Uvod u programiranje u Turbo Pascalu 7 (1)

Pregled razvoja programskih jezika

Prvi računari bili su vrlo složeni za korišćenje. Njih su koristili isključivo stručnjaci koji su bili osposobljeni za komunikaciju s računarom. Ta komunikacija se sastojala od dva osnovna koraka: davanje uputstava računaru i čitanje rezultata obrade. I dok se čitanje rezultata vrlo brzo učinilo koliko-toliko snošljivim uvođenjem štampača na kojima su se rezultati ispisivali, unošenje uputstava – programiranje – se sastojalo od mukotrpnog upisivanja niza nula i jedinica. Ti nizovi su davali računaru uputstva kao što su: „saberi dva broja", „premesti podatak s neke memorijske lokacije na drugu", „skoči na neku instrukciju izvan normalnog redosleda instrukcija" i slično. Kako je takve programe bilo vrlo složeno pisati, a još složenije čitati i ispravljati, ubrzo su se pojavili prvi programerski alati nazvani *asembleri* (engl. *assembler*s).

U asemblerskom jeziku svaka mašinska instrukcija predstavljena je mnemonikom koji je razumljiv ljudima koji čitaju program. Tako se sabiranje najčešće obavlja mnemonikom ADD, dok se premeštanje podataka obavlja mnemonikom MOV. Time se postigla bolja čitljivost programa, ali i dalje je bilo vrlo složeno pisati programe i ispravljati ih jer je bilo potrebno davati sve, pa i najmanja uputstva računaru za svaku pojedinu operaciju. Javlja se problem koji će kasnije, nakon niza godina, dovesti i do pojave tzv. viših programskih jezika, tj. potrebno je razviti programerski alat koji će osloboditi programera rutinskih poslova i omogućiti mu da se usredi na problem koji rešava.

Zbog toga se pojavljuje niz viših programska jezika, koji preuzimaju na sebe neke „dosadne" programerske poslove. Tako je FORTRAN bio posebno pogodan za matematičke proračune, zatim BASIC koji se vrlo brzo učio, COBOL koji je bio po pravilu namenjen upravljanju bazama podataka.

Šta je program i kako ga napisati?

Elektronski računari postali su danas pribor kojim se svakodnevno koristimo kako bismo sebi olakšali posao ili se zabavili. Prvi navod će mnogi poricati, navodeći kao kontraargument činjenicu da im je za podizanje novca u banci pre trebalo znatno manje vremena nego otkad su šalteri kompjuterizovani. Međutim, činjenica je da su mnogi poslovi danas nezamislivi bez računara; u krajnjoj liniji, dokaz za to je tekst koji upravo čitate koji je u potpunosti napisan pomoću računara.

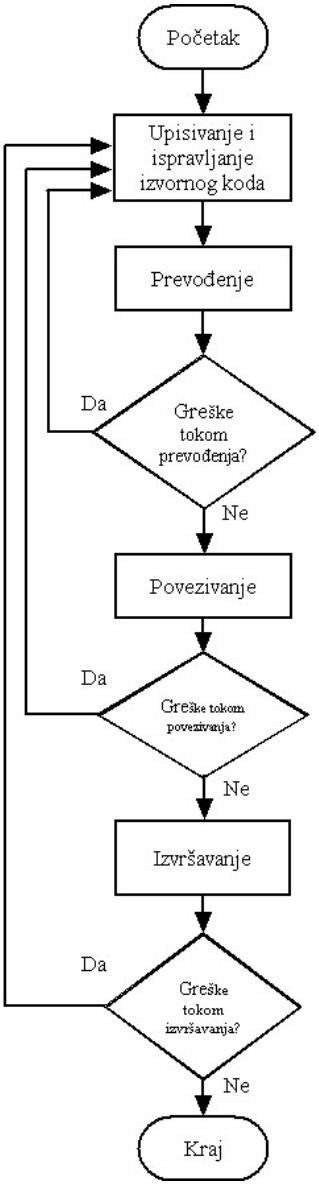
Sam računar, čak i kada se uključi na niskonaponsku mrežu, nije u stanju da uradi ništa korisno. Na današnjim računarima se ne može čak ni zagrejati jelo, što je inače bilo moguće na računarima sa elektronskim cevima. Ono što vam nedostaje je pamet neophodna za koristan rad računara: programi, programi, … i samo programi. Pod programom podrazumevamo niz naredbi u mašinskom jeziku koje procesor u vašem računaru izvršava i shodno njima obrađuje podatke, izvodi matematičke proračune, ispisuje tekstove, iscrtava krive na ekranu itd. Pokretanjem programa sa diska, diskete ili kompakt diska (CD-ROM-a), program se učitava u radnu memoriju računara i procesor počinje sa mukotrpnim postupkom njegovog izvršavanja.

Programi koje pokrećete na računaru su u *izvršnom obliku* (engl. *executabl*e), razumljivom samo procesoru vašeg (i njemu sličnih) računara. U suštini, mašinski kôd se sastoji od nizova binarnih cifara: nula i jedinica.

Budući da su današnji programi tipično dužine nekoliko megabajta, naslućujete da ih autori nisu pisali direktno u mašinskom kôdu. Gotovo svi današnji programi se pišu u nekom od *viših programskih jezika* (FORTRAN, BASIC, Pascal, C) koji su donekle razumljivi i ljudima (barem onima koji imaju nešto pojma o engleskom jeziku). Naredbe u tim jezicima se sastoje od mnemonika. Kombinovanjem tih naredbi programer slaže *izvorni kôd* (engl. *Source cod*e) programa, koji se pomoću posebnih programa *prevodilaca* (engl. *compile*r) i *povezivača* (engl. *linker)* prevodi u izvršni kôd. Prema tome, pisanje programa u užem smislu podrazumijeva pisanje izvornog kôda. Međutim, kako pisanje koda nije samo sebi svrha, pod pisanjem programa u širem smislu podrazumeva se i prevođenje, odnosno povezivanje programa u izvršni kôd. Prema tome, možemo govoriti o četiri faze kod pravljenja programa:

1. pisanje izvornog kôda
2. prevođenje izvornog kôda,
3. povezivanje u izvršni kôd i
4. testiranje programa.

Da bi se za neki program moglo reći da je uspešno napravljen, treba uspešno proći kroz sve četiri faze. Kao i svaki drugi posao, i pisanje programa zahteva određeno znanje i veštinu. Prilikom pisanja programera vrebaju Scile i Haribde, danas poznatije pod nazivom greške ili *bugovi* (engl. *bug* - stenica) u programu. Ako se pojave greške u nekoj od faza izrade programa, izvorni kôd treba doraditi i ponoviti sve prethodne faze. Zbog toga postupak izrade programa nije pravolinijski, već manje-više podseća na mukotrpno kretanje u krug. Na slici 1. je šematski prikazan celokupni ciklus izrade programa, od njegovog početka, pa sve do njegovog završeta. Analizirajmo najvažnije faze izrade programa.



Slika 1. Tipičan razvoj programa

Prva faza programa je pisanje izvornog kôda. U principu, izvorni kôd se može pisati u bilo kom programu za uređivanje teksta (engl. *text editor*), međutim, većina današnjih prevodilaca i povezivača se isporučuje kao celina zajedno s ugrađenim programom za upisivanje i ispravljanje izvornog kôda. Te programske celine poznatije su pod nazivom *integrisano razvojno okruženje* (engl. *integrated development environment* - *IDE*).

Nakon što je pisanje izvornog kôda završeno, on se čuva u datoteci izvornog kôda na disku. Toj datoteci se obično daje nekakvo smisleno ime, pri čemu se ono za kôdove pisane u programskom jeziku Pascal obično proširuje nastavkom `.pas`, na primer `primer.pas`. Nastavak (tip datoteke) je potreban samo zato da bismo izvorni kôd kasnije mogli lakše pronaći. Zatim sledi prevođenje izvornog kôda. U integrisanom razvojnom okruženju program za prevođenje se pokreće pritiskom na neki taster na tastaturi, pritiskom odgovarajućeg tastera na tastaturi ili izborom opcije iz menija sa vrha prozora programa – ako prevodilac nije integrisan, pozivanje je nešto složenije.

Prevodilac tokom prevođenja proverava sintaksu napisanog izvornog kôda i u slučaju uočenih ili naslućenih grešaka ispisuje odgovarajuće poruke o greškama, odnosno

upozorenja. Greške koje prijavi prevodilac nazivaju se *greškama pri prevođenju* (engl. *compile-time error*s). Nakon toga programer će pokušati ispraviti sve navedene greške i ponovo prevesti izvorni kôd – sve dok prevođenje koda ne bude uspešno okončano, ne može se pristupiti povezivanju koda. Prevođenjem izvornog dobija se datoteka *objektnog kôda* (engl. *object cod*e), koja se lako može prepoznata po tome što obično ima nastavak `.o` ili `.obj` (u našem primeru bi to bilo `primer.obj`).

Nakon što su ispravljene sve greške uočene prilikom prevođenja i kôd ispravno preveden, pristupa se povezivanju objektnih kôdova u izvršni. U većini slučajeva objektni kôd dobijen prevođenjem programerovog izvornog kôda treba povezati sa postojećim *bibliotekama* (engl. *librarie*s). Biblioteke su datoteke u kojima se nalaze već prevedene gotove funkcije ili podaci. One se isporučuju zajedno s prevodiocem, mogu se posebno kupiti ili ih programer može sam razviti. Bibliotekama se izbegava ponovno pisanje vrlo često korišćenih operacija. Tipičan primer za to jesu biblioteka matematičkih funkcija koja se uvek isporučuje uz prevodilac, a u kojoj su definisane sve funkcije poput trigonometrijskih, eksponencijalnih i slično.

Prilikom povezivanja proverava se mogu li se svi pozivi kodova realizovati u izvršnom kodu. Ukoliko povezivač tokom povezivanja uoči neku nepravilnost, ispisaće poruku o greški i onemogućiti generisanje izvršnog koda. Ove greške nazivaju se *greškama pri povezivanju* (engl. *link-time error*s) – sada programer mora prionuti na ispravljanje grešaka koje su nastale pri povezivanju. Nakon što se isprave sve greške, kôd treba ponovno prevesti i povezati.

Uspešnim povezivanjem dobija se izvršni kôd. Međutim, takav izvršni kôd još uvek ne garantuje da će program raditi ono što ste zamislili. Na primer, može se dogoditi da program radi pravilno za neke vrednosti podataka, ali da se za druge ponaša nepredvidivo. U tom se slučaju radi o *greškama pri izvođenju* (engl. *run-time error*s).

Da bi program bio potpuno korektan, programer mora da istestira program da bi uočio i ispravio te greške, što znači ponavljanje celog postupka u lancu „ispravljanje izvornog koda-prevođenje-povezivanje-testiranje”. Kod jednostavnijih programa broj ponavljanja će biti manji i smanjivaće se proporcionalno sa rastućim iskustvom programera. Međutim, kako raste složenost programa, tako se povećava broj mogućih grešaka i celi postupak izrade programa neiskusnom programeru može postati mukotrpan.

Za ispravljanje grešaka pri izvođenju, programeru na raspolaganju stoje programi za otkrivanje grešaka (engl. *debugge*r). Radi se o programima koji omogućavaju prekid izvođenja izvršnog koda programa koji testiramo na unapred zadatim naredbama, izvođenje programa naredbu po naredbu, ispis i promene trenutnih vrednosti pojedinih podataka u programu. Najjednostavniji programi za otkrivanje grešaka ispisuju izvršni kôd u obliku mašinskih naredbi. Međutim, većina današnjih naprednih programa za otkrivanje grešaka su *simbolički* (engl. *symbolic debugge*r) – iako se izvodi prevedeni, mašinski kôd, izvođenje programa se prati preko izvornog kôda pisanog u višem programskom jeziku. To omogućava vrlo lagano lociranje i ispravljanje grešaka u programu.

Osim grešaka, prevodilac i povezivač redovno prijavljuju i upozorenja. Ona ne onemogućavaju nastavak prevođenja, odnosno povezivanja koda, ali predstavljaju potencijalnu opasnost. Upozorenja se mogu podeliti u dve grupe. Prvu grupu čine upozorenja koja javljaju da kôd nije potpuno korektan. Prevodilac ili povezivač će zanemariti našu grešku i prema svom nahođenju generisati kôd. Drugu grupu čine poruke koje upozoravaju da

„nisu sigurni da je ono što smo napisali upravo ono što smo želeli napisati", tj. radi se o dobronamernim upozorenjima na zamke koje mogu proizići iz načina na koji smo program napisali. Iako će, uprkos upozorenjima, program biti preveden i povezan (možda čak i korektno), pedantan programer neće ta upozorenja nikada zanemariti – ona često upućuju na uzrok grešaka pri izvođenju gotovog programa. Za precizno tumačenje poruka o greškama i upozorenja neophodna je dokumentacija koja se isporučuje uz prevodilac i povezivač.

Da zaključimo: „Šta nam, dakle, treba za pisanje programa na jeziku Turbo Pascal?" Osim računara, programa za pisanje i ispravljanje teksta, prevodioca i povezivača trebaju vam još samo tri stvari: interesovanje, puno slobodnog vremena i dobra literatura. Interesovanje verovatno postoji, ako ste sa čitanjem ovog teksta stigli čak dovde. Slobodno vreme će vam trebati da isprobate primere i da se sami oprobate u bespućima Turbo Pascala.

Uvod u Turbo Pascal

Zašto je Pascal dobar izbor programskog jezika? Odgovor će se sam nametnuti, ako se kaže bar nekoliko reči o nastanku jezika Pascal.

Njegov tvorac Niklaus Wirth (profesor na Eidgenossische Technische Hochschule, Cirih, Švajcarska) izborom imena jezika počastio je velikog francuskog naučnika i filozofa Blaisea Pascala (1623 — 1662) koji je izgradio jedan od prvih mehaničkih računara.

Stvarajući programski jezik Pascal, Wirth je težio da nauči učenika da programira tako da ne mora da menja logiku stvari koje je već ranije naučio, da ono što želi da izrazi bude i u programu čitljivo, jasno i pregledno. Omogućiti učeniku da svoje misli saopštava programom tako, da naredbe slede misli, da se njihov tok ne gubi u iznenadnim skokovima, da se razumeju bez muke. Wirth je želeo da se programiranje uči takvim jezikom u kojem će se misli lako prenositi, a s druge strane, da program jasno odražava misli programera. Imajući na raspolaganju više programske jezike kao što su Fortran, Algol i Basic Wirth nije bio zadovoljan. Pokušao je da svoje želje zadovolji ubacivanjem novih stvari u Algol. Ubrzo je bio nezadovoljan krparenjem Algola i odlučio je da napiše novi jezik. S radom je počeo šezdesetih godina, a prvi program-prevodilac proradio je u kasnim sedamdesetim godinama.

Wirth je prvi proklamovao prednosti strukturnog programiranja. One se sastoje u mogućnosti razlaganja problema u manje celine (module ili blokove) od kojih svaka sme da ima samo jedan ulaz i jedan izlaz izbegavajući naredbu GO TO. Informacije se prenose u sledu iz bloka u blok, dok se ne dođe do konačnog rešenja.

Wirth je želeo kao prvo, jezik koji će odražavati principe sistematičnog kreiranja i drugo, da njegovo ostvarenje na računaru bude pouzdano i efikasno. To je i ostvario Pascalom, zajednički radeći sa svojim kolegama, a baza mu je bila Algol.

Wirth smatra da učenje programiranja zapravo znači učenje struktura podataka, toka informacija i kontrole.

Programski jezici kao što su Fortran i Basic nemaju takve osobine i konstrukcije da programer može direktno izraziti programom svoje ideje. Često se nameće potreba da

razmišlja o datom ostvarenju na računaru, pa njemu prilagođava svoje ideje. Oba jezika zahtevaju upotrebu GO TO naredbe, koja doprinosi „nečitljivosti" programa, gubi se sled misli. Dalje je veliki nedostatak što nema strukturiranih podataka, koji dozvoljavaju, da se jedan složeni podatak sastoji od različitih jednostavnih tipova podataka, pa se moraju stvarati komplikovane konstrukcije „paralelnih polja".

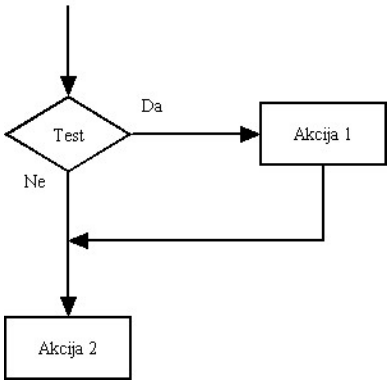
Dobre osobine Pascala u odnosu na druge jezike mogu se sažeto sagledati s obzirom na tri elementa:

1) nomenklatura jezika,
2) tok programa i
3) struktura podataka

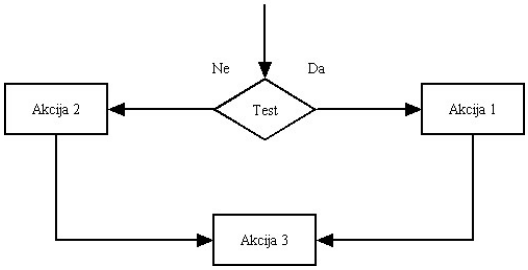
Pođimo redom. Prvo, primer *nomenklature.* Svakom programeru početniku izraz $a = a + 1$, s kojim se susreće u Fortranu i Basicu, znači nešto što se kosi s već pre usvojenom matematičkom logikom. Pascal za taj slučaj ima operator dodeljivanja (znak je :=) koji se razlikuje od relacijskog operatora jednakosti (=), pa se tako gornji izraz piše $a := a + 1$, a čita „a je zamenjeno vrednošću $a + 1$" ili „a postaje $a + 1$". S druge strane $a = a + 1$ u Pascalu daje vrednost *laž,* što je u skladu s matematičkom logikom.

Sada o *toku programa.* Wirth smatra da se GO TO može izbeći, ako se pristupi sistematičnom kreiranju i posmatranju mogućih situacija u programu. Njih najlakše predstavljamo dijagramom toka, sasvim uopšteno nevezano za određeni jezik:

a) Slučaj grananja



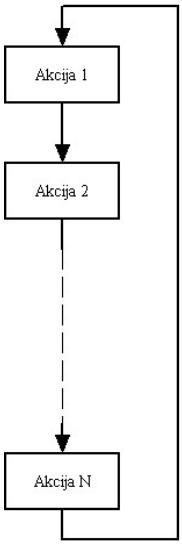
Akcija 1 se preduzima samo u slučaju da je ispunjen uslov koji se ispituje testom.



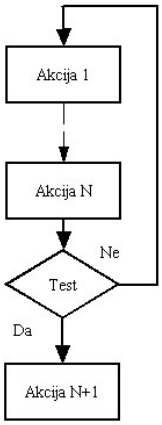
Moguće su dve akcije u zavisnosti od rezultata testa.

b) Slučaj ponavljanja

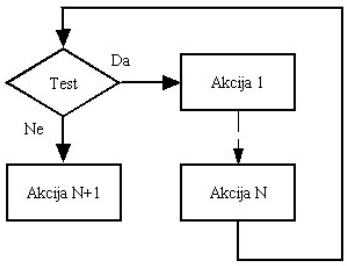
Bezuslovno ponavljanje	Uslovno ponavljanje (1)	Uslovno ponavljanje (2)



Akcije od 1 do N ponavljaju se određeni broj puta.

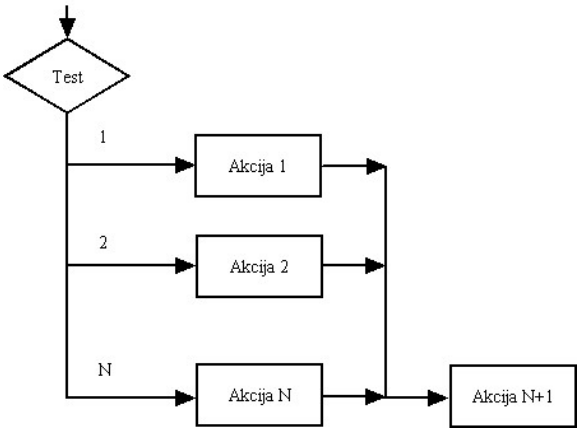


Ponavljanje do akcije N izvrši se bar jedanput, a zatim se u zavisnosti od rezultata testa nastavlja ili prekida.



Ponavljanje zavisi od testa, može se desiti da se obavi ili ne obavi nijednom.

c) Slučaj selekcije



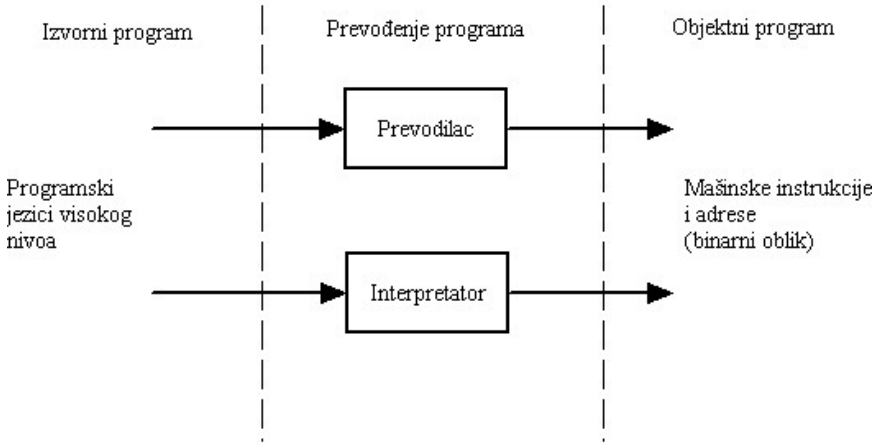
Tok programa zavisi od selektora čiju vrednost određuje test.

Svi su ovi tokovi ostvarivi u Pascalu bez naredbe GO TO primenom naredbi FOR, IF . . THEN . . ELSE, REPEAT, WHILE, CASE. One omogućuju da se i letimičnim čitanjem dobro napisanog programa Pascal može pretpostaviti njegovo ponašanje, tj. statički plan koda određuje njegovo dinamičko ponašanje. To je vrlo značajno za programera, jer on mora ispravljati greške u programu i mora održavati program. Ukoliko je moguće obuhvatiti što više informacija jednostavnim čitanjem programa, to će biti lakše i brže izdvojiti greške.

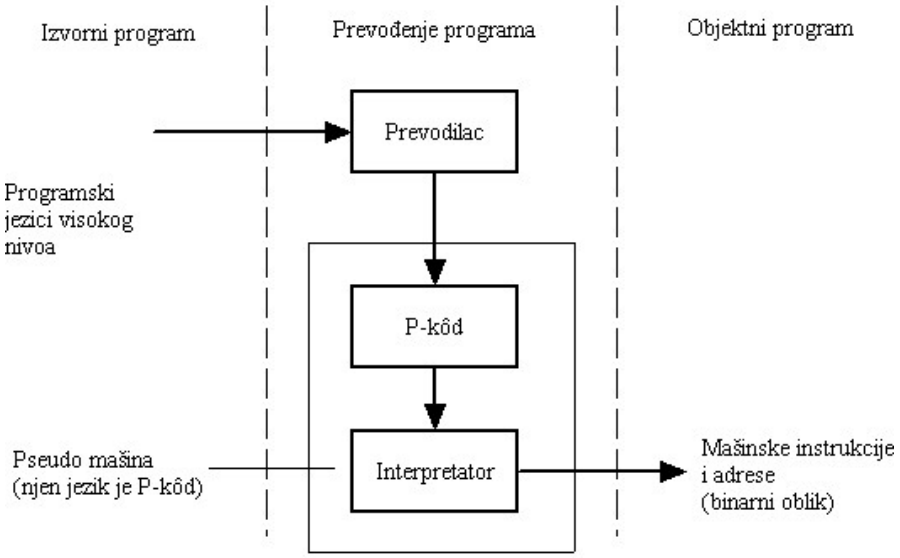
Pascal ima bogate mogućnosti izražavanja različitih *tipova podataka.* Uz standardne numeričke podatke u obliku celih i realnih brojeva te znakova, programer može sam definisati svoj tip podataka. Zatim, vrlo jednostavno rešava zapis (slog) u kojem se nalazi više tipova podataka združenih u celovit podatak. Na primer, podatak o učeniku zahteva podatke: ime i prezime, adresu, godinu rođenja, pol, razred, uspeh itd.. U Pascalu se to može izraziti strukturiranim tipom podataka RECORD. Na taj način Pascal omogućuje programeru da nađe direktan put izražavanja svojih misli programom. U drugim (starijim) programskim jezicima programer mora stvarati, održavati i dokumentovati strukture, koje su već deo samog Pascala. Zapravo, to omogućuje dobar program prevodilac (kompajler)! Na tome je Wirth takođe radio, i bio svestan mogućeg sukoba između pogodnosti jezika i njegove izvodljivosti na računaru, odnosno, ono što će značiti ugodno proširenje za programera može postati poteškoća za stvaraoca prevodioca. Međutim, i tu je Pascal jak! *Prvo,* promenljive u Pascalu mogu imati lokalno značenje za jedan blok u programu, na primer, u funkciji ili proceduri (koje su nezavisni delovi u odnosu na druge blokove). S druge strane lokalne promenljive za pisca programa prevodioca znače organizaciju memorije na osnovu potreba. Deo memorije koji je blok zauzimao pri izvršavanju, oslobađa se nakon izvršavanja i može se dodeliti drugim delovima programa. *Drugo,* podintervalni tip podataka omogućuje programeru da veoma opisno može izražavati podatke, a stvaraocu prevodioca daje mogućnost da podatke memoriše na manjem prostoru memorije jer se prevodiocem svakom podatku dodeljuje redni broj i on se memoriše.

U Pascalov prevodilac Wirth nije ugradio eksponencijalni operator. To je koristan operator, ali i vrlo skup! Pascal ga ostvaruje upotrebom petlje u programu.

Važna osobina viših programskih jezika je njihova prenosivost (portabilnost). I u tom pogledu Pascal je povoljan, budući da je Wirth veliki zagovornik prenosive sistemske podrške. Naime, izvorni program u Pascalu prevodiocem prelazi u njegov pseudo kod, P-kôd, koji se zatim izvodi interpretacijom. Svojstvo Pascala da generiše P-kôd dopfinosi prenosivosti sa jednog računara na drugi. Sledeća šema prikazuje način izvršavanja programa u višim programskim jezicima.



Za Pascal bi se mogao napraviti (u suglasnosti s navedenom šemom), ovakav prikaz:



Podaci

Podacima se naziva uopšteno sve ono što se može obraditi računarom ili dobiti kao rezultat obrade. Mogu to biti činjenice, brojevi, slova, situacije i slično.

Svaki program sadrži u sebi podatke koje obrađuje. Njih možemo podeliti na Nepromenljive podatke, tj. *konstant*e, i promenljive podatke, tj. *promenljive (varijable)*. Najjednostavniji primer konstanti su brojevi (5, 1 0, 3.4 59). Promenljive su podaci koji mogu menjati svoj iznos. Stoga se oni u izvornom kôdu predstavljaju ne svojim iznosom već simboličkom oznakom, *imenom promenljive*.

Svaki podatak ima dodeljenu oznaku tipa koja govori o tome kako se dotični podatak čuva u memoriji računara, koji su njegovi dozvoljeni rasponi vrednosti, kakve se operacije mogu izvesti sa tim podatkom i slično. Tako razlikujemo celobrojne, realne, logičke, pokazivačke podatke. U narednim odeljcima upoznaćemo se sa ugrađenim tipovima podataka i pripadajućim operatorima.

U računaru su svi podaci predstavljeni binarnim oblikom tj. određenim brojem bitova. U višim programskim jezicima postoje različiti tipovi podataka, da bi se lakše moglo izraziti sve ono što se želi obraditi računarom. Računar mora prepoznati, a zatim prihvatiti različite tipove podataka. On ima različitu internu reprezentaciju za različite tipove podataka. Tip podataka određuje skup vrednosti koje podatak može imati.

U Pascalu postoji nekoliko tipova podataka. Mogu se podeliti u tri grupe:

1) jednostavni,
2) složeni ili strukturirani,
3) pokazivači (engl. pointer).

Jednostavni tip uključuje četiri skalarna tipa podataka: celobrojni, realni, Booleov i znakovni (engl. integer, real, Boolean, char). Uz to, Pascal dozvoljava da korisnik sam definiše svoj tip podataka (engl. user defined type).

Strukturirani tipovi podataka su polja, zapisi, datoteke i skupovi (engl. arrays, records, files, sets).

Podaci tipa *pokazivač* koriste se za dinamičke promenljive.

U narednim odeljcima biće obrađeni navedeni tipovi podataka.

Celobrojni tip

Za prikaz celih brojeva koristi se celobrojni tip podataka. Na primer, celobrojne kostante su:

30 351
-67 +21

Znak + je ispred konstante proizvoljan. Ako ne postoji nikakav predznak pretpostavlja se da je konstanta pozitivna.

Napomena:
Najveća celobrojna konstanta u obradi zavisi od računara na kojem se radi, na primer, raspon na jednom računaru može biti od —32767 do 32767, dok na drugom može biti od —281474976710655 do iste pozitivne vrednosti.

Realni tip

Brojevi koji imaju decimalni deo, istorijski se nazivaju realni. Na primer, realne konstante su:

8.5 -4.73 3.777 -7.2

Realna konstanta u Pascalu ne sme počinjati ili završavati decimalnom tačkom, na primer:

459. .357

Umesto toga treba da se napiše 459.0 i 0.357

U Pascalu je dozvoljeno pisanje realnih vrednosti s pokretnom tačkom (engl. floating-point). Takav način pisanja progodan je za vrlo velike i vrlo male vrednosti, na primer 750000000.0 ili 0.000000781. Tako se izbegava pisanje velikog broja nula.

Primer:

8.5E + 7 odgovara **8.5** $\mathbf{10^7}$**.**

Pisanje brojeva u obliku pokretne tačke poznato je i pod nazivom eksponencijalna notacija. Eksponent pokazuje za koliko mesta treba decimalnu tačku pomeriti u levo ili desno. Ako je eksponent pozitivan tačka se pomera udesno, a ako je negativan tačka se pomera ulevo.

Predznak + iza znaka E može se izostaviti. Evo još nekoliko primera prikaza realnih konstanti:

Prikaz broja u eksponencijalnom obliku	Konvencionalni prikaz broja
5.2E4	5200
3.141 E2	314.1
3.57 E-2	0.035
777.9E -	7

Konstanta napisana u eksponencijalnom obliku uvek prikazuje realni broj. To je slučaj i kada se decimalna tAčka izostavi.

Napomena:

Pascal nema decimalni zarez, nego decimalnu tačku. Zbog toga i svi decimalni brojevi u tekstu imaju tačku, a ne zarez.

Na primer, sledeći brojevi napisani u istom redu su ekvivalentni:

754E - 1	**754.0E - 1**	**75.4**
1E3	**1.0E3**	**1000.0**
1E-4	**1.0E-4**	**0.0001**

Broj 235 i 235.0 prikazuje istu vrednost u običnoj aritmetici, međutim, tako napisane vrednosti u Pascalu znače različite tipove podataka. One će biti i različito prikazane unutar računara pa će i rezultat obrade biti različit. One se zato ne smeju poistovetiti u programu Pascal.

Tačnost računanja u računaru nije ista ako se računa s realnim brojevima ili s celim brojevima. Ako se koristi zapisivanje brojeva u eksponencijalnom obliku tačnost zavisi od broja bitova koji se koriste za zapis mantise. Veličina broja zavisi od broja bitova koji služe za zapis eksponenta. U praksi se uvek radi zaokruživanje decimalnih brojeva zbog čega nastaje izvesna greška.

Računanje u realnoj aritmetici u računaru troši više vremena od celobrojne aritmetike.

Booleov tip

Računar u radu proverava različite uslove i s obzirom na rezultat ispitivanja nastavlja rad. Zbog toga se upotrebljava tip podataka koji ima dve vrednosti, prikazane sljedećim konstantama:

False	True

Laž	istina

Često se nazivaju logičkim vrednostima, ali u Pascalu ih nazivaju Booleove vrednosti, u čast engleskog matematičara Georga BOOLE-a, koji je prvi razvio logičku algebru u 19. veku. U računaru se Booleove vrednosti prikazuju jednim bitom, koji je obično 0 za laž, a 1 za istinito.

Znakovni tip

Ovu grupu podataka čine znakovi:

1. slova abecede,
2. numerički znakovi od 0 do 9,
3. znakovi interpunkcija i
4. specijalni znakovi.

Različiti računari koriste različite skupove znakova i njihovih kodova. Najčešće korišćeni kodovi za znakove su ASCII i EBCDIC.

Znakovna konstanta sastoji se od znaka ograđenog jednostrukim navodnicima, na primer:

'A' 'a' *'S' '* + *' '-'* 'X'

Jednostruki navodnici su neophodni da bi računar znao, na primer, da + znači znakovnu konstantu za razliku od oznake sabiranja, ili da je 5 znakovna konstanta a ne celobrojni podatak 5.

Više znakova kao što su reči ili rečenice nazivaju se *nizovi* (engl. string). U programu se niz znakova može koristiti kao konstanta s time da se niz znakova omeđi navodnicima. Na primer:

'RAČUNARSTVO' '2003'

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (2)

Ispisivanje podataka

Da bi se dobio rezultat obrade koriste se naredbe za ispisivanje (odnosno, štampanje). Računar može dati ispis u različitim formatima. Svaki računar ima svoj „standardni" format za ispisivanje podataka. Međutim, u višim programskim jezicima postoji mogućnost da i programer sam odredi format ispisivanja preko različitih formata za ispis. Takav je slučaj i u Pascalu. Svaki računar koji radi u Pascalu ima svoj „standardni" format i mogućnost da programer kreira format ispisa.

Ispisivanje celobrojnih vrednosti

a) naredba WRITE

Ova naredba ispisuje vrednost koja je napisana u zagradi iza reči WRITE. Na primer:

```
WRITE (1, 2, 3, 4, 5)
```

ispisuje:

```
1       2       3       4       5
```

Kad želimo da računar izvrši nekoliko naredbi jednu iza druge, tada pišemo
naredbe pojedinačno, odvajajući ih tačka-zarezom. Obično se samo jedna naredba piše u jednom redu.

```
WRITE     (1, 2);
WRITE     (3);
WRITE     (4, 5)
```

b) naredba WRITELN

Ova naredba omogućuje ispis u novom redu. Na primer, za naredbe:

```
WRITELN  (1, 2);
WRITELN  (3);
WRITELN  (4, 5)
```

ispis će biti u obliku:

```
1       2
3
4       5
```

Ako želite da se preskoči red pri ispisu, piše se samo WRITELN, na primer:

WRITELN (-10, 5, 8); WRITELN (-4); WRITELN; WRITELN (-100, 35)

Nakon izvršenja svih naredbi za ispisivanje biće preskočen jedan red između drugog i trećeg reda, znači ispis će imati oblik:

-10 5 8
-4

-100 35

U upotrebi je češće naredba WRITELN nego WRITE. Ako želimo sami da odredimo oblik ispisa (na primer, razmak između brojeva) može se upotrebiti oznaka za proizvoljno dugo polje ispisa (engl. field width parameters).

Sledeći primer to ilustruje:

WRITELN (-5 :10)

Ovo znači da će se ispisati vrednost -5, a za ispis je rezervisano 10 mesta

								-	5

Evo još jednog primera:

WRITELN (15 :8, -100 :8, 1 :8)

						1	5					-	1	0	0								1

Ispisivanje realnih brojeva

Kad se realni brojevi ispisuju bez oznake za željenu dužinu ispisa većinom se ispisuju vrednosti u eksponencijalnom obliku.

Na primer:

WRITELN (1.5, -8.25)

1.500000000000E + 00 -8.250000000000E + 00

Za konvencionalno ispisivanje možemo rezervisati dva područja proizvoljno duga i to tako, da prvi broj pokazuje ukupan broj mesta za ispis, a drugi broj pokazuje koliko je od toga decimalnih mesta.

Na primer:

WRITELN (5.4 :7 :1, -8.25 :8 :2)

ispisuje:

				5	.	4				-	8	.	2	5

Drugi primer:

WRITELN (3.14 :7 :3, 9.58 :8 :4, 1.0 :6 :3)

ispisuje:

		3	.	1	4	0			9	.	5	0	0	0		1	.	0	0	0

Ispisivanje Booleovih vrednosti

Booleove vrednosti se ispisuju koristeći reči: istina i laž (true, false). Na primer:

WRITELN (true, false)

TRUE FALSE

Ispisivanje znakova i nizova

Ista naredba može ispisati i znakove i nizove znakova, na primer:

WRITELN ('A', 'b', 'Abe')

ispsisuje

AbAbe

Moguće je i ispisivanje znakova s razmakom, na primer:

WRITELN ('A', 'b' :5, 'Abe' :5)

ispisuje:

A

				b			A	b	e

Pre nego što pređemo na strukturu Pascal programa i pisanje programa upoznaćemo se sa integrisanim razvojnim okruženjem Turbo Pascala.

Integrisano razvojno okruženje

Turbo pascal obezbeđuje dva nezavisna metoda pravljenja Pascal programa. Možete da koristite svoj editor (program za pisanje i prepravljanje ASCII teksta) i samostalni Turbo Pascal kompajler (prevodilac), ili možete da koristite integrisano razvojno okruženje (engl. Integrated Development Environment – IDE) koje sadrži moćan editor, Pascal kompajler, linker (povezivač) i debager (program za otkrivanje i ispravljanje grešaka).

Sa stanovišta korisnika, IDE izgleda kao tekst editor koji sadrži komandu Compile i pridružene funkcije Debug. Ovo znači da ne morate da napuštate ovo okruženje, jednostavno tu pišete svoj kôd, prevodite ga, ispravljate greške i ponovo prevodite sve dok ne dobijete zadovoljavajući rezultat. Za vreme izvršavanja programa, IDE obezbeđuje debager na nivou izvornog koda. Koristeći IDE debager, možete da pomerate editorski pokazivač na određeni red izvornog koda i postavite prekidnu tačku za privremeno zaustavljanje izvršavanja programa, a zatim da nastavite jednokoračni hod kroz izvorni kôd, proveravajući vrednosti promenljivih u prozoru Watch. Šta više, možete i da menjate vrednosti promenljivih za vreme izvršavanja programa.

Kada da koristite IDE?

Uopšteno govoreći, želećete da koristite IDE kadgod je to moguće. Međutim, u zavisnosti od veličine vašeg programa i memorijskog kapaciteta biće premašene granice koje IDE zahteva. U tom slučaju, možete svoj program podeliti u manje celine i raditi u IDE-u ili koristiti samostalni kompajler.

Pokretanje Turbo Pascal razvojnog okruženja

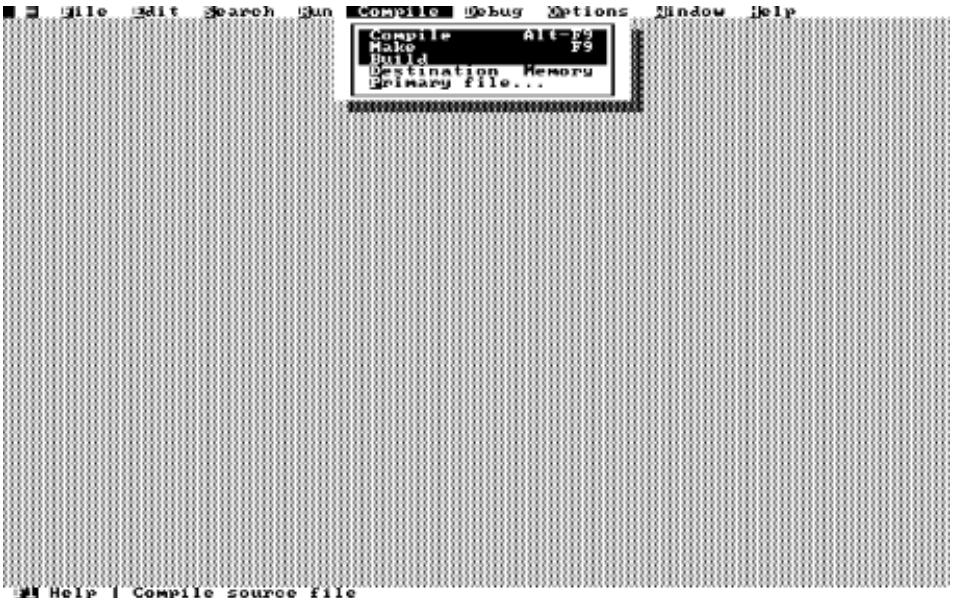
Pod pretpostavkom da ste instalirali Turbo Pascal u poddirektorijum \TP, kao što je u ovom primeru, da biste pristupili razvojnom okruženju na komandnoj liniji ispisaćete sledeće:

C:\TP> TURBO

Ukoliko ste instalirali Turbo Pascal u neki drugi direktorijum tada treba da stavite tu lokaciju u DOS Path naredbu ili da se prebacite u taj direktorijum da biste mogli da pristupite.

Biranje stavki menija

Stavke menija u razvojnom okruženju možete birati na više načina. Najlakši način je pomoću miša; postavite pokazivač miša na željenu stavku (komandu) i pritisnete levo dugme miša. Na ekranu se prikazuje padajući meni i vi postavljate pokazivač na stavku koja vam je potrebna i dvo-klikom miša je izabirate. Na sledećoj slici prikazan je padajući meni Compile razvojnog okruženja Turbo Pascal.



Slika 2.1

Takođe, korišćenjem tastature možete pristupiti stavkama menija tako što ćete istovremeno pritisnuti taster Alt i istaknuto slovo odgovarajuće stavke menija. Zatim, korišćenjem pokazivačkih tastera (tastera sa strelicama) možete se kretati kroz menije. Kada dođete do željene stavke, pritiskom na taster Enter aktivirate označenu stavku.

Pored toga, neke stavke menija možete aktivirati pritiskom tasterske kombinacije koja je označena pored tih stavki u meniju (ispisana sa desne strane tih stavki menija).

Na kraju, ukoliko želite, možete pristupiti liniji menija pritiskom funkcijskog tastera F10. Time označavate meni File u gornjem levom uglu ekrana. Upotrebite tastere sa strelicama da biste došli do željenih stavki u menijima, a pritiskom na Enter ih aktivirate.

Uređivanje, snimanje i prevođenje programa

Da biste napravili novi program, izaberite u meniju File stavku New. Na ekranu se pojavljuje prazan prozor sa naslovom NONAME00.PAS centriranim na gornjoj ivici. Tekstualni kursor (pokazivač) će se pojaviti u gornjem levom uglu prozora.

Sada možete da počnete sa direktnim pisanjem svog prvog Pascal programa u IDE editoru. Krenimo od najprostijeg mogućeg primera:

```
Program Zdravo;
Begin
        Writeln(‘Zdravo programeri!’)
End;
```

Posle upisivanja ovog programa, možete ga snimiti na disk pomoću komande Save As iz menija File. Okvir za dijalog koji će se potom pojaviti, tražiće od vas da navedete naziv datoteke u koju će program biti snimljen. Upišite naziv datoteke i zatim ili pritisnite taster Tab dva puta da biste

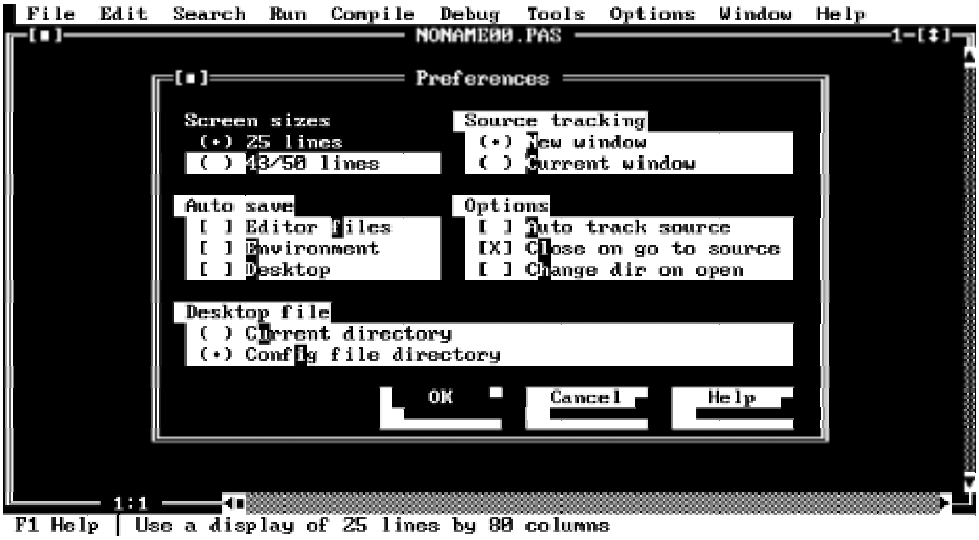
se prebacili da dugmeta OK i pritisnite taster Enter, ili upotrebite miša i dva puta kliknite na dugme OK. Bez obzira na primenjeni metod, rezultat je isti, novi program je snimljen na disk.

Okvir za dijalog

Okviri za dijalog se koriste u IDE i svim Turbo Pascal pomoćnim programima za obezbeđivanje ulaza i biranje opcija. Dijalozi sadrže nekoliko tipova ulaznih kontrola, uključujući:

- Ulaznu liniju ili polja za upisivanje tekstualnih odziva.
- Radio dugmad za odabir samo jedne stavke iz grupe stavki.
- Polja za potvrdu za omogućavanje ili onemogućavanja velikog broja opcija.
- Liste, za prikazivanje, biranje i kretanje kroz liste sa informacijama.
- Dugmad, kao što su OK ili Cancel.

Većina dijaloga su tzv. modalni dijalozi što znači da dok su oni prikazani vi ne možete da obavljate bilo koju drugu funkciju osim da kompletirate okvir za dijalog. Kada se dijaqlog prvi put prikaže, pokazivač je ili vidljiv u nekom ulaznom polju, ili je nešto istaknuto da označi da je na tome težište u tom okviru za dijalog.

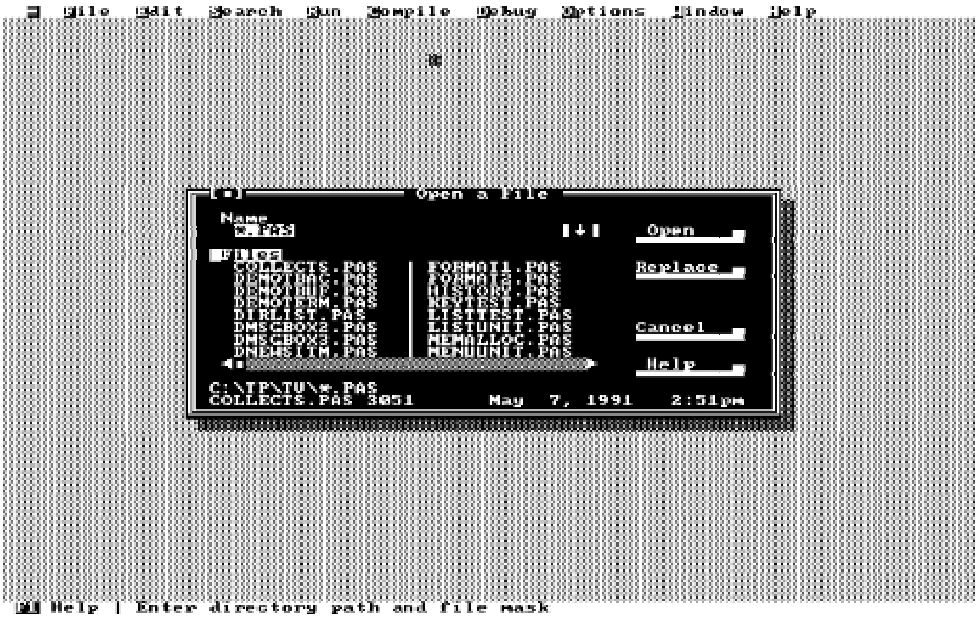


Slika 2.2

Na primer, kada je pokazivač u polju za potvrdu, možete videti da pokazivač trepće između zagrada '[ ]' koje omeđuju izbor polja za potvrdu.

Generalno, pritiskom na taster Tab krećete se između kontrola okvira za dijalog ili koristite miša da kliknete na određenu stavku. Međutim, grupe radio dugmadi i polja za potvrde se smatraju jednom kontrolom, tako da pritiskanjem taster Tab dok ste unutar grupe prebacujete se na sledeću kontrolu, a ne na sledeći izbor unutar grupe. Prema tome, koristite tastere sa strelicama da biste se kretali unutar grupe radio dugmadi ili polja za potvrdu, a taster Tab koristite da se prebacite sa jedne kontrole na drugu. U sledećoj tabeli dat je spisak kontrola okvira za dijalog i objašnjenja za njihovo korišćenje.

Taster Tab	Pomera pokazivač na sledeću kontrolu u nizu.
Tasteri Shift-Tab	Pomera pokazivač na prethodnu kontrolu u nizu.
Tasteri sa strelicama	Pomeraju pokazivač između stavki polja za potvrdu i radio dugmadi.
Taster razmaknica (Space)	Bira ili poništava radio dugme ili polje za potvrdu.
Taster Enter	Bira označenu kontrolu.
Klik mišem	Omogućava ili onemogućava stavku radio dugme ili polje za potvrdu, bira kontrolu tipa dugme, premešta akciju na stavku na koju je kliknuto.
Istaknuta tasterska kombinacija	Stavke koje sadrže istaknutu tastersku kombinaciju mogu se izabrati direktnim pritiskom na te tastere, usled čega se akcija prebacuje na izabranu stavku.
Radio dugmad	Ova kontrola vam omogućava da izaberete samo jednu stavku od više ponuđenih. Kada birate radio dugme pritiskom tastera razmaknice ili mišem, ili pritiskom na istaknutu slovnu oznaku, sva ostala radio dugmad unutar te grupe postaju neaktivna.
Polja za potvrdu	Ove kontrole se koriste za omogućavanje ili onemogućavanje bilo kog broja stavki. Obično su grupisane i tasterskim strelicama možete da se krećete između ponuđenih stavki. Pritisnite taster razmaknicu da izaberete stavku, ili kada je stavka već izabrana, pritisnite razmaknicu da poništite taj izbor.
Ulazno polje	Ulazna polja vam omogućuju da upišete proizvoljnu tekstualnu informaciju i da koristite tasterske strelice za pomeranje ulevo i udesno unutar teksta, povratni taster i taster Delete da obrišete tekst, taster Home da se prebacite na početak polja a taster End da se prebacite na kraj polja. Većina ovih polja obezbeđuje horizontalno pomeranje sadržaja (skrolovanje), što vam omogućuje da upišete tekst koji je širi od prikazanog ulaznog polja. Kada se to desi, na oba kraja polja pojaviće se odgovarajuće strelice da pokažu da ima više teksta nego što je prikazano. Koristite odgovarajuće tastere za kretanje kroz tekst ulevo, odnosno udesno, ili pritiskajte mišem na odgovarajuću strelicu.
Okvir liste	Oni se koriste za izbor stavke sa velikog spiska. Na sledećoj slici prikazan je okvir za dijalog Open u IDE okruženju. Tipična upotreba ovog okvira za dijalog je za otvaranje datoteke koja se nalazi u direktorijumu čiji sadržaj se prikazuje u listi. Ako postoji više stavki nego što može da stane u polje liste, prikazuje se ili na dnu ili na desnoj strani okvira traka za pomeranje sadržaja. Da biste koristili okvir liste, upotrebite taster Tab da biste prebacili pokazivač u okvir liste, ili koristite miša da kliknete na stavku unutar liste. Kada se već nalazite u okviru liste, možete koristiti standardne navigacione tastere.



Slika 2.3

Koristite taster Ins da se prebacujete iz režima umetanja u režim prepisivanja znakova.

Korišćenje editora

Editor razvojnog okruženja je mesto gde ćete provesti veći deo svog vremena pišući kôd, prepravljajući postojeće programe i ispravljajući greške. Editoru pristupate izborom opcije New u meniju File ili otvaranjem postojeće datoteke, takođe iz menija File. Alternativno, možete otvoriti postojeću datoteku navođenjem njenog naziva na komandnoj liniji kada prvi put pokrenete IDE kucajući sledeću komandu:

C:\TP>TURBO SHELL.PAS

koja pokreće Turbo i učitava postojeći program SHELL.PAS za uređivanje.

Kada ste u editoru, na raspolaganju vam je niz komandi sa tastature za uređivanje teksta. Kako upisujete tekst, pokazivač se pomera duž ekrana, na desno. Kada dođete do kraja reda, pritisnite taster Enter da biste prešli u novi red, mada će editor pomerati prikaz horizontalno ukoliko treba da upišete izvorni kôd čija širina je veća od širine ekrana.

Pritiskom na taster Ins, editor se prebacuje iz režima umetanja u režim prepisivanja preko postojećeg sadržaja i obrnuto.

Navigacija u editoru

Horizontalna traka za pomeranje sadžaja prikazuje tekući položaj datoteke, u odnosu na početak i kraj svojih redova, a vertikalna traka za pomeranje sadržaja prikazuje tekući položaj datoteke u

odnosu na prvi i poslednji red u datoteci. Svaka traka prikazuje i kvadratni marker koji možete mišem da prevlačite da bi se brzo premestili na novu lokaciju u datoteci. Da biste se pomerali red po red, pritisnite taster sa strelicom na gore (↑) ili strelicom na dole (↓), ili kliknite mišem na prazan prostor na traci za pomeranje sadržaja.

Kombinacije tastera za navigaciju u editoru

↑ ili Ctrl+E	Pomera naviše za jedan red
↓ ili Ctrl+X	Pomera naniže za jedan red
← ili Ctrl+S	Pomera ulevo za jedan znak
→ ili Ctrl+D	Pomera udesno za jedan znak
Ctrl+← ili Ctrl+A	Pomera za jednu reč ulevo
Ctrl+→ ili Ctrl+F	Pomera za jednu reč udesno
PgUp ili Ctrl+R	Pomera za jedan pun ekran naviše
PgDn ili Ctrl+C	Pomera za jedan pun ekran naniže
Home	Pomera na početak tekućeg reda
End	Pomera na kraj tekučeg reda
Ctrl+Home	Pomera na početak tekućeg prozora
Ctrl+End	Pomera na dno tekućeg prozora
Ctrl+Q+B	Pomera na početak izabranog tekstualnog bloka
Ctrl+Q+K	Pomera na kraj izabranog tekstualnog bloka
Tab ili Ctrl+I	Pomera na sledeću tabulatorsku poziciju u redu

Označavanje teksta i rad sa blokovima teksta

Označavanje (selektovanje) teksta možete uraditi mišem tako što postavite pokazivač na početak željenog tekstualnog bloka, držite pritisnuto levo dugme miša i povlačite pokazivač do kraja željenog bloka teksta. Tekst će biti vidno istaknut kako ga označavate (selektujete) povlačenjem pokazivača miša.

Tekstualni blok možete označiti i pomoću tastature. Držite pritisnut taster Shift i istovremeno pritiskajte navigacione tastere dok ne označite željeni blok teksta.

Označeni blok teksta možete zatim kopirati ili iseći. Turbo Pascal raspolaže klipbordom, tj. privremenim magacinom u koji možete smestiti označeni tekst koji ste iskopirali ili isekli. Kada je neki blok isečen (Cut) iz tekuće datoteke, on je iz nje uklonjen i smešten u klipbord. Ako je neki blok kopiran (Copy), on ostaje u tekućoj datoteci, a kopija se smešta u klipbord. Isecanje koristite da obrišete blokove ili da ih privremeno prebacite u klipbord da bi mogli kasnije da ih stavite na drugo mesto u datoteci. Kopiranje koristite da duplirate tekst.

Komande za isecanje (Cut) i kopiranje (Copy) nalaze se u padajućem meniju Edit. Takođe, mogu se aktivirati preko tastature. Tasterska kombinacija za Cut je Shift+Del, a za Copy je Ctrl+Ins.

Sadržaj klipborda može se staviti na novu lokaciju ili čak drugu datoteku tako što ćete u meniju Edit izabrati komandu Paste ili istovremeno pritisnuti tastere Shift i Ins (Shift+Inst). Koristite komandu Show Clipboard da pregledate tekući sadržaj klipborda.

Ukoliko želite da isečete blok teksta bez smeštanja u klipbord, izaberite komandu Clear u meniju Edit.

Meni File

Meni File se koristi za otvaranje postojećih programskih datoteka, snimanje datoteka na disk (disketu), pravljenje novih programskih datoteka i štampanje.

File/Open
Da biste otvorili postojeću programsku datoteku, možete ili navesti na komandnoj liniji kada pokrećete Turbo Pascal, ili koristiti komandu Open iz menija File. Komanda Open prikazuje na ekranu standardan okvir za dijalog, koji izlistava sve datoteke u tekućem direktorijumu koje odgovaraju po tipu, i obezbeđuje polje u koje možete upisati naziv datoteke.

Kao i kod većine okvira za dijalog, možete koristiti i miša da kliknete na određenu ekransku kontrolu ili da koristite taster Tab da se prebacite sa jedne ekranske kontrole na sledeću. Naravno, vi ćete verovatno upisati naziv datoteke koju želite da otvorite, ili ćete pritisnuti taster Tab da biste prešli na listu Files. Kada ste prešli na listu, možete koristiti tastere sa strelicama ili miš da označite željenu datoteku, a zatim pritisnuti dugmad OK ili Replace da otvorite tu datoteku. Da biste se brzo kretali kroz listu Files, koristite tastere PgUp ili PgDn. Takođe, možete mišem pomerati klizač na traci za pomeranje sadržaja da biste pregledali kompletnu listu.

Razlika između OK ili Replace je u tome što kada izaberete OK, IDE će otvoriti novi prozor u editoru pre nego što otvori datoteku. Ovo vam omogućuje da više datoteka bude vidljivo na ekranu u isto vreme. Ukoliko izaberete Replace, datoteka će biti učitana u postojeći prozor editora, ili ako ima više editorskih prozora, datoteka će biti učitana u trenutno aktivni prozor.

Kada se okvir za dijalog pojavi prvi put, pokazivač se inicijalno pojavljuje u ulaznom polju Name sa istaknutom oznakom *.PAS. Možete obrisati ovu oznaku ako pritisnete bilo koji taster i upisati određen naziv datoteke. Takođe, možete se prebaciti u neki drugi poddirektorijum upisivanjem putanje tog poddirektorijuma u polje Name, međutim, ovime se samo privremeno menja podrazumevani direktorijum. Iskoristite komandu Change dir menija File da promenite podrazumevani direktorijum dok koristite razvojno okruženje.

Ako ste prethodno otvarali druge datoteke, tada strelica usmerena na dole, koja se nalazi desno od polja Name, prikazuje listu prethodno otvaranih datoteka. Ovo olakšava ponovno otvaranje datoteke koju ste ranije već koristili.

Izbaerite opciju New da otvorite novi, prazan editorski prozor da biste u njega počeli da upisujete novu programsku datoteku. Novi editorski prozor će imati podrazumevani naziv NONAMExx.PAS, gde je xx brojčana oznaka u rasponu od 0 do 99.

File/Save
Da biste snimili sadržaj editorskog prozora na disk, izaberite jednu od komandi za snimanje: Save, Save As ili Save All.
Ako ste upisivali u novu datoteku ili želite da snimite sadržaj editorskog prozora u novu datoteku, izaberite stavku Save As iz menija File. U komandnom odzivniku upišite naziv nove datoteke gde će sadržaj editorskog prozora biti snimljen.
Izaberite Save da brzo snimite sadržaj aktivnog editorskog prozora na disk, koristeći naziv koji je već dodeljen datoteci. Ako editorski prozor ima naziv NONAMExx.PAS, IDE će automatski prikazati komandni odzivnik Save koji od vas traži da navedete jedinstven naziv za datoteku.
Možete pritisnuti i taster F2 da uradite brzo snimanje dok radite u editorskom prozoru.
Opcija Save All snima svaki od editorskih prozora koji su bili promenjeni.

File/Print
Koristite ovu opciju da odštampate sadržaj aktivnog prozora na DOS PRN: printer uređaju (obično ekvivalent sa LPT1:). Ukoliko želite da odštampate samo deo teksta, prvo označite taj deo teksta i zatim pritisnite istovremeno tastere Ctrl+K+P.

File/Exit
Da biste izašli iz Turbo pascala izaberite opciju Exit u meniju File ili istovremeno pritisnite tastere Alt+X. Ukoliko niste snimili na disk neku od datoteka čiji sadržaj je u međuvremenu promenjen, Turbo Pascal će vas upozoriti i ponuditi da snimite promene na disk.

Meni Run

Meni Run sadrži opcije za pokretanje i upravljanje izvršavanjem vašeg programa. Opcije menija Run možete koristiti posle prevođenja (kompajliranja; meni Compile) programa ili izvršiti program direktnim zadavanjem komande Run. Ako program nije bio kompajliran posle zadnje izmene u editoru, tada će istovremeno biti i kompajliran i izvršiti se.

Run/Run (Alt+R)
Ako izvorni program treba da se kompajlira, komanda Run će pozvati kompajler, zatim će vaš program početi da se izvršava. Možete istovremeno pritisnuti tastere Ctrl+Break jednom ili dva puta, po potrebi, da biste zaustavili izvršavanje programa. IDE će tada postaviti pokazivač na lokaciju gde je program zaustavljen pri izvršavanju.

Ako je program privremeno zaustavljen za vreme korišćenja ugrađenog debagera, izaberite komandu Run da biste ponovo pokrenuli izvršavanje programa.

Na izvršavanje programa utiču i komande Run/Parameters i stavke menija Options (Compiler, Memory sizes, Linker i Debugger).

Run/Program Reset (Ctrl+F2)
Dok koristite debagerska svojstva razvojnog okruženja, možete u potpunosti zaustaviti i ponovo pokrenuti (resetovati) izvršavanje programa izborom ove komande menija Run. Kada je jednom izabrana ova komanda, obnavljanje izvršavanja programa od početka ostvaruje se izborom komande Run.

Run/Go to Cursor (F4)
U bilo kom trenutku kada želite da koristite komandu Run da bi pokrenuli izvršavanje svog programa, možete izabrati komandu Go to Cursor koja izvršava program do lokacije gde se nalazi pokazivač u IDE okruženju. Kada izvršavanje dostigne tekuću lokaciju pokazivača, program će se zaustaviti i vi možete da koristite debagerska svojstva da proverite vrednosti promenljivih ili da nastavite sa ispravljanjem programa. Takođe, mogli ste da postavite prekidnu tačku (engl. breakpoint; meni Debug/Breakpoints) i pokrenete izvršavanje programa dok se ne zaustavi u prekidnoj tački. Razlika između ova dva slučaja je u tome što se sa Go to Cursor postavlja privremena tačka prekida, dok se sa Breakpoints postavlja prekidna tačka trajnijeg karaktera, koja je važeća sve dok se ne obriše.

Run/Trace Into (F7) i Run/Step Over (F8)
Da biste izvršavali izvorni program korak po korak, tj. red po red, izaberite ili komandu Trace Into ili Step Over. To su slične komande, samo što će Trace Into nastaviti da izvršava program red po red kako se pojavljuju procedure niskog nivoa kojima debager mora da pristupa, a Step Over će pozivati procedure ali ih neće pratiti i neće se zaustaviti sve do reda koji se nalazi neposredno ispod reda iz koga je procedura pozvana.

Run/Parameters
Ako vaš program koristi parametre sa komandne linije, tj. koristi argumente, možete upotrebiti opciju Parameters da postavite parametre za izvršavanje programa. Kada se program pokrene, ovi parametri su na raspolaganju kao da su upisani na DOS komandnoj liniji.

Meni Compile

Meni Compile se koristi kompajliranje i povezivanje („linkovanje”) izvršnog programa. Takođe, na proces kompajliranja mogu da utiču i stavke menija Run i Options.

Compile/Compile (Alt+F9)
Koristite opciju Compile da kompajlirate sadržaj tekućeg editorskog prozora. Koristite Build ili Make ili Run (u meniju Run) kada je potrebno da ponovo kompajlirate druge izvorne datoteke koje će se koristiti pri završnom povezivanju.

Compile/Make (F9)
Koristite opciju Make da kompajlirate i povežete kompletnu izvršnu (.EXE) datoteku. Make automatski proverava i kompajlira, ako je potrebno, bilo koju nezavisnu programsku jedinicu (*.TPU), spoljne objektne datoteke (*.OBJ) i uključene datoteke unutar jedinica. Kadgod Make funkcija pronađe izvornu datoteku koja je starija od datoteke koja se u tom trenutku kompajlira (.OBJ ili .TPU datoteka), tada se te datoteke automatski ponovo kompajliraju tako da njihov kompajlirani oblik odgovara poslednjim promenama. Make je „inteligentna” funkcija u tom smislu što kompajlira samo one datoteke koje treba da se kompajliraju; nepromenjene jedinice se ne kompajliraju ponovo.

Compile/Build
Komanda Build je kao Make, izuzev što Build ponovo kompajlira svaku zavisnu jedinicu, bez obzira da li je ili nije potrebno ponovno kompajliranje.

Compile/Destination
Ovu funkciju koristite da saopštite razvojnom okruženju da kompajlira i poveže program direktno u memoriji, što je najbrže, ili da napravi .EXE datoteku na disku. Bez obzira na ovaj izbor, sve jedinice koje prave .TPU datoteke uvek se kompajliraju na disk. Bez obzira koju ste od ove dve mogućnosti izabrali, možete da izvršite svoj program koristeći komandu Run.

Compile/Primary File
Kada pišete program koji koristi više jedinica i uključene datoteke, konkretno kada je njih nekoliko istovremeno učitano u više editorskih prozora, potrebno je da saopštite kompajleru koja od tih datoteka je primarna ili glavna datoteka. Koristite komandu Primary File da označite koja datoteka je glavna datoteka i koja poziva sve druge. Komande Make i Build koriste ovu datoteku da odrede koja datoteka prisvaja odgovarajuće jedinice i da obave „inteligentno" ponovno kompajliranje koda.

Meni Debug i ugrađeni debager

Meni Debug obezbeđuje pristup ugrađenom debageru razvojnog okruženja. Zajedno sa opcijama menija Run, ugrađeni debager omogućava postepen prolaz kroz izvorni kôd programa, red po red, praćenje vrednosti promenljivih, kao i njihovo menjanja, postavljanje prekidnih tačaka da bi se zaustavilo izvršavanje na određenim lokacijama.

Dva funkcije koje se najčešće koriste su opcija Breakpoints, koja se koristi za postavljanje tačaka za zaustavljanje programa i opcija Watches, koja se koristi za pregledanje vrednosti promenljivih. Takođe, postoji mogućnost da promenite vrednosti promenljivih korišćenjem komande Evaluate/Modify.

Debug/Evaluate/Modify (Ctrl+F4)
Ova funkcija se koristi da izračunate izraz, koji sadrži aktuelne vrednosti promenljivih i opciono da dodelite nove vrednosti programskim promenljivim za vreme izvršavanja programa.

Evaluate/Modufy prikazuje okvir za dijalog u koji možete da upišete izraz (Expression), a prikazuje i izračunati izraz u polu Result (čiji sadržaj takođe može da se prepravlja). Možete da upišete novu vrednost ili izraz u polje New value i izaberete dugme Modify da ažurirate sadržaje promenljivih (mora biti promenljiva ili element niza) koje su upisane u polje Expression.

Modul za izračunavanje izraza može da formatira rezultate u saglasnosti sa specifikacijama formata prikazanim u narednoj tabeli. Za nizove, možete da specificirate izraz za broj ponavljanja da nabrojite više elemenata niza i to tako što navedete niz ili naziv pokazivača iza koga sledi zapeta i zatim izraz za broj ponavljanja. Na primer:

Xniz[0], 3

Prikazuje prva tri elementa niza Xniz.

Specifikatori formata komande Evaluate/Modify

,	Koristite iza niza ili pokazivačke (pointerske) promenljive da specificirate broj ponavljanja (pogledajte prethodni primer).
C	ASCII znakovi u opsegu od 0 do 31 prikazuju se korišćenjem notacije # da se označi koja ASCII vrednost je smeštena u bajt. Dodavanjem ,C posle izraza, ove vrednosti će biti prikazane kao stvarni znaci.
D	Koristite ,D da prikažete sve celobrojne vrednosti kao decimalne brojeve.
Fn	Koristite ,Fn da promenite podrazumevani broj decimala u prikazu sa 11 cifara na broj zadat sa n, gde ne može biti u rasponu od 2 do 18.
M	Dodavanjem ,M posle izraza, vrednost izraza se koristi kao memorijska adresa (izraz mora izračunati adresnu promenljivu ili pokazivačku promenljivu pošto će se koristiti na levoj strani operatora dodeljivanja (:=)). Normalno, format ,M će prikazati heksadekadno jedan bajt memorije. Međutim, ako iza ,M sledi neki od specifikatora, možete prikazati naredne bajtove kao da sadrže celobrojnu, realnu, pokazivačku ili string vrednost.
P	Podrazumeva se da se pokazivačke vrednosti prikazuju kao Ptr(segment, offset). Navodeći ,P posle izraza, pokazivačka vrednost će biti prikazana kao segment:offset.
R	Zapis (slog) promenljivih prikazuje se tako što su vrednosti polja razdvojene zapetama. Dodavanjem ,R zapis se prikazuje tako da sadrži naziv promenljive.
S	Primorava vrednosti tipa string ili character koje su u opsegu od 0 do 31 da budu prikazane kao #nn. Kada se koristi sa formatom ,M , S utiče da se bajtovi memorije tretiraju kao string.
$, H ili X	Prikazuje sve celobrojne (integer) vrednosti kao heksadecimalne, na primer kao $1A.

Debug/Watches

Korišćenjem funkcije Watches, možete da dodate promenljive i izraze u prozor Watch, gde će njihovi sadržaji biti prikazani svo vreme. Kad izaberete opciju Watches na ekranu se prikazuje podmeni koji sadrži stavke Add Watch, Delete Watch, Edit Watch i Remove all watches. Ove funkcije su niže opisane.

Debug/Watches – Add Watch (Ctrl+F7)

Izaberite ovu opciju da biste upisali novu promenljivu ili izraz u prozor Watch (prozor za posmatranje vrednosti). Obično ćete upisati naziv promenljive, kao što je Filename ili pokazivač, List^.Name ili čak naziv zapisa. Takođe, možete upisati izraze, kao što je I*J ili slično.
Pored prečice sa tastature Ctrl+F7 za dodavanje vrednosti koje ćete posmatrati, ako je prozor Watch aktivan ili u žiži (u prvom planu), možete pritisnuti taster Ins da ubacite izraze za posmatranje, jedan ispod drugog.

Debug/Watches – Delete Watch

Kada je prozor Watch aktivan prozor, možete pojedinačno obrisati izraze koje posmatrate tako što postavite pokazivač na željeni izraz i izaberite u meniju stavku Delete watch, ili na tastaturi pritisnite taster Del ili Ctrl+Y.

Debug/Watches – Edit Watch
Da biste promenili postojeće izraze koje pratite, izaberite Edit Watch i upišite novi izraz u okviru za dijalog Edit Watch. Da biste ovo brže uradili, pomerite pokazivač na postojeći izraz koji pratite u prozoru Watch i pritisnite taster Enter. Na taj način aktivira se prikaz prozora Edit Watch.

Debug/Watches – Remove all Watches
Koristite opciju Remove all Watches da obrišete sadržaje prozora Watch.

Debug/Toggle Breakpoint (Ctrl+F8)
Da biste postavili prekidnu tačku, izaberite opciju Toggle Breakpoint. Ako je pokazivač na istaknutom redu koji već ima prekidnu tačku, izaberite opciju Toggle Breakpoint da obrišete prekidnu tačku.

Debug/Breakpoints
Komanda Breakpoints se koristi da prepravite ili dodate prekidne tačke u listu trenutno aktivnih prekidnih tačaka, da obrišete prekidne tačke ili da brzo pronađete i pogledate izvorni kôd gde je prekidna tačka postavljena. Po aktiviranju komande Breakpoints na ekranu se pojavljuje okvir za dijalog koji prikazuje listu aktivnih prekidnih tačaka (broj reda gde je postavljena i uslove koji su dodeljeni prekidnoj tački), i grupu dugmadi da biste mogli da baratate listom.

Dugme Edit Breakpoints
Izaberite dugme Edit Breakpoints da promenite postojeću prekidnu tačku ili da dodate novu. Edit Breakpoints prikazuje na vrhu okvira za dijalog Breakpoints novi okvir za dijalog sa sledećim poljima:

Condition field: Možete upisati uslov za testiranje u ovo polje, kao što je I>10, koji ograničava uticaj prekidne tačke na promenljivu I, tj. kada ima veću vrednost od 10.

Pass count: Upisivanjem vrednosti u polje Pass count, prekidna tačka će biti preskočena taj broj puta pre nego što bude aktivirana. Ovo svojstvo je posebno korisno kada otklanjate grešku u sredini petlje.

Filename: To je naziv izvorne datoteke koja sadrži tu prekidnu tačku.

Line number: Specificira broj reda u kojem je locirana prekidna tačka.

Dugme Delete
Unutar liste prekidnih tačaka, pomerite istaknutu horizontalnu pokazivačku traku na prekidnu tačku koju želite da obrišete, a zatim izaberite dugme Delete. Zapazite da okvir za dijalog Breakpoints nema dugme Cancel tako da prekidne tačke koje obrišete trajno su uklonjene.

Dugem View
Da biste locirali izvorni kôd konkretne prekidne tačke, pokažite na prekidnu tačku u listi prekidnih tačaka i izaberite dugme View. Ovime se vrši prebacivanje aktivnog prozora na datoteku koja sadrži prekidnu tačku i prikazuje red izvornog koda gde je prekidna tačka postavljena.

Dugme Clear All
Koristite ovo dugme da obrišete sve prekidne tačke programa.

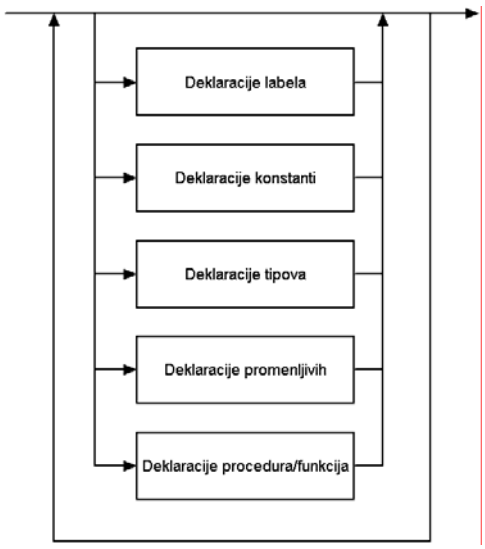
Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (3)

3. PASCAL PROGRAM

Osnovna struktura Pascal programa ima sledeći oblik:

PROGRAM Naziv_programa (Lista_datoteka)
USES
    (* Nazivi modularnih jedinica (UNIT-a) koje koristi *)
LABEL
    (* Deklaracija labela *)
CONST
    (* Deklaracije konstanti *)
TYPE
    (* Deklaracije tipa *)
VAR
    (* Deklaracije promenljivih *)
(* Definicije podprograma *)
BEGIN
    (* Izvršne naredbe *)
END.



Slika 3.1 Sintaksni dijagram koji pokazuje kako se deklaracioni elementi Pascal jezika mogu pojaviti u bilo kom redosledu.

Evo jednog kompletnog programa u Pascalu:

```
PROGRAM pozdrav (output);
{ovaj program šalje poruke korisniku}

BEGIN
        WRITELN  ('Dobar dan!');
        WRITELN ('Ja sam vaš saradnik RAČUNAR!');
        WRITELN  ('Mogu li vam pomoći?')
END.
```

Kad računar izvrši ovaj program biće ispisano:

```
Dobar dan!
Ja sam vaš prijatelj RAČUNAR!
Mogu li vam pomoći?
```

Napomena: O fontovima i našim slovima biće reči u jednom od narednih nastavaka.

Program se sastoji od tri dela:
1. naslova programa,
2. komentara i
3. naredbi.

1. Naslov programa

Naslov programa ima oblik:

```
PROGRAM  pozdrav (output);
```

Reč PROGRAM označava naslov programa i to je rezervisana reč u Pascal jeziku. Rezervisane reči su deo samog Pascala i korisnik ih ne sme upotrebljavati u programu kao imena promenljivih i procedura. Druga reč u naslovu je pozdrav i označava ime programa. To ime daje programer.

Imena se u programskim jezicima nazivaju **identifikatorima.**

Imena se daju programima, promenljivama i procedurama.

Imena se stvaraju na sledeći način:

1. Identifikator mora počinjati sa slovom abecede.
2. Iza prvog znaka ostali znakovi u identifikatoru mogu biti bilo koji alflanumerički znakovi. Ne smeju se upotrebljavati drugi znakovi, na primer, praznine ili znakovi interpunkcije.

3. Identifikator ne sme biti isti kao rezervisana reč.

   Rezervisane reči u Pascalu su:

and	**end**	**nil**	**shr**
asm	**file**	**not**	**string**
array	**for**	**object**	**then**
begin	**function**	**of**	**to**
case	**goto**	**or**	**type**
const	**if**	**packed**	**unit**
constructor	**implementation**	**procedure**	**until**
destructor	**in**	**program**	**uses**
div	**inline**	**record**	**var**
do	**interface**	**repeat**	**while**
downto	**label**	**set**	**with**
else	**mod**	**shl**	**xor**

   One pomažu računaru da razume šta želimo da program uradi.

4. Identiflkatori u Pascalu imaju proizvoljnu dužinu. Međutim, Pascal prevodilac razlikuje identifikatore do dužine od osam znakova. To znači da programer može koristiti po volji dugo ime, ali prevodilac ignoriše sve znakove iza osmog znaka.

Takođe, niže su navedene ključne reči koje su rezervisane od strane Turbo Pascala ali mogu da se redefinišu u programu, što nije preporučljivo da radite.

absolute	**external**	**forward**	**near**
assembler	**far**	**interrupt**	**private**
virtual			

Reč **output** u zagradama datog primera označava *datoteku* (file) u koju se prenosi izlaz iz programa. Potrebno je pobliže označiti pojam datoteke.

Pojam datoteke se odnosi na organizovani skup podataka u memoriji računara. Uopšteno, moguće je datoteku snimiti i na ostalim medijumima ulazno-izlaznih uređaja.

Datoteke koje program koristi potrebno je navesti unutar zagrada naredbe PROGRAM.

Svakom Pascal programu automatski se pridružuju dve datoteke: **input i output.** Datoteka **input** se pridružuje primarnom sistemskom ulaznom uređaju, a **output** primarnom sistemskom izlaznom uređaju. Ako program sadrži i ulazne i izlazne akcije tada naredba koja sadrži ime programa treba da sadrži i imena ulazne i izlazne sistemske datoteke **- input i output:**

```
PROGRAM  pozdrav (input, output);
```

Naslov programa završava tačka-zapetom.

2. Komentar

Red `{Ovaj program ispisuje poruke korisniku}` predstavlja komentar. Komentar se sastoji od teksta koga bira programer. Komentar je namenjen čoveku koji čita program, a ne računaru.

Tokom prevođenja programa u pseudo kôd, komentari se ignorišu. Komentari su isključivo namenjeni korisnicima programa u cilju boljeg razumevanja funkcija programa. Komentar se može staviti na bilo koje mesto u programu.

Treba napomenuti da je u Turbo Pascalu označavanje {komentar}alternativa označavanju (* komentar *). Preporučujem da koristite standardnu oznaku za komentar (* *), a da lastavičaste zagrade ostavite za otklanjanje grešaka. Valja znati da ugnježđivanje komentara nije dozvoljeno. Ukoliko napišete sledeće:

(* (* *) *)

biće prijavljena greška zato što kompajler uparuje prvi par znakova (* sa prvim parom znakova *), a ignoriše sve što se između njih nalazi. Stoga, drugi par znakova *) neće imati svog parnjaka (*.

Uglavnom komentar se piše na dva načina:

1. Komentar napisan u zasebnom redu (kao u gornjem primeru) objašnjava bliže sadržaj programa ili dela programa.
2. Komentar napisan u istom redu s nekom naredbom kao na primer: **WRITELN**

   {preskok reda}

Komentar pomaže korisniku koji nije programer da bolje razume šta program radi, a i samom programeru može biti od koristi kad se koristi

programom koji je napisao pre par meseci. Dodatnoj razumljivosti programa doprinosi pogodan izbor imena identifikatora. Na primer, u navedenom primeru za ime programa se koristi „pozdrav", a ne, na primer yzx81, što korisniku programa ništa ne znači.

3. Naredbe

Naredbe sadrže uputstvo za računar, tj. šta treba da uradi. U Pascalu se dele na neizvršne i izvršne.

Neizvršne naredbe služe za deklaraciju svega onoga što će se u programu koristiti, prvenstveno su to promenljive.

Izvršne naredbe čine „radni" deo programa. To su aritmetičko-logičke, upravljačke i ulazno-izlazne naredbe.

3.1. Aritmetika u Pascalu

U Pascalu, svaka kombinacija simbola koja može prikazati neku vrednost naziva se **izraz**. Tako su, na primer, sledeće vrste konstanti primeri su jednostavnih izraza:

72 TRUE -3.5 a

Sledeći način predstavljanja vrednosti pokazuje kako se može računati s njima, na primer, 8 + 5 i 8 - 5 pokazuju vrednosti 13 i 3. To su, takođe, izrazi. U izrazu 8 + 5 znak + je operator, a 8 i 5 su operandi. U aritmetici govorimo o aritmetičkim izrazima; to su izrazi u kojima su vrednosti celobrojni ili realni brojevi.

Saglasno tome govorimo o aritmetičkim operatorima, koji operišu s realnim ili celobrojnim operandima, a rezultat je realna ili celobrojna vrednost.

Aritmetički operatori u Pascalu su:

+ sabiranje
- oduzimanje
* množenje
/, div, mod deljenje

Operatori +, -, * i / primenjuju se na obe vrste brojeva, (realne i cele). Za + , - i * ako su oba operanda celobrojna i rezultat će biti celobrojan. Međutim, ako je jedan ili oba operanda realni broj, rezultat će biti realni broj. Sliedeći primer to pokazuje za + :

Izraz	**Vrednost**
8 + 5	13
8 + 5.0	13.0
8.0 + 5	13.0
8.0 + 5.0	13.0

Takva šema ne zadovoljava deljenje. To je zato što, na primer, ako su oba operanda celobrojna rezultat deljenja može biti realan broj, na primer, 3 podeljeno sa 2, rezultat je 1,5. Zato operator deljenja / daje za rezultat realni broj, bez obzira na tip operanada:

Izraz	**Vrednost**
3/2	1.5
3/2.0	1.5
3.0/2	1.5
3.0/2.0	1.5

Međutim, moguće je i deljenje celobrojnih vrednosti čiji rezultat je celi broj. To je metoda deljenja sa ostatkom - tako smo svi računali u osnovnoj školi dok nismo naučili decimalne brojeve. Na primer, 10 :3 = 3 ostatak 1.

U Pascalu postoje dva operatora za deljenje metodom količnika sa ostatkom. Operator koji računa količnik zove se div, a operator koji računa ostatak zove se **mod.** Primer:

Izraz	**Vrednost**
10 div 3	**3**

10 mod 3 1

Naredbe WRITE i WRITELN su primer izvršnih naredbi i ne mogu se koristiti za ispisivanje vrednosti bilo kog izraza, a ne samo za vrednosti konstanti. Na primer:

WRITELN (2 + 7, 7-2, 7x2,7/2); WRITELN (7 div 2, 7 mod 2)

Rezultat obrade biće ispisan kao:

9 5 14 3.500000000000E + 00
3 1

Ovde treba istaći da je samo vrednost 7/2 ispisana kao realni broj. Možemo isti primer ispisati sa zadatim formatom ispisa, tj.:

WRITELN (2+7 :5, 7-2 :5, 7 x 2 :5, 7/2 :7 :1, 7 div 2 :5, 7 mod 2 :5)

				9				5			1	4				3	.	5				3				1	

4. DEKLARACIJE PROMENLJIVIH

Za svaku promenljivu upotrebljenu u programu potrebno je odrediti tokom izvođenja programa tip podatka (vrednosti) koji promenljiva može imati. Ima više razloga za to. Vrednosti različitih tipova podataka zahtevaju memorijske lokacije različitih veličina. Neki se operatori mogu primeniti na više od jednog tipa vrednosti.

Računar tokom kompajliranja i izvođenja programa proverava tipove podataka kako bi se dobio tačan rezultat.

Proverom računar može otkriti, na primer, besmislice kao što je deljenje slova!

Zbog toga program na početku i sadrži *deklaracije promenljivih.* Rečeno je ranije da se u Pascal programima koriste sledeći tipovi podataka: *celobrojni, realni, logički* i *znakovni* s time da se u deklaracionom delu programa označavaju s rezervisanim rečima:

integer real Boolean char

Deklaracija promenljivih dolazi u programu između naslova programa i naredbi. Na primer, evo deklaracije jedne realne i jedne celobrojne promenljive:

```
VAR
    x : real;
    i : integer;
```

Za deklaraciju promenljivih koristi se rezervisana riječ VAR. U navedenom primeru promenljiva *x* se deklariše kao realna a i kao celobrojna.

Promenljiva *x* može imati samo realnu vrednost, a promenljiva *i* samo celobrojnu. Često se kraće kaže, da je *x* realna promenljiva, a *i* celobrojna promenljiva. Više promenljivih istog tipa može se deklarisati zajednički jednom rezervisanom reči, na primer:

```
VAR
```

```
x, y, z : real;
i, j, k : integer;
```

deklarisane su tri promenljive realnog tipa i tri promenljive celobrojnog tipa.

Primer deklarisanih promenljivih u programu:

```
PROGRAM primer (output);
   {primer deklaracije promenljivih}
VAR
  x : real;
  i  : integer;
BEGIN
.
.
.
.
END.
```

Tačkicama između BEGIN i END označena su mesta gde dolaze naredbe.

U Pascalu se može svaka deklarisana promenljiva pobliže opisati komentarom u istom redu da bi bila jasnija. Na prmer:

```
VAR
radnik : integer;   {broj zaposlenih}
sati,               (radni sati}
rata     : real;
```

Ako se izostave komentari može se kraće pisati:

```
VAR
radnik    : integer; sati, rata   : real;
```

Sledeći primer deklariše sve spomenute tipove promenljivih:

```
VAR
    x, y, z : real;
    i, j, k    : integer;
    p, q      : Boolean;
    c         : char;
```

Tako se uvek eksplicitno podrazumeva da su x, *y* i *z* realne promenljive, zatim i, 7, *k* celobrojne, itd.

4.1. Pridruživanje

Pridruživanje (dodeljivanje) je operacija kojom se smešta vrednost u memorijsku lokaciju. Od trenutka kada je vrednost smeštena u memorijsku lokaciju, ona postaje vrednost odgovarajuće promenljive. Kažemo da je pridružena nova vrednost promenljivoj.

Operator pridruživanja u Pascalu je :=, a naredba pridruživanja ima sledeći oblik:

promenljiva := izraz

Kada računar izvrši naredbu pridruživanja izraz na desnoj strani postaje vrednost promenljive na levoj strani.
Operator := se može čitati kao „postaje".

Konstanta je jednostavna vrsta izraza. Na primer:

```
i :=100;
x:=5.4;
p:=true;
c:=A
```

Kada računar izvrši gornje operatore pridruživanja dobije se:

promenljiva	vrednost
i	100
x	5.4
p	true
c	A

Naknadno pridruživanje vrednosti menja promenljivu, na primer:

```
x:=4.2
p:=false
```

pa će rezultat biti:

promenljiva	vrednost
i	100
x	4.2
p	false
c	A

Samo one promenljive kojima se vrednost promenila, tj. kojima je pridružena nova vrednost, promenjene su, a ostale imaju staru vrednost. Naravno, vrednost pridružena promenljivoj mora biti istog tipa kao što je specificirano za promenljivu u deklaraciji promenljive.

Ima jedan izuzetak od tog pravila. Za svaku celobrojnu vrednost imamo odgovarajuću realnu vrednost. Na primer, za celobrojnu veličinu 25 odgovarajuća je realna 25.0; za 135 je 135.0 itd.

Pascal dozvoljava označavanje celobrojnih vrednosti realnim promenljivama. Kada je pridruživanje izvršeno, celobrojna vrijednost se pretvara u odgovarajuću realnu.

Prema tome:

x:=35;

y:=135;

Z.= 0

postaje nakon pridruživanja

promenljiva	**vrednost**
x	**35.0**
y	**135.0**
z	**0.0**

Obrnuto nije dozvoljeno; *ne možemo realnoj vrednosti pridružiti celobrojnu vrednost.*

Izrazi koji sadrže operatore mogu, naravno, biti upotrebljeni u naredbi pridruživanja. Svaki izraz je određen i njegova vrednost je pridružena promenljivoj na levoj strani znaka := . To pokazuju sledeći primeri:

i:=7+4

j :=*7-4;*

k :=* *7* **4;

x := 30/8

što nakon izvođenja daje:

promenljiva	**vrednost**
i	**11**
j	**3**
k	**28**
x	**3.75**

5. IZRAZI

Izraz je bilo koja kombinacija simbola koja predstavlja neku vrednost. Promenljiva prikazuje vrednost, tj. vrednost promenljive koja joj je trenutno pridružena. Tako promenljive kao što su konstante, predstavljaju izraze i takođe mogu biti upotrebljene kao delovi složenijih izraza. Sledeći primeri predstavljaju izraze:

i
j
i + 5
10 - j
i+j
i/j

Ako *i* ima vrednost 9 a *j 6* onda se vrednosti sledećih izraza mogu prikazati tablicom:

izraz	**vrednost**
i	9
j	6
i + 5	14
10-j	4
i + j	15
i/ j	1.5

Izrazi koji sadrže promenljive mogu biti korišćeni svuda gde su izrazi dopušteni. Na primer, oni se mogu koristiti u naredbama WRITE i WRITELN:

WRITELN (i + 5, 10-j, i + j, i/j)

što će dati ispis u obliku:

14 4 15 1.500000000000E + 00

Izrazi koji sadrže promenljive mogu se takođe koristiti u naredbi pridruživanja, na primer:

k:=i+j

x:=i/j

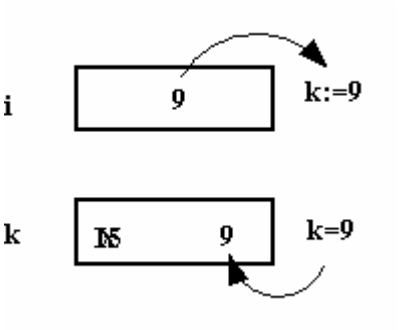
gde je onda uz i dalje iste vrednosti *i* i *j* (6 i 9)

promenljiva	**vrednosti**
k	**15**
x	**1.5**

Moguć je sledeći slučaj:

k:=i

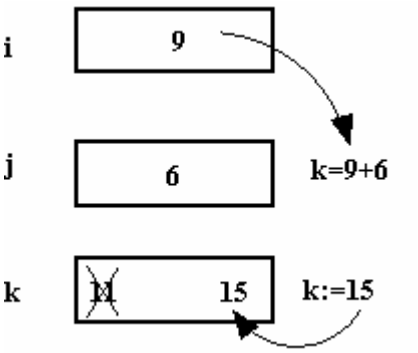
Ako je vrednost *i* 9 tada je *k:=9.* (Nova vrednost 9 zamenjuje staru 15). To se može prikazati slikovito:



gde je naredba pridruživanja u suštini operacija kopiranja. Vrednost u memoriji na lokaciji *i* kopira se na lokaciju *k.*

Sledeći primer:

k:=i+j



11 je stara vrednost promenljive *k.*

Ponekad će kod početnika naredba pridruživanja izazvati zbrku, kao na primer u slučaju

i:=i+j

gde se iste promenljive javljaju s obe strane izraza. Vrednosti *i* i *j* su dobijene iz memorije i zamenjuju promenljive:

i:=9+6

pa je

i:=15

Prema tome ,,stara vrednost" za *i* dobija sada novu vrednost 15, nakon što je izvršena naredba pridruživanja. Dve su važne primene u programiranju naredbi ove vrste: brojanje i izračunavanje zbira (suma). Na primer, ako želimo da brojimo učestalost nekog događaja, možemo upotrebiti celobrojnu promenjivu / u funkciji brojača. Počećemo sa ,,čišćenjem" brojača sa i :=0. Svaki put kad se broji pojava događaja izvršiće se

i:=i+1

što će svaki put povećati vrednost *i* za jedan.

Akumuliranje zbira (sume) je vrlo sličan postupak. Pretpostavimo da želimo uzeti *x* kao akumulator-registar, koji čuva zbir u mašini koji sabira. Na primer, neka su vrednosti koje želimo sabrati 3.5, 2.7, 1.9, 6.3. Počećemo sa čišćenjem akumulatora

X :=0.0

nakon čega će sledeće naredbe s komentarom pokazati tok sabiranja:

x :=0.0 {vrednost x je 0.0}
x := x + 3.5 {vrednost x je 3.5}

.

.

x:=x+6.3 (vrednost x je 14.4}

5.1. Izrazi s više operatora

Do sada su spominjani izrazi oblika:

6-3 i+j 7 * i k+12

od kojih svaki ima samo jedan operator. Upotrebljavajući ovako jednostavne izraze mogu se računati i složeniji izrazi, ali u više koraka s više naredbi, na primer, ako želimo sabratii vrednosti *i*, *j*, *k,* te rezultat obeležiti celobrojnom promenljivom sum, napisaćemo:

sum :=i + j
sum :=sum + k

no, logičnije je da se to napiše u obliku:

sum :=i + j + k

i tako dobije isti rezultat koristeći samo jednu naredbu umesto dve. Redosled promenljivih u izrazu ne utiče na rezultat.

Međutim, ako imamo izraz u obliku:

i + j * k

moramo biti puno pažljiviji! Pretpostavimo da su promenljivama *i*, *j*, *k* pridruženi podaci 2, 3, 4. Rezultat može biti različit. To pokazuje tablica:

Sabiranje prvo	**Množenje prvo**
1 + j * k	**i + j * k**
2 + 3 *4	**2 + 3 *4**
5 * 4	**2 + 12**
20	**14**

Znači potrebno je istaći pravila o prioritetu operatora.

Prioritet aritmetičkih operatora u Pascalu je:

***, /, div, mod**

\+ , -

Evo nekoliko primera za vežbu:

a) 7 * 7 - 4 * 3 = ?

b) 9 * 5 + 7 div 3 =?

c) 1.5*1.1+4.5/9.0-0.1 =?

d) 3.6/1.2*3.0 = ?

5.2. **Upotreba zagrada**

Ukoliko je u izrazu nekoliko operatora iste grupe prioriteta, a želimo da se ipak izvrši množenje pre deljenja, upotrebljavamo zagrade.

Primer:

3.6/(1.2 * 3.0)

ili drugi primer

3 * (8 + 56) * 2

daju prednost pri izračunavanju operatorima u zagradama.

Dozvoljena je upotreba više zagrada, kao na primer:

4 * (5 + 4 * (6 - 2)).

Sledeći program ilustruje upotrebu izraza sa više operatora. Podaci su dužina i širina oblasti, a izračunava se njezina površina i obim. Učitavanje podataka izvršiće se naredbom **read,** koja će kasnije biti detaljnije opisana.

```
PROGRAM polje;
{računanje površine i obima oblasti}
  VAR
        dužina, širina, površina, obim:real;


  BEGIN
     read (dužina, širina);
     površina =dužina * širina;
     obim :=2.0 x (dužina + širina);
     writeln ('Dužina:', dužina: 7:2);
     writeln ('Širina:', širina: 7:2);
     writeln ('Površina:', površina: 7:
     writeln ('Obim:', obim: 7:2)
  END.
```

5.3. BOOLEOVI izrazi

U ovom poglavlju obratiću pažnju na metode kojima se može kontrolisati redosled izvođenja delova programa. U tu svrhu koriste se naredbe sa uslovima.

Pod *uslovima* mislimo na izraze koji mogu poprimiti vrednosti istina ili laž. Na primer, izraz:

vrednost *i* je ista kao vrednost *j*

je istina ako je na primer, $i = 5$ i $j = 5$, a laž ako je $i = 4$ a $j = 2$. Da bi se odredilo da li je naredba istinita ili lažna računar mora stvarno uporediti vrednosti *i* i *j*.

Stanje istine ili laži predstavlja se u Pascalu Booleovim izrazom i to izrazom koji daje vrednost istina ili laž. Izraz daje vrednost istina ako je odgovarajući uslov zadovoljen (istinit) a vrednost laž ako je odgovarajući uslov nezadovoljen (lažan).

5.4. Relacioni operatori

Vratimo se na gornji primer: **vrednost *i* je ista kao vrednost *j*.**

Simbolički to možemo napisati $i=j$. Želimo li to da razmatramo kao jedan izraz imaćemo vrednost *istina* ako je vrednost *i* jednaka vrednosti promenljive *j*, a vrednost *laž* u suprotnom. Znak jednakosti = naziva se relacioni operator (engl. relational operator).

Za ilustraciju operatora = evo nekoliko primera:

izraz	**vrednost**

3 = 5	laž
3 = 3	istina
5 = 9	laž
0 = 0	istina

Pascal ima 7 relacionih operatora, razmatraćemo 6, jer se sedmi operator odnosi na poseban tip podataka koji će kasnije biti objašnjen. Za svaki operator dat je Booleov izraz konstruisan s operatorom i odgovarajućim uslovom. Booleov izraz daje vrednost **istina** ako je odgovarajući uslov istinit i vrednost laž ako je odgovarajući uslov lažan:

izraz	**uslov**
i = j	**vrednost i je jednaka vrednosti j**
i < j	**vrednost i prethodi vrednosti j**
i > *j*	**vrednost i sledi iza vrednosti j**
j < = j	**vrednost i prethodi ili je jednaka vrednosti j**
/ > = j	**vrednost i sledi iza ili je jednaka vrednosti j**
i < > *j*	**vrednost i nije jednaka vrednosti j**

Relacioni operatori mogii biti primenjeni na sva 4 standardna tipa podataka a ti su: celobrojni, realni, Booleov i znakovni. Pre nego što bi mogli primeniti date definicije, moramo reći šta znači „prethodi" (engl. precedes) i „sledi" (engl. follows) za vrednosti svakog tipa.

Za celobrojne i realne veličine „prethodi" i „sledi" dodeljuje se uobičajeni numerički poredak. Mi kažemo da jedan broj prethodi drugom ako je prvi broj manji od drugoga, i da jedan broj sledi drugi ako je prvi broj veći od drugoga. Za celobrojne i realne brojeve možemo zameniti „manji od" umesto prethodi i „veći od" za sledi.

Sledeći primeri ilustruju upotrebu relacionih operatora sa celobrojnim i realnim veličinama:

izraz	**vrednost**
3 < 5	**istina**
7 < >7	**laž**
3.9 < 8	**istina**
3.5 < 6.9	**istina**
4.2 < = 8.3	**istina**
7.4 > 9.2	**laž**

Za Booleove vrednosti uzeto je da laž prethodi istini:

izraz vrednost

laž < istina istina
laž > istina laž

Za svaki Pascal postoji uređeni niz koji daje poredak za znakove (engl. collating sequence). U svakom slučaju, možemo očekivati da su slova poređana po abecedi, a brojevi po redu.

Poredak (redosled) znakova se može razlikovati od jednog do drugog tipa računara, na primer, za mašinu (PC) koja koristi ASCII-kôd znakovi su u sledećem poretku:

! " # $ % & ' ( ) * + , - • / 0123456789 : ; < = >?

@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]6-abcdefghijklmnopqrstuvwxyz[|] ~

Relacioni operatori imaju niži prioritet od bilo kog aritmetičkog operatora, pa će se u izrazima gde se javljaju i relacioni i aritmetički operatori, aritmetički primenjivati pre relacionih operatora.

Evo šeme prioriteta: x, /, div, mod ... **visoki prioritet**

+, -

= , <, >, <==, >=, < > .... niski prioritet

Sledeći primeri pokazuju redosled izvršavanja kada su uključeni i aritmetički i relacioni operatori:

Primer:

3 + 5 < 4 * 2 - 1
3 + 5 < 8 _ 1
8 < 7
laž

Primer:

10 div 3 = 2
3 = 3
istina

5.5. Booleovi operatori

Booleovi operatori NE, ILI, I (NOT, OR, 'AND) uzimaju Booleove vrednosti kao operande i

pridružuju Booleove vrednosti kao rezultat. Operator NE menja istinu u laž i laž u istinu. Tako je na primer, NOTp laž ako je vrednost *p* istina, i istina ako je vrednost *p* laž. Možemo definisati NE operator prema sledećoj tabeli:

Izraz	**vrednost**
NE istina	**laž**
NE laž	**istina**

Izraz *p* ILI *q* je istina kada je vrednost *p* istina ili je vrednost *q* istina ili oba.

Možemo definisati ILI operator sledećom tabelom:

Izraz	**vrednost**
istina ILI istina	**istina**
istina ILI laž	**istina**
laž ILI istina	**istina**
laž ILI laž	**laž**

Izraz *p* **I** *q* ima vrednost istina samo onda kada su obe vrednosti *(p* i *q)* istina. Tabela definicije I operatora:

Izraz	**vrednost**
istina I istina	**istina**
istina I laž	**laž**
laž I istina	**laž**
laž I laž	**laž**

Pre nego što upotrebimo Booleove operatore u složenijim izrazima, evo šeme prioriteta među njima:

- NOT ima najveći prioritet među operatorima.
- AND i OR su prioriteti slični množenju i sabiranju.

Sledeća tabela daje pregled prioriteta svih do sada spomenutih operatora:

NOT ..... **najviši prioritet**
/, *, div, mod, AND
+ , -, OR
=, <, >, <=», >=, <>, ... **niži prioritet**

Booleovi operatori se često upotrebljavaju u izrazima kao na primer:

(i=j) OR (i < k + 3)

što ćemo interpretirati kao: vrednost *i* je jednaka vrednosti 7, ili vrednost *i* je manja nego vrednost *k* + 3.

Zagrade u izrazima su obavezne. To je zato što Booleovi operatori imaju viši prioritet nego relacioni operatori. Ako nema zagrada primeniće se OR na vrednosti *i* i *j* što bi dalo pogrešan rezultat, budući da se Booleovi operatori u zadatku na njih ne primenjuju.

Zagrade obezbeđuju da se relacioni operatori primene pre Booleovih operatora.

Zagrade, takođe, doprinose preglednosti izraza.

Pogledajmo sada Booleov izraz koji uključuje sva pravila koja su nabrojana:

(11 <3*2 + 5) OR (6 * 2 = 4 * 3) AND NOT (2 + 2 = 4)

Redosled izvršavanja je: najpre rešiti ono što je u zagradama poštujući prioritet operatora.

(11 <6 + 5) OR (12 = 12) AND NOT (2 + 2 = 4)
(11 <11) OR (12 = 12) AND NOT (4 = 4)

Kad smo rešili delove u zagradama potrebno je primeniti relacione operatore:

false OR true AND NOT true

Booleov operator NOT je najvišeg prioriteta, znači:

false OR true AND false

Sledeći po prioritetu je AND pa zatim OR:

false OR false false

Konačna vrednost je **false!**

6. STANDARDNE FUNKCIJE

Često je potrebno da se izvrše operacije za koje nisu definisani operatori. Za taj slučaj koristimo funkcije, kod kojih je operacija prikazana identifikatorom umesto znakom.

Pascal ima brojne standardne funkcije. Takođe, možemo i sami definisati svoje vlastite funkcije.

Evo, na primer, jedne Pascalove funkcije: **sqr,** koja računa kvadrat broja.

Da bismo koristili funkciju moramo napisati funkcijsku oznaku, koja određuje ime funkcije, a iza toga treba napisati u zagradi vrednost nad kojom će se funkcija izvršiti. Na primer:

sqr (5)

znači kvadriranje će se primeniti na vrednost 5. Vrednost u zagradi naziva se stvarni parametar funkcije. Tako sqr (5) prikazuje vrednost 25 na isti način kao što 3 + 8 prikazuje 11 ili 3 x 7 prikazuje vrednost 21.

Funkcije mogu biti upotrebljene kao delovi većih izraza. Vrednost funkcije se izračunava pre nego što se primeni operator. Na primer:

Izraz	**vrednost**
sqr (2) + 3	**7**
2 * sqr (3)	**18**
2 * sqr (4) + 3	**35**

Parametar funkcije može biti i izraz. U tom slučaju prvo se izračunava parametar, a zatim na njega primenjuje funkcija.

izraz	**vrednost**
sqr (3 + 5)	**64**
sqr (2 x 3 - 2)	**16**
2 x sqr (4 x 3 - 7) + 1	**51**

Evo nekoliko najčešće upotrebljavanih funkcija:

FUNKCIJA	**DEFINICIJA**
abs (x)	Izračunava apsolutnu vrednost stvarnog parametra, na primer, abs (5) je 5, abs (- 5) je 5.
sqr (x)	Izračunava kvadrat stvarnog parametra, na primer, sgr (3) je 9.

sqrt (x)	Izračunava kvadratni koren stvarnog parametra, na primer, sqrt (9) je 3,0, sqrt (2.25) je 1,5.
trunc (x)	Pretvara realni broj u celobrojni zanemarujući decimalni deo, na primer, trunc (3.25) je 3.
round (x)	Pretvara realni broj u celobrojni zaokružujući realnu vrednost na celobrojnu, na primer, round (3.25) je 3, round (3.5) je 4.
sin (x)	Izračunava trigonometrijsku funkciju, a stvarni parametar mora biti u radijanima.
cos (x)	
arctan (x)	
Ln (x)	Izračunava prirodni logaritam stvarnog parametra, na primer, In (10) je 2,3.
exp (x)	Izračunava eksponencijalnu funkciju čiji je eksponent x; na primer exp (2) je 7,34

Stvarni parametri za abs i sqr mogu biti celobrojni i realni brojevi. Rezultat će biti istog tipa kao što je parametar.

Za sqrt parametar, takođe, može biti celobrojan ili realan, ali rezultat je uvek realan broj.

Za trunc i round, naravno, stvarni parametri su uvek realni brojevi, a rezultat je celobrojan.

Potrebno je nešto više reći o funkcijama **trunc i round.** Ranije je rečeno da je u Pascalu dozvoljeno pridruživanje kao:

x:=25

budući da svaki celi broj kao 25 ima odgovarajući realni broj; tako se 25 može pretvoriti u realni broj pa će vrednost *x* postati 25,0.

Pridruživanje realnog broja celobrojnoj promenljivoj je u Pascalu zabranjeno.

Postoje dva standardna načina pretvaranja realnog broja u celobrojni. Jedan način je odsecanje (engl. truncation) decimalnog dela broja, na primer, 6,72 postaje 6. Drugi način je zaokruživanje (engl. rounding) realnog broja na najbliži celi broj, na primer 6,5 će postati 7, a 6,72 će dati zaokruživanjem isto 7.

Funkcije **trunc i round** izvršavaju operaciju odsecanja i zaokruživanja. Tako naredbe:

```
i:= trunc (6.72);
j:=round (6.72)
```

nakon izvršenja daće vrednosti 6 za i, a vrednost *j* postaje 7.

7. ULAZNE NAREDBE

Dve se naredbe koriste za učitavanje vrednosti u Pascalu: READ i READLN. Učitavanje može biti preko ulazne datoteke ili preko tastature.

Na primer:

```
READ (i, j, k)
```

znači učitavanje promenljivih i, *j* i *k*. Pretpostavimo da su podaci upisani preko tastature:

```
25      6      -100
```

tada će nakon izvršenja naredbe READ promenljive imati sledeće vrednosti:

promenljiva	vrednost
i	25
j	6
k	-100

U naredbi READ podaci se učitavaju po redu kako su u naredbi navedeni. Na primer:

```
READ (i, j);
READ (k, x);
READ (y, z)
```

Ako su ulazni podaci:

```
25    30    400
3.5   7.8   2.0
```

promenljive će imati sledeće vrednosti:

promenljiva	vrednost
i	25
j	30
k	400
x	3.5
y	7.8
z	2.0

Naredba READLN učitava vrednosti kako su specificirane promenljivama, a nakon toga se prelazi na sledeći red, tj. zanemaruju se svi ostali podaci u tekućem redu.

Na primer, neka se izvrše naredbe:

```
READLN (i, j);
READLN (x, y)
```

ako su zadati podaci:

10 20 4.5 19
6.9 8.5 17 3.5

Vrijednosti promenljivih biće:

promenljiva	**vrednost**
i	10
j	20
x	6.9
y	8.5

Sledeći program objedinjuje sve do sada spomenute i objašnjene naredbe u jednu celinu.

Program izračunava ukupnu sumu novca koju će korisnik kredita otplatiti ako diže potrošački kredit u iznosu 2 000 din uz kamatu 4%.

Iznos kredita: 2000.00
Kamata: 4.0
Iznos kamate: 80.00
Ukupan iznos plaćanja: 2080.00 (total)

Program počinje s učitavanjem iznosa kredita i kamatom. **READ (iznos, kamata)**

Pre nego što se upotrebi kamata u računanju, procenat mora biti pretvoren u decimalni broj 0.04, što je naravno učinjeno deljenjem sa 100:

dcmlkam := kamata /100.0

Dalje računamo iznos kamate:

iznkam := iznos * kamata

Ukupan iznos koji se plaća je:

total := iznos + iznkam

Program treba završiti ispisom podataka i rezultata. Kompletan program izgleda ovako:

```
PROGRAM kredit (ulaz,izlaz);
{ovaj program racuna kamatu, i total u kreditu}
VAR
    iznos, {iznos kredita}
    kamata, {kamata u postotku)
    dcmlkam, {kamata decimalno)
    iznkam, {iznos kamate}
```

```
    total: real; {ukupan iznos placanja}
BEGIN
    read (iznos, kamata);
    dcmlkam := kamata / 100.0;
    iznkam :=iznos * dcmlkam;
    total := iznos + iznkam;
    vvriteln ('lznos kredita:', iznos : 8 : 2);
    vvriteln ('Kamata:', kamata:4:1);
    writeln ('lznos kamata:' iznkam: : 5 : 2);
    writeln (Puni iznos placanja:', total:8:2)
END.
```

Druga verzija istog programa može biti u interaktivnom obliku gde korisnik dobija na ekranu uputstva koji podatak treba da upiše.

```
PROGRAM kredit (input, output);
{interaktivna verzija}
VAR
        isto
BEGIN
    write ('lznos kredita?');
    readln (iznos);
    writeln ('kamata u procentima?');
    readln (kamata);
    dcmlkam := kamata/100.0;
    iznkam :=iznos * dcmlkam;
    total := iznos + iznkam;
    writeln ('lznos kamate je', iznkam:5:2);
    writeln ('Puni iznos placanja je', total:8:2)
END.
```

Rezultat dijaloga između programa i korisnika bio bi:

Iznos kredita ? 2000.00
Kamata u procentima? 4.0
Iznos kamate je 80.00
Puni iznos placanja je 2080.00

Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (4)

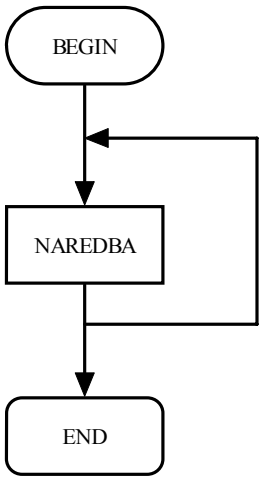
7. Naredbe za ponavljanje

U programima često postoji potreba da se deo programa više puta uzastopno izvede. U tu svrhu koriste se tzv. naredbe za ponavljanje. Pascal koristi tri vrste naredbi za ponavljanje:

- FOR,
- WHILE,
- REPEAT - UNTIL.

7.1. Naredba FOR

Naredba FOR omogućuje bezuslovno ponavljanje određenog dela programa toliko puta koliko programer želi. To se može prikazati dijagramom toka na sledeći način:



Evo prvog primera naredbe FOR:

```
FOR i := 1 TO 5 DO
    write ('+')
```

Kad je izvršena naredba FOR, naredba WRITE će se izvršiti pet puta i računar će ispisati:

```
+     +     +     +     +
```

Naredba **write** je zapravo deo FOR naredbe. U prvom izvršavanju vrednost *i* je 1, u drugom ponavljanju *i* je 2 itd.

Na primer, kad se izvrši:

```
FOR i=1 TO 5 DO
    WRITE (i:4)
```

računar će ispisati :

```
1    2    3    4    5
```

Ako upotrebimo DOWNTO umesto TO računar će brojati unazad:

```
FOR h=5 DOWNTO 1  DO
    WRITE (i:4)
```

rezultat će biti:

```
5    4    3    2    1
```

Vrednosti koje pokazuju početnu i graničnu vrednost u ponavljanju mogu biti i izrazi:

```
FOR i := + 3 + 2  TO  4 x 3 - 1  DO
    WRITE (i:4)
```

računar će ispisati:

```
5    6    7    8    9    10
```

Evo još jednog primera upotrebe FOR naredbe:

```
FOR i:=1 TO 5 DO
  BEGIN
        read (j);
        write (j:4)
  END
```

Ukoliko su podaci, na primer :

```
7    9    3    8    4    1    4    5    0
```

rezultat obrade će biti:

```
7    9    3    8    4
```

Naredba koja se ponavlja razmatra se kao deo FOR naredbe. To znači da tačka-zapeta odvaja naredbu FOR od susednih naredbi koje dolaze pre FOR i posle naredbe koja se ponavlja. Na primer, razmotrimo sledeće:

```
x:=3.5;
FOR i:=1 TO 5 DO write (i:4);
y:=7.9
```

Pišući naredbu, koja se ponavlja, u istom redu kao ostatak naredbe FOR, istakli smo, da je **write (i:4)** deo naredbe FOR. Međutim, obično se piše naredba koja se ponavlja u posebnom redu, ali je stavljanje interpunkcije isto:

```
x:=3.5;
FOR i:=1  TO 5  DO
    WRITE (i:4);
    y:=7.9
```

Isti princip važi kad je naredba za ponavljanje složena naredba. U tom slučaju naredba FOR se sastoji od svega što je između reči FOR do reči END koja završava složenu naredbu.

Tačka-zapeta odvaja naredbu FOR od sledeće naredbe koja dolazi posle reči END.

Primer:

```
x:=3.5;
FOR i == 1 TO 5 DO
  BEGIN
    read (j);
    write (j:4)
  END;
y:=7.9
```

Za ilustraciju upotrebe naredbe FOR u programu, možemo uzeti, na primer, program ,,polje" u kojem ćemo ispisati površinu i obim za pet oblasti. Uzećemo da su podaci dati u retku datoteke, sa dužinom i širinom jedne oblasti. Podaci mogu biti, na primer:

```
 8.6    7.5
 6.9   .5.2
 9.3    8.6
17.4   14.3
11.5    1.2
```

Površina prve oblasti je 8.6 puta 7.5, druge je 6.9 puta 5.2 itd. Evo i kompletnog programa:

```
PROGRAM polja (input, output);
    {racunanje povrsine i obima za pet oblasti}
VAR
    i: integer;
    duzina, sirina, povrsina, obim : real;
BEGIN
    writeln ('Duzina':10, 'Sirina':10, 'Povrsina':10, 'Obim':10);
    writeln;
    FOR i:=1 TO 5 DO
    BEGIN
        readln (duzina, sirina); povrsina:=duzina x sirina; obim :=2.0 x (duzina + sirina);
        writeln (duzina: 10:1, sirina:10:1, povrsina: 10:2, obim:10:2)
    END
END.
```

Ako se program izvrši sa ranije datim podacima ispis će biti:

Duzina	Sirina	Povrsina	Obim
8.6	7.5	64.50	32.20
6.9	5.2	35.88	24.20
9.3	8.6	79.98	35.80
17.4	14.3	248.82	63.40
11.5	1.2	13.80	25.40

U ovom primeru je zadato ponavljanje 5 puta. Međutim, broj ponavljanja može se učitati kao podatak pre FOR naredbe, pa će program biti univerzalniji.

Na primer, ako je broj ponavljanja *n*, naredba FOR će imati oblik:

```
FOR i:=1 TO n DO
```

gde će se *n* učitati naredbom read (n) pre FOR naredbe.

Prethodni program ,,polja" može se uzeti kao primer različitog broja ponavljanja. Na primer, prvi put želimo da obradimo 5 oblasti, a drugi put samo 3 oblasti. U tom slučaju podaci su

za 5 oblasti		a za 3	
5		3	
8.6	7.5	8.6	7.5
6.9	5.2	6.9	5.2
9.3	8.6	9.3	8.6
17.4	14.3		
11.5	1.2		

Želimo da učitamo prvu grupu i koristimo vrednost 5 da bi označili broj skupine podataka koji će biti u postupku. Za to možemo upotrebiti naredbu:

read (broj)

i tada koristimo broj naredbi FOR koja kontroliše ostatak koji se obrađuje:

```
PROGRAM polje1 (input, output);
    {povrsina i obim oblasti}
    VAR
    i,
    broj: integer; {broj oblasti}
    duzina, sirina, povrsina, obim: real;
BEGIN
    writeln ('Duzina': 10, 'Sirina':10, 'Povrsina': 10,
        'Opseg':10);
        writeln; readln (broj);
        FOR n=1 TO broj
        DO BEGIN
```

```
                readln (duzina, sirina);
                povrsina:=duzina x sirina;
                obim :=2.0 x (duzina + sirina);
                writeln (duzina: 10:1, sirina:10:1,povrsina: 10:2,
                obim:10:2)
                 END
END.
```

Ako bismo, na primer, želeli sabrati sve površine oblasti i sve obime, modifikovaćemo program. Obratimo najpre pažnju na površinu. Sačuvaćemo ukupnu površinu u memorijskoj lokaciji nazvanoj „total". Na početku programa očistićemo tu lokaciju:

```
total:=0.0
```

Svaki put će nova površina biti izračunata i dodata u lokaciju „total":

```
total := total + površina
```

Tako će se sabrati sve površine, pa će na kraju u „total" biti suma površina. Na isti način će se sabrati i obimi u lokaciji koju ćemo nazvati „totalo":

```
PROGRAM polje2 (input, output);
{izracunavanje povrsine i obima oblasti te ukupne povrsine
    i ukupnoga obima svih oblasti}
VAR
    i,
    broj :integer; (broj oblasti}
    dužina, širina, površina, obim,
    totala, (ukupna površina}
    totalo:real; {ukupan obim}

BEGIN
  writeln ('Duzina': 10,'Sirina': 10,  'Povrsina': 10,
  'Obim':10);
  writeln;
  total:=0.0;              {očistiti akumulatore}
  totalo:=0.0;
  readln (broj);      {dati broj ploha}
  FOR h=1 TO broj DO
    BEGIN
      readln (duzina, sirina); povrsina:= duzina x sirina; obim =2.0 x
    (duzina + sirina);
      writeln (duzina: 10:1, sirina:10:1,
      povrsina: 10:2, obim:10:2);
      totala := totala + površina;
      totalo := totalo + opseg
```

```
        END;
        writeln;
        writeln (totala: 30:2, totalo: 10:2)
    END.
```

Pretpostavimo da su podaci za ovaj program

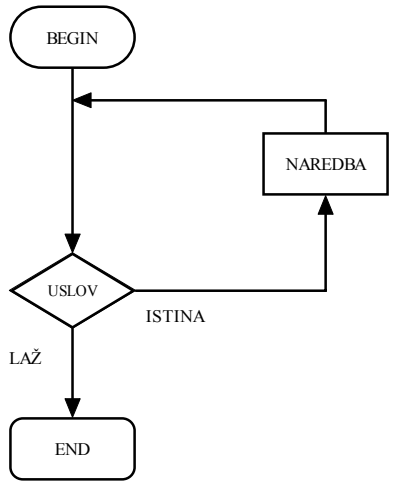
```
3
8.6     7.5
6.9     5.2
9.3     8.6
```

Rezultat obrade će biti:

DUZINA	SIRINA	POVRSINA	OBIM
8.6	7.5	64.50	32.20
6.9	5.2	35.88	24.20
9.3	8.6	79.98	35.80
		180.36	92.20

7.2. Naredba WHILE

Za korišćenje naredbe FOR moramo znati broj ponavljanja. On se mora naznačiti pre nego što se i jedno ponavljanje izvrši. Ali ponekad želimo ponavljati naredbe sve dotle dok je ispunjen neki uslov. U tom slučaju ne znamo unapred koliko će biti ponavljanja. Dijagram toka pokazuje opisani slučaj:



Ovde ćemo upotrebiti naredbu WHILE za ponavljanje. Njen opšti oblik je:

```
WHILE Booleov izraz DO
```

```
naredba
```

Booleov izraz procenjuje uslov, pa ako je vrednost Booleovog izraza ISTIN A naredba se izvršava. Taj proces se ponavlja tako dugo dok Booleov izraz ima vrednost ISTINA. Kad postane LAŽ ponavljanje prestaje i računar ide na sledeću naredbu u programu. Kao i kod FOR naredbe, naredba za ponavljanje može biti jednostavna ili složena naredba. Na primer, razmotrimo sledeće:

```
i:=0;
WHILE i < =20 DO
    WRITELN  (i);
    i:=i + 5;
    WRITELN  ('Ponavljanje završeno')
```

U ovom slučaju naredba *i: = i + 5* će se izvršavati ponavljanjem. Svakim izvršenjem raste vrjednost *i* za 5, pa će imati sledeće vrednosti: 0, 5, 10, 15, 20. Pre svakog ponavljanja vrednost *i* se proverava da li je manja ili jednaka 20. Kad je vrednost *i* 0, 5, 10, 15 ili 20 ovaj test prođe i ponavljanje se izvršava. Ali kada je vrednost *i* 25, vrednost *i* < = 20 je LAŽ, pa je ponavljanje završeno.

Možemo uzeti postupno vrednosti *i* koristeći složenu naredbu kao naredbu za ponavljanje:

```
i:=0;
WHILE i < =20 DO
    BEGIN
        write (i:4);
        i := i + 5
    END
```

Rezultat će biti:

```
0       5       10      15      20
```

Zadnja vrednost *i* = 25 nije ispisana. Zašto? Budući da naredba WHILE proverava vrednost Booleovim izrazom pre svakog izvršavanja naredbe za ponavljanje, ne izvršava se čim se prvi put dođe do Booleovog izraza LAŽ.

Na primer, u slučaju:

```
i:=21;
WHILE i < =20 DO
    BEGIN
        write (i:4);
```

```
        i:=i+5
    END
```

niti jedna vrednost neće biti ispisana, budući da je kod prvog ispitivanja uslova $i <= 20$ vrednost LAŽ.

Predikati

Funkcija koje daju Booleove vrednosti poznate su pod nazivom „PREDICATES". Pascal ima tri standardna predikata (jedan od njih se upotrebljava uz naredbu WHILE). Sledeća tabela daje tri predikata:

Predikat	definicija
odd	Stvarni parametar je celobrojan. Vrednost odd (i) je ISTIN A ako je vrednost i neparan broj, a u suprotnom LAŽ. Tako na primer odd (4) jednako je LAŽ, a odd (5) jednako je ISTINA.
oef	Stvarni parametar je datoteka kao što je na primer input. Vrednost eof (f) je istina ako nema više podataka koji su ostali za učitavanje sa datoteke f („oef" znači „end of file").
eoln	Stvarni parametar je datoteka. Vrednost eoln (f) je ISTINA ako se dostigne kraj reda („eoln" znači „end of line"). Kada računar ide na novi red vrednost eoln je LAŽ.

Pascal u slučaju navedenih predikata dozvoljava izostavljanje stvarnih parametara. Kad je parametar izostavljen Pascal pretpostavlja *input.*

Tako:

eof **je ekvivalentno** eof (input)
eoln **je ekvivalentno** eoln (input)

Velika je prednost predikata eof. Kada ga koristite ne morate da vodite računa o tome koliko je podataka u grupi koji će ući u obradu. Računar podatke uzima dokle god ih ima.

Upotrebom eof predikata može se izbeći brojanje grupe podataka. Jednostavno nastavlja se obrada dok je još podatak u procesu - to je dok je vrednost eof (input) LAŽ ili dok je NOT eof (input) ISTINA.

Evo jednostavnog programa za platni spisak. Za svakog zaposlenog program učitava

njegov ID broj, radne sate, iznos satnice. Program ispisuje ove informacije zajedno s iznosom koji treba zaposlenom isplatiti.

```
PROGRAM obracun (input, uotput);
    {racunanje plata radnika}
VAR
idbroj: integer;
sati, {radni sati}
plsata, {iznos placen za sat}
dohodak; real; {plata}
BEGIN
    writeln ('ID broj':10, 'sati':10, 'cena sata':12,
            'dohodak':10);
    writeln;
    WHILE NOT eof (input) DO
        BEGIN
            readln (idbroj, sati, plsata); dohodak :=sati x plsata;
            writeln (idbroj:10, sati:10:1,
            plsata: 10:2, dohodak: 10:2)
        END
END.
```

Postoji još jedan način za otkrivanje kraja podataka. Koristi se graničnik (izvorno sentinel) podataka. To je podatak posebno izabrane vrednosti, pa kad ga računar učita registruje se kraj podataka.

Izmenimo program Obračun da koristi graničnik. Zadnji podatak će biti sastavljen od ID broja 999999. Ta vrednost ID broja je znak, da je to zadnji podatak, inače ID broj predstavlja matični broj radnika čiji će se dohodak računati.

```
PROGRAM obracun (input, output);
{program će koristiti graničnik za detekciju kraja podataka}
CONST
granicnik = 999999;
VAR
idbroj: integer;
sati, plsata, dohodak: real;
BEGIN
    writeln ('ID broj':10, 'sati':10, 'cena sata':12,
            'dohodak':10);
    writeln;
    read (idbroj);
WHILE idbroj <  > granicnik DO
    BEGIN
        readln (sati, plsata);
        dohodak := sati x plsata;
```

```
            writeln (idbroj:10, sati:10:1, plsata:10:2,
            dohodak:10:2); read (idbroj)
        END
    END.
```

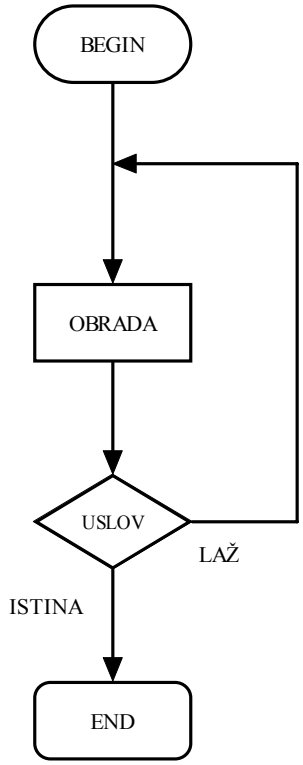
Napomena:
ID broj se učitava iz ostatka podataka za sledećeg zaposlenog, pa na taj način proverava da li ID broj ima ili nema vrednost graničnika, pre nego što se pokuša učitavanje i obrada ostalih podataka. Zbog toga se ID broj za prvog zaposlenog učitava s odvojenom READ naredbom, a nakon što je obrađen podatak za jednog zaposlenog, učitava se ID broj za sledećeg zaposlenog.

7.3. Naredba REPEAT

Pascal dozvoljava još jednu naredbu za kontrolu ponavljanja, to je REPEAT naredba, koja ima sledeći oblik:

```
REPEAT

    naredbe

UNTIL Booleov izraz
```

Dijagram toka za ovu naredbu je:



Bilo koji broj naredbi odvojen tačka-zarezom može biti smešten između REPEAT i

UNTIL. Budući da reči - REPEAT i UNTIL čine oblik poput para zagrada, BEGIN i END nisu potrebni.

Naredbe između REPEAT i UNTIL se izvršavaju nakon što je procenjen Booleov izraz. Ako je vrednost Booleovog izraza ISTINA nema više ponavljanja, pa računar ide na sledeću naredbu u programu. Ako je vrednost Booleovog izraza LAŽ naredbe ponavljanja se ponovo izvršavaju itd.

Na primer, naredbe:

```
i:=0;
REPEAT
      write   (i:4);
      i:=i + 5
UNTIL i >20
```

izazivaju ispis:

```
0        5        10        15        20
```

Posle vrednosti 20 naredba:

```
i := i  + 5
```

menja vrednost *i* u 25. Sada vrednost:

```
i >20
```

postaje ISTINA i nema više ponavljanja.

Najvažnija odlika naredbe REPEAT je da se vrednost Booleovog izraza proverava posle izvršenja naredbi ponavljanja, a ne pre. Ovo znači, da se naredbe ponavljanja uvek izvršavaju najmanje jednom. Možemo napraviti poređenje s dve grupe sličnih naredbi, jedna upotrebljava WHILE, a druga REPEAT:

```
i := 21                          i:= 21;
WHILE i < =20 DO                 REPEAT
      BEGIN                            write (i:4);
        write (i:4);                   i:=i + 5
        i:=i + 5                 UNTIL i > 20
END
```

Naredbe levo neće izazvati nikakav ispis. Naredbe desno uslovljavaju ispis 21. Kada se izvrše naredbe na levoj strani, vrednost *i* postaje 21, a kada se izvrše na desnoj strani *i* postaje 26. Razlika između ova dva primera je u primeru na levoj strani, gde se naredbe ponavljanja ne izvršavaju, dok u primeru na desnoj strani imamo izvršeno jedno ponavljanje. To je zato što naredba WHILE ispituje (proverava) Booleov izraz pre svakog izvršavanja naredbi ponavljanja, a REPEAT ispituje posle.

8. Uslovne naredbe

Uslovne naredbe omogućuju selektivno izvođenje delova programa. U ovu grupu naredbi svrstane su IF - THEN - ELSE i CASE naredbe.

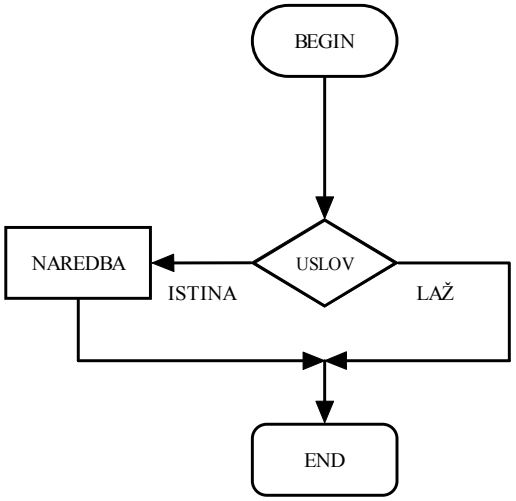
8.1. Jednostruki izbor

Kod jednostrukog izbora računar proverava vrednost Booleovog izraza pre izvršavanja određene naredbe. Ako je vrednost izraza ISTINA naredba se izvršava, a ako je LAŽ naredba se ne izvršava i izvodi se sledeća naredba u programu.

U Pascalu se za jednostruki izbor upotrebljava naredba IF - THEN koja ima sledeći oblik:

```
IF Booleov izraz THEN
    naredba
```

Dijagram toka za jednostruki izbor je:



Sledeći primer pokazuje opisanu naredbu. Želimo da učitamo ID brojeve i mesečnu zaradu zaposlenih, te da ispišemo ID brojeve i zarade koje su veće od 1000. Učitaćemo podatke za sve zaposlene i upotrebiti IF naredbu da bi se ID broj i mesečna zarada ispisali samo onda kad prelaze 10000.

```
PROGRAM  izbor (input, output);
{ispisati ID brojeve i zarade koje su veće od 10000}
CONST
    limit = 10000.0;
VAR
    idbroj: integer;
    zarada:real;
BEGIN
    writeln ('ID broj':10, 'ZARADA':10);
    writeln;
    WHILE NOT eof (input) DO
        BEGIN
            readln (idbroj, zarada);
            IF zarada  >   limit THEN
```

```
                writeln (idbroj:10, zarada:10:2)
            END
        END.
```

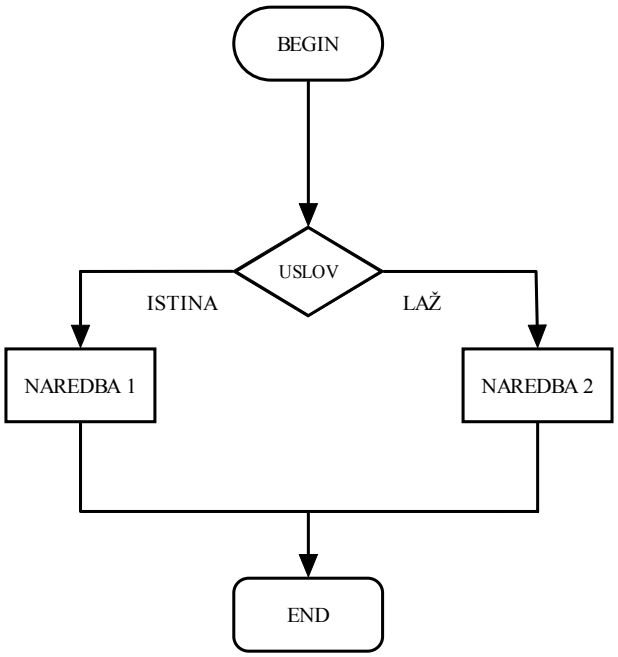
8.2. **Dvojni izbor**

Kod dvojnog izbora postoji mogućnost izvršavanja jedne od dve naredbe, zavisno od uslova koji je ispunjen.

Naredba IF - THEN - ELSE ostvaruje dvojni izbor. Njen oblik je:

```
IF Booleov izraz THEN
    naredba 1
ELSE
    naredba 2
```

Dvojni izbor može se prikazati dijagramom toka kao:



I ovde, kad se izvodi IF naredba procenjuje se Booleov izraz. Ako je izraz ISTINA izvršava se naredba 1. Ako je izraz LAŽ izvršava se naredba 2. Ove dve naredbe mogu biti jednostavne ili složene.

Evo primera: platni spisak zaposlenih. Platu računamo prema broju sati i ceni sata, tj.

```
plata:=sati x cena
```

Uzima se 40 redovnih sati, a prekovremeni se uzimaju 1,5 sat za 1 sat, pa će za onog ko radi preko 40 sati obračun biti:

```
plata :=40.0 x cena + 1.5 x cena (sati - 40.0)
```

Znači da imamo dve naredbe za računanje plate: jednu za radnike koji rade 40 sati ili manje, a drugu za radnike koji rade više od 40 sati.

```
PROGRAM platnispisak (input, output);
VAR
idbroj: integer;
sati, cena, plata:real;
BEGIN
    writeln ('ID broj':10, 'SATI':10, 'CENA':10,
             'PLATA':10);
             writeln;
    WHILE NOT eof (input) DO
        BEGIN
            readln (idbroj, sati, cena);
            IF sati > 40.00 THEN
               plata :=40.0 x cena + 1.5 x cena x (sati - 40.0)
            ELSE
               plata := sