     REAL  rand
    INTEGER BROJ
! POGADJAMO BROJ
! IGRAJU RACUNAR I COVEK, RACUNAR ZADA BROJ  A COVEK POGADJA

          WRITE(*,*) 'RACUNAR ZADAJE BROJ IZMEDJU 1 I 100'
    WRITE(*,*)'UNESI broj sa 4 cifri,SEME ZA GENERATOR SLUC.BROJEVA'
           READ(*,*)ICSEED
           CALL SEED(ICSEED)
          CALL RANDOM(rand)
           WRITE(*,*)'SLUCAJNI BROJ JE =',RAND
          BROJ=INT(100*RAND+1)
C         WRITE(*,*)'BROJ',BROJ
    WRITE(*,*)'COVEK POKUSAVA DA POGODI BROJ'
          WRITE(*,*) 'UNESI CEO BROJ'
          READ *, MOJBROJ
     K=1
               DO
                  IF(MOJBROJ==BROJ)EXIT
                      IF(MOJBROJ>BROJ)THEN
                      PRINT *,  'PREVISOKO'
                    ELSE
                     PRINT *,'PRENISKO'
                     END IF
                  READ *,MOJBROJ
                   K=K+1
                   IF(K.GT.7)WRITE(*,*)'HA,HA,HA,HAA'
                 END DO
    WRITE(*,*)' TO JE TO CESTITAM'
    WRITE(*,*)'MOJ BROJ',MOJBROJ,'  TVOJ BROJ  ',BROJ
          WRITE(*,*) 'BROJ POKUSAJA JE ',K
          END
```

4.3. PETLJA U PETLJI

Moguće je da se u jednoj petlji nadje druga petlja. Ovo su ugnježdene ili ciklične petlje. Pri tome se za jednu vrednost brojača spoljašnje petlje izvršava cela unutrašnja petlja. U ranijim verzijama bilo je ograničenja koliko cikličnih struktura može da se upotrebi, dok u FORTRAN90 nema ograničenja. Korišćenje velikog broj cikličnih struktura može biti problematično i ne preporučujem više od cikličnosti 4. reda. Ako se iz unutrašnje petlje izadje sa EXIT, onda se nastavlja izvršavanje spoljašnje petlje.

ZADACI

1. Kreirati skup od zadatog broja slučajnih brojeva čije su vrednosti od 0 do 1000. Naći najveći i najmanji kreirani broj

Rešenje:

```
PROGRAM NAJVECI_NAJMANJI
!PROGRAM GENERISE ZADATI BROJ SLUCAJNIH BROJEVA IZMEDJU 1 I 1000.
!ODREDJUJE SE NAJVECI I NAJMANJI BROJ I STAMPAJU SE NJIHOVE VREDNOSTI
REAL NAJMANJI, NAJVECI
CALL RANDOM_SEED()
PRINT *, 'KOLIKO SLUCAJNIH BROJEVA'
READ *, N
NAJMANJI= 1001.
NAJVECI=-1.
      DO I=1,N
       CALL RANDOM_NUMBER(RAND)
        SLUCAJNIBROJ= RAND*1000
          IF(SLUCAJNIBROJ<NAJMANJI)NAJMANJI=SLUCAJNIBROJ
            IF(SLUCAJNIBROJ>NAJVECI)NAJVECI=SLUCAJNIBROJ
              END DO
PRINT *, 'NAJMANJI',NAJMANJI
PRINT *, 'NAJVECI',NAJVECI
END
```

Naredba CALL RANOM_SEED() omogućuje da se pri svakom narednom startovanju programa dobija drugi niz slučajnih brojeva. Za odredjivanje najmanjeg broja uzme se početna vrednost NAJMANJI=1001 koja se ne može dogoditi u toku izvršavanja. Zatm se generisani brojevi porede i kad se nadje broj manji od broja NAJMANJI, onda broj NAJMANJI uzima novu vrednost. Obrnuto se postupa pri odredjivanju najvećeg broja; kao početni najveći broj bira se neka vrlo mala vrednost koja se ne može dogoditi u toku rada. Ovde je izabrano -1.
Povećanjem broja slučajnih brojeva, najmanji se približava nuli, a najveći jedinici.

2. Napisati program koji računa sumu cifara nekog celog broja. Broj može da ima najviše do pet cifara. Sledeći program obavlja ovaj zadatak vrlo efikasno

```
PROGRAM SUMACIFARA
! PROGRAM RACUNA SUMU CIFARA NEKOG BROJA
INTEGER CIFRA, SUMA, BROJ, BROJCIFARA
SUMA=0.
PRINT *, ' UNESI BROJ CIFARA I SAM BROJ'
READ*, BROJCIFARA, BROJ
N=IABS(BROJ)
DO I=1,BROJCIFARA
 IF(N==0)EXIT
   CIFRA=MOD(N,10)
     SUMA=SUMA+CIFRA
       N=N/10
      END DO
PRINT *, BROJ, SUMA
END
```

Objašnjenje programa: *brojcifara* je ukupan broj cifara u nekom broju (na primer broj 324 je trocifren); *Broj* je broj čije cifre treba sabrati; N=IABS(BROJ) funkcija daje apsolutnu vrednost unetog broja (u postavci zadatka nije rečeno da je broj pozitivan) i ta se vrednost pridružuje promenljivi N. Sledi DO petlja koja se obavlja onoliko puta koliko ima cifara u broju. Ako se pri unošenju u READ naredbi ne podudara uneti broj cifara i stvarni broj cifara u broju, program ne radi korektno. Funkcija

MOD(N,10) daje ostatak deljenja broja N sa 10. Tako ako je unet broj BROJ=753, onda je N=753 i MOD(753/10)= 3. Time je odredjena prva cifra broja. Sledi sabiranje cifara a zatim deljenje broja N sa 10. Nova vrednost broja N je 75 (deljenje celih brojeva daje ceo broj). Sada se petlja ponavlja na isti nacin. Kao samostalan zadatak može se napraviti sličan program ali za slučaj kada broj nije ceo.

3. Kao rezultat merenja dobijen je niz rezultata (x_1,y_1), (x_2,y_2), . . ,(x_n,y_n). Pretpostavlja se da je zavisnost y od x linearna prema funkciji y=ax+b. Odrediti pravu, tj., njene parametre *a* i *b* tako da ona najbolje aproksimira dati skup merenih vrednosti.

   Poznati metod za nalaženje vrednosti a i b je da se minimizira suma kvadrata greške date kao $Greska^2 = \sum_{i=1}^{n}\left[y_i - (ax_i + b)\right]^2$.

Izraz u zagradi predstavlja razliku ordinate prave u nekoj tački x_i i merene vrednosti. Metod minimiziranja, poznat kao metoda najmanjih kvadrata daje sledeće izraze za a i b.

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \qquad \text{i} \qquad b = \frac{1}{n}\left[\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i\right]$$

Program je

```
PROGRAM METOD_NAJMANJIH_KVADRATA
!LINEARNO FITOVANJE METODOM NAJMANJIH KVADRATA
!RACUNA KOEFICIJENAT PRAVCA I SLOBODAN CLAN PRAVE
!KOJA NAJBOLJE APROKSIMIRA SKUP MERENIH REZULTATA

OPEN(10,FILE='EKSP_REZULTATI.DAT')   !FAJLA SA PODACIMA U DVE KOLONE
                                     !PODACI TREBA DA SU DATI PO PAROVIMA
Xi,Yi


SUMX=0. ; SUMY=0. ;SUMXY=0. ;SUMXSQ=0.

N=0                                  !BROJAC BROJA TACAKA
DO
READ(10,*, ERR=100) X, Y
N=N+1                !BROJI BROJ TACAKA, POVECAVA SE ZA 1 PRI SVAKOM USPESNOM
CITANJU
SUMX=SUMX + X
SUMY=SUMY + Y
SUMXY=SUMXY + X*Y
SUMSQ=SUMSQ + X**2
END DO

100 CONTINUE

A =(N*SUMXY-SUMX*SUMY)/(N*SUMSQ-SUMX**2)   !KOEFICIJENAT PRAVCA
```

```
B =1./REAL(N) *(SUMY-A*SUMX)                !SLOBODAN CLAN PRAVE

PRINT *, ' KOEFICIJENAT PRAVCA', A
PRINT *, ' SLOBODAN CLAN PRAVE', B
END
```

Merene vrednosti su smeštene u fajlu EKSP_REZULTATI.DAT odakle se učitavaju naredbom READ. Demonstrirano je čitanje nepoznatog broja podataka i njihovo prebrojavanje.

4. Napisati program za izračunavanje kvadratnog korena nekog broja po približnoj Newton-Raphson ov metodi

Po ovoj metodi sukcesivno se računa niz vrednosti na sledeći način

k= 0 $S_1=X/2$

k=1 $S_2=(S_1+X/S_1)/2$

k=2 $S_3=(S_2+X/S_2)/2$

.

.

k=n $S_{n+1}=(S_n+X/S_n)/2$

```
PROGRAM NEWTON_RAPHSON
!RACUNA KVADRATNI KOREN BROJA X PO ITERATIVNOM POSTUPKU
! Sk+1=(Sk+X/Sk)/2
! S1=X/2
PRINT *, 'UNETI BROJ'
READ *, X
PRINT *, 'UNETI ZELJENU TACNOST'
READ *, DELTAS
S1=X/2.
  DO
   S2=(S1+X/S1)/2.
    DELTA=S2-S1
     IF(ABS(DELTA)<DELTAS)EXIT
    S1=S2
  END DO
PRINT *, 'PRIBLIZNA FORMULA',S2
PRINT *, 'UNUTRASNJA FUNKCIJA', SQRT(X)
END
```

Račun se obavlja sve dok apsolutna vrednost razlike dva susedna broja ne bude manja od neke ranije zadate vrednosti. Ovakve procedure se nazivaju iteracije. Bez izlaznog uslova, ili ako je nemoguće postići izlazni kriterijum, program bi se večito vrteo u DO petlji. Izlaz iz ovakve situacije je pritisak na »CTRL C« čime se prekida rad računara.

V

JOŠ NEKI TIPOVI PROMENLJIVIH

V.1. RAD SA SLOVNIM PROMENLJIVIM (KARAKTERNE PROMENLJIVE)

U Fortranu je, kao i drugim jezicima, omogućen rad sa promenljivima koje se sastoje od slova i drugih nenumeričkih simbola. Takve promenljive se često nazivaju stringovi ili karakterne promenljive. Iako je Fortran prvenstveno namenjen numeričkim izračunavanjima, ipak je otvorena i ovakva mogućnost koja ovaj jezik čini bogatijim.

Postoje karakterne konstante i karakterne promenljive. Karakterna konstanta je bilo koji niz sastavljen od slova i drugih simbola (karaktera) koji se nalazi izmedju apostrofa. Na primer 'A', ili 'JA' ili 'BEOGRAD' i slično jesu karakterne konstante, tj stringovi. Dužina stringa je broj karaktera izmedju dva apostrofa, pri čemu se računaju i prazna mesta. Dužina prethodnih stringova je 1, 2 i 7, respektivno.

Slovne promenljive (ili karakterne promenljive) se koriste da se sačuva niz karaktera. One se definišu posebnom deklaracionom naredbom koja se navodi na početku programa pre prve izvršne naredbe. Sintaksa te deklaracione naredbe je
CHARACTER *n *lista slovnih promenljivih*.
gde je n dužina promenljivih navedenih u listi. Na primer
CHARACTER *5 *ime, a,* deklariše da su promenljive pod nazivom *ime* i *a* karakterne promenljive dužine 5. Ako se dužina neke promenljive ne specificira podrazumeva se da je njena dužina jednaka 1. Na primer
CHARACTER *z , a*3* deklariše karakternu promenljivu *z* dužine 1, i karakternu promenljivu *a* dužine 3.

U sledecem primeru definisane su dve karakterne promenljive, *odgovor* duzine 2 i *starost* duzine 8. Fortran ovde pravi razliku izmedju malih i velikih slova pa je uprogramirano da se odgovori dati malim slovima prevedu u velika.

```
PROGRAM KARAKTERI
CHARACTER *2 ODGOVOR
CHARACTER *8 STAROST
 PRINT *, 'DA LI IMAS BRATA'
PRINT *, 'ODGOVORI SA DA ILI SA NE'
READ *, ODGOVOR
IF(ODGOVOR=='da') ODGOVOR ='DA'
IF(ODGOVOR=='ne') ODGOVOR ='NE'
IF( ODGOVOR=='DA')THEN
PRINT * ,' DA LI JE ON MLADJI ILI STARIJI OD TEBE '
PRINT *, 'odgovori sa S ili sa  M ili sa B'
print *, ' B je za  blizance'
READ *, STAROST
IF(STAROST=='s') STAROST='S'
IF(STAROST=='m') STAROST='M'
IF(STAROST=='b') STAROST='B'
IF(STAROST.NE.'S'.AND.STAROST.NE.'M'.AND.STAROST.NE.'B') THEN
PRINT *, ' NEMOGUC SLUCAJ, MORA BITI STARIJI, MLADJI ILI BLIZANAC'
END IF
IF (STAROST=='S')STAROST='STARIJI '
IF (STAROST=='M')STAROST='MLADJI  '
IF(STAROST=='STARIJI  '.OR.STAROST=='MLADJI    ')PRINT *, 'TI IMAS BRATA KOJI JE ',
STAROST, ' OD TEBE'
```

```
IF(STAROST=='B')PRINT *, ' VI STE BLIZANCI'
END IF
IF( ODGOVOR=='NE')THEN
PRINT *, ' TI NEMAS BRATA'
END IF
END
```

Obratiti pažnju da se vrednost karakterne promenljive stavlja izmedju apostrofa pri ispitivanju uslova.

V.1.1.Formatirani unos/izlaz karakternih promenljivih

Za unos i štampanje karakternih promenljivih koristi se takozvani A opis. Dintaksa je An, gde n broj pozicija koje se rezervišu za unos neko karakterne promenljive. Na primer

```
CHARACTER*5 IME
READ(*,10) IME
10 FORMAT(A5)
```

Ovim je rezervisano 5 mesta za unos imena. Ovde je moguće da se desi da je broj mesta rezervisan za unos manji od dužine karakterne promenljive. U tom slučaju se odseca deo koji je van opisa formata. Na primer,

```
CHARACTER IME*6,
READ(*,10) IME
10 FORMAT(A3)
```

Čak i ako se naprimer, unese MILAN promenljiva ime dobija vrednost MIL.
U obrnutom slučaju, kada je u naredbi character definisan kraći niz nego što je broj mesta u opisu, uzima se onoliko simbola sdesna koliko je zadata dužina karakterne promenljive. Na primer

```
CHARACTER IME*3,
READ(*,10) IME
10 FORMAT(A5)
```

Ako se na ulazu unese MILAN, onda promenljiva IME dobija vrednost LAN.

V.1.2. Podstringovi

Podstring datog stringa je neki njegov deo: Na primer 'DRAGOMIR' je string a podstring je na primer 'MIR'. Podstring se izdvaja navodjenjem stringa iza koga se u zagradama navedu dva broja u sintaksi (n:m). Time se iz zadatog stringa izdvajaju svi karakteri počev od n tog do m tog. Sledeći program ilustruje izdvajanje podstringa

```
PROGRAM KARAKTERI_01
CHARACTER *8 IME1
CHARACTER *4, PODIME
READ *, IME1              ! UNETI NEKO IME DO 8 KARAKTERA
PRINT *, ' UNESI DVA CELA BROJA I<J<=8, KOJA IZDVAJAJU PODSTRING'
READ *, I, J
PODIME=IME1(I:J)
PRINT *, PODIME
END
```

Ako se za promenljivu *ime1* unese na primer BEOGRADE (ima 8 karaktera), a za I=4 i za J=7, onda promenljiva *podime* uzima vrednost GRAD. Ako je I=1 a J=3 onda je *podime* BEO.

V.1.3. Operacije sa karakternim promenljivima

Konkatenacija

Operacijom konkatenacije može se formirati jedan stringa od dva postojeća. Ovo je u stvari spajanje dva stringa. Konkatenacija se obelažava sa dve kose crte, //. Na primer BEO//GRAD daje BEOGRAD. Sledeći program ilustruje operaciju konkatenacije

```
PROGRAM KARAKTERI_02
CHARACTER PRVI*3, DRUGI*4, CEO*7
READ *, PRVI
READ *, DRUGI
CEO=PRVI//DRUGI          !KONKATENACIJA
PRINT *, CEO
END
```

Ako string prvi dobije na ulazu vrednost MIO, a string drugi dobije vrednost DRAG, onda je vrednost promenljive ceo MIODRAG. Ukoliko se pri unosu ukuca veći broj karaktera nego što je naznačeno u deklaraciji CHARACTER, višak će biti jednostavno zanemaren. U obrnutom slučaju, kada se unese manji broj podataka nego što je predvidjeno u deklaraciji, na nedostajućim mestima se podrazumevaju blanko.

Poredjenje slovnih promenljivih

Slovne promenljive se mogu porediti. Poredjenje se obavlja prema ASCII codu. Naime, svaki karakter (slova, brojevi i ostali simboli) ima odredjenu vrednost u ASCII tabeli. (ASCII cod tabela se može naći na mnogo mesta u literaturi ai u MSDEV helpu; na primer klikni na HELP, zatim na SEARCH, i ukucajte ASCII). Na primer veliko slovo A ima broj 65 u ASCII tabeli, a malo slovo a broj 97. Uzima se da je "veća" ona slovna promenljiva koja ima veću ASCII vrednost. Unutrašnja funkcija fortrana ACHAR(I) daje simbol koji ima broj I u ASCII kodu. Na primer ACHAR(59) daje za rezultat ; (tačka zarez). Takodje ACHAR(90) je Z. Slična je i funkcija CHAR(I). Obrnuto dejstvo ima funkcija IACHAR(string). Ovde se unosi string dužine 1, i funkcija daje broj tog karaktera u ASCII codu. Na primer IACHAR('P') je 90.

Ako se string sastoji od većeg broja simbola onda se sumiraju ASCII vrednosti svih simbola u tom stringu. Dva stringa sa više od jednog simbola se porede po ukupnoj ASCII vrednosti.

Funkcije poredjenja

Funkcija LGE(*string1,string2*) poredi stringove 1 i 2. Kao rezultat dobija se T (true, tačno) ili F (false, netačno). Poredjenje se obavlja prema ASCII kodu. Vrednost T se dobija ako je *string1* veći ili jednak u poredjenju sa *string2*. Ako su stringovi različite dužine onda se kraći string dopunjava praznim mestima do jednake dužine sa dužim stringom. Slična je funkcija LGT(*string1,string2*) koja na izlazu daje T ako je string 1 veći od stringa 2.

Funkcija LLE(*string1, string2*) daje na izlazu T ako je string1 manji ili jednak stringu2. Takodje funkcija LLT(*string1,string2*) daje na izlazu T ako je string1 manji od stringa 2.

V.1.4. Unutrašnje funkcije fortrana za rad sa slovnim promenljivima

Pored funkcija poredjenja i funkcija koje daju položaj simbola u ASCII kodu datih gore, postoje i ostale koje omogućuju manipulaciju sa stringovima.

Funkcija **LEN**(*imeslovnepromenljive*) daje dužinu, tj. broj karaktera date slovne promenljive.

Funkcija **INDEX**. Koristi se u sintaksi RESULTAT=INDEX(*string, podstring*). Daje poziciju u stringu odakle počinje podstring. Slovne promenljive se stavljaju izmedju apostrofa. Na primer INDEX('kragujevac','vac') daje 8.

Funkcija **LEN_TRIM**(*slovnapromenljiva*) daje stvarnu dužinu slovne promenljive, tj stvarni broj karaktera ne računajući blanko mesta.

Funkcija **REPEAT** (*string*, *n*) obavlja konkatenacija datog stringa. Kao rezultat dobija se novi string. Na primer REPEAT('*ja*',2) daje string *jaja*.

Funkcija **SCAN**(*string,set*). *String* i *set* (karakterne promenljive) zadaje korisnik, ili su rezultat prethodnog računanja. Kao rezultat dobija se prva pozicija u *string*u na kojoj se javlja bilo koji simbol zadat u set-u. Na primer SCAN(fortran,o) daje 2 jer se slovo o pojavljuje na drugoj poziciji u stringu fortran. Scan(fortran,tr) daje 3 jer se slovo r pojavljuje na trećoj poziciji. Ako nema podudaranja onda se kao reazultat dobija 0.

ZADACI

1. Zadata su dva stringa jednake dužine 5. Napisati program koji će u prvi string da umetne drugi počev od n te pozicije, i štampati rezultat. Program koji obavlja ovu operaciju je

```
PROGRAM KARAKTERI_02
CHARACTER PRVI*5, UMETNUT*5,REZULTAT*10
READ*, PRVI
READ*, UMETNUT
READ*, N                    !pozicija od koje se u prvi string umece drugi
PRINT *, PRVI
PRINT *, UMETNUT
IF(N<=1)THEN
 REZULTAT=UMETNUT//PRVI              !LEVA KONKATENACIJA
 ELSE IF(N==LEN(PRVI))THEN
 REZULTAT=PRVI//UMETNUT   !DESNA KONKATENACIJA
 ELSE
 REZULTAT=PRVI(1:N-1)//UMETNUT//PRVI(N:) !KONKATENACIJA OD N-TE POZICIJE
END IF
PRINT *, REZULTAT
END
```

2. Napisati progam koji u datom stringu briše zadati podstring i rezultat se dodeljuje novom stringu.

```
PROGRAM KARAKTERI_03
!program brise podstring u datom stringu
CHARACTER PRVI*7, BRISAN*3,REZULTAT*4
READ*, PRVI   !String iz koga ce se obrisati podstring pod imenom brisan
READ*, BRISAN           !ovaj podstring se brise i stringa prvi
N=INDEX(PRVI, BRISAN)          !otkriva od koje pozicije u stringu prvi pocinje podrsting brisan
PRINT *, N
IF(N==0) THEN
PRINT *, ' NIJE PODSTRING DATOG STRINGA, NEMA BRISANJA'
```

```
ELSE
REZULTAT=PRVI(1:N-1)//PRVI(N+LEN(BRISAN):LEN(PRVI))   !brisanje
PRINT *, REZULTAT
END IF
END
```

V.2. LOGIČKE PROMENLJIVE

Često se u radu pojavljuju veličine koje mogu da imaju samo dve vrednosti, kao na primer, tačno ili netačno; da ili ne i sl. U Fortranu je omogućen rad sa ovakvim veličinama i one se nazivaju logičke promenljive.
Logičke promenljive mogu da imaju jednu od dve vrednosti, .TRUE. ili .FALSE. (tačno ili netačno). Da bi se znalo da je neka veličina logička, potrebno je na početku programa deklarisatu tu veličinu kao logičku. Deklaraciona naredba je
LOGICAL :: lista_logičkih _promenljivih
Naredba se stavlja na početku programa pre prve izvršne naredbe.
Na primer , naredba, LOGICAL :: *tvrdjenje, a,b* definiše tri promenljice tvrdjenje, a i b kao logičke.
Logička promenljive *tvrdjenje* može biti tačna i tada se piše tvrdjenje=.TRUE. , ili netačna kada se piše tvrdjenje=.FALSE. Obrati pažnju na tačke pre i posle službene reči fortrana true i false.

Postoji nekoliko operacija koji se primenjuju na logičke promenljive. To su:
-negacija .NOT.*p* je tačna ako je logička promenljiva (ili izraz) p tačna,
-konjukcija. *a*.AND.*b* je tačna ako su obe logičke promenljive *a* i *b* tačne,
-disjunkcija *a*.OR.*b* je tačna ako je bar jedna od logičkih promenljivih *a* ili *b* tačna,
-ekvivalencija *a*.EQV.*b* je tačna ako su obe logičke promenljive tačne ili obe netačne,
-negacija ekvivalencije a.NEQV.b je netačna u onim slučajevima u kojima je ekvivalencija tačna,

Operacije nad logičkim promenljivima imaju prioritet, slično aritmetičkim operacijama. Prioritet se može menjati korišćenjem zagrada.

Operacije izmedju logičkih promenljivih i ostalih vrsta promenljivih kao što su realne ili celobrojne ne dovode do korektnih rezultata. Pokušaj mešanja logički sa drugim promenljivima izaziva pojavu greške pri radu.

Logičke promenljive se mogu naći u ulazno izlaznim naredbama. Na primer sledeći program učitava dve logičke promenljive, izvrši odredjene operacije nad njima i štampa dobijene rezultate.

```
PROGRAM LOGICKE_01
LOGICAL :: A,B,C,D,E
PRINT *, 'UNESI VREDNOSTI DVE LOGICKE PROMENLJIVE'
PRINT *, ' ZA TACNO, UNOSI SE SAMO T'
PRINT *, 'ZA NETACNO UNOSI SE SAMO F'
READ *, A,B
C=A.AND.B
D=.NOT.A
```

```
E=A.OR.B
PRINT *, 'KONJUKCIJA, A I B JE   ',C
PRINT *, 'NEGACIJA OD A JE      ',D
PRINT *, 'DISJUKCIJA, A ILI B JE ',E
END
```

Obrati paznju da se za vrednosti logičkih promenljivih unosi ili samo T (za tačno) ili samo F (za netačno). Medjutim u samom programu ne može se staviti da je neka logička promenljiva, recimo A, jednaka sa F ili sa T. Kompajler ne prepoznaje ovo F ili T kao netačno ili tačno. Da bi se u programu zadala vrednost logičke promenljive mora se zapisati sledeće
ime_log_prom =.TRUE. ili *ime_log_prom* =.FALSE.

Pri unosu ili štampanju logičkih veličina mogu se koristiti FORMAT naredbe. Opis koji se koristi za logičke veličine ima sintaksu Ln, gde je L oznaka opisa, a n je broj mesta koja se koriste za unos ili štampu date veličine.

Primer
Mika, Pera i Janko su vlasnici neke kompanije sa odredjenim procentom ucesca PM, PP i PJ. odlučivanju poštuje se pravilo da je neka odluka doneta ako za nju glasaju vlasnici koji imaju više od nekog procenata kapitala. Napraviti program koji odredjuje da li je neki predlog prihvaćen ili ne. Ulazni podaci su procenti kapitala za pojedine vlasnike i minimalna vrednost procenata akcija iznad koje se odluka smatra prihvaćenom.
Ako neki od vlasnika glasa DA za predlog, onda se to predstavlja sa .TRUE. i obratno.
Program koji obavlja ovo ispitivanje je

```
PROGRAM LOGICKE_02
!DEMONSTRACIJA KORISCENJA LOGICKIH VELICINA
LOGICAL :: MIKA, PERA, JANKO, ODLUKA
INTEGER :: BROJ_PREDLOGA
REAL :: MINIMUM
OPEN(10, FILE='PREDLOZI.DAT')
PRINT *, ' UNESI PROCENTE AKCIJA ZA TRI VLASNIKA'
READ *, PM,PP, PJ
PRINT *, MINIMALNA VREDNOST ZBIRA PROCENATA IZNAD KOJE JE ODLUKA PRIHVACENA'
READ *, MINIMUM
IF(PM+PP+PJ==100.)THEN
DO
READ(10,*, ERR=100)BROJ_PREDLOGA,MIKA,PERA,JANKO
S1=0.
S2=0.
S3=0.
IF(MIKA)  S1=PM
IF(PERA)  S2=PP
IF(JANKO) S3=PJ
SUM=S1+S2+S3
IF(SUM>=MINIMUM)THEN
ODLUKA=.TRUE.
ELSE
ODLUKA=.FALSE.
END IF
IF(ODLUKA)THEN
WRITE(*,20) BROJ_PREDLOGA, ODLUKA
```

```
ELSE
WRITE(*,21) BROJ_PREDLOGA,ODLUKA
END IF
END DO
ELSE
PRINT *, 'SUMA MORA BITI JEDNAKA 100 %'
END IF
20 FORMAT (' ' , 'PREDLOG BROJ ', I2,2X, L2, ' PRIHVACEN')
21 FORMAT (' ' , 'PREDLOG BROJ ', I2,2X, L2, ' NIJE PRIHVACEN')
100 CONTINUE
END
```

Fajla predlozi.dat sadrži predloge vlasnika kompanije u sledećoj formi

```
1  T  T  T
2  T  T  F
3  T  F  T
4  T  F  F
5  F  T  T
6  F  T  F
7  F  F  T
8  F  F  F
```

gde je prvi broj, broj predloga, a simboli T i F su glasovi za ili protiv predloga.

V.3. KOMPLEKSNE PROMENLJIVE

Iz matematike je poznato da se kompleksni broj z, definiše kao

$$z=a+ib$$

gde su a i b, neki realni brojevi a i je imaginarna jedinica i jednaka je $i=\sqrt{-1}$.
Broj *a* je realni deo, a broj *b* je imaginarni deo kompleksnog broja. U Fortranu je omogućen rad sa kompleksnim brojevima. I ranije verzije fortrana su imale istu mogućnost, ali su sada dodate neke nove funkcije u ovoj oblasti. Kompleksni broj se u fortranu tretira kao par dva realna broja. Na početku programa potrebno je naglasiti da je neka promenljiva komplesnog tipa. To se obavlja deklaracionom naredbom
COMPLEX lista_promenljivih
koja se navodi pre prve izvršne naredbe programa.
Naredba COMPLEX se može pojaviti i u implicitnoj deklaraciji
IMPLICIT COMLEX (lista početnih slova)
Sve promenljive koje počinju slovima navedenim u listu su kompleksne promenljive.

Kompleksne promenljive se mogu pojaviti u izlazno ulaznim naredbama.
Na primer

```
PROGRAM KOMP
COMPLEX A
READ *,A
PRINT *,A
END
```

Pri unosu kompleksnih veličina potrebno je ukucati dva broja. Oni se stavljaju u zagradama i odvajaju se zarezom. Tako, ako je kompleksni broj A=3+2i , na ulazu bi

se ukucalo (3., 2.). Razmak iza zareza nije neophodan. Na izlazu se kompleksni brojevi takodje štampaju u zagradama, te bi se dobilo (3.000000,2.000000).

Ukoliko se drugačije ne specificira, brojevi a i b, koji definišu kompleksni broj su obične tačnosti, tj. REAL(4). Ako se želi dvostruka tačnost brojeva a i b, onda se u deklaracionoj naredbi stavi COMPLEX(8).

V.3.1.Operacije sa kompleksnim brojevima

U fortranu su omogućene aritmetičke operacije sa kompleksnim brojevima i one su analogne odgovarajućim operacijama u matematici.
Na primer ako su definisana dva kompleksna broja z_1 i z_2; $z_1=a_1+ib_1$ i $z_2=a_2+ib_2$ onda su aritmetičke operacije definisane kao

$z_3=z_1\pm z_2= (a_1\pm a_2)+i\ (b_1\pm b_2)$
$z_4=z_1*z_2= (a_1a_2-b_1b_2)+i(a_1b_2+a_2b_1).$

$$\frac{z_1}{z_2}=\frac{a_1a_2+b_1b_2}{a_2^2+b_2^2}+i\frac{a_2b_1-a_1b_2}{a_2^2+b_2^2}$$

Pri računanju sa kompleksnim brojevima primenjuju se gornja pravila.
Pored ovako definisanih aritmetičkih operacija postoje posebne unutrašnje funkcije fortrana koje se odnose samo na kompleksne brojeve.
Funkcija REAL(Z) daje realni deo kompleksnog broja Z (to jest komponentu *a*).
Funkcija AIMAG(Z) daje imaginarni deo kompleksnog broja Z (komponenta b).
Funkcija CONJG(Z) daje konjugovano kompleksni broj broja z (razlikuje se za znak imaginarnog dela)
Funkcija CMPLX(A,B) kombinuje dva realna broja A i B u jedan kompleksan broj A+iB.
Funkcija CSQRT(Z) daje koren kompleksnog broja ako su realni i imaginarni delovi broja REAL(4).
On je jednak $(Re^2+Im^2)^{0.5}(\cos\alpha/2+i\sin\alpha/2)$, gde je $\alpha=arctg(Im/Re)$. Na primer kompleksni broj Z=1+i se u trigonometrijskoj formi predstavlja kao $Z=2^{0.5}(\cos45^0+i\sin45^0)$. Koren ovog broja je $CSQRT(Z)=2^{0.25}(\cos22.5^0+i\sin22.5^0)=$ =(1.098684,4.550899E-1).

Funkcija CDSQRT(z) računa kvadratni koren ako su realni i im. deo u dvostrukoj tačnosti.

Primer. Napisati program za rešavanje kvadratne jednačine, uzimajući u obzir da diskriminanta može biti manja od nule za neku set koeficijenata.

```
PROGRAM KOMPLEX_01
!RESAVA KVADRATNU JEDNACINU
COMPLEX DISKR, RESULT, BC, DVAA, KOREN1, KOREN2
REAL A,B,C
PRINT *, 'UNETI REALNE KOEFICIJENTE KVADRATNE JEDNACINE'
READ *, A,B,C
        IF(A==0.)THEN
        KOREN1=-C/B
        PRINT *, 'JEDNO RESENJE JE ', KOREN1
                ELSE
        DISKR=CMPLX(B**2-4*A*C,0.0)
        RESULT=CSQRT(DISKR)
        DVAA=CMPLX(2.*A,0.0)
        BC=CMPLX(B,0.0)
```

```
        KOREN1=(-BC+RESULT)/DVAA
        KOREN2=(-BC-RESULT)/DVAA
        PRINT *, 'KORENI SU ', KOREN1,KOREN2
        END IF
END
```

Pre nego što je obavljena operacija brojevi su prevedeni u kompleksne funkcijom CMPLX. Program radi i ako su rešenja realna, u tom slučaju se imaginarni deo rešenja je jednak nuli i to se dobija na izlazu.
Korišćenje kompleksnih brojeva je naročito korisno i uobičajeno u elektrotehnici.

V.4. KINDS I DVOSTRUKA TAČNOST

Koncept KINDS je nova karakteristika uvedena u FORTRAN u 90. Znatna sličnost sa FORTRAN om 77 postoji u DOUBLE PRECISION i zbog toga je ovde prvo dato objašnjenje značenja DOUBLE PRECISION.

Ranije je rečeno da se realni brojevi pamte u jednom memorijskom registru od 32 bita. Pri tome je broj značajnih cifara do 6. Medjutim, vrlo često ovo je nedovoljna tačnost. Zato je uvedena mogučnost da se relani brojevi pamte u dva memorijska registra čime je broj značajnih cifara povećan do 15. To je dvostruka tačnost. Ovo se ostvaruje naredbom
DOUBLE PRECISION lista,
gde se u listi navode imena realnih promenljivih koje su dvostruke tačnosti. Ovo je jedna od deklaracionih naredbi, kao što su REAL, INTEGER, LOGICAL, CHARACTER koje su objašnjene ranije, i kao takva navodi se na početku programa pre prve izvršne naredbe.
Tako naredba DOUBLE PRECISION A,B,X,Z definiše promenljive A,B,X,Z kao promenljive dvostruke tačnosti. Naredba DOBULE PRECISION se može naći u IMPLICIT naredbi. Moguć je zapis,
IMPLICIT DOUBLEPRECISION (A-H,O-Z). Ovim su sve promenljive koje počinju slovima od A do H i od O do Z definisane kao promenljive dvostruke tačnosti.
Koncept dvostruke tačnosti je proširen u FORTRANu90 i na ostale vrste promenljivih. Takodje uvedene je nova deklaraciona naredba koje zamenjuju DOUBLEPRECISION i koja ima isti značaj; to je naredba REAL(8). Može se postaviti pitanje zašto je ovde stavljen broj 8. Razlog je sledeći. Jedan memorisjki registar od 32 bita ima 4 BAJTA. Dva memorijska registra imaju ukupno 64 bita, ili 8 BAJTA. Tako naredba REAL(8) lista_promenljivih, znači da se premenljive navedene u listi memorišu u dva memorijska registra odnosno u 8 BAJTA.

U FORTRANu 77 podrazumevalo da su sve realne veličine obične tačnosti (ako se ne specificira da su one dvostruke tačnosti). U FORTRANu 90 se mogu pojedine veličine specificirati u običnoj tačnosti naredbom REAL(4) lista_promenljivih. Ovo je potrebno obaviti ako se isključi unutrašnja konvencija fortrana naredbom IMPLICIT NONE.

Pored deklarisanja tipa realnih promenljivih, u FORTRANu90 je uvedena mogućnost specificiranja vrste celobrojnih promenljivih. Za ovo se koriste naredbe
INTEGER(2) lista_promenljivih

i
INTEGER(4) lista_promenljivih.

Promenljive navedene u listi iza INTEGER(2) se čuvaju u dva BAJTA, tj. pola memorijskog registra, a promenljive u INTEGER(4) u 4 BAJTA odnosno u celom jednom memorijskom registru. Pored kinda 2 i 4 za cele brojeve se može koristiti i INTEGER(1).

Sada možemo da definišemo šta je to KIND. To je broj bajta koji se koriste za memorisanje neke veličine. Prevod ove reči bi bio, vrsta, sorta ili soj.
Uvedena je posebna unutrašnja funkcija fortrana čija je sintaksa KIND(X), koja kao izlaz daje kind (vrstu) ispitivane veličine X. Ovo je jedna od tzv. INQUIRY funkcija. To je posebne vrsta unutrašnjih funkcija fortrana koje ne obavljaju nikakav račun već daju informacije o argumentu. Sama reč INQUIRY bi se mogla prevesti kao *upit* ili *pitanje*. Pored funkcije KIND(X) postoje i druge INQUIRY funkcije koje su u vezi sa vrstom brojeva. Neke od njih su

HUGE(X) daje najveći pozitivni broj za vrstu brojeva kojoj pripada argument X.

EPSILON(X) daje najmanji broj koji nije zanemarljiv u poredjenju sa 1, za vrstu brojeva kojoj pripada X. Vrednost dobijena funkcijom EPSILON treba da bude najmanja vrednost veličine nekog intervala, recimo, u numeričkoj integraciji, ili pri nekom porastu i sl. Ako se izabere manja vrednost može se doći u situaciju da račun ne može da se obavi ili se obavlja pogrešno. Na primer sledeći segment programa

```
REAL(4) X
X=1.
DO
X=X+EPSILON(x)
WRITE (*,*) X
END DO
```

izaziva rast veličine X za $1.192093 \cdot 10^{-7}$ pri svakom izvršavanju petlje. Ako se u sumaciji stavi X=X+EPSILON(X)/2. veličina X stalno ostaje jednaka 1, jer je EPSILON(X)/2 premala vrednost i računar je zaokružuje na nulu.

MAXEXPONENT(X) daje stepen broja 2 koji daje najveći broj u vrsti brojeva kojoj pripada X. MINEXPONENT(X) daje najveći negativni eksponent broja 2.

TINY(X) daje najmanju pozitivnu vrednost vrste brojeva kojoj pripada X.

PRECISION(x) daje broj značajnih decimalnih mesta za vrstu brojeva kojoj pripada X.
RANGE(X) daje maksimalni decimalni eksponent za vrstu brojeva kojoj pripada X.
Ove funkcije su demonstrirane u sledećem programu.

```
PROGRAM KINDS_1
!DEMONSTRIRA RAZLICITE INQUIRY  FUNKCIJE
INTEGER(1) I
INTEGER(2) J
INTEGER(4) K
REAL(4) X
REAL(8) Y
```

```
PRINT *, '             CELI BROJEVI '
WRITE(*,10) KIND(I), KIND(J), KIND(K)
WRITE(*,20) HUGE(I), HUGE(J), HUGE(K)
PRINT *, ' '
PRINT *, '                 REALNI BROJEVI'
PRINT *,' VRSTA BROJA   ', KIND(X), KIND(Y)
PRINT *, 'NAJVECI BROJ',HUGE(X), HUGE(Y)
PRINT *, 'NEZANEMARLJIV',EPSILON(X),EPSILON(Y)
PRINT *, 'NAJVECI EKSPONENT',MAXEXPONENT(X), MAXEXPONENT(Y)
PRINT *, 'NAJMANJI EKSPONENT',MINEXPONENT(X), MINEXPONENT(Y)
PRINT *, 'ZNACAJNIH CIFARA',PRECISION(X), PRECISION(Y)
PRINT *, 'OPSEG          ',RANGE(X), RANGE(Y)
10 FORMAT (' ' ,' VRSTA BROJA',I3, 2(8X, I3))
20 FORMAT (' ' , 12X,I3, 5X, I6,5X,I12)
END
```

Većina unutrašnjih funkcija fortrana ima varijantu koja je namenjena za računanje u dvostrukoj tačnosti. Pri tome se kod nekih funkcija traži da i argument bude u dvostrukoj tačnosti, a kod nekih ne. Ukoliko se navede neodgovarajuća vrsta argumenta u unutrašnjoj funkciji može doći do javljanja greške pri kompiliranju ili pri izvršavanju programa. Moguće je da se ne javi nikakva greška ali da se račun obavlja pogrešno, što je najgori slučaj. Najčešće se funkcija dvostruke tačnosti dobija stavljanjem slova D ispred uobičajenog imena funkcije. Na primer, SIN(X) računa sinus ugla X u običnoj tačnosti. Dvostruka tačnost se dobija funkcijom DSIN(X) - argument funkcije, X mora biti dvostruke tačnosti. Slična je situacija i sa ostalim unutrašnjim funkcijama. U tabeli su date neke unutrašnje funkcije fortana u dvostrukoj tačnosti.

Funkcija	Računa	Tip argumenta	Tip rezultata
DSIND(*x)*	Sinus ugla x u stepenima	REAL(8)	REAL(8)
DSIN(x)	Sinus ugla x u radijanima	REAL(8)	REAL(8)
DASIN(x)	Arkus sinus od x	REAL(8)	REAL(8)
DTAN(X)	Tangens ugla x u radijan.	REAL(8)	REAL(8)
DCOTAN(x)	Kotangens ugla x	REAL(8)	REAL(8)
DEXP(x)	e^x	REAL(8)	REAL(8)
DLOG(x)	ln x	REAL(8)	REAL(8)
DLOG10(x)	$Log_{10}x$	REAL(8)	REAL(8)
DSQRT(x)	$X^{0.5}$	REAL(8)	REAL(8)

V.4.1. Funkcije za konverzije i manipulacije brojevima

U operaciji izmedju realnog broja obične tačnosti i realnog broja dvostruke tačnosti dobija se kao rezultat broj dvostruke tačnosti. Računar prvo prevede broj obične tačnosti u dvostruku tačnost pa nakon toga obavlja računanje. Medjutim takav način prevodjenja je nepovoljan jer se gubi u vremenu. Pored toga u nekim funkcijama koristi se tačno odredjena vrsta brojeva i potrebno je obaviti konverziju pre poziva te funkcije. U FORTRANu su omogućene funkcije koje prevode brojeve iz jedne u drugu vrstu (konverzija). Ranije smo se već sreli sa funkcijama INT(X) koja realni broj prevodi u integer, i REAL(I) i FLOAT(I) koje prevode integer u realni broj. Varijante funkcije INT su INT1, INT2 i INT4 koje kao rezultat daju cele brojeve vrste 1, 2 i 4 respektivno.
Funkcija DBLE(X) prevodi neki broj u REAL(8). Argument X može biti ceo broj, ili REAL(4).

Funkcija IDINT(D) prevodi broj REAL(8) u ceo broj odsecanjem decimalnog dela.
Funkcija SNGL(X) prevodi broj REAL(8) (dvostruke tačnost) u REAL(4) (obična tačnost). Pri tome se broj zaokružuje na 6 toj decimali. Na primer sledeći program računa broj π u dvostrukoj tačnosti (PIDVT), a kasnije se prevodi u broj obične tačnosti (PIOT). Uporedi rezultate.

```
PROGRAM BROJ_PI
REAL(4)PIOT
REAL(8)PIDVT
PIDVT=4.D0*DATAN(1.D0)
PRINT *,'BROJ PI U DVOSTRU. TACNOSTI JE JE  ', PIDVT
PIOT=SNGL(PIDVT)
PRINT *,'BROJ PI U OBICNOJ   TACNOSTI JE JE ', PIOT
END PROGRAM BROJ_PI
```

Obrati pažnju da je korišćeno 4.D0, što je zapis broja 4 u dvostrukoj tačnosti. Simbol D0 ima smisao 10^0. U ranijem tekstu za predstavljanje eksponenta broja 10 korišćen je simbol E. Tako, pisano je E+05, da bi se predstavilo 10^5. Pri korišćenju dvostruke tačnosti koristi se simbol D umesto E.

Takodje, pri računanju arctang(1) pisano je DATAN(1.D0) čime je 1 već napisano u dvostrukoj tačnosti. Funkcija DATAN(x) traži argument u dvostrukoj tačnosti i zapisi DATAN(1) ili DATAN(1.) su nekorektni, i izazvaju grešku pri kompiliranju. Korektan je i zapis DATAN(DBLE(1.)) gde se prvo broj 1 prevodi u broj dvostruke tačnosti pa se onda obavlja računanje arcustangensa.

Zadatak. Napisati program koji računa broj π prema Lajbnicovoj formuli

$$\pi = 4 \cdot (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ...)$$

U istoj petlji računati π na dva načina, u REAL(4) i u REAL(8). Uporedi rezultate. Štampati svaki 1000-ti tekući rezultat. Na ulazu zadati broj članova, N, koji se sabiraju u formuli.

Program je

```
PROGRAM KINDS_3
!RACUNA BROJ PI PO LAJBNICOVOJ FORMULI
IMPLICIT NONE
REAL(4)  PI,  CLAN                !BROJ PI I CLAN REDA U OBICNOJ TACNOSTI
REAL (8) PID, CLAND               !BROJ PI I CLAN REDA U DVOSTRUKOJ TACNOSTI
INTEGER(4) I, MULT, N
PI=1.
MULT=1
PID=1.D0
PRINT *, ' UNETI BROJ SABIRAKA '
READ *, N
          DO I=1,N
          CLAN=1./(2.*I+1)* MULT
          PI=PI-CLAN
          CLAND=1.D0/(2.D0*DBLE(I)+1.D0)*MULT
          PID=PID-CLAND
          MULT=MULT*(-1)
          IF(MOD(I,1000)==0) PRINT *,I,4*PI, 4.D0*PID
          END DO
 PRINT *, 'UZETO ', I-1 , ' CLANOVA'
PRINT *,'BROJ PI U OBICNOJ . TACNOSTI ', 4.*PI
PRINT *,'BROJ PI U DVOSTRUKOJ TACNOSTI', 4.D0*PID
END
```

Funkcija SELECTED_INT_KIND(n) daje kao rezultat kind (vrstu) celih brojeva koji je potrebno koristiti za predstavljanje broja sa n cifara. Na primer da bi se predstavili brojevi do 9999, (sa 4 cifri), dovoljno je koristiti INTEGER(2). Za brojeve veće od 9999 treba koristiti INTEGER(4). Ovde je ostavljena odredjena nepreciznost u samom FORTRANu jer je najveći ceo broj u INTEGER(2) jednak 32767 i do te vrednosti mogu se koristiti KIND =2. Ali ako se celobrojna veličina definiše sa INTEGER(2) a u toku računa ona dobije vrednost veću od 32767 računar javlja grešku i zaustavlja dalji račun. Funkcija SELECTED_INT_KIND(n) daje vrednost -1 ako je n veće 9, što znači da se ne mogu koristiti celi brojevi sa više od 9 cifara.

Funkcija SELECT_REAL_KIND(r) je slična prethodnoj ali se odnosi na relne brojeve.

Pored konverzionih i funkcija upita postoje i funkcije za *manipulaciju brojevima.* To su funkcije AINT, NINT, ANINT i dr.
Funkcija NINT(r) zaokružuje realni broj na najbliži ceo broj. Na primer NINT(5.901) je jednako 6 (ceo broj 6 bez tačke) .
Funkcija ANINT(r) odseca decimalni deo broja. ANINT(5.901) je 5.0.
Funkcija AINT(r) takodje odseca decimalni deo broja.
U ovim funkcijama rezultat je iste vrste kao i argument r. Ako se želi rezultat druge vrste onda se može iza argumenta *r*, koristiti opcioni argument KIND, kao na primer ANINT(r, KIND) ili AINT(r, KIND) pri šemu KIND definiše vrstu rezultata. Ako se stavi ANINT(r,8) onda je izlazni rezultat REAL(8).

VI
NIZOVI

VI.1. OSNOVNI POJMOVI

Često se pojavlju podaci koji su istorodni ili pripadaju nekoj zajedničkoj grupi. Takve podatke je pogodno tretirati zajedno na neki način. Zbog toga je u programskim jezicima, pa i u FORTRAN u omogućen rad sa grupom podataka. Grupa podatka se naziva niz. FORTRAN90 je obogaćen mnogim funkcijama koje se odnose samo na nizove. Smatra se da su najveća unapredjenja u odnosu na ranije verzije fortrana učinjene upravo u ovoj oblasti.

Na primer, u jednoj grupi se nalazi 100 osoba. Znamo visine tih osoba i potrebno je obaviti statističku analizu podataka o njihovim visinama. Problem koji se ovde pojavljuje je tretman velikog broja istorodnih podataka. Može se visina svake osobe označiti posebnom promenljivm, na primer, *vis1, vis2, ... vis100*. Pri računanju srednje vrednosti bilo bi potrebno napisati

srvr=(vis1+vis2+..... +vis100)/100.

Naravno da je ovakav zapis nepovoljan, jer broj podataka može biti i znatno veći od 100. Ovakav tretman može biti i praktično nemoguć ako je, recimo broj podataka nekoliko hiljada ili više. Nizovi omogućavaju lak i efikasan tretman ovakvih grupa podataka. Niz ima svoje ime koje zadaje programer i koje se podvrgava pravilima zadavanja imena ostalih promenljivih u fortranu.

Na početku programa potrebno je naglasiti da je neko ime, ime niza. To se radi naredbom:

DIMENSION *imeniza(brojelemenata)*

koja se stavlja pre prve izvršne naredbe programa. Na primer naredba,

DIMENSION *visina(100)*

definiše niz pod imenom *visina* sa ukupno 100 elemenata. Ova naredba se naziva dimenzionisanje niza i ona odredjuje ukupan broj elemenata niza. Elementi niza se pamte u susednim memorijskim registrima tako da niz zauzima kompaktan prostor u memoriji računara. Drugi način dimenzionisanja niza je bez korišćenja reči DIMENSION, ali uz deklarisanje tipa promenljive. Tako, ekvivalentna su sledeća dva zapisa,

```
REAL  VISINA
DIMENSION VISINA(100)
```

ili samo

```
REAL VISINA(100)
```

U jednoj istoj DIMENSION naredbi moguće je deklarisati više nizova. Zapis

DIMENSION A(10), B(5)

definise dva niza, A sa 10 i B sa 5 elemenata. Ako dva niza imaju isti broj elemenata moguće je zapisati ovako:

DIMENSION VISINA(100), TEZINA(100),

ali je u ovom slučaju moguć i kraći zapis uz istovremeno deklarisanje tipa promenljive koričćenjem simbola dva puta dve tačke, ::,

```
REAL, DIMENSION (100) :: VISINA, TEZINA.
```

Pri radu sa nizovima, moguće je operisati sa celim nizom, ili sa njegovim pojedinačnim elementima. Zapis VISINA(5) navodi visinu pete osobe, a zapis TEZINA(34), navodi tezinu 34. osobe. Takodje C=VISINA(11)/2. znači visinu jedanaeste osobe podeliti sa dva i vrednost pridružiti promenljivoj C (koja ne mora biti element niza). Zapisom VISINA(I) navodi se I ti elemenat niza VISINA. Promenljiva I se naziva indeks niza. Tako, računanje srednje vrednosti visine grupe od 100 osoba obavlja sledeći segment programa

```
PROGRAM SREDNJA_VISINA_GRUPE
REAL, DIMENSION(100) ::  VISINA
OPEN (10, FILE='ULAZ_VISINA.DAT')
        DO I=1,100
        READ (10,*) VISINA(I)
        END DO
SUMA=0.
        DO I=1,100
        SUMA=SUMA+VISINA(I)
        END DO
SREDNJAVISINA=SUMA/REAL(100)
PRINT *, 'SREDNJA VISINA ', SREDNJAVISINA
END
```

Podaci o visinama pojedinaca se učitavaju iz fajle ULAZ_VISINA.DAT koja je pripremljena ranije. Ukucavanje 100 podataka sa tastature bi bilo dosta nepraktično, a pored toga, greške bi bile skoro neminovne. Gornji program je rešen sa dve DO petlje, ali se može rešiti i na kraći način, (ovde je intencija na preglednost teksta).

Nizovi mogu biti različitih vrsta (KINDSa) zavisno od toga kakvi su njegovi elementi. Tako, postoje INTEGER(1), INTEGER(2), INTEGER(4), REAL(4), REAL(8), LOGICAL, COMPLEX(4), COMPLEX(8) i CHARACTER ni nizovi. Deklarisanje KINDSa obavlja se pre same reči DIMENSION. Može se pisati na primer
INTEGER(2) A1(1000) čime je definisan niz A1 kao celobrojan sa 1000 elemenata.

Veličina niza (*size*) je ukupan broj elemenata tog niza. Dobija se množenjem broja elemenata duž svih dimenzija niza.

U prethodnim deklaracionim DIMENSION naredbama definisana je samo gornja granica indesa niza; pretpostavljano je da je donja granica 1. Očekivano je da se pojedinci broje počev od broja 1. Medjutim, omogućeno je da se definiše i donja granica indeksa niza (koja je različita od 1) naredbom.

DIMENSION *imeniza(donja_granica:gornja_granica)*

Stavljaju se dve tačke izmedju donje i gornje granice indeksa niza. Naredba
DIMENSION A(-5:5), B(0:10) definiše dva niza A i B. Niz A ima 11 elemenata i indeks niza se kreće od -5 do 5 (0 je uključena). Niz B takodje ima 11 elemenata, stim što indeks niza može da uzme vrednosti od 0 do 10. Ako se ne navede donja granica niza podrazumeva se da je ona jednaka 1.

Primer
Dat je skup od 20 podataka dobijenih merenjem neke veličine. Izračunati srednju vrednost i standardnu devijaciju dobijenu merenjem.

Prema definicijama srednja vrednost, $\bar{x}$ i standardna devijacija σ su date kao

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad \text{i} \quad \sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}}$$

Program je:

```
PROGRAM STATISTIKA_1
INTEGER, PARAMETER ::BROJ=20
DIMENSION X(BROJ)
OPEN(10,FILE='INPUT.DAT')
DO I=1,BROJ
READ(10,*) X(I)
PRINT *, I, X(I)
END DO
SUMA=0.
DO I=1,BROJ
SUMA=SUMA+X(I)
END DO
SRVR=SUMA/REAL(BROJ)
PRINT *, 'SREDNJA VREDNOST JE=', SRVR
SUMAKV=0
DO I=1,BROJ
RAZLIKA=X(I)-SRVR
RAZLIKA_KV=RAZLIKA**2
SUMAKV=SUMAKV+RAZLIKA_KV
END DO
STDEV=SQRT(SUMAKV/REAL(BROJ))
PRINT *, 'STANDARDNA DEVIJACIJA JE', STDEV
END PROGRAM STATISTIKA_1
```

I ovaj zadatak se može rešiti jednostavnije primenom samo jedne DO petlje, ali uz drugačiju (ekvivalentnu) definiciju standardne devijacije. Jedan od novih elemenata u prethodnom programu je korišćenje naredbe PARAMETER BROJ=20. Posle toga se niz dimenzioniše sa brojem elemenata BROJ. Ovakav način definisanja niza omogućava lakšu promenu programa, ako je broj podataka različit od 20. Dovoljno je promeniti samu jednu vrednost u programu, i ponovo kompilirati program. Medjutim, sledeći segment programa

```
INTEGER BROJ
READ *, BROJ
DIMENSION X(BROJ)
```

ne prolazi kompiliranje jer se naredba DIMENSION javlja nakon izvršne naredbe READ. Ranije je pomenuto da se dimenzionisanje nizova obavlja pre prve izvršne naredbe. Kasnije ćemo objasniti rad sa nizovima čija se broj elemenata definiše tek u toku rada programa.

Prethodno definisani nizovi su jednodimenzionalni, odnosno imaju samo jednu indeksnu promenljivu. Takvi nizovi se često nazivaju vektori, (iako je u fizičkom smislu vektor definisan sa 3 podatka). Termin vektor ovde ima smisao jednodimenzionog niza sa brojem podataka (koji može biti različit od 3), a koji je

definisan u deklaracionoj naredbi. Pored jednodimenzionalnih nizova u FORTRAN-u je omogućen rad i sa višedimenzionalnim nizovima. Dvodimenzionalni nizovi imaju dve indeksne promenljive i u matematičkom smislu odgovaraju im matrice, trodimenzionalni imaju tri indeksne promenljive, i tako dalje. U Fortranu je omogućen rad sa nizovima do 7 dimenzija.

VI.2. JOŠ NEKI POJMOVI POVEZANI SA NIZOVIMA

Rang niza je broj dimenzija tog niza. Npr. matrica ima rang =2.

Oblik niza (shape) daje broj elemenata duž svih dimenzija nekog niza; to je jednodimenzinalni koji ima onoliko elemenata koliko dati niz ima dimenzija, a vrednosti elementi su jednaki broju elemenata ispitivanog niza u datoj dimenziji. Na primer matrica sa tri reda i tri kolone, kakve se često sreću u matematici pri rešavanju sistema od tri linearne jednačine, ima oblik (3, 3).
Postoji posebna unutrašnja funkcija koja daje oblik nekog niza. To je funkcija RESULT=SHAPE(*imeniza*)
Na primer, ako je definisana neka matrica, *A*, sa 100 x 200 elemenata onda će naredba PRINT *, SHAPE(A) dati na ekranu 100,200. Ovo je demonstrirano u sledećem programu:

```
PROGRAM OBLIKNIZA
DIMENSION :: A(100,200), B(10,20,30), C(-5:10)
PRINT *, SHAPE(A)
PRINT *, SHAPE(B)
PRINT *, SHAPE(C)
END
```

Rezultat izvršavanja ovog programa je

```
 100   200
 10  20  30
 16
```

Niz nulte veličine, je omogućen u Fortranu 90. Može se postaviti pitanje čemu služi niz koji nema ni jedan element. Jasno je da niko neće na početku programa da postavi niz nulte dužine. Medjutim, postoje slučajevi kada neka konstanta ulazi kao donja ili gornja granica indeksa niza. U toku računa može se dogoditi da su gornja i donja vrednost medjusobno jednake pa bi dužina niza bila nula. Da bi se izbegla greška koja bi se u tom slučaju neophodno javila u toku rada omogućeno je postojanje nizova nulte dužine.

VI.3. MANIPULACIJE SA CELIM NIZOVIMA-ELEMENTARNE FUNKCIJE KOJE SE ODNOSE NA NIZOVE

Svaki niz ima svoje ime i kao takvo ono predstavlja ceo niz. Na primer, deklarišu se tri niza sa po 10 elemenata REAL, DIMENSION (10) :: A,B,C

Navodjenje samo imena niza podrazumeva se ceo niz. Tako su omogućene *operacije sa celim nizovima.* Ovo inače nije bio slučaj u ranijim verzijama FORTRANa.

Na primer naredba A=B+C obavlja sabiranje svih elemenata niza B sa odgovarajućim elementima niza C, i rezultat se pridružuje elementima niza A. Tako
A(1)=B(1)+C(1), A(2)=B(2)+C(2) i tako do A(10)=B(10)+C(10).
Naredba A= B+C zamenjuje jednu DO petlju koja bi trebala da se zapiše kao

```
DO I=1,10
A(I)=B(I)+C(I)
END DO
```

Moguće su i druge algebarske operacija, kao što su oduzimanje množenje i deljenje. Operacije izmedju celih nizova su moguće ako su oni iste veličine i istog ranga, kao što je to u prethodnom slučaju (svi nizovi imaju po 10 elemenata i imaju rang 1). Ako su nizovi različite veličine ili ranga, program ne prolazi kompliranje ili se pojavljuje greška u toku izvršavanja programa.
Algebarske operacije su moguće izmedju niza i skalara. Na primer, ako su A i B imena niza, onda je moguća sledeća naredba
B=2.*A
svaki član niza A biće pomnožen sa 2 i vrednosti će biti dodeljene elementima niza B. Moguće su i druge operacije, deljenje, sabiranje oduzimanje i sl. koje se primenjuju na sve elemente niza.
Na primer, naredba A=A+1, povećava vrednost svih elemenata niza A za 1.

Ceo niz se može naći u ulazno izlaznim naredbama READ, WRITE i PRINT. Naredba READ *,A podrazumeva da će se ceo niz uneti sa tastature, redom počev od prvog do poslednjeg člana niza A. Slično naredba PRINT *, B dovodi do štampanja celog niza B, elementi se štampaju redom od prvog do zadnjeg.
Sledeći program demonstrira unos i štampu celih nizova.

```
PROGRAM SREDNJA_VISINA_GRUPE
IMPLICIT NONE
REAL, DIMENSION(5) :: A,B,C
READ *, A
B=2.5*A
C=A**B
PRINT *, A
PRINT *, B
PRINT *, C
END
```

Na početku se definišu tri niza, A, B i C sa po 5 elementa. Niz A se učitava sa tastature, naredbom READ*,A. Unos podataka nije formatiran te se mogu svi elementi uneti kako je korisniku zgodno (svi u jednom redu odvojeni praznim mestom, ili u nekoliko redova pri čemu se pritiska eneter nakon svakog reda). Niz B se dobija množenjem elemenata niza A sa 2.5, a niz C se dobija stepenovanjem elemenata niza A elementima niza B. Sva tri niza se štampaju na ekranu. Pri tome se prvo odštampa ceo niz A, pa zatim ceo niz B i na kraju ceo niz C.
Ako se radi o dvodimenzionalnim ili višedimenzionalnim nizovima treba imati u vidu specijalni način na koji se u ovakvim slučajevima niz tretira u fortranu. Naime, leviji indeksi se brže menjaju od desnih. Ako je, na primer neki niz A dvodimenzionalan, recimo ima oblik (3,2), onda naredba READ*, A podrazumeva sledeći redosled unosa podataka
A(1,1),A(2,1),A(3,1),A(1,2),A(2,2),A(3,2)

Drugim rečima, matrice se unose kolona po kolona, a ne vrsta po vrsta.
Redosled unošenja se može promeniti na sledeći način
READ *, ((A(I,J), J=1,3),I=1,2)
ovo se naziva i implicitna DO naredba. Izvršava se tako što indeks I uzme vrednost 1, a indeks J se menja od 1 do 3, a zatim indeks I dobije vrednost 2 i onda se opet J promeni od 1 do 3. Ovaj zapis je ekvivalentan sledećem zapisu

READ *, A(1,1),A(1,2),A(1,3)
READ *, A(2,1),A(2,2),A(2,3)

Niz se može biti argument nekih elementarnih unutrašnjih funkcija fortrana (ne svih). Na primer naredba SIN(A) računa sinuse svih elemenata niza A. Ista je situacija i sa ostalim trigonometrijskim funkcijama. Takodje EXP(A) računa eksponente svih članova niza A.

Pored elementarnih unutrašnjih funkcija čiji argumenti mogu biti nizovi, postoje posebne funkcije koje se odnose samo na nizove. Jedna od njih je gore pomenuta funkcija SHAPE.

Funkcija RESHAPE

Ova naredba preoblikuje niz. Sintaksa je
RESULT= RESHAPE(izvorni_niz,shape)
Izvorni niz je ime niza koji se preoblikuje. Shape je oblik niza u koji će dati niz biti preoblikovan. RESULT je ime novo-dobijenog niza. Pri preoblikovanju izvorni niz se pretvori u jednodimenzionalni niz, (tako da se leviji indeks menja brže od desnog, što u slučaju dvodimenzionalnog niza znači kolona po kolona) a zatim se tako dobijeni elementi pridružuju novom nizu, (takodje uz poštovanje da se leviji indeks menja brže).
Primer preoblikovanja niza je dat u sledećem programu.

```
PROGRAM PRIMER_PREOBLIKOVANJA
IMPLICIT NONE
REAL A (2,3) ,B(3,2)
INTEGER I,J
READ *, ((A(I,J), J=1,3),I=1,2)     !unosenje vrsta po vrsta implicitnom DO naredom
PRINT *, A(1,1),A(1,2), A(1,3)      !stampa prvu vrstu matrice A
PRINT *, A(2,1),A(2,2), A(2,3)      !stampa drugu vrstu matrice A
B=RESHAPE(A,(/3,2/))                !pretvara niz 2x3 u niz 3x2
PRINT *, B(1,1),B(1,2)              !prva vrsta niza B
PRINT *, B(2,1),B(2,2)              !druga  vrsta niza B
PRINT *, B(3,1),B(3,2)              !treca  vrsta niza B
END
```

Ako se za niz A unesu sledeće vrednosti, 1,2,3,4,5,6, onda matrica izlgeda kao

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Transformacija ove matrice kolona po kolona u jednodimenzionalni niz daje

$A_{transformiran}$=(1 4 2 5 3 6)

Sada se ovi brojevi pridružuju matrici B, takodje kolona po kolona te je matrica B

$$B = \begin{bmatrix} 1 & 5 \\ 4 & 3 \\ 2 & 6 \end{bmatrix}$$

Konačno, se na izlazu se matrica B štampa vrsta po vrsta i dobija se

1 5
4 3
2 6.

Naredba RESHAPE takodje ima veći broj opcija koje ovde nećemo sve da obradjujemo.

Nalaženje najvećeg i najmanjeg elementa u nizu

Često je potrebno odrediti najveći ili najmanji element nekog niza. Postoje posebne elementarne unutrašnje funkcije koje ovo izvršavaju. Najveća vrednost dobija se funkcijom

MAXVAL(ime_niza)

a najmanja vrednost

MINVAL(ime_niza).

Naredba PRINT *, MINVAL(A),MAXVAL(A)

štampa najmanju i najveću vrednost niza A (bez obzira na oblik, veličinu i rang tog niza).

Ako je niz jednodimenzionalni štampa se jedna vrednost koja je najveća u tom nizu. Ako je niz višedimenzionalni može se specificirati dimenzija duž koje se traži najveći element. Ako je niz A dvodimenzionalni iz prethodnog primera, onda naredba

PRINT *,MAXVAL(A,1) dovodi do štampanja tri vrednosti, koje predstavljaju najveće brojeve duž kolona - ovde broj 1 predstavlja prvu dimenziju. Naredba PRINT *, MAXVAL(A,2) dovodi do štampanja dve vrednosti koje su najveće duž vrsta matrice A.

Ukoliko se ne specificira dimenzija duž koje se traži najveća vrednost, onda se dobija najveća vrednost u celom nizu.

Slično razmatranja važe i za funkciju MINVAL, s tim što se dobijaju najmanje vrednosti.

Sumiranje niza. Omogućena je posebna funkcija za sabiranje elementa niza. Sintaksa funkcije je SUM (ime_niza) ako je niz jednodimenzionalan. U tom slučaju kao rezultat se dobija suma svih članova tog niza.

Ako je niz višedimenzioni, onda se sumiranje može obaviti duž odredjene dimenzije tog niza. Naravno može se sumirati ceo niz. Sledeći program demonstrira sumiranje jednodimenzionalnog niza, kao i sumiranje dvodimenzionalnog niza po vrstama i kolonama.

```
PROGRAM SUMIRANJE_NIZA
IMPLICIT NONE
INTEGER NIZ(10), B(2,5)
```

```
INTEGER SUMA, I,J
DATA NIZ/1, 3, 5, 3, 4, 2, 5, 6, 7, 8 /
WRITE(*,5) NIZ
SUMA=SUM(NIZ)
PRINT *, 'SUMA POCETNOG NIZA JE ' , SUMA
B=RESHAPE(NIZ,(/2,5/))
PRINT *, ' PREOBLIKOVAN NIZ 2x5 JE '
WRITE(*,10) ((B(I,J),J=1,5),I=1,2)

WRITE(*,*)'SUMA NIZA JE ', SUM(B)
WRITE(*,15) SUM(B,DIM=1)  !SABIRANJE PO KOLONAMA
PRINT *, ' '
WRITE(*,20)SUM(B,DIM=2) !SABIRANJE PO VRSTAMA
PRINT *, ' '
PRINT *, 'PROIZVOD NIZA JE ', PRODUCT(B)
PRINT *, ' '
WRITE(*,25)PRODUCT(B,DIM=1)          ! PROIZVOD PO KOLONAMA
PRINT *, ' '
WRITE(*,30) PRODUCT(B,DIM=2)         ! PROIZVOD PO VRSTAMA
PRINT *, ' '
5 FORMAT (' POCETNI NIZ',10(I3,2X))
10 FORMAT (5(I3,2X))
15 FORMAT(' SUMA PO KOLONAMA',5(I5,2X))
20 FORMAT(' SUMA PO VRSTAMA', 2(I5,2X))
25 FORMAT(' PROIZVOD KOLONA NIZA', 5(I6,2X))
30 FORMAT(' PROIZVOD VRSTA NIZA ',2(I5,2X))
END
```

Ako je niz dvodimenzioni, onda funkcija SUM(B,DIM=1) dovodi do sumiranja po prvom indeksu, (drugi indeks je konstantan) a to je sumiranje po kolonama.
Reč DIM označava dimenziju niza duž kojeg se obavlja sumiranje i sama reč DIM se može izostaviti.

Proizvod niza se dobija funkcijom PRODUCT(ime_niza,dimenzija). Ime niza je niz čiji se svi članovi množe duž odredjene dimenzije. Ova funkcija je demonstrirana takodje u programu Sumiranje_niza.f90.

Korišćenje maske. U prethodnim funkcijama, jedna od opcija je takozvana maska. Maska je logička promenljiva (može da ima dve vrednosti T ili F) takodje je niz koja treba da ima isti oblik, rang i veličinu kao i niz u kome se koristi. Navodjenje reči MASK izaziva proveru svakog elementa niza pojedinačno. Ako dati element zadovoljava uslov zadat u maski onda se elementu niza MASK pridružuje vrednost T, a u suprotnom vrednost F. Drugim rečima formira se logički niz istog oblika, ranga i veličine kao i niz koji se ispituje; na mestima gde element niza zadovoljava logički uslov postavljas se T. Zapis PRODUCT(B,DIM=1, MASK=B<4) se izvršava na sledeći način. Računa se proizvod elemenata niza duž svih kolona, ali se u obzir računaju samo oni elementi koji su manji od 4. Množe se samo elementi koji zadovoljavaju uslov zadat u MASK. Oni elementi koji su ≥4 se zamenjuju jedinicom pri množenju. Pri sumiranju funkcijom SUM, elementi koji ne zadovoljavaju uslov zadat u maski se zamenjuju nulom. MASK se može upotrebiti i u mnogim drugim funkcijama.

Funkcije MINLOC i MAXLOC

Ove funkcije daju položaj najmanjeg i najvećeg elementa niza. Pod polažajem se podrazumeva vrednost indeksa ekstremnog elementa. Sintaksa je MINLOC(ime_niza, maska) i MAXLOC(ime_niza, maska).

Funkcija TRANSPOSE

Ovom funkcijom se obavlja transponovanje dvodimenzionalnog niza; primenjuje se samo za dvodimenzionalne nizove, tj. matrice. Transponovanje matrice podrazumeva zamenu vrsta kolonama i obratno. Program TRANSPO demonstrira transponovanje matrice 3x3.

```
PROGRAM TRANSPO
REAL A(3,3), B(3,3), C(3,3)
READ *, ((A(I,J), J=1,3),I=1,3)  !UCITAVA MATRICU A VRSTU PO VRSTU
B= TRANSPOSE(A)                  !TRANSPONUJE MATRICU A U MATRICU B
DO I=1,3
WRITE (*,10) (B(I,J), J=1,3)
END DO
PRINT *, 'PROIZVOD MATRICA A I B JE'
C=MATMUL(A,B)                    !MATRICNO MNOZENJE A PUTA B
WRITE (*,10) ((C(I,J), J=1,3),I=1,3)
10 FORMAT (3(' ' ,F7.3,2X))
END
```

Skalarni proizvod dva vektora

Prema definiciji, skalarni proizvod vektora a i b je $\vec{a}\ i\ \vec{b}$ je

$$\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y + a_z b_z$$

U Fortanu je omogucena funkcija koja obavlja ovakvo mnozenje. To je interna funkcija DOT_PRODUCT(niz1,niz2). Broj elemenata mora biti isti u oba niza. Obratiti paznju da je ovde rezultat jedan broj, kao i kod skalarnog proizvoda stvarnih vektora. Rezultat obicnog mnozenja dva niza NIZ1*NIZ2 je treci niz sa istim brojem elemenata kao i nizovi koji se mnoze, pri čemu su njihovi elementi izmnoženi.

Funkcija MATMUL

Funkcija MATMUL obavlja mnozenje matrica. Sintaksa je ime3=MATMUL(ime1,ime2), gde su ime1 i ime2 imena dva dvodimenzionalna niza koja se matrično množe. Rezultat se pridružuje trećoj matrici pod nazivom ime3. Matrice koje se množe moraju da zadovoljavaju uslov množenja matrica, tj da je broj kolona prve matrice jednak broju vrsta druge. U suprotnom javlja se greška pri kompiliranju. Jedna od matrica koje se množe može da bude jednodimenzionalna, ali ne obe. Množenje matrica je takodje demonstrirano u programu TRANSPO.

Funkcija CSHIFT

Ova funkcija obavlja cirkularno premeštanje članova niza. Sintaksa je, CSHIFT(imeniza, shift, dim). Ako je shift +1, onda se izvršava levi shift, a ako je shift =-1 onda je desni shift. Obavezan je unos +1 ili -1 za vrednost pomeraja, (tj shifta). Na primer

```
DATA NIZ/1, 3, 5/
PRINT *, CSHIFT(NIZ, 1)
```

daje na ekranu 3 5 1 . Jedinica je prebačena iza petice, kao da je niz pomeren ulevo.
Takodje

PRINT *, CSHIFT(NIZ, -1) na ekranu daje 5 1 3. Petica je prebačena ispred jedinice i ostatak niza je pomeren udesno.
Kod jednodimenzionalnih nizova ne definiše se dimenzija duž koje se obavlja cirkularni pomeraj. Kod višedimenzionalnih nizova definiše se dimenzija na koju se odnosi cirkularni pomeraj

Funkcija PACK
Sintaksa ove funkcije je PACK(imeniza,MASK). Izvršava se tako što se svi elementi niza navedenog u PACK funkciji koji zadovoljavaju logički uslov zadat u MASK smeste u jednodimenzionalni niz.

U svim prethodnim slučajevima MASK je logički niz istog oblika kao i niz koji se tretira. Funkcija se tretira u onim slučajevima kada je MASK tačna.
Osobina maske kao i funkcija PACK su demonstrirani u programu PAKOVANJE.

```
PROGRAM PAKOVANJE
REAL  A(3,3)
LOGICAL MASK(3,3)
DATA A/1., 4., 7., 2., 5., 8., 3., 6., 9./
WRITE(*,10) ((A(I,J),J=1,3),I=1,3)
MASK= A>3.
WRITE(*,15) ((MASK(I,J),J=1,3),I=1,3)
PRINT *, PACK(A, MASK)
10 FORMAT( 3(' ' , F6.3, 2X))
15 FORMAT( 3(' ' , L3,2X))
END
```

Pored navedenih unutrašnjih funkcija postoji još odredjeni broj njih, ali ih nećemo obradjivati sve u detalje. Ove funkcije omogućuju lake i jednostavne odgovore na složena pitanja o podacima smeštenim u neku tabelu bez potrebe primene komplikovanih grananja i DO petlji. Dat je primer obrade rezultata studenata.

Primer: Četri studenata je radilo pet različitih testova. Rezultati su dati od 0 do 100 u obliku tabele, gde se jedna vrsta odnosi na jednog studenta, a jedna kolona na jedan test. Na primer rezultati prvog studenta su na prvom testu 34, a na drugom 66, na trećem 54 i td. Rezultat trećeg studenta na drugom testu je 49. ..

$$TEST = \begin{bmatrix} 34 & 66 & 54 & 68 & 33 \\ 67 & 88 & 75 & 89 & 91 \\ 43 & 49 & 51 & 59 & 32 \\ 86 & 81 & 93 & 96 & 76 \end{bmatrix}$$

Može se postaviti veći broj različitih pitanja.
1. Štampati tabelu prolaza, ako je za prolaz potrebno više od 50 poena. Ovo se rešava uvodjenjem logičke promenljive MASK(4,5) i naredbom MASK=TEST≥50.
2. Koja je najbolja ocena pojedinih studenta. Ovo se rešava funkcijom MAXVAL(TEST,DIM=2). Na izlazu se dobija 68 91 59 96 što su najbolje ocene pojedinih studenata na bilo kojem od testa.

3. Koja je najslabija ocena pojedinih studenata na testovima. Ovo se rešava funkcijom MINVAL(TEST,DIM=2). Dobija se 33 67 32 76.
4. Koja je srednja ocena nekog studenta. Ovde se primenjuje sumiranje i deljenje sa brojem testova SUM(TEST, DIM=2)/FLOAT(5).
5. Koja je srednja ocena cele grupe studenata na svakom od testova. SUM(TEST, DIM=1)/FLOAT(4).
6. Koji je srednji rezultat cele grupe studenata na svim testovima. SUM(TEST)/SIZE(TEST).
Program PRIMER_NIZ1 obavlja sve ove operacije

```
PROGRAM PRIMER_NIZ1
INTEGER TEST(4,5)
LOGICAL MASK(4,5)
OPEN(10,FILE='REZULTATI.DAT')
READ (10,*) ((TEST(I,J), J=1,5), I=1,4)
MASK= TEST>=50
WRITE(*,15)((MASK(I,J),J=1,5),I=1,4)
WRITE(*,20) MAXVAL(TEST,DIM=2)
WRITE(*,25) MINVAL(TEST, DIM=2)
WRITE(*,30) SUM(TEST,DIM=2)/FLOAT(5)
WRITE(*,35) SUM(TEST, DIM=1)/FLOAT(4)
WRITE(*,40) SUM(TEST)/FLOAT(SIZE(TEST) )
15 FORMAT( 5(' ',L2,2X))
20 FORMAT(' ','NAJBOLJE OCENE STUDENATA SU',    5(' ' I3,2X))
25 FORMAT(' ','NAJSLABIJE OCENE STUDENATA SU',  5(' ' I3,2X))
30 FORMAT(' ','SREDNJE OCENE STUDENATA SU ',        4(' ' F7.2,2X))
35 FORMAT(' ','SREDNJE OCENE PO TESTOVIMA SU ', 5(' ' F7.2,2X))
40 FORMAT(' ','SREDNJA OCENA CELE GRUPA NA SVIM TESTOVIMA ', F7.2)
END
```

Kao što se vidi nema nikakvih DO petlji, posebnih ispitivanja, poredjenja niti grananja u ovom programu što bi inače bio slučaj u prethodnim verzijama FORTRANa.

VI.3.1. Sortiranje niza

Sortiranje niza, odnosno uredjivanje po rastućim ili opadajućim vrednostima je značajno pitanje u programiranju. Sortiranje većih nizova traži dosta računarskog vremena. Postoji veći broj algoritama za sortiranje sa različitom efikasnošću. U mnogim problemima i zadacima potrebno je obaviti sortiranje niza. U fortranu je omogućena posebna unutrašnja funkcija koja obavlja sortiranje jednodimenzionalnog niza po rastućim vrednostima. Sintaksa ove funkcije je
CALL SORTQQ(adresa_niza, broj,vrsta_niza)
adresa niza je broj memorijskog registra koji sadrži dati niza i dobija se unutrašnjom funkcijom LOC(ime_niza),
broj je broj elemenata niza koji se sortira,
i konačno, vrsta niza se dobija po sledećem rasporedu:

Ako je niz celobrojan onda se stavlja SRT$INTEGER2
Ako je niz realan obićne tačnosti stavlja se SRT$REAL4
Realan niz dvostruke tačnosti stavlja se SRT$REAL8.
Ovde ima još mogučnosti zavisno od vrste podataka u nizu ali još uvek nismo obradili sve postojeće tipove podataka.

Primer. Dati niz od 10 elemenata sortirati po rastući vrednostima korišćenjem naredbe SORTQQ. Zatim od tog niza formirati drugi niz po opadajućim vrednostima. Štampati rastući i opadajući niz.

```
PROGRAM SORTIRA_NIZ
USE MSFLIB
INTEGER(2)  NIZ(10), NIZOPADAJUCI(10)
INTEGER(2)  N

DATA NIZ/14, 9, 6, 16, 17, 19, 5, 15, 22, 4/
!    SORTIRA NIZ
CALL SORTQQ (LOC(NIZ),10,SRT$INTEGER2)
!STAMPA RASTUCI NIZ
DO N = 1, 10
PRINT *, N, NIZ(N)
END DO
!FORMIRA OPADAJUCI NIZ
DO N=1,10
NIZOPADAJUCI(10-N+1)=NIZ(N)
END DO
PRINT *,  '  '
!STAMPA OPADAJUCI NIZ
DO N = 1, 10
PRINT *, N, NIZOPADAJUCI(N)
END DO
END
```

Naredba USE MSFLIB omogućuje korišćenje biblioteka *microsoft fortran library*. Jedna od funkcija iz ovih biblioteka je i SORTQQ koja obavlja sortiranje niza. Bez ove naredbe program ne bi mogao da funkcioniše i sortiranje se ne bi obavljalo. Naredbom DATA NIZ/14, 9, 6, 16, 17, 19, 5, 15, 22, 4/ zadaju se vrednosti elemenata niza NIZ.

VI.4. DINAMIČKI NIZOVI

U ranijim verzijama fortrana korišćeni su samo statički nizovi. Broj njihovih elemenata je morao da bude poznat pre početka izvršavanja programa. Jedna od mogućnosti delimičnog prevazilaženja ovog problema je pokazana u primeru ranije, gde se broj dimenzija definiše u naredbi PARAMETER, a zatim se niz dimenzioniše sa tim brojem. Medjutim ovakvo dimenzionisanje zahteva naknadnu kompilaciju program i nepogodno je. Jedna od glavnih kritika starijih verzija FORTRANa je bila nemogućnost rada sa nizovima čiji je broj elemenata nepoznat pre početka izvršavanja programa. U FORTRAN90 omogućen je rad sa dinamičkim nizovima čija se dimenzija odredjuje ili zadaje u toku izvršavanja programa. Na početku izvršavanja programa, pre prve izvršne naredbe navodi se lista dinamičkih nizova iza naredbe ALLOCATABLE. Sintaksa je sledeća

```
ALLOCATABLE :: ime1(:), ime2(:,:), ime3(:,:,:)
```

Takodje je moguće deklarisati i tip nizova, na sledeći način

```
REAL, ALLOCATABLE  :: lista_imena_nizova
```

ili

INTEGER, ALLOCATABLE :: lista_imena_nizova
i sl.

Ime1,ime2 i ime3 su imena nizova. Iza imena koriste se dve tačke u zagradi; ako je navedeno samo (:) onda to znači da je niz jednodimenzionalan, kao što je ime1, (:,:) znači da je niz dvodimenzionalan, to je ime2 i td. Pri definisanju dinamičkih nizova, deklariše se njihov rang (tj. broj dimenzija).
U samom programu mora se definisati broj elemenata u svakoj dimenzija dinamičkih nizova. Ovo se obavlja naredbom

ALLOCATE (IME1(N1), IME2(N2,N3), IME3(N4,N5,N6))
N_I su broj elemenata po pojedinim dimenzijama. Imena nizova se obavezno stavljaju u zagradama. Brojevi N se mogu odrediti nekim računom u programu ili mogu biti zadati naredbom READ. Pokušaj korišćenja dinamičkog niza čija veličina nije odredjena u ALLOCATE izaziva pojavu greške pri izvršavanju programa.

Primer. Koordinate nepoznatog broja tačaka u ravni su date kao parovi brojeva u fajli Podaci.dat. Odrediti donju tačku (najmanja vrednost ordinate), gornju tačku (najveća vrednost ordinate), naj-leviju tačku (tačku sa najmanjom apscisom), naj-desniju tačku (tačku sa najvećom apscisom). Koja je tačka najudaljenija od koordinatnog početka. Program koji ovo izvršava je

```
PROGRAM PRIMER_NIZ2
REAL, ALLOCATABLE :: X(:),Y(:), RASTOJANJE(:)
INTEGER LOKACIJA(1)
OPEN(10, FILE='PODACI.DAT')
I=0
DO
READ(10,*,ERR=20)XLAZI
I=I+1
END DO
20 N=I
PRINT *, ' UKUPNO TACAKA ',N
PRINT *, ' '
ALLOCATE (X(N),Y(N),RASTOJANJE(N))
REWIND 10
DO I=1,N
READ(10,*) X(I),Y(I)
END DO
RASTOJANJE=SQRT(X**2+Y**2)
LOKACIJA=MINLOC(X)
PRINT *, 'NAJLEVIJA TACKA', X(LOKACIJA(1)),Y(LOKACIJA(1))
LOKACIJA=MAXLOC(X)
PRINT *, 'NAJDESNIJA TACKA', X(LOKACIJA(1)),Y(LOKACIJA(1))
LOKACIJA=MINLOC(Y)
PRINT *, 'DONJA TACKA', X(LOKACIJA(1)),Y(LOKACIJA(1))
LOKACIJA=MAXLOC(Y)
PRINT *, 'GORNJA TACKA', X(LOKACIJA(1)),Y(LOKACIJA(1))
LOKACIJA=MAXLOC(RASTOJANJE)
PRINT *, 'NAJUDALJENIJA TACKA JE', X(LOKACIJA(1)),Y(LOKACIJA(1))
PRINT *, ' I NA RASTOJANJU JE ', MAXVAL(RASTOJANJE),'  OD KOORD. POCETKA'
END
```

Prva DO petlja služi da prebroji koliko ima ukupno tačaka. Koristi se lazna veličina xlazi. Pri svakom uspešnom čitanju broj I se povećava za 1. Kasnije se nizovi dimenzionišu prema ukupnom broju koordinata u naredbi ALLOCATABLE. Kasnije su demonstrirane funkcije MINLOC i MAXLOC koje nalaze položaj minimalnog i maksimalnog elementa niza. Obrati pažnju da se kao rezultat primene funkcije MAXLOC na niz X dobija drugi jednodimenzionalni niz nazvan LOKACIJA koji ima onoliko elemenata koliko niz X ima dimenzija. Kako su X i Y jednodimenzionalni, to niz LOKACIJA ima samo jedan elemenat.

Primeri
Dat je niz B sa 5 elemenata. Učitati drugi niz A sa nepoznatim brojem elemenata. Štampati samo one elemente niza B kojih ima i u nizu A.

```
PROGRAM PRIMER_NIZ3
INTEGER, ALLOCATABLE:: A(:)
INTEGER  B(5)
LOGICAL  D(5)
D=.TRUE.
DATA  B/3, 5, 7, 9, 11/
PRINT *, ' KOLIKO ELEMENATA IMA NIZ KOJI SE POREDI SA B'
READ *, N
ALLOCATE ( A(N))
PRINT *, ' UNESI ELEMENTE NIZA A'
READ *, A
 DO I=1,5
  DO J=1,N
   IF(A(J)==B(I))THEN
    IF(D(I)) PRINT*, B(I)
     D(I)=.FALSE.
    END IF
  END DO
 END DO
END
```

Koristi se logički niz, D sa 5 elemenata da se označi da je neki elemenat niza B već štampan.

VI.5. MANIPULACIJA SA DELOVIMA NIZA. KONSTRUKTORI NIZOVA

Pored manipulacije sa celim nizom, otvorena je i mogućnost rada sa delovima (segmentima) niza, takodje bez primene DO petlji.
Na primer niz A ima 10 elemenata. Početne vrednosti tog niza se mogu zadati u DATA naredbi
DATA A/1.,2.,3.,4.,5.,6,.7.,8,.9.,10./
Ovo je jedna vrsta konstrukcije niza. Neki drugi niz B sa 5 elemenata zadat je narebom B=A(3:7)
Ovim je segment niza A pridružen nizu B, tako da je B(1)=A(3), B(2)=A(4), B(3)=A(5) B(4)=A(6) i , B(5)=A(7). Tako, niz B ima elemente, 3., 4., 5., 6., 7.
Primer manipulacije sa delovima nizova dat je u sledećem programu Primer_niz4

```
PROGRAM PRIMER_NIZ4
REAL, DIMENSION(10):: A,E
REAL, DIMENSION (2,5):: B
REAL, DIMENSION (5):: C,D,F
DATA A/1.,2.,3.,4.,5.,6.,7.,8.,9.,10./
```

```
PRINT *, ' POLAZNI VEKTOR JE '
WRITE(*,20)A
PRINT *, ' '
B(1,1:5)=A(6:10)
B(2,1:5)=A(1:5)
PRINT *, ' MATRICA B JE '
WRITE(*,10) ((B(I,J),J=1,5),I=1,2)
PRINT *, ' '
C=B(1,1:5)        !VEKTOR C JE PRVA VRSTA MATRICE B
PRINT *, 'VEKTOR C JE'
WRITE(*,10) C
D=B(2,1:5)        !VEKTOR D JE DRUGA VRSTA MATRICE B
PRINT *, ' '
PRINT *, 'VEKTOR D JE'
WRITE(*,10) D
E=(/C,D/)
PRINT *, ' '
PRINT *, ' VEKTOR E JE'
WRITE(*,20)E
PRINT *, ' '
F=E(3:7)
PRINT *, ' VEKTOR F JE'
WRITE(*,10)F
10 FORMAT( 5(' ',F4.1,2X))
20 FORMAT( 10(' ',F4.1,2X))
END PROGRAM PRIMER_NIZ4
```

Pri inicijalizaciji mogu se koristiti kraći zapisi ako se neke vrednosti ponavljaju. Na primer
REAL A(100)
DATA A/100*1./
Svi elementi niza A imaju vrednost =1. Još kraći zapis je samo
A=1.
Medjutim, ako nisu svi elementi niza jednaki medjusobno može se zapisati i ovako.
REAL A(100)
DATA A/60*1.,40*2./
Prvih 60 elemenata niza dobija početnu vrednost 60 a drugih 40 elemanata vrednost 2. Ovo je moglo da se zapiše preko konstruktora
A(1:60)=1. i A(61:100)=2.

Konstrukcija se može obaviti i sa dvodimenzionalnim (i višedimenzionalnim) nizovima. Korišćenje ovakvih konstrukcija u potpunosti odstranjuje potrebu korišćenja DO naredbi. Vrednosti segmenta jednog niza se mogu direktno pridružiti elementima drugog niza.
Na primer data su dva niza A(3,5) i B(4,3). Grafički su predstavljeni u sledećim tabelama.

A11	A12	A13	***A14***	***A15***	B11	B12	B13
A21	A22	A23	***A24***	***A25***	B21	***B22***	***B23***
A31	A32	A33	***A34***	***A35***	B31	***B32***	***B33***
					B41	***B42***	***B43***

Naredba B(1:3,1)=A(2,1:3) pridružiće elemente niza A nizu B na sledeći način B11=A21; B21=A22 i B31=A23 (zadimljeni delovi tabela). Takodje naredba B(2:4,2:3)=A(1:3,4:5) zadaje vrednosti elemenata niza B na sledeći način
B(2,2)= A(1,4) , B(2,3)=A(1,5)
B(3,2)= A(2,4) , B(3,3)=A(2,5)
B(4,2)= A(3,4) , B(4,3)=A(3,5)

```
PROGRAM PRIMER_NIZ4
REAL  A(3,5), B(4,3)
B=0.
DATA A/1.,2.,3.,4.,5.,6.,7.,8.,9.,10.,11.,12.,13.,14.,15/
PRINT *, 'MATRICA A JE'
WRITE(*,10) ((A(I,J),J=1,5),I=1,3)
B(2:4,2:3)=A(1:3,4:5)
PRINT *, ' MATRICA B JE'
WRITE(*,20)((B(I,J),J=1,3),I=1,4)
10 FORMAT ( 5(' ' ,F5.2))
20 FORMAT ( 3(' ' ,F5.2)); END
```

VI.5.1.Poredjenje nizova.

Nizovi se mogu porediti- ovo možda izgleda malo neobično, kako porediti dve strukture sa većim brojem elemenata. Poredjenje se obavlja element po element. Pri tome nizovi moraju da imaju isti broj elemenata i tip podataka mora biti isti. Kao rezultat poredjenja dobija se treci logički niz koji sadrži rezultat poredjenja (njegovi elementi su T ili F).
Na primer definisana su dva niza A i B sa po 5 elemanata
DATA A/1.,2.,3.,4.,5./
DATA B/7., -2., -4., 4., 3./
Naredba PRINT *, A<B daje na izlazu T F F F F
PRINT *, A==B , na izlazu daje F F F T F
Moguć je i zapis C=A>B gde je C niz definisan u deklaracionoj naredbi kao logički sa 5 elemenata.

Primer. Neki niz pod imenom NIZ se formira u programu tako što se slučajni brojevi množe sa 1000 nakon čega se uzima samo ceo deo tako dobijenog broja. Niz je nesortiran i vrednosti elemenata se ne pojavljuju više od jedanput. Drugi niz, pod imenom NELEMENT sa manjim brojem elemenata unosi korisnik. Traži se da li se neki element drugog niza sadrži u prvom nizu. Štampati položaj tog elementa u nizu NIZ, i sam taj element. Program je

```
PROGRAM PRIMER_NIZ7
IMPLICIT NONE
!TRAZENJE VREDNOSTI U NEKOM NIZU
INTEGER, ALLOCATABLE :: NIZ(:),NELEMENT(:)
INTEGER N, NTRAZI
INTEGER  I, J, MOGUC
REAL SLBROJ
LOGICAL NASO
PRINT *,  'KOLIKO CLANOVA IMA NIZ- DOZVOLJENO DO 1000 ELEMENATA'
READ *, N
PRINT *, 'KOLIKO SE ELEMENATA TRAZI'
READ *, NTRAZI
```

```
ALLOCATE ( NIZ(N),NELEMENT(NTRAZI) )
CALL RANDOM_SEED()
CALL RANDOM_NUMBER(SLBROJ)
NIZ(1)=INT(SLBROJ*1000)
I=2
DO
NASO=.FALSE.
CALL RANDOM_NUMBER(SLBROJ)
        MOGUC=INT(SLBROJ*1000)
        DO J=1,I-1              !petlja eliminise one elemente koji su se ranije vec pojavili.
        IF(MOGUC==NIZ(J))NASO=.TRUE.
        END DO
IF(NASO)CYCLE
NIZ(I)=MOGUC
I=I+1
IF(I>N)EXIT
END DO
PRINT *, 'FORMIRAN NIZ'
WRITE(*,10) NIZ
10 FORMAT (10(I4,2X))
        PRINT *, 'UNESI ', NTRAZI,'  ELEMENATA KOJI SE TRAZE'
        READ(*,*) (NELEMENT(I),I=1,NTRAZI)

NASO=.FALSE.
DO I=1,NTRAZI
J=1
DO
IF(NIZ(J)==NELEMENT(I))THEN
NASO=.TRUE.
PRINT *,'ELEMENT BROJ',I,NELEMENT(I),'  NADJEN NA POZICIJI', J
EXIT
END IF
J=J+1
IF(J>=N)EXIT
END DO
END DO
IF(.NOT.NASO)PRINT *, 'NIJE NASO NI JEDAN ELEMENAT U NIZU'
END
```

Ovakva vrsta traženja naziva se linearna pretraga. Ona se mora primeniti kada niz nije sortiran. Pri traženju u sortiranom nizu mogu se primeniti efikasniji metodi kao što je binarno pretraga.

Zadatak. Napisati program u slučaju kada se elementi niza NIZ mogu ponavljati više puta.
Binarna pretraga se primenjuje u slučaju kada je niz sortiran. Pri tome se niz prvo podeli u dva dela i zatim se ispita da li je traženi broj veći ili manji od središnog elementa niza. Tako se odredi u kome je delu broj i odrede se granice (tj indeksi prvog i zadnjeg elementa tog dela niza). Zatim se deo u kome se broj nalazi deli opet na pola i postupak se nastavlja dok se razlika izmedju gornje i donje granice ne dovede do 1. Onda se ispituje da li je broj jednak donjoj ili gornjoj granici ili se nalazi izmedju njih. U programu koji ovo obavlja primenjeno je formiranje niza pomoću slučajnih brojeva kao i u prethodnom primeru. Korisnik može da izmeni program i da unese niz po želji.

```
PROGRAM PRIMER_NIZ8
USE MSFLIB
```

```
IMPLICIT NONE
!TRAZENJE VREDNOSTI U NEKOM NIZU
INTEGER, ALLOCATABLE :: NIZ(:)
INTEGER N,  NELEMENT
INTEGER  I, J, MOGUC, PRVI,ZADNJI, SREDNJI, POLOZAJ
REAL SLBROJ
LOGICAL NASO
PRINT *,  'KOLIKO CLANOVA IMA NIZ- DOZVOLJENO DO 1000 ELEMENATA'
READ *, N
ALLOCATE ( NIZ(N))
!CALL RANDOM_SEED()  !aktivirati ako se želi drugi niz pri svakom izvršavanju.
CALL RANDOM_NUMBER(SLBROJ)
NIZ(1)=INT(SLBROJ*1000)
I=2
DO
NASO=.FALSE.
CALL RANDOM_NUMBER(SLBROJ)
        MOGUC=INT(SLBROJ*1000)
                DO J=1,I-1                  !eliminise ponavljanje istog elementa
                IF(MOGUC==NIZ(J))NASO=.TRUE.
                END DO
IF(NASO)CYCLE
NIZ(I)=MOGUC
I=I+1
IF(I>N)EXIT
END DO
CALL SORTQQ(LOC(NIZ), N, SRT$INTEGER4)
PRINT *, 'FORMIRAN NIZ'
WRITE(*,10) NIZ
10 FORMAT (10(I5,2X))
PRINT *, 'UNESI BROJ KOJI SE TRAZI U DATOM NIZU'
READ(*,*) NELEMENT
NASO=.FALSE.
PRVI=1
ZADNJI=N
NASO=.FALSE.
I=1
 DO                          !Binarna pretraga
   SREDNJI=(PRVI+ZADNJI)/2          !deli niz na dva dela
        IF(NIZ(SREDNJI)>NELEMENT)THEN
        ZADNJI=SREDNJI
        ELSE
        PRVI=SREDNJI
        END IF
        IF(ZADNJI-PRVI==1)THEN
                IF(NELEMENT==NIZ(ZADNJI)) THEN
                POLOZAJ=ZADNJI
                NASO=.TRUE.
                EXIT
                ELSEIF(NELEMENT==NIZ(PRVI))   THEN
                POLOZAJ=PRVI
                NASO=.TRUE.
                EXIT
          ELSE
          NASO=.FALSE.
          POLOZAJ=PRVI
PRINT *, 'BROJ',NELEMENT,' JE IZMEDJU ',PRVI,ZADNJI, '   ELEMENTA  U NIZU '
        END IF
        EXIT
        END IF
```

```
END DO
IF(NASO) PRINT *,'BROJ ', NELEMENT, ' JE NA MESTU ',POLOZAJ,' U NIZU'
END
```

Program može da se reši i prostrije ako se zna da traženi broj sigurno postoji u nizu. U gornjim primerima traženja porede se celi brojevi. U slučaju poredjenja realnih brojeva mogu da nastupe odredjeni problemi povezani sa ograničenom tačnošću sa kojom se brojevi predstavljaju u računaru.

Eratostenovo sito.

Prosti brojevi su celi pozitivni brojevi koji su deljivi samo sa jedinicom i samim sobom. Takvi su na primer brojevi 2, 3, 5, 7 i td. U matematici je pokazano da ne postoji najveći prost broj, tj, postoji beskonačno mnogo prostih brojeva. Broj 2 je takodje prost jer je deljiv samo sa 1 i sam sa sobom. To je jedini paran broj koji je prost. Svi ostali parni brojevi nisu prosti. Nalaženje prostih brojeva može se obaviti linearno, tj ispitivati da li je broj deljiv sa nekim brojem koji je manji od njega. Medjutim ovakav pristup je neefikasan. Jedan od načina nalaženja prostih brojeva u nekom intervalu brojeva je primena tzv. Eratostenovog sita. Znamo da je broj 2 prost i ne treba da ga dalje ispitujemo, te tako u nizu u kome čuvamo proste brojeve prvo smestimo broj 2. Sve ostale parne brojeve ne moramo dalje da testiramo jer sigurno nisu prosti. Sada programiramo petlju koja testira sve naparne brojeve u zadatom intervalu i koja nalazi proste brojeve. Nije potrebno testirati brojeve da li su deljivi sa 1 i sa 2 (jer testiramo samo neparne breve). Prvi broj sa kojim testiramo je broj 3. Ako se broj 3 sadrži u ispitivanom broju nije potrebno dalje ispitivanje jer broj nije prost. Ako broj nije deljiv sa 3 sledi ispitivanje deljivosti sa sledećim brojem. Najveći broj sa kojim testiramo je kvadratni koren ispitivanog broja. Program je dat

```
PROGRAM PRIMER_NIZ9
!EREATOSTENOVO SITO
INTEGER, ALLOCATABLE :: PROST(:)
INTEGER  N,DELILAC,NP, NKOREN
LOGICAL NIJEPROST
PRINT *, 'UNESI DONJU I GORNJU GRANICU ISPITIVANOG INTERVALA'
PRINT *, ' DONJA GRANICA TREBA DA JE NEPARAN BROJ'
READ *, NDONJI, NGORNJI
INTERVAL=NGORNJI-NDONJI+1
ALLOCATE (PROST(INTERVAL))
PROST=0
PROST(1)=2
PROST(2)=3
PROST(3)=5
PROST(4)=7
NP=4   !BROJAC PROSTIH BROJEVA
DO N=NDONJI, NGORNJI,2                  !PETLJA MENJA BROJEVE KOJI SE TESTIRAJU
NKOREN=(N)**0.5
NIJEPROST=.FALSE.
DO DELILAC=3,NKOREN,2                   !PETLJA MENJA BROJEVE KOJIMA SE DELI
        IF(MOD(N,DELILAC)==0)THEN
        NIJEPROST=.TRUE.
        EXIT
        END IF
        END DO
IF(NIJEPROST)CYCLE
```

```
NP=NP+1
PROST(NP)=N
END DO
WRITE(*,90)
90 FORMAT( ' ','REDNI BROJ',3X,  'PROST BROJ')
DO I=1,NP
WRITE(*,100) I, PROST(I)
100 FORMAT (' ' 5X, I3, 2X, I5)
END DO
        END
```

Ovaj primer demonstrira jednu od mogućih upotreba nizova; niz se koristi kada je potrebno sačuvati neku izračunatu vrednost za kasniji rad. U ovom slučaju čuvane su vrednosti prostih brojeva.

Zadatak. Data su dva dvodimenzinalna niza dimenzija (3,4). Napisati program koji će da zameni sadržaj ta dva niza.
Da bi se rešio ovaj zadatak potrebno je uvesti treći niz Z gde se privremeno čuvaju vrednosti jednog od dva niza, na primer niza X. Zatim se vrednosti niza Z pridruže nizu X i na kraju se vrednosti niza Z (što je inicijalno bio niz X) pridruže nizu Y. Vrednosti elemenata nizova X i Y uneti iz fajle ULAZ.DAT.

```
PROGRAM PRIMER_NIZ10
REAL  X(3,4),Y(3,4), Z(3,4)
OPEN(10,FILE='ULAZ.DAT')
READ(10,*) )(X(I,J),J=1,4), I=1,3)
READ(10,*) (Y(I,J),J=1,4),  I=1,3)
Z=X
X=Y
Y=Z
PRINT *, ' '
WRITE(*,*)((X(I,J),J=1,4),I=1,3)
WRITE(*,*)((Y(I,J),J=1,4),I=1,3)
END
```

Zadatak. Koordinate (x,y,z) nepoznatog broja tačaka u prostoru su date u nekoj fajli *Koordinate.dat*. (u jednom redu su tri koordinate jedne tačke) Učitati koordinate tačaka i sortirati ih prema rastućoj vrednosti koordinate z (prostije rečeno poredjati tačke po visini).

Sortiranje jednodimenzionalnog niza se može obaviti korišćenjem naredbe SORTQQ, što je objašnjeno ranije. Medjutim, u ovom problemu postoje vezani podaci (related data) tako da bi nakon sortiranja niza koordinata po Z trebalo ispitivati koordinate koje koordinate x i y pripadaju odgovarajućoj vrednosti koordinate z i obavljati potrebna premeštanja. Kako su koordinate realni brojevi može doći do problema pri ispitivanju jednakosti i zato je ovde potrebno primeniti drugi način sortiranja. Sortiranje koje je dato u sledećem programu se naziva BUBBLE SORT, ili mehurasto sortiranje. Naziv je dobilo jer se brojevi premeštaju u toku sortiranja prema početku ili kraju niza zavisno od njihove vrednosti. Niz se prolazi najviše onoliko puta koliko ima brojeva, ili dok se svi brojevi ne poredjaju kako treba. Uvedena je logička promenljiva PREMESTIO koja nosi informaciju da je neki element niza premešten i niz se ponovo prolazi. Ako ova logička promenljiva ima vrednost .FALSE. onda to znači da nije bilo premeštanja u nizu, tj. niz je već uredjen. U programu se ispituju vrednosti koordinate z i ona se prvo premešta, a zajedno sa njom premeštaju se i

vrednosti koordinata x i y. Pre izvršavanja programa treba formirati ulaznu fajlu sa koordinatama odredjenog broja tačaka.

```
PROGRAM SORTIRANJE
LOGICAL PREMESTIO
ALLOCATABLE :: X(:), Y(:),Z(:)
OPEN(10, FILE='KOORDINATE.DAT')      !koordinate tacaka, jedna tacka u jednom redu
OPEN(25, FILE='IZLAZ.DAT')           !pise sortirane podatke
I=0
DO  !broji koliko ima tacaka
READ(10,*,ERR=15) XCITA
I=I+1
END DO
15 PRINT *, 'Ima ukupno tacaka ',I
N=I
ALLOCATE (X(N),Y(N),Z(N) )
REWIND 10
DO I=1,N                    !petlja ucitava koordinate tacaka
READ(10,*)X(I),Y(I),Z(I)
END DO
DO                          !spoljasnja petlja,
PREMESTIO=.FALSE.
DO I=1,N-1                  !unutrasnja petlja, premesta elemente.
        IF(Z(I).GT.Z(I+1)) THEN
        TEMP=Z(I)
        Z(I)=Z(I+1)
        Z(I+1)=TEMP
        TEMP1=Y(I)
        Y(I)=Y(I+1)
        Y(I+1)=TEMP1
        TEMP2=X(I)
        X(I)=X(I+1)
        X(I+1)=TEMP2
        PREMESTIO=.TRUE.
        END IF
END DO
IF(.NOT.PREMESTIO)EXIT
END DO
DO I=1,N
WRITE(25,*)X(I),Y(I),Z(I)
END DO
END
```

Unutrašnja DO petlja ispituje da li je jedan element niza Z veći od sledećeg elementa u tom nizu. Ako jeste, onda oni zamene mesta u nizu Z. Zajedno sa tom promenom vrši se i zamena mesta u nizovima X i Y. Pri zameni, logički prekidač PREMESTIO dobija vrednost .TRUE. čime se spoljašnja petlja opet izvršava.
Algoritam bubble sortiranja je dosta neefikasan i pri radu sa većim nizova zahteva dosta računarskog vremena. Postoje efikasniji algoritmi za sortiranje. Jedan od mogućih načina je korišćenje MAXLOC funkcije, koja otkriva položaj maksimalnog elementa u nizu. Program koji ovo obavlja je

```
PROGRAM PRIMER_NIZ12
!sortiranje pomocu funkcije MAXLOC
INTEGER P(1)
ALLOCATABLE :: X(:), Y(:)
PRINT *, 'UNESI BROJ ELEMENATA NIZA X'
```

```
READ *, N
ALLOCATE (X(N),Y(N))
DO I=1,N
READ*, X(I)
END DO
XMIN=MINVAL(X)
DO I=1,N
P=MAXLOC(X)
Y(N-I+1)=X(P(1))
X(P)=XMIN-1.
END DO
PRINT *, ' '
PRINT *, Y
END
```

Funkcija Maxloc otkriva polozaj najveceg elementa u nizu, i ta vrednost se čuva kao jedini elemenat jednodimenzionalnog niza P (*p*olozaj). Taj element se smešta u drugi niz Y koji se puni počev od najveće vrednosti. Sada se primenjuje ¨varka¨ odnosno, vrednost najvećeg elementa se smanji za 1 ispod minimalnog elementa niza tako da u sledećem prolasku funkcija MAXLOC otkriva sledeći najveći element i smešta ga u niz Y. Slično, niz se može sortirati po opadajućim vrednostima korišćenjem funkcije MINLOC.
Predlaže se studentu da uporedi efikasnost (to jest utrošeno računarsko vreme) tri vrste sortiranja, BUBBLE, SORTQQ i pomoću funkcije MAXLOC za sortiranje niza od većeg broja elemenata, recimo $5 \cdot 10^4$ ili $5 \cdot 10^5$. Veći broj brojeva se može lako kreirati korišćenjem generatora slučajnih brojeva RANDOM_NUMBER, čime se otklanja potreba ukucavanja enormno velikog broja brojeva.

Niz može biti i karakternog tipa. U sledećem primeru demonstrirano je korišćenje karakternih nizova. Zadatak se sastoji u sledećem: uneti imena i prezimena 5 studenata i njihove ocene na nekom ispitu. Klasifikovati ih po uspehu (najbolji da je na vrhu liste i štampati rezultat).

```
PROGRAM PRIMER_NIZ13
CHARACTER*6,IME(5),PREZIME(5), TEMP_IME, TEMP_PREZIME
INTEGER OCENA(5), TEMPO
LOGICAL PREMESTIO
DO I=1,5
READ *, IME(I),PREZIME(I),OCENA(I)
END DO
!DO I=1,5                    !AKTIVIRATI AKO SE ZELI PROVERA UNOSA
!PRINT *, IME(I),PREZIME(I),OCENA(I)
!END DO
DO                  !BUBBLE SORT PO OPADAJUCIM VREDNOSTIMA
PREMESTIO=.FALSE.
DO I=1,4
IF(OCENA(I)<OCENA(I+1))THEN
TEMPO=OCENA(I)
TEMP_IME=IME(I)
TEMP_PREZIME=PREZIME(I)
OCENA(I)=OCENA(I+1)
IME(I)=IME(I+1)
PREZIME(I)=PREZIME(I+1)
OCENA(I+1)=TEMPO
IME(I+1)=TEMP_IME
PREZIME(I+1)=TEMP_PREZIME
PREMESTIO=.TRUE.
```

```
END IF
END DO
IF(.NOT.PREMESTIO)EXIT
END DO
PRINT *, ' '
DO I=1,5
PRINT *, IME(I),PREZIME(I),OCENA(I)
END DO
END
```

VII

IZVEDENI TIPOVI PODATAKA

Do sada smo upoznali ukupno 6 tipova podataka koji si mogući u FORTRANU: Integer, Real(4), Real(8), Logical, Complex i Character. Medjutim u Fortranu 90 je omogućeno da korisnik kreira svoje sopstvene tipove podataka. To su izvedeni tipovi podataka, i oni se nazivaju još i tipovi kreirani od strane korisnika. Oni se mogu koristiti za kombinovanje više vrsta podataka pod jednim imenom. Izvedeni tipovi podataka omogućuju pogodnu manipulaciju nad skupom povezanih podataka. Nizovi omogućuju manipulaciju većeg broja podataka, ali je nedostatak što svi moraju da budu istog tipa. U izvedenim tipovima podataka moguće je definisati više komponenti koje mogu biti različite vrste. U ranijim verzijama Fortrana to je bilo moguće kombinacijom postojećih naredbi, ali nije bilo mogućnosti direktnog kreiranja novih vrsta podataka. Ova mogućnost se smatra značajnim unapredjenjem u odnosu na FORTRAN77. Na primer, pod imenom hemijskog jedinjenja može se smestiti više podataka kao atomski broj, relativna atomska masa, tačka topljenja, i mnogi drugi podaci.

Naredba koja omogućuje kreiranje izvedenog tipa podataka je TYPE. Sintaksa je

TYPE ime
. . . . .
END TYPE ime

Umesto ..... stavljaju se deklaracione naredbe iza kojih dolaze imena promenljivih.
U slučaju hemijskih elemenata to bi izgledalo ovako:

```
PROGRAM TIP_01
        TYPE ELEMENT
        CHARACTER(10)IME
        INTEGER(2) Z
        REAL(4) MASA
        END TYPE ELEMENT
TYPE (ELEMENT):: EL(5)
EL(1)%IME='H'
EL(1)%Z=1
EL(1)%MASA=1.003
EL(2)%IME='He'
EL(2)%Z=2
EL(2)%MASA=4.013
EL(3)%IME='Li'
EL(3)%Z=3
EL(3)%MASA=7.013
EL(4)%IME='Be'
EL(4)%Z=4
EL(4)%MASA=9.013
EL(5)%IME='B'
EL(5)%Z=5
EL(5)%MASA=10.013
write(*,10) EL(1)
write(*,10) EL(2)
write(*,10) EL(3)
write(*,10) EL(4)
write(*,10) EL(5)
10 format(' ', A3, 2X, I3, 2X,F6.3)
END
```

Naredbom TYPE ELEMENT definisan je novi tip promenljive pod imenom ELEMENT. Taj novi tip ima tri komponente, prva je karakternog tipa, i to je IME elementa, druga komponenta je celobrojna Z i to je (atomski broj) i treća je realna promenljiva masa. Ovim je samo definisana matrica kako izgleda tip ELEMENT, ali on još uvek nije definisan.
Definisanje ELEMENTA obavlja se naredbom TYPE (ELEMENT) :: EL(5) definiše niz od pet elementa EL(1), EL(2), EL(3), E(4) i EL(5). Ovo se još naziva i definisanje strukture. Svaki od ovih elemenata ima tri komponente, ime, atomski broj i masu kako je definisano u naredbi TYPE ELEMENT.

Sledi zadavanje podataka za svaki od ovih elemenata, i kao primer uzeto je prvih pet elementa periodnog sistema.

Zadavanje vrednosti se obavlja naredbom EL(1)%IME='H' . Obratiti pažnju da se stavlja znak % izmedju EL(1) i IME. Takodje ime se 'H' se stavlja izmedju apostrofa jer je karakternog tipa. Moraju se zadati vrednosti svih komponenti koje čine datu promenljivu ELEMENT.

Na kraju se obavlja štampanje podataka o svim definisanim elementima. Da bi se štampa obavila dovoljno je navesti samo Print *, EL(1) čime se odštampaju vrednosti svih komponenti za prvi element. U gornjem primeru korišćen je formatirani izlaz.

Mogu se obavljati računske operacije sa pojedinim komponentama izvedenog tipa promenljive. Sledeći program ilustruje primenu izvedenih tipova podataka na obračun plate za 5 radnika neke kompanije.

```
PROGRAM TIP_02
!PROGRAM OBRACUNA PLATE
TYPE PLATA
        CHARACTER(10)       IME
        CHARACTER(10)       PREZIME
        INTEGER(4)          STAZ
        INTEGER(4)          BODOVA
        REAL(4)                     POREZ
        REAL(4)                     IZNOS_BRUTO
        REAL(4)                     IZNOS_NETO
END TYPE PLATA
TYPE (PLATA):: RADNIK(5)
RADNIK(1)%IME    ='MARKO'
RADNIK(1)%PREZIME ='MARKOVIC'
RADNIK(1)%STAZ   =23
RADNIK(1)%BODOVA  =150

RADNIK(2)%IME    ='PERA'
RADNIK(2)%PREZIME ='PETROVIC'
RADNIK(2)%STAZ   =12
RADNIK(2)%BODOVA  =180

RADNIK(3)%IME    ='ZORICA'
RADNIK(3)%PREZIME ='JEKIC'
RADNIK(3)%STAZ   =33
RADNIK(3)%BODOVA  =95


RADNIK(4)%IME    ='STOJA'
```

```
RADNIK(4)%PREZIME ='JELIC'
RADNIK(4)%STAZ    =2
RADNIK(4)%BODOVA  =205

RADNIK(5)%IME     ='IVAN'
RADNIK(5)%PREZIME ='JOVIC'
RADNIK(5)%STAZ    =18
RADNIK(5)%BODOVA  =125

PRINT *,'UNESI VREDNOST BODA'
READ*, VREDNOSTBODA

DO I=1,5
RADNIK(I)%IZNOS_BRUTO=RADNIK(I)%BODOVA*VREDNOSTBODA+RADNIK(I)%STAZ*
0.02*VREDNOSTBODA
END DO

DO  I=1,5
J=INT(RADNIK(I)%IZNOS_BRUTO)
IF(J<10000) RADNIK(I)%POREZ= 0.05
IF(J>=10000.AND.J<20000) RADNIK(I)%POREZ= 0.1
IF(J>=20000) RADNIK(I)%POREZ= 0.15
RADNIK(I)%IZNOS_NETO=RADNIK(I)%IZNOS_BRUTO*(1.-RADNIK(I)%POREZ)
END DO

WRITE(*,5)
5 FORMAT (' ' , 'RB',2X, ' IME',8X, 'PREZIME', 2X,' STAZ', X, 'BOD', 2X,'POREZ', 2X,'BRUTO PL', 3X,'NETO PL')
DO I=1,5
WRITE(*,10)I, RADNIK(I)
END DO
10 FORMAT(I3,2X,A10,2X,A10,I3,2X,I3,2X,F5.2,2X,F10.2,2X,F10.2)
END
```

Prvo se definiše nova vrsta promenljive, nazvana PLATA. Ona ima 7 komponenti, ime radnika, prezime radnika, ukupan staž radnika, broj bodova za dato radno mesto, porez, bruto plata i neto plata. Naredbom TYPE (PLATA):: RADNIK(5) definiše se niz nazvan RADNIK sa 5 elemenata, i svaki od 5 radnika je okarakterisan sa 7 prethodnih komponenti. Sledi deo programa gde se zadaju vrednosti pojedinih komponenti za svakog radnika. Posle toga računa se bruto plata kao proizvod broja bodova i vrednosti jednog boda, uvećano za 0.02 % za svaku godinu staža. Porez se obračunava po progresivnoj skali 5 %, 10 % ili 15 % zavisno od bruto plate. Na kraju se iznos poreza oduzima od bruto plate i računa se neto plata. Na kraju se štampaju rezultati. Obrati pažnju da se jednom naredbom PRINT *, RADNIK(I) štampaju vrednosti svih 7 pripadajućih komponenti. Ovde se mogu ugraditi i ostali elementi pri računanju plata kao što su razni doprinosi, krediti i ostalo, dodavanjem novih komponenti u izvedenom tipu podataka PLATA. Računanje je krajnje jednostavno i može se obaviti u svega par redova. Primenjen je formatirani izlaz radi preglednosti ali se ona može poremetiti ako se primene jako velike ili jako male vrednosti boda. Takodje, broj radnika se može povećati, a unos podataka obaviti iz fajle, a ne direktno u programu. Unos iz fajle bi izgledao ovako

```
OPEN(100, FILE='ULAZ_PLATA.DAT')
DO I=1,5
READ(100,*) RADNIK(I)%IME, RADNIK(I)%PREZIME, RADNIK(I)%STAZ, RADNIK(I)%BODOVA
END DO
```

a sama fajle Ulaz_plata.dat je

```
MARKO        MARKOVIC     23  150
PERA         PETROVIC     12  180
```

```
ZORICA          EKIC            33   95
STOJA           JELIC            2   205
IVAN            JOVIC           18  125
```

Obrati pažnju da karakterne promenljive ne moraju da budu uokvirene apostrofima, kao što je slučaj kada se njihove vrednosti zadaju u programu.

Pri definisanju izvedenog tipa u deklaracionom delu, mogu se koristiti druge konstrukcije (takodje izvedeni tip podataka), čija je definicija data ranije. Na primer moguće je pisati na sledeći način:

```
PROGRAM TIP_04

TYPE IZOTOP
CHARACTER*10 IMEIZOTOPA
INTEGER BROJNUKLEONA
REAL OBILNOST
END TYPE IZOTOP

TYPE ELEMENT
CHARACTER(10)IME
INTEGER(2) Z
REAL(4) MASA
TYPE(IZOTOP)::GLAVNI_IZOTOP
END TYPE ELEMENT

TYPE(IZOTOP)::GLAVNI
TYPE(ELEMENT)::EL(1)

GLAVNI%IMEIZOTOPA='H1'
GLAVNI%BROJNUKLEONA=1
GLAVNI%OBILNOST=98.

EL(1)%IME='H'
EL(1)%Z=1
EL(1)%MASA=1.003
EL(1)%GLAVNI_IZOTOP= GLAVNI

PRINT *, EL(1)%IME
PRINT *, EL(1)%Z
PRINT *, EL(1)%MASA
PRINT *, EL(1)%GLAVNI_IZOTOP
END
```

Prvo se definiše izvedeni tip IZOTOP sa tri komponente, imenom, brojem nukleona i obilnošću. Zatim se definiše tip ELEMENT, sa četiri komponenti od kojih je četvrta komponenta izvedeni tip IZOTOP i dobija ime GLAVNI_IZOTOP. Strukture se definišu na uobičajeni način stim što se prvo definiše podstruktura IZOTOP. U izlaznom delu štampaju se komponente EL(1) jedna po jedna.

Korišćenje izvedenih tipova podataka i struktura omogućuje rad sa tabeliranim podacima gde su rezultati po raznim kolonama različitog tipa.

VIII

POINTERI

Sama engleska reč ¨pointer¨ se može prevesti kao pokazivač. U ranijim verzijama Fortrana nije postojao koncept pointera i to je sasvim nov pojam u Fortranu. U drugim jezicima kao što su Pascal i C, omogućen je rad sa pointerima ali imaju različit smisao.

Svakoj promenljivoj se može pridružiti pointer. Pointer se može smatrati drugim imenom neke promenljive (takoreći -nadimak). Tako, imamo *ime* promenljive i njen *nadimak*. Ako se želi da se nekoj promenljivoj pridruži pointer onda se to naglašava u deklaracionoj naredbi, rečju TARGET (meta).

TARGET :: IME

Pointer se deklariše u naredbi:

POINTER :: NADIMAK

Konačno, pridruživanje pointera promenljivoj se obavlja naredbom

NADIMAK=>IME

Ispred TARGET i POINTER može se staviti tip promenljive, REAL(4), REAL(8), INTEGER i sl. U tom slučaju se odvajaju zarezom, INTEGER, TARGET :: BROJ. Moguće je istovremeno sa deklarisanjem tipa i TARGETa, obaviti inicijalizaciju vrednosti promenljive, INTEGER, TARGET::BROJ=100. Ovo naravno nije obavezno i može se vrednost broja definisati i kasnije u samom programu.

Ovakva inicijalizacija vrednosti pointera nije dozvoljena.

Sve što se dešava sa nekom promenljivom, dešava se i sa njenim pointerom, i obrnuto. Ovo je demonstrirano u sledećem programu

```
PROGRAM POINTER_01
INTEGER, TARGET::  IME
INTEGER, POINTER :: NADIMAK
READ *, IME
NADIMAK => IME
IME = NADIMAK+3
PRINT*, IME, NADIMAK
NADIMAK=1
PRINT*, IME, NADIMAK
END
```

Definisane su promenljiva IME i pointer NADIMAK. Zatim je promenljivoj IME pridružen pointer NADIMAK naredbom NADIMAK => IME. Na ulazu se promenljivoj IME zada neka vrednost (na primer 5) i štampaju se vrednosti promenljive i pointera (obe imaju vrednost 8). Naredbom NADIMAK=1 se promeni vrednost pointera i sledeći red opet štampa vrednost promenljive i pointera i one su sada obe =1.

Pointer može da bude u jednom od tri stanja (statusa). Neposredno po deklarisanju, svi pointeri su *nedefinisani* (odnosno ne zna se na koju se promenljivu odnose). Pokušaj korišćenja takvog pointera bez prethodnog pridruživanja nekoj promenljivoj izaziva grešku u toku izvršavanja programa.

Naredbom *pointer => imepromenljive* obavlja se pridruživanje pointera promenljivoj; ovo je *asocirani* pointer.

Konačno pointer može biti u *nultom* (*null*) stanju kada nije pridružen ni jednoj promenljivoj. Postoji posebna naredba kojom se anulira pridruživanje pointera nekoj

promenljivoj. To je naredba NULLIFY (*lista imenapointera*). Svi pointeri koji su u listi iza NULLIFY ne ukazuju ni na jednu promenljivu. Pokušaj korišćenja ovih pointera, (na primer štampanje njihovih vrednosti) izaziva grešku u izvršavanju programa.

Ispitivanje da li je neki pointer pridružen nekoj promenljivoj obavlja se unutrašnjom funkcijom fortana ASSOCIATED(*imepointera,[imepromenljive]*). Ime promenljive je navedeno u uglastim zagradama, što je ustanovljena konvencija za opcione parametre. Takvi parametri se mogu uneti a ne moraju. Ova funkcija daje logički izlaz T ili F zavisno od toga da li je pointer pridružen ili ne nekoj promenljivoj. Ako se ne unese ime promenljive, rezultat poziva funkcije ASSOCIATED (IMEPOINTERA) biće T (true) ako je pointer pridružen bilo kojoj promenljivoj u programu, a F (false) ako nije. U slučaju da se uvede ime promenljive u naredbi, onda je odgovor T ako je pointer pridružen toj promenljivi, a F ako joj nije pridružen. Na primer program

```
PROGRAM POINTER_02
INTEGER, TARGET:: IME
INTEGER, POINTER :: NADIMAK
INTEGER,TARGET :: DRUGA
NADIMAK => IME
PRINT *, ASSOCIATED (NADIMAK)
PRINT *, ASSOCIATED(NADIMAK,IME)
PRINT *, ASSOCIATED(NADIMAK,DRUGA)
NULLIFY(NADIMAK)
PRINT *, ASSOCIATED (NADIMAK)
END
```

daje na izlazu T T F F. Prvi T znači da je pointer NADIMAK pridružen nekoj promenljivoj u programu, drugi T znači da je pridružen promenljivoj IME. Prvo F znači da pointer nije pridružen promenljivoj DRUGA. Konačno, drugo F znači da pointer nije pridružen ni jednoj promenljivoj jer je pre toga anuliran funkcijom NULLIFY(Nadimak).

IX

PODPROGRAMI

Skup naredbi koje bi se ponavljale više puta u programu ili u raznim programima se izdvajaju u posebnu programsku jedinicu koja se naziva podprogram. Podprogram je izdvojena programska jedinica. Postoji više razloga za pisanje i korišćenje podprograma.
Prvi razlog je što je pogodno jedan komplikovan program podeliti u više manjih programskih jedinica. Proučavanje i kontrola su na taj način jako olakšani; može se detaljno studirati i testirati jedan po jedan podprogram bez prevelike brige o tome šta se dešava u drugim delovima programa. Takodje, podprogrami se mogu razvijati nezavisno od glavnog programa. Više različitih programera može raditi istovremeno na raznim podprogramima, koji se kasnije mogu kombinovati u jednom ili više programa.
Drugi razlog je što se jednom napisan podprogram može primeniti u većem broju programa. Takodje može se skup podprograma grupisati u jedan modul koji se može koristiti u više različitih softvera.
Treći razlog je zaštita informacija. U podprogramu se mogu koristiti podaci koji su nedostupni iz glavnog programa i na taj način može se sprečiti njihova nenamerna izmena ili zloupotreba.

Svaki fortranski software **sadrži samo jedan glavni program**, a može da sadrži proizvoljan broj podprograma i modula.

Podprogrami se mogu klasifikovati na sledeći način:
- ugradjene funkcije (intrinsic function);
- aritmetičke naredbe;
- funkcijski podprogrami i
- opšti podprogrami.

Ranije smo se već sreli sa ugradjenim, ili unutrašnjim funkcijama fortrana. Takve su na primer SIN, COS, MOD, i dr.

IX.1.ARITMETIČKA NAREDBA

Aritmetička naredbase piše na početku programa pre prve izvršne naredbe. Ovo nije posebna izdvojena programska jedinica tako da nema sve osobine podprojekta. U njoj se koriste lažni argumenti (ponekada se koristi i termin, fiktivni argumenti). Ovi argumenti se upotrebljavaju da bi se definisala sama naredba. Pozivanje aritmetičke naredbe se obavlja iz programa i pri pozivu se koriste stvarni argumenti čije su vrednosti poznate u samom trenutku pozivanja. Pri pozivu se vrednosti stvarnih argumenata pridružuju lažnim argumentima, zatim se obavlja računanje prema pravilu zadatom u aritmetičkoj naredbi i vraća se u program na mestu u kome je obavljen poziv. Korišćenje aritmetičke naredbe je dato u sledećem programu.
Zadatak. Tabelirati vrednosti funkcije $y(x)=3x^3+2x^2+x-1$ u intervalu od –10 do + 10 sa korakom od 0.1

```
 PROGRAM  ARITM_NAREDBA_1
!DEMONSTRIRA KORISCENJE ARITMETICKE NAREDBE
```

```
Y(Z)=3.*Z**3+2.*Z**2+Z-1.
OPEN(10, FILE='IZLAZ.DAT')
DO X=-10., 10., 0.1
A=Y(X)
WRITE(10,*)X,A
END DO
END
```

Naredbom Y(Z)=3.*Z**3+2.*Z**2+Z-1. , definiše se pravilo (propis) po kome se računa veličina Y. Promenljiva Z je lažni argument i koristi se da se definiše pravilo. Naredba A=Y(X) je poziv aritmetičke naredbe, vrednost promenljive X je poznata. Ta vrednost se pridružuje lažnom argumentu Z, obavlja se računanje i vraća se u program na mestu poziva.

Aritmetička naredba se koristi u slučajevima kada je celokupan račun moguće obaviti jednom naredbom. Ona se mora pojaviti u programu u kome se koristi. Ako se u njoj pojavljuju i neki drugi parametri, koji nisu lažni argumenti, oni moraju da budu definisani u programu. Aritmetička naredba se smatra zastarelom i buduće verzije fortrana je neće sadržati, te je treba izbegavati pri pisanju novih programa. Koncept lažnih i stvarnih argumenata iznet u prethodnom tekstu, se koristi i u drugim vrstama podprograma.

IX.2. FUNKCIJSKI PODPROGRAMI

Funkcijski podprogrami (ili funkcije) se koriste kada se računanje ne može obaviti jednom naredbom ili u jednom redu. Funkcijski podprogram je posebna programska jedinica i može se pisati odvojeno od glavnog programa. Programer zadaje ime funkcijskog podprograma prema pravilima zadavanja imena u fortranu koja su objašnjena ranije. Pri pisanju funkcijskog podprograma koriste se lažni argumenti, slično kao i kod aritmetičke naredbe. Pozivanje funkcijskog podprograma se obavlja u glavnom programu i to sa stvarnim argumentima. *Rezultat računanja se pridružuje imenu funkcijskog podprograma.*

Na primer, razmotrimo sledeću funkciju

$F(x) = 1 \quad ako\ je \quad x > 1$

$F(x) = x^2 \quad ako\ je \quad \geq 1 x \geq -1$

$F(x) = -1 \quad ako\ je \quad x < 1$

Potrebno je napraviti program koji izračunava funkciju Z= (F(a)+F(b))/F(a*b).

```
PROGRAM FJSKI
!DEMONSTRIRA FUNKCIJSKI PODPROGRAM
READ *, A,B
Z=(F(A)+F(B))/F(A*B)    !POZIV SA STVARNIM ARGUMENTIMA
PRINT *, Z
END
FUNCTION F(X)             !ovo je funkcijski podprogram
IF(X<-1) C=-1.
```

```
IF(X>1) C=1.
IF(X<=1.AND.X>=-1.)C=X**2
F=C
RETURN
END
```

U funkcijskom podprogramu nazvanom F, odredjuje se vrednost veličine C prema zadatom obrascu tj prema zadatim jednačinama. Zatim se izračunata vrednost C, dodeljuje imenu F funkcijskog podprograma. U glavnom programu se izračunava zadata kombinacija.

Naredba RETURN znači povratak u glavni program. Ova naredba je opciona i ne mora da se koristi. Ukoliko se ova naredba ne stavi u podprogramu, povratak u glavni program se obavlja kada se naidje na naredbu END podprograma.

Zapazimo da se funkcijski podprogram piše iza naredbe END koja predstavlja fizički kraj glavnog programa. Organizacija programa i podprograma je moguća i na sledeći način: podprogram se može *dodati* radnom prostoru projekta kao nezavisna jedinica. Dodavanje se obavlja naredbom INSERT. Znači moguć je takav raspored glavnog programa i podprograma gde u isto vreme u radnom prostoru projekta figuriše jedan glavni program i jedan ili više podprograma. Pri dodavanju podprograma radnom prostoru, koristi se ekstenzija .F90 iza imena podprograma (u prethodnom slučaju dodaje se fajla F.F90). Moguće je dodati veći broj podprograma koji se pozivaju iz glavnog programa.

Promenljive koje se koriste za komunikaciju izmedju glavnog programa i podprograma (fiktivni i stvarni argumenti) su takozvane globalne promenljive. Pored njih u podprogramu mogu biti definisane i druge promenljive potrebne za rad. Ovakve promenljive definisane u podprogramu su *lokalnog* karaktera, tj. definisane su samo u okviru datog podprograma. Njihove vrednosti van podprograma su nedefinisane. Isto važi i za promenljive definisane u glavnom programu. Tako, moguće je imati dve *različite promenljive* sa istim imenom, jednu u glavnom programu, a drugu u nekom od podprograma. Moguće je koristiti isto ime za fiktivni i stvarni argument, ali nije obavezno.

Fukcijski podprogrami se koriste kada se kao rezultat računanja dobija jedna vrednost i ona se pridružuje imenu podprograma. U samom podprogramu mora se bar jednom naći ime funkcijskog podprograma na levoj strani jednačine, gde se neka izračunata vrednost pridružuje imenu.

U prethodnom primeru podprogram F ima samo jedan argument. Moguće je koristiti više argumenata. Fiktivni i stvarni argumenti se moraju slagati po broju redu i vrsti. Ukoliko ovaj zahtev nije zadovoljen, javlja se greška pri kompliranju programa. Moguće je da je argument ime niza i onda se niz mora definisati i u glavnom programu i u podprogramu. Takodje, moguće je da se pri pozivu na mestu stvarnog argumenta koristi izraz. Pri pozivu izračunava se vrednost izraza i dodeljuje fiktivnom argumentu.

Korišćenje fiktivnih i stvarnih argumenata je jedan od načina prenošenja informacija iz glavnog programa u podprogram i obratno. Postoje i drugi načini prenošenja informacija i o tome će biti reči kasnije.

Ime funkcijskog podprograma može biti dvostruke tačnosti. U tom slučaju se u glavnom programu navodi ime podprograma iza naredbe deklaracije dvostruke tačnosti, kao na primer REAL (8) F. Pri tome se i u definiciji podprograma navodi

REAL (8) FUNCTION F(X)
ili DOUBLE PRECISION FUNCTION F(X).

Takodje, ime funkcijskog podprograma može biti i LOGICAL, INTEGER ili REAL(4).
Funkcijski podprogram opisan u prethodnom tekstu je spoljašnji, i koristi se izraz eksterni podprogram. Da bi se naznačio eksterni karakter nekog podprograma može se koristiti službena reč fortrana EXTERNAL (znači spoljašnji) koja se navodi na početku programa iza naredbe PROGRAM, a pre prve izvršne naredbe, kao na primer:

```
PROGRAM FJSKI
EXTERNAL F
```

Eksterni podprogram se navodi iza naredbe END glavnog programa ili se postavlja kao nezavisna programska jedinica u radnom prostoru projekta.

U sledećem podprogramu računa se skalarni proizvod ili intenzitet vektorskog proizvoda dva vektora (prema želji korisnika).

```
PROGRAM FJSKI_1
REAL,DIMENSION (3):: A,B
CHARACTER *1, TIP
PRINT *, 'UNESI KOMPONENTE PRVOG VEKTORA'
READ *, A
PRINT *, 'UNESI KOMPONENTE DRUGOG VEKTORA'
READ *, B
DO
PRINT *, 'ZA SKALARNI PROIZVOD UNESI S, A ZA VEKTORSKI V'
READ *, TIP
IF(TIP/='S'.AND.TIP/='V')CYCLE
PRINT *, TIP
EXIT
END DO
PROI=PROIZVOD(TIP,A,B)
IF(TIP=='S') PRINT * , 'SKALARNI  PROIZVOD'
IF(TIP=='V') PRINT * , 'INTENZITET VEKTORSKOG PROIZVODA'
PRINT *, 'PROIZVOD JE ', PROI
END PROGRAM FJSKI_1

FUNCTION PROIZVOD(K,X,Y)
CHARACTER *1,K
REAL,DIMENSION (3):: X,Y
PRINT *, K
 IF(K=='S')THEN
 PROIZVOD=X(1)*Y(1)+X(2)*Y(2)+X(3)*Y(3)
 RETURN
 END IF
 IF(K=='V')THEN
 VX=X(2)*Y(3)-X(3)*Y(2)
 VY=X(3)*Y(1)-X(1)*Y(3)
 VZ=X(1)*Y(2)-X(2)*Y(1)
 PROIZVOD=SQRT(VX**2+VY**2+VZ**2)
 END IF
 END FUNCTION PROIZVOD
```

Funkcijski podprogram PROIZVOD koristi tri argumenta. Prvi argument K je karakternog tipa, a druga dva su nizovi X i Y svaki sa po tri elementa.

U prethodnim primerima rezultat računanja funkcijskog podprograma je uvek pridruživan imenu tog podprograma. Ovaj koncept vuče poreklo još iz ranijih verzija fortrana. U Fortranu 90 izvršena je izvesna modifikacija uvodjenjem službene reči RESULT. Sintaksa je RESULT (ime_rezultata) koja se stavlja u istom redu gde i FUNCTION ime_podprograma. Izračunata vrednost se pridružuje promenljivoj ime_rezultata. Tako može se pisati sledeći red

FUNCTION ime_podprograma RESULT(ime_rezultata).

Pri korišćenju reči RESULT mora se negde u podprogramu dodeliti vrednost promenljivoj koja ima ime ime_resultata. Celokupna sintaksa ima oblik

FUNCTION ime_podprograma RESULT(ime_rezultata)
.
.
ime_rezultata=izraz
END FUNCTION ime_podprograma

Ime rezultata je lokalnog tipa i nije definisano van podprograma. U ovom slučaju nije neophodno u podprogramu pridružiti izračunatu vrednost imenu podprograma-to se obavlja automatski. Podprogram se i dalje poziva na isti način kao i ranije i ime podprograma nosi izračunatu vrednost. Može se postaviti pitanje, čemu služi izdvojeno ime rezultata ako već ime funkcijskog podprograma nosi izlaznu vrednost. U fortranu 90 omogućeno je da se jedan podprogram poziva iz samog tog podprograma. To su takozvane rekurzivne procedure (RECURSIVE). Rekurzivna procedura je demonstrirana u računanju faktorijala broja I u sledećem programu

```
PROGRAM FJSKI_3
INTEGER FACTORIAL
READ *, I
PRINT *, FACTORIAL(I)
END

RECURSIVE INTEGER FUNCTION FACTORIAL(N) RESULT(FAC_N)
IMPLICIT NONE
INTEGER :: N
SELECT CASE (N)
CASE(1)
FAC_N=1
CASE DEFAULT
FAC_N=N*FACTORIAL(N-1)
END SELECT
END FUNCTION FACTORIAL
```

Ispred imena podprograma u kome se poziva taj isti podprogram stavlja se službena reč fortrana RECURSIVE. Rezultat računanja se pridružuje promenljivoj FAC_N koja je

ime rezultata. Bez korišćenja posebnog imena za rezultat ova rekurzivna procedura ne bi bila moguća.

IX.3. OPŠTI PODPROGRAMI

Funkcijski podprogrami daju kao izlaz jedan broj, koji je pridružen imenu same promenljive. Medjutim, često je potrebno da se kao rezultat računanja dobije više od jednog broja, ili čak mnogo brojeva. U takvim slučajevima koriste se opšti podprogrami. Slično funkcijskim podprogramima, i opšti podprogrami se mogu pisati iza naredbe END koja označava kraj glavnog programa, ili se mogu dodati radnom prostoru projekta. Veličine definisane u podprogramu su lokalnog karaktera, i imaju definisane vrednosti samo u okviru podprograma. Za definisanje opšteg podprograma koristi se sledeća sintaksa

SUBROUTINE ime_podprograma(lista argumenata)
.
.
.
END SUBROUTINE ime

Ime podprograma definiše programer po svojoj volji, poštujući pravila zadavanja imena u fortranu. Kako se imenu opšteg podprograma ne pridružuje nikakva promenljiva ovde nema smisla govoriti o tipu (real, integer i sl) kao kod funkcijskig podprograma. Pri pisanju podprograma koriste se fiktivni argumenti, a pri pozivanju stvarni. Poziv opšteg podprograma u glavnom programu se ostvaruje naredbom call

CALL ime_podprograma(lista stvarnih argumenata)

Za razliku od funkcijskog podprograma, ovde se definišu ulazni i izlazni argumenti- izračunata vrednost koja predstavlja izlazni rezultat se pridružuje izlaznim argumentima a ne imenu podprograma. Pri pozivanju podprograma moraju biti definisani ulazni argumenti čije se vrednosti pridružuju fiktivnim argumentima. Po završenom računu, prirdužuju se vrednosti izlaznim argumentima i vraća se u glavni program na prvoj liniji iza poziva.

Fiktivni i stvarni argumenti se i ovde moraju slagati po redu, broju i vrsti. Argumenti mogu biti imena nizova i tada se moraju dimenzionisati i u glavnom programu i u podprogramu. Takodje, stvarni argument može biti izraz, čija se vrednost izračunava pri pozivanju, a zatim se pridružuje fiktivnom argumentu.
U sledećem programu obavlja se množenje dve matrice u podprogramu nazvanom MNOZENJE_MATRICA.

```
PROGRAM OPSTI
!DEMONSTRIRA OPSTI PODPROGRAM
REAL, ALLOCATABLE :: MAT1(:,:),MAT2(:,:),REZ(:,:)
print *,'MNOZENJE MATRICA'
READ *, I,J,K
ALLOCATE (MAT1(I,J),MAT2(J,K),REZ(I,K))
PRINT *, 'UNESI PRVU MATRICU VRSTA PO VRSTA'
READ *, MAT1
```

```
PRINT *, 'UNESI DRUGU MATRICU VRSTA PO VRSTA'
READ *, MAT2
CALL MNOZENJE_MATRICA(MAT1,MAT2,REZ,I,J,K)
PRINT *, 'REZULTAT MNOZENJA JE'
PRINT *, REZ
PAUSE  'PROVERA PREKO INTERNE FUNKCIJE FORTRANA, PRESS ENTER'
REZ=MATMUL(MAT1,MAT2)
PRINT *, REZ
END
SUBROUTINE MNOZENJE_MATRICA(A,B,Y,L,M,N)
!MNOZI DVE MATRICE,  A(L,M) SA B(M,N) I DAJE Y(L,N)
INTEGER I,J,K,L,M,N
REAL, INTENT(IN), DIMENSION(L,M)::A
REAL, INTENT(IN), DIMENSION(M,N)::B
REAL, INTENT(OUT),DIMENSION(L,N)::Y
Y=0.0
 DO K=1,N
  DO I=1,M
   DO J=1,L
    Y(J,K)=Y(J,K)+A(J,I)*B(I,K)
   END DO
  END DO
 END DO
END SUBROUTINE MNOZENJE_MATRICA
```

Reči IN i OUT definišu parametre navedene iza njih kao ulazne odnosno izlazne u nekom opštem podprogramu.

Naredba REAL, INTENT(IN), DIMENSION(L,M)::A definiše realni dvodimenzionalni niz A(L,M) kao ulazni parameter. Slično je i sa nizom B(M,N). Službena reč IN odredjuje da je parametar sa imenom iza nje ulazna veličina i ona se ne može menjati u podprogramu.

Naredbom REAL, INTENT(OUT), DIMENSION(L,N)::Y definiše se dvodimenzionalni niz Y(L,N) kao izlazni parametar podprograma MNOZENJE_MATRICA. Veličine navedene iza INTENT(OUT) nemaju definisane vrednosti pre pozivanja podrograma i one se izračunavaju u podprogramu.

Moguće je i da je jedan parametar istovremeno i ulazni i izlazni i u tom slučaju se koristi reč INOUT. Takav parametar ima odredjenu vrednost pre pozivanja i njegova vrednost se može promentiti u podprogramu. Preporučuje se korišćenja reči IN, OUT i INOUT radi bolje preglednosti, jasnoće i lakše kontrole programa.

Nije obavezno definisati izlazne i ulazne parametre u podprogramu preko IN OUT i INOUT. U tom slučaju svi parametri su i ulazni i izlazni i mogu biti definisani i pre poziva podprograma, a mogu se menjati u podprogramu. Sledeći segment demonstrira ovu osobinu parametara bez korišćenja IN, OUT reči.

```
PROGRAM OPSTI_2
READ *, A,B
CALL TEST(A,B,C)
PRINT *, A,B,C
END

SUBROUTINE TEST(X,Y,Z)
X=X+1.
Y=Y+1.
```

```
Z=X*Y
END SUBROUTINE TEST
```

Podprogram TEST koristi tri parametra x,y i z. Pri pozivanju definisana su samo dva, X i Y. Njihove vrednosti se menjaju u toku računanja u podprogramu i izmenjene vrednosti se vraćaju u glavni podprogram.

Pri pozivanju podprograma, kao argument se može koristiti ime nekog drugog funkcijskog podprograma.

Na primer: CALL TEST(TEST1(A),...) je korektan zapis, pri čemu je TEST1 neki funkcijski podprogram. U ovom slučaju će se prvo obaviti račun u podprogramu TEST1 za zadatu vrednost argumenta A, a zatim se poziva opšti podprogram TEST. Takodje, moguće je iz jednog podprograma pozivati druge podprograme, kako funkcijske, tako i opšte.

IX.4. MODULI

Modul je skup podprograma (ili jedan podprogram) i predstavlja nezavisnu programsku jedinicu. Drugim rečima, modul je »kontejner« podprograma. Takodje, može biti i kontejner podataka koji ne bi trebalo da budu menjani od strane glavnog programa. Modul se »uvozi« u glavni program korišćenjem naredbe **USE**. Važna karakteristika je kontrola pristupa promenljivima u modulu. Svaki objekat, procedura ili promenljiva u modulu koja je definisana kao PUBLIC postaje dostupna u glavnom programu kada je modul uvežen. Ako se neki objekat deklariše kao PRIVATE, on je onda nedostupan van modula. Obe naredbe PUBLIC i PRIVATE su opcione. Ako se ni jedna ne navede onda su sve promenljive i procedure definisane u modulu dostupne iz glavnog programa. Ako se navede samo PRIVATE lista_imena, onda su promenljive navedene u listi nedostupne, dok su ostale dostupne iz glavnog programa

Modul ima specifikacijski deo i deo koji sadrži jedan ili više podprograma. Podprogrami u modulu se mogu pozivati direktno iz glavnog programa, ili iz drugih podprograma istog modula. Pri linkovanju procesor kombinuje glavni program i sve potrebne podprograme i module, kao i unutrašnje funkcije u jedan izvršni fajl. Modul se dodaje radnom prostoru projekta preko naredbe INSERT.

Sledeći program demonstrira korišćenje modula. U glavnom programu se unose dva podatka, a njihov zbir, razlika, proizvod i količnik se računaju u podprogramu RACUN. Stepen količnika se računa u podprogramu STEPEN. Ova dva podprograma su izdvojena u poseban modul koji se naziva SKUPPR.

```
PROGRAM MOD1
! DEMONSTRIRA KORISCENJE MODULA
USE SKUPPR
REAL KOL, KVADRAT
READ *, X,Y
CALL RACUN(X,Y,ZBIR, RAZL, PROIZV, KOL)
PRINT *, ZBIR, RAZL, PROIZV, KOL
KVADRAT=STEPEN(KOL)
PRINT *, KVADRAT
END
```

```
MODULE SKUPPR
 PUBLIC PRVI,A,B,Z,R,P,K,X,STEPEN
CONTAINS
SUBROUTINE RACUN(A,B,Z,R,P,K)
REAL K
Z=A+B
R=A-B
P=A*B
```

```
K=A/B                                   STEPEN=X**2
END SUBROUTINE RACUN                    END FUNCTION STEPEN
FUNCTION STEPEN(X)                      END MODULE SKUPPR
```

Modul Skuppr treba da ima ekstenziju .F90 ako se dodaje radnom prostoru projekta.

IX.5. INTERNI PODPROGRAMI

U prethodnom tekstu opisivani su eksterni podprogrami, koji se pišu odvojeno od glavnog programa. Pored ovakvih podprograma postoje i interni podprogrami koji se pišu u okviru glavnog programa. Pri definisanju internih podprograma kao poslednja naredba glavnog programa stavlja se CONTAINS. Nakon te naredbe slede podprogrami, može ih biti više.Na kraju se stavlja naredba END PROGRAM ime_glavnog_programa Sintaksa je sledeća

```
PROGRAM GLAVNI
.
.
.
CONTAINS
        FUNCTION PODPROGRAM1
        .
        .
        END FUNCTION PODPROGRAM1
SUBROUTINE PODPROGRAM2
        .
        .
END SUBROUTINE PODPROGRAM2

END PROGRAM GLAVNI
```

Kao primer korišćenja internog podprograma daje se sledeći program u kome se računaju koordinate neke tačke u sisemu koji je rotiran za neki ugao ALFA.

```
PROGRAM ROTACIJA_KOORDINATA
!DEMONSTRIRA KORISCENJE INTERNIH PODPROGRAMA
PRINT  *, 'UNESI KOORDINATE TACKE'
READ *, X,Y
PRINT *, 'UNESI UGAO ROTACIJE U STEPENIMA'
READ *, ALFA
CALL ROTATION
PRINT *,'KOORDINATE TACKE U ROTIRANOM SISTEMU SU'
PRINT *, XP,YP
CONTAINS
SUBROUTINE ROTATION
XP=X*SIND(ALFA)+Y*COSD(ALFA)
YP=-X*COSD(ALFA)+Y*COSD(ALFA)
END SUBROUTINE ROTATION
END PROGRAM ROTACIJA_KOORDINATA
```

Prenos vrednosti iz glavnog programa u podprogram se ne obavlja preko fiktivnih i stvarnih argumenata. Veličine X i Y definisane u glavnom programu imaju definisanu

vrednost i u podprogramu. Takodje, veličine XP,YP izračunate u podprogramu imaju odredjene vrednosti i u glavnom programu. Prenos podataka iz glavnog programa u podprogram i obratno je automatski i nema potrebe za korišćenjem argumenata. Naravno da je moguće koristiti argumente i u ovakvim podprogramima.

Interni podprogrami su sastavni deo programa i kompiliraju se i linkuju zajedno.

IX6.NAREDBA COMMON

Jedan od problema koji se pojavljuje pri prenosu podataka iz glavnog programa u podprogram i obratno je slučaj kada je potrebno preneti veliku količinu podataka. Jasno je da korišćenje fiktivnih i stvarnih argumenata nije praktično u ovom slučaju. Ovaj problem je rešavan korišćenjem COMMON naredbe. Sama reč COMMON znači zajednički i ova naredba se smatra zastarelom te je ne treba koristiti pri pisanju novih programa. Sintaksa je COMMON naredbe je sledeća

PROGRAM GLAVNI
COMMON *lista1*
.
.
.
END PROGRAM GLAVNI

SUBROUTINE PPG(fiktivni_argumenti)
COMMON *lista2*
.
.
END SUBROUTINE PPG

Broj podataka u listama lista1 i lista2 ne mora biti isti (mada je poželjno) ali se nabrojana imena moraju se slagati po vrsti. Vrednost prve promenljive u listi *lista1* koja je definisana u glavnom programu se direktno prenosi na prvu promenljivu u listi *lista2*. Izjednačavanje vrednosti promenljivih u listama se nastavlja za sve promenljive u listama. Svaka promena bilo koje promenljive, bilo gde u programu ili u podprogramima ima uticaj na svim mestima gde se ta veličina pojavljuje.

```
PROGRAM ZAJEDNICKI
! DEMONSTRIRA NAREDBU COMMON
REAL :: A,B,C
COMMON B,C
READ *, A
CALL RACUN1(A)
PRINT *, A,B,C
END
SUBROUTINE RACUN1(X)
REAL X,Y,Z
COMMON Y,Z
Y=X+1.
Z=X-1.
END SUBROUTINE RACUN1
```

Promenljiva B u glavnom podprogramu ima istu vrednost kao i promenljiva Y u podprogramu takodje, ista je situacija sa promenljiva C i Z.
Promenljive B i C čine jednu zajedničku zonu. Ovakve zone nemaju ime i nazivaju se neimenovane.
Pored neimenovanih zona moguće je raditi i sa imenovanim zonama. Sintaksa je sledeća
COMMON /ime_zone/ lista_imena_promenljivih.

Pri korišćenju internih podprograma, naredbe COMMON nisu potrebne jer je prenos podataka automatski. S druge strane, kod eksternih podprograma naredba COMMON je vrlo štetna jer onemogućava prenosivost podprograma i otežava njihovo korišćenje od strane različitih programa. Pored toga, naredbe COMMON su bile čest izvor greški i zato se ne preporučuje njihovo korišćenje.

IX.7. INTERFACE

Pri korišćenju internih podprograma lako je proveriti da li se fiktivni i stvarni argumenti slažu po redu, broju i vrsti. Medjutim, pri korišćenju eksternih podprograma, ili podprograma iz nekih biblioteka često je teško proveriti da li su argumenti kompatibilni. U takvim slučajevima preporučuje se korišćenje posebnog programskog bloka koji se naziva INTERFACE i koji se stavlja na početku glavnog programa. U ovom programskom bloku daje se spisak podprograma i argumenata, ali bez ikakvog računanja. Dat je primer primene interface i opšteg podprograma na računanje kvadratne jednačine.

```
PROGRAM OPSTI_3
IMPLICIT NONE

INTERFACE
SUBROUTINE INTERACT(A,B,C,OK)
IMPLICIT NONE
REAL, INTENT(OUT)::A
REAL, INTENT(OUT)::B
REAL, INTENT(OUT)::C
LOGICAL, INTENT(OUT):: OK
END SUBROUTINE INTERACT

SUBROUTINE RESI(X,Y,Z,KOREN1,KOREN2, NEUSPEH)
IMPLICIT NONE
REAL, INTENT(IN)::X
REAL, INTENT(IN)::Y
REAL, INTENT(IN)::Z
REAL, INTENT(OUT)::KOREN1
REAL, INTENT(OUT)::KOREN2
INTEGER, INTENT(INOUT) :: NEUSPEH
END SUBROUTINE RESI
ENDINTERFACE



REAL :: P,Q,R, KOREN1, KOREN2
INTEGER :: NEUSPEH =0
```

```
 LOGICAL :: OK=.TRUE.
 CALL INTERACT(P,Q,R,OK)
 IF(OK)THEN
 CALL RESI(P,Q,R, KOREN1,KOREN2, NEUSPEH)
 IF(NEUSPEH==1) THEN
   PRINT *, ' NEMA REALNIH RESENJA'
 ELSE
   PRINT *, ' KORENI SU ', KOREN1, KOREN2
          END IF
          ELSE
          PRINT *, ' POGRESNI PODACI NA ULAZU'
          END IF
          END PROGRAM OPSTI_3

SUBROUTINE INTERACT(A,B,C,OK)
 REAL, INTENT(OUT)::A
 REAL, INTENT(OUT)::B
 REAL, INTENT(OUT)::C
 LOGICAL, INTENT(OUT):: OK
INTEGER :: STANJEUNOSA=0
PRINT *, 'UNESI KOEFICIJENT A,B I C'
READ(*,*, IOSTAT=STANJEUNOSA) A,B,C
IF(STANJEUNOSA==0)THEN
OK=.TRUE.
ELSE
OK=.FALSE.
END IF
END SUBROUTINE INTERACT

SUBROUTINE RESI(X,Y,Z, KOREN1, KOREN2, NEUSPEH)
IMPLICIT NONE

 REAL, INTENT(IN)::X
 REAL, INTENT(IN)::Y
 REAL, INTENT(IN)::Z
 REAL, INTENT(OUT)::KOREN1
 REAL, INTENT(OUT)::KOREN2
 INTEGER, INTENT(INOUT) :: NEUSPEH
!Lokalne varijable
  REAL :: DISKRIMINANTA,PDKOREN
 REAL :: A2
 DISKRIMINANTA=Y**2-4.*X*Z
 A2=X*2
 IF( DISKRIMINANTA<0.)THEN
 NEUSPEH=1
 ELSE
 PDKOREN=SQRT(DISKRIMINANTA)
 KOREN1=(-Y+PDKOREN)/A2
 KOREN2=(-Y-PDKOREN)/A2
 END IF
 END SUBROUTINE RESI
```

Na početku glavnog program dat je interface gde su deklarisana dva podprograma. Prvi od njih *interact*, kontroliše korektnost unosa podataka. Ima tri ulazna podatka, A,B i C, koji treba da budu koeficijenti kvadratne jednačine, i jednu izlaznu logičku veličinu OK, koja je .TRUE. ako je unos podataka korektan. Za proveru korektnosti unosa koristi se IOSTAT, koji ima vrednost 0 ako je unos korektan, i ima vrednost različitu od 0, ako je unos nekorektan. Vrednost IOSTAT se pridružuje promenljivoj *STANJEUNOSA* na osnovu koje se dalje odredjuje vrednost logičke promenljice OK (true ili false). (Vrednost IOSTAT pri pogrešnom unosu podataka zavisi od vrste načinjene greške).
Drugi podprogram RESI, ima 6 parametara, od kojih su tri ulazni, X,Y i Z, a dva su izlazna, KOREN1 i KOREN2, i konačno, NEUSPEH je i ulazni i izlazni parametar.
Postoje slučajevi kada je korišćenje interface obavezno.

Naredba SAVE
Lokalne promenljive nemaju definisane vrednosti pre poziva podprograma. Pri izvršavanju podprogama njihove vrednosti se odredjuju na osnovu nekog računa, a pri povratku u glavni program one se anuliraju. Naredba SAVE *lista*, koja se stavlja u podprogramu omogućuje da se sačuvaju vrednostu lokalnih promenljivih izmedju poziva.

PRIMERI

P1.Dat je neki niz brojeva. Napisati program koji računa sumu kvadrata recipročnih vrednosti i sumu kubova recipročnih vrednosti tih brojeva. Sumiranje obaviti u podprogramu.

```
 PROGRAM OPSTI_4
IMPLICIT NONE
REAL, ALLOCATABLE:: A(:)
REAL:: SUMREC2,SUMREC3
INTEGER :: NUMBER,I
INTERFACE
SUBROUTINE SABIRANJE(X,SUMA2,SUMA3,N)
IMPLICIT NONE
INTEGER, INTENT(IN):: N
REAL, INTENT(IN), DIMENSION(1:N) :: X
REAL, INTENT(OUT):: SUMA2,SUMA3
END SUBROUTINE SABIRANJE
END INTERFACE
PRINT *, ' KOLIKO BROJEVA'
READ *, NUMBER
ALLOCATE (A(NUMBER))
DO I=1,NUMBER
READ *, A(I)
END DO
CALL SABIRANJE(A,SUMREC2,SUMREC3,NUMBER)
PRINT *, 'SUMA KVADRATA RECIPROCNIH VREDNOSTI JE', SUMREC2
PRINT *, 'SUMA KUBOVA RECIPROCNIH VREDNOSTI JE', SUMREC3
END PROGRAM OPSTI_4

SUBROUTINE SABIRANJE(X,SUMA2,SUMA3,N)
```

```
IMPLICIT NONE
INTEGER, INTENT(IN):: N
REAL, INTENT(IN), DIMENSION(1:N) :: X
REAL, INTENT(OUT):: SUMA2,SUMA3
INTEGER :: I
SUMA2=0.
SUMA3=0.
DO I=1,N
SUMA2=SUMA2+1/X(I)**2
SUMA3=SUMA3+1/X(I)**3
END DO
END SUBROUTINE SABIRANJE
```

U prethodom programu je demonstrirano korišćenje niza kao argumenta podprograma. Jednodimenzionalni Niz A u glavnom programu je stvarni argument pri pozivu podprograma. Da bi program bio opštijeg karaktera korišćeni su dinamički nizovi (allocatable). Niz X u podprogramu je fiktivni argument, i mora takodje biti jednodimenzioni sa istim brojem elemenata, što se definiše naredbom *REAL, INTENT(IN), DIMENSION(1:N) :: X.*

P2. Kvadratni i kubni koren nekog broja se mogu računati prema iterativnom postupku

$$za\ kvadratni\ koren\ x' = \frac{x + a/x}{2} \quad i\ za\ kubni\ x' = \frac{2x + a/x^2}{3}$$

gde se x' dobija iz x iz prethodne aproksimacije. Ovo znači da je potrebna nulta aproksimacija za koju se uzima da je x'=x/2, jer iteracija brzo konvergira i nulta aproksimacija ne utiče previše na brzinu računanja niti na tačnost rezultata. Drugo pitanje je kada izaći iz iteracije, tj. kada prekinuti iterativni postupak. Iteraciju treba nastaviti dok je razlika izmedju dva susedna člana dovoljno mala. U tu svrhu izračunaće se količnik dve uzastopne vrednosti za x i ako su one bliske izaći će se iz iteracije. Poredi se apsolutna vrednost razlike količnika x'/x i 1 sa zadatom tolerabilnom vrednošću ε. Ako je ta razlika manja od ε izlazi se iz iteracije.

```
PROGRAM KOREN
IMPLICIT NONE
REAL ::KVKOREN
REAL ::KUBKOREN
REAL ::EPSILON
REAL :: BROJ
PRINT *, 'UNESI BROJ I TACNOST'
READ *, BROJ, EPSILON
PRINT *, 'KVADRATNI KOREN', KVKOREN(BROJ, EPSILON)
PRINT *, 'KUBKOREN',KUBKOREN(BROJ,EPSILON)
END  PROGRAM  KOREN

REAL FUNCTION KVKOREN(A,EPS)
X=A/2.
DO
XP=(X+A/X)/2.
RAZLIKA=ABS(X/XP-1.0)
X=XP
IF(RAZLIKA<EPS)EXIT
```

```
END DO
KVKOREN=XP
END FUNCTION KVKOREN

REAL FUNCTION KUBKOREN(A,EPS)
X=A/2.
DO
XP=(2.*X+A/X**2)/3.
RAZLIKA=ABS(X/XP-1.)
X=XP
IF(RAZLIKA<EPS)EXIT
END DO
KUBKOREN=XP
END FUNCTION KUBKOREN
```

P3.Napisati podprogram koji nalazi najmanji član dvodimenzionalnog niza, i štampa sam taj član i položaj člana u matrici

```
        PROGRAM PPGRAM_2
        IMPLICIT NONE
        REAL ::NAJMANJI
        INTEGER:: N,M,I,J
        INTEGER POZX,POZY !POLOZAJ NAJMANJEG ELEMENTA U MATRICI
        REAL , ALLOCATABLE ::ULAZNINIZ(:,:)
        PRINT *, 'UNESI DIMENZIJE MATRICE'
        READ *, N,M
        ALLOCATE (ULAZNINIZ(N,M))
        PRINT *, 'UNOSI MATRICU VRSTU PO VRSTU'
        DO I=1,N
        READ(*,*) (ULAZNINIZ(I,J),J=1,M)
        END DO
  CALL MALI(ULAZNINIZ,N,M,NAJMANJI,POZX,POZY)
        PRINT*, NAJMANJI
        PRINT *, POZX,POZY
  END
        SUBROUTINE MALI(NIZ,K,L,SMALLEST,POZICIJAX,POZICIJAY)
        IMPLICIT NONE
        INTEGER, INTENT(IN):: K,L
        REAL, INTENT(IN), DIMENSION(K,L):: NIZ
        INTEGER, INTENT (OUT):: POZICIJAX,POZICIJAY
        REAL, INTENT(OUT):: SMALLEST
        REAL :: I,J
        SMALLEST=NIZ(1,1)
        POZICIJAX=1
  POZICIJAY=1
                DO I=1,K
                DO J=1,L
                IF(SMALLEST>NIZ(I,J))THEN
                   SMALLEST=NIZ(I,J)
                    POZICIJAX=I
                     POZICIJAY=J
                      END IF
                END DO
                END DO
        END
```

X

GRAFIKA

U ranijim verzijama fortrana nisu postojale nikakve grafičke rutine. Ovo je bio ozbiljan nedostatak fotrana 77. Tako, nije bilo moguće grafički predstaviti rezultate čak ni najprostijeg računa. Postojali su izvesni programi, tzv. grafike bez grafike, gde je nizom simbola (kao što je na primer slovo X) predstavljan grafički izlaz. Pri tome bilo je potrebno programirati mesto pojave svakog pojedinačnog simbola. U fortranu 90 omogućen je rad sa grafikom. Moguće je crtati najrazličitije i vrlo komplikovane slike. Moguće je programirati ONLINE prezentaciju računanja. Takodje, omogućen je i rad sa bojama kao i kombinovanje sa pojedinim grafičkim paketima kao što je OpenGL. Mnogobrojne grafičke opcije su na raspolaganju u fortranu 90. U ovom tekstu date su osnove grafike bez namere da se obuhvate sve postojeće rutine i opcije.

Da bi se omogućio rad sa grafikom neophodno je pri kreiranju projekta izabrati QuickWin Application ili Standard Graphics Application.

U QuickWin applikaciji omogućen je rad sa jednim i/ili više grafičkih podprozora, pojavljuje se programabilni meni, klizači. Pored toga, QuickWin omogućuje tekstualne prozore, grafiku zasnovanu na pixelima ili na realnim koordinatama, menije definisane korisnikom, programiranje klika mišem, dijaloge i još mnoge druge na grafici zasnovane opcije. Takodje, moguće je kombinovati grafiku i tekst u istom prozoru. Na raspolaganju su i različiti fontovi pri korišćenju teksta.

U standard graphics application omogućen je rad sa jednim prozorom, koji pokriva celu raspoloživu površinu u kome se mogu pojaviti i grafički i tekstualni objekti. Pored toga uz prozor postoje klizači. Ne pojavljuje se programabilni meni niti su mogući višestruki pod prozori. Po defaultu, prozor koji se otvara pri startovanju grafičkog programa u ovoj vrsti aplikacije je maksimalne veličine (zauzima ceo ekran).

Aktiviranje grafičkih opcija obavlja se naredbom USE MSFLIB, koja se stavlja na početku programa iza naredbe PROGRAM *ime*, a pre deklaracionih naredbi. MSFLIB je akronim od (Microsoft Fortran Library) Microsoftove fortranske biblioteke. Ovom komandom postaju pristupačne API funkcije (application interface) kao i ostale grafičke biblioteke u GDI (Graphic Device Interface) iz Windows Operativnog Sistema.

Po defaultu grafički prozor koji se otvara pri startovanju programa sa grafikom je 640x480 pixela, ima 30 linija sa po 80 karaktera u jednoj liniji i veličinu simbola 8x16. Prozor se može maksimizirati, minimizirati i spustiti na task bar. Ove vrednosti mogu da budu drugačije zavisno od korišćenog računarskog sistema. Informacije o default grafičkim karakteristikama prozora se mogu dobiti preko sledeće naredbe:

STATUS=GETWINDOWCONFIG(WC)

Sledeći program štampa raspoložive informacije o grafičkom prozoru koje se dobijaju prethodnom naredbom.

```
PROGRAM GR1
USE MSFLIB
LOGICAL(4)  status
TYPE (windowconfig) wc
status = GETWINDOWCONFIG(wc)
print *, 'BROJ TEKSTUALNIH LINIJA',  wc.numtextrows
print *, 'BROJ PIKSELA PO X OSI', wc.numxpixels
```

```
print *, 'BROJ PIKSELA PO Y OSI', wc.numypixels
print *, 'BROJ SIMBOLA TEKSTA U LINIJI', WC.numtextcols
PRINT *, 'VELICINA SLOVA',  WC.FONTSIZE
PRINT *, 'BROJ BITA PO JEDNOM PIXELU', WC.bitsperpixel
END
```

Veličina slova se zadaje u heksadecimalnom kodu. Grafičke parametere je moguće promeniti naredbom SETWINDOWCONFIG. Pozivanjem ove naredbe moraju se definisati svi grafički parametri; definisanje dela parametara ne dovodi do postavljanja novih grafičkih parametara. Ukoliko se ne želi promena vrednosti parametra onda se stavlja –1. Ovo je ilustrovano sledećim primerom

```
PROGRAM GR2
USE MSFLIB
LOGICAL statusmode
TYPE (windowconfig) wc
wc.numxpixels = 1000
wc.numypixels = 300
wc.numtextcols = -1
wc.numtextrows = -1
wc.numcolors = -1
wc.title = 'graficki prozor'
wc.fontsize = #000A000C      ! 10 X 12
statusmode = SETWINDOWCONFIG(wc)
IF (.NOT. statusmode) statusmode = SETWINDOWCONFIG(wc)
PRINT *,'broj pixela po x osi', wc.numxpixels
print *, 'broj pixela po y osi',wc.numypixels
PRINT *, 'naslov prozora je  ', wc.title
END PROGRAM GR2
```

X.1.KOORDINATNI SISTEMI

X.1.1.Pixel koordinatni sistem

Grafika u fortranu 90 ima na raspolaganju tri koordinatna sistema. Prvi je zasnovan na pixelima. Centar koordinatnog sistema je u gornjem levom uglu ekrana. X osa je horizontalna i orijentisana na desno, a Y osa je vertikalno naniže. Ako je ekran u rezoluciji 800x600, onda donja desna tačka ekrana ima koordinate (800,600). Gornja desna tačka je (800,0), a donja leva (0,600). Kako broj pixela ne može biti negativan, u pixel koordinatnom sistemu nema negativnih vrednosti. Pixel koordinatni sistem se naziva još i fizički, jer je zasnovan na fizičkim koordinatama pixela.

X.1.2.Windows koordinatni sistem

Pixel koordinatni sistem je dosta nepovoljan, jer je veličina podataka ograničena, a i sam izbor Y ose, vertikalno naniže, je suprotan uobičajenom izboru. Pobrojana ograničenja su prevazidjena u *windows* koordinatnom sistemu. Ovaj koordinatni sistem se aktivira naredbom

```
status = SETWINDOW(.TRUE.,x1,y1 ,x2,y2)
```

Prvi logički switch, koji može biti .TRUE. ili .FALSE. odredjuje usmerenost Y ose. Izbor .TRUE. znači da je Y osa usmerena vertikalno na više, a tačka sa koordinatama (x1,y1) je leva donja tačke, a (x2,y2) su koordinate desne gornje tačke. Ako je logički switch .FALSE. onda je Y osa usmerena vertikalno naniže i (x1,y1) je leva gornja tačka ekrana, a (x2,y2) desno donja tačka. Koordinate x1,x2,y1,y2 su u dvostrukoj tačnosti.

X.1.3.Textualne koordinate

Koordinate teksta se odredjuju brojem reda i brojem pozicije u redu. Mesto pojave teksta definiše se naredbom CALL SETTEXTPOSITION (*broj reda*, *broj pozicija*, ime), Tekst se pojavljuje u redu sa zadatim brojem, počev od zadate pozicije. '*ime*' je izvedeni tip podataka i da bi ova naredba funkcionisala potrebno je na početku programa postaviti TYPE (rccoord) *ime*.

X.2. CRTANJE PRAVE LINIJE

Naredba za crtanje prave linije u pixel koordinatnom sistemu je STATUS=LINETO(I, J) gde su I i J pixel koordinate (celobrojne vrednosti). Linija se iscrtava od *trenutno aktivne pozicije* do tačke sa pixel koordinatama (I,J). U slučaju da aktivna pozicija nije definisana na početku, onda je ona u centru ekrana. Povlačenjem linije od centra do tačke definisane u naredbi LINETO(I,J), aktivna pozicija se premešta u tačku (I,J). Aktivnu poziciju treba uslovno shvatiti kao mesto na ekranu na kome se nalazi kursor.

Naredba za crtanje prave linije u windows koordinatnom sistemu je STATUS=LINETO_W(X1,Y1,X2,Y2) (dodaje se ekstenzija _w na naredbu za crtanje u pixel koordinatama). Koordinate tačaka x1,y1,x2,y2 su u dvostrukoj tačnosti. Ostali elementi su isti kao i u pixel koordinatama.

X.2.1.Podešavanje aktivne pozicije

Jasno je da je prethodni način definisanja aktivne pozicije nepovoljan. Postoji posebna naredba za premeštanje aktivne pozicije u tačku sa željenim koordinatama. Ova naredba je CALL MOVETO(I,J,XY) gde su I i J pixel koordinate tačke u koju se premešta aktivna pozicija, a XY je izvedeni tip podataka. Da bi ova naredba funkcionisala potrebno je na početku programa napisati sledeću naredbu: TYPE (xycoord)xy. U windows koordinatnom sistemu naredba je CALL MOVETO_W(x,y,wxy) gde su x i y windows koordinate (i zadaju se u dvortrukoj tačnosti), a wxy je izvedeni tip podataka koji se definiše naredbom TYPE (wxycoord)wxy i postavlja se na početku programa.

X.3. POSTAVLJANJE BOJA

U fortranu 90 omogućeno je korišćenje boja. Broj i kvalitet boja zavise od korišćenog monitora i ostalih elementa sistema. Na raspolaganju su dva sistema boja. Prvi je korišćenje palete boja pri čemu je u jednom momentu moguće koristiti do 256 različitih boja. Drugi sistem je takozvani RGB (akronim od red green blue), koji omogućuje mešanje tri osnovne boje (crveno zeleno plavo) u odnosima po volji programera.
Biranje boja iz palete se obavlja pomoću tri naredbe
REZULTAT=SETCOLOR(broj_boje)

REZULTAT=SETBKCOLOR(broj_boje)
REZULTAT=SETTEXTCOLOR(broj_boje)

Primena prve naredba SETCOLOR rezultira u pojavi boje odredjene brojem u zagradi. Rezultat zavisi od monitora i grafičke kartice i preporučuje se eksperimentisanje sa različitim brojevima boja. Naredba se odnosi na sve grafičke elemente koji se iscrtavaju iza naredbe SETCOLOR.
Druga naredba SETBKCOLOR odredjuje boju pozadine teksta i grafike.
Treća naredba SETTEXTCOLOR definiše boju teksta i odnosi se na sve tekstualne poruke iza ove narebe.
U drugom RGB sistemu tri osnovne naredbe su SETCOLORRGB, SETBKCOLORRGB, and SETTEXTCOLORRGB (dodaje se sufix RGB na prethodne naredbe). Sintaksa ovih naredbi je sledeća
REZULTAT=SETCOLORRGB(#BBGGRR)
BBGGRR su heksadecimalni brojevi (pre njih postavlja se znak #). Prva dva broja definišu intenzitet plave boje (blue). Ako se napiše SETCOLORRGB(#FF0000) na ekranu će se dobiti plava boja maksimalnog intenziteta. Druge dve oznake GG odredjuju intenzitet zelene boje. Naredba SETCOLORRGB(#00FF00) daje zelenu boju najvećeg intenziteta, a SETCOLORRGB(#0000FF) daje crvenu boju. (F je oznaka za broj heksadecimalnom brojčanom sistemu).
Bela boja se dobija naredbom SETCOLORRGB(#FFFFFF), pri čemu se sve tri osnovne boje mešaju sa istim maksimalnim intenzitetom. Različite nijanse sive boje se mogu dobiti mešanjem sve tri osnovne boje u jednakim ali ne maksimalnim intenzitetima.
Broj boje u naredbi SETCOLORRGB se može zadati i bez korišćenja simbola #. Pri tome se unosi broj boje (koji je integer (4)). Preporučujemo eksperimentisanje sa različitim brojevima, jer rezultat zavisi od računarskog sistema.

Primer
Nacrtati grafik polinoma trećeg reda koji je pomnožen kvadratom funkcije sin2x. $y(x)=(ax^3+bx^2+cx+d)\sin^2 2x$ u intervalu vrednosti x od –10 do +10. Koeficijenti a,b,c i d se zadaju na ulazu.

```
PROGRAM GR3
USE MSFLIB
LOGICAL STATUS
TYPE (WXYCOORD)WXY
TYPE (RCCOORD)TEXT
TYPE (qwinfo) winfo
REAL KORAK
winfo.TYPE = QWIN$MAX
result = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
result = SETWSIZEQQ(0, winfo)
CALL SETTEXTPOSITION(INT2(1),INT2(2),TEXT)
XPOCETNI=-10.
XKRAJNJI=10.
YPOCETNI=FJA(A,B,C,D,XPOCETNI)
CALL MOVETO_W(DBLE(YPOCETNI),DBLE(XPOCETNI),WXY)
 BOJA_POZADINE=SETBKCOLORRGB(#FF0000)
 BOJATEXTA=SETTEXTCOLORRGB(#00FF00)
PRINT *, 'UNESI KOEFICIJENTE A,B,C I D'
```

```
READ *, A,B,C,D
PAUSE 'KLIKNI NA ENTER DA OBRISES EKRAN'
CALL CLEARSCREEN(GCLEARSCREEN)  !nareba cisti ekran
YMAX=1.E-10
YMIN=1.E+10
KORAK=(XKRAJNJI-XPOCETNI)/1000.
DO X=XPOCETNI,XKRAJNJI, KORAK       !petlja odredjuje najvecu i najmanju
VREDNOST=FJA(A,B,C,D,X)                    !vrednost funkcije u datom intervalu
IF(VREDNOST<YMIN)YMIN=VREDNOST
IF(VREDNOST>YMAX)YMAX=VREDNOST
END DO
BOJA = SETCOLORRGB(#FF00FF)
IF(YMIN>0.)THEN
  YMIN=YMIN*0.2
ELSE
  YMIN=YMIN*1.1
END IF
STATUS=SETWINDOW(.TRUE., XPOCETNI*1.1,YMIN,XKRAJNJI*1.1,YMAX*1.1)
CALL MOVETO_W(0.D0, DBLE(YMAX),WXY)  !postavlja koordinatne ose
STATUS= LINETO_W(0.D0,YMIN)
CALL MOVETO_W(DBLE(XPOCETNI),0.D0,WXY)
STATUS=LINETO_W(DBLE(XKRAJNJI),0.D0)
CALL MOVETO_W(DBLE(XPOCETNI),DBLE(FJA(A,B,C,D,XPOCETNI)),WXY)
DO X=XPOCETNI,XKRAJNJI, KORAK
VREDNOST=FJA(A,B,C,D,X)
STATUS=LINETO_W(DBLE(X),DBLE(VREDNOST))
CALL SLEEPQQ(10)        !PROGRAM PAUZIRA 10 MILISEKUNDI
END DO
CALL BEEPQQ (2000, 2000)
END
FUNCTION FJA(A,B,C,D,X)
FJA=(A*X**3+B*X**2+C*X+D)*(sin(2*x))**2
END FUNCTION FJA
```

Skup naredbi winfo.TYPE = QWIN$MAX
result = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
result = SETWSIZEQQ(0, winfo)
datih na početku programa služi da se maksimizira prozor u kome se pojavljuje grafika. Prva DO petlja odredjuje maksimalnu i minimalnu vrednost funkcije u intervalu od -10 do 10. Zatim se definišu windows koordinate koje su za 10 % odnosno 20 % veće od maksimalnih odnosno minimalnih vrednosti funkcije. Ovo se koristi da ne bi grafik funkcije išao tačno do donje odnosno gornje ivice ekrana. Takodje, x koordinate su za 10 % veće iz istog razloga. Nakon toga ctraju se koordinatne ose.
Sledeća DO petlja računa vrednosti funkcije u 1000 koraka i crtaju se prave linije od jedne do druge tačke. Kako je korak mali, dobija se utisak da se iscrtava kriva. Naredba CALL SLEEPQQ(vreme u milisekundama) zaustavlja programa za zadato vreme u zagradama. Na ovaj način može se pratiti crtanje linije na ekranu.
Naredba CALL BEEPQQ (2000, 3000) stvara zvučni signal po završetku crtanja; signal ima frekvenciju od 2000 Hz i traje 3000 ms. Ova dva parametra se mogu menjati pri pozivanju naredbe BEEPQQ.
Naredba LINETO_W(wx,wy) crta pravu punu liniju do tačke sa windows koordinatama wx,wz. Moguće je crtati raznovrsne linije i za to se koristi naredba

CALL SETLINESTYLE(#heksadecimalni_broj). Ako je heksadecimalni broj #FFFF onda je default i crta se neprekidna linija. #FF00 crta duge isprekidane linije a #F0F0 kratke isprekidane linije. Ova naredba deluje na sve grafičke elemente koji se pojavljuju posle nje (ne samo na linije, već i na ostale osnovne geometrijske figure).

X.4. CRTANJE OSTALIH OSNOVNIH GRAFIČKIH ELEMENATA

Pored crtanja prave linije, omogućeno je jednostavno crtanje ostalih osnovnih geometrijskih figura jednostavnim pozivanjem grafičkih funkcija. Sve ove funkcije se javljaju u dubletu, jedna se odnosi na pixel koordinate a druga sa sufiksom _W na windows koordinate.

X.4.1. Crtanje elipsi i krugova

Naredba za crtanje elipsi je

REZULTAT=ELLIPSE(KONTROLOR, X1,Y1,X2,Y2)

gde je X1,Y1 levi gornji ugao pravougaonika u koji je upisana elipsa, a X2,Y2 je desni donji ugao tog pravougaonika; ove koordinate se zadaju kao celi brojevi. U windows koordinatama naredba je

REZULTAT=ELLIPSE(KONTROLOR, WX1,WY1,WX2,WY2)

gde su WX1,WY1 windows koordinate gornjeg levog, a WX2,WY2 windows koordinate donjeg desnog ugla pravougaonika u kojem je upisana elipsa, u ovom slučaju koordinate se daju kao REAL(8).

KONTROLOR može biti $GBORDER, kada se crta samo elipsa, ili $GFILLINTERIOR kada se unutrašnja površina elipse oboji odredjenom bojom koja je zadata ranije u programu.

Za crtanje kruga ne postoji posebna naredba već se to obavlja crtanjem elipse u kvadratu. Da bi se na ekranu dobio krug potrebno je da odnos dimenzija ekrana po x i po y osi bude približan 4:3.

X.4.2.Crtanje pravougaonika

Naredbe za crtanje pravougaonika su

REZULTAT=RECTANGLE($GBORDER,X1,Y1,X2,Y2)

i REZULTAT=RECTANGLE_W($GBORDER,WX1,WY1,WX2,WY2)

sa sličnim značenjem kao i kod elipse. Umesto reči rezultat može se upotrebiti na primer PRAVOUGAONIK ili nešto slično po želji korisnika. Naredba SETLINESTYLE deluje na pravouganike nacrtane naredbom RECTANGLE.

X.4.3.Crtanje lukova

Crtanje lukova se obavlja naredbom

LUK=ARC(X1,Y1,X2,Y2,X3,Y3,X4,Y4) ili

LUK=ARC_W(WX1,WY1,WX2,WY2,WX3,WY3,WX4,WY4).

X1,Y1 su koordinate levog gornjeg temena, a, X2,Y2 su koordinate desnog donjeg temena pravougaonika u koji je upisan luk;

X3,Y3 su komponente startnog vektora, vektor spaja prvu tačku luka sa centrom pravougaonika, a luk se povlači od te tačke.

X4,Y4 komponente završnog vektora; završni vektor spaja zadnju tačku luka sa centrom pravougaonika.
Centar luka je u centru pravougaonika koji uokviruje luk.
Primer. Nacrtati, elipsu. Nacrtati dve koncentrične kružnice u različitim bojama, druga ima manji poluprečnik za 2 % i da je ispunjena. Nacrtati proizvoljan pravougaonik i proizvoljan luk.

```
PROGRAM GR3
USE MSFLIB
LOGICAL STATUS
TYPE (qwinfo) winfo
winfo.TYPE = QWIN$MAX
result = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
result = SETWSIZEQQ(0, winfo)
STATUS=SETWINDOW(.TRUE., -19.,-15.,19., 15.)
BOJAPOZADINE =SETBKCOLORRGB(#999999)
BOJATEXTA=SETTEXTCOLORRGB(#FF0000)
BOJA=SETCOLORRGB(#FF00FF)
PRINT *, 'ELIPSA, KRUGOVI, PRAVOUGAONIK I LUK'
REZULTAT=ELLIPSE_W($GBORDER, -12., -2., -10.,4.)
KRUG=ELLIPSE_W($GFILLINTERIOR, -2.*0.98, -2.*0.98, 2.*0.98,2.*0.98)
REZULTAT=SETCOLORRGB(#00FF00)
KRUG=ELLIPSE_W($GBORDER, -2., -2., 2., 2.)
PRAVOUGAONIK=RECTANGLE_W($GBORDER,3., 3., 5., 5.)
BOJA=SETCOLORRGB(#775533)
LUK=ARC_W(-14., -10., 12.,10., -2.,-1.,1.,1.)
END
```

X.4.4.Crtanje poligona

Pologona linija se crta pomoću naredbi
REZULTAT= POLYGON(kontrolor, tačke, broj tačaka)
ili u windows koordinatama
REZULTAT= POLYGON_w(kontrolor, wtačke, broj tačaka)

Kao i ranije, kontrolor može biti $GBORDER, za crtanje okvira poligona, ili $GFILLINTERIOR, za ispunu poligona. Broj tačaka je broj temena poligona; tačke ili wtačke je izvedeni tip podataka koji sadrži koordinate temena poligona(u pixel ili windows koordinatnom sistemu). Na početku programa se definiše tip podataka

TYPE (xycoord) ime_poligona(broj_tačaka) za pixel koordinate, ili
TYPE (wxycoord) ime_poligona(broj_tačaka) za windows koordinate.
Koordinate temena se definišu naredbama
ime_poligona(1)xcoord=vrednost x koordinate prve tačke
ime_poligona(1)ycoord=vrednost y koordinate prve tačke
.
.
ime_poligona(n)xcoord=vrednost x koordinate zadnje n te tačke
ime_poligona(n)ycoord=vrednost y koordinate zadnje n te tačke

Kod windows koordinata koristi se promenljive *wx* i *wy* u dvostrukoj tačnosti.

Primer. Nacrtati petougao sa koordinatama temena (-5,-5),(5, -5),(5,5),(-5,10),(-10,1)
Program je

```
PROGRAM PETOUGAONIK
USE MSFLIB
LOGICAL STATUS
TYPE (qwinfo) winfo
TYPE (WXYCOORD) PETOUGAO(5)
winfo.TYPE = QWIN$MAX
result = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
result = SETWSIZEQQ(0, winfo)
STATUS=SETWINDOW(.TRUE., -20.,-15.,20., 15.)
BOJA=SETCOLORRGB(#FF00FF)
 PETOUGAO(1).WX=-5.
 PETOUGAO(1).WY=-5.
 PETOUGAO(2).WX=5.
 PETOUGAO(2).WY=-5.
 PETOUGAO(3).WX=5.
 PETOUGAO(3).WY=5.
 PETOUGAO(4).WX=-5.
 PETOUGAO(4).WY=12.
 PETOUGAO(5).WX=-10.
 PETOUGAO(5).WY=-1.
 REZULTAT=POLYGON_W($GBORDER,PETOUGAO,5)
 END
```

Naredba *TYPE (WXYCOORD) PETOUGAO(5)* definiše izvedeni tip promenljive pod imenom PETOUGAO, i to kao niz sa 5 elemenata. Izvedeni tip podataka WXYCOORD je oblika

```
 TYPE  wxycoord
 REAL(8)  wx
 REAL(8)  wy
 END TYPE wxycoord
```

tj. sastoji se od dve promenljive u dvostrukoj tačnosti.

LITERATURA

Ian Chivers and Jane Sleightholme. INTRODUCTION FORTRAN 90. Springer. Berlin 1995

V. Rajaraman. COMPUTER PROGRAMMING IN FORTRAN 77. (WITH AN INTRODUCTION TO FORTRAN 90). Prentice Hall of India. New Delhi 1999.

Loren P. Messner. ESSENTIAL FORTRAN 90/95. International Thompson Publishing Company. Boston MA USA 1997.

Rama N. Reddy and Carol A. Ziegler. FORTRAN 77 WITH 90: APPLICATIONS FOR SCIENTISTS AND ENGINEERS. Second edition. West Publising Company. Mineapolis 1994.

Stacey L. Edgar. FORTRAN FOR THE 90S. Computer Science Press. New York. 1992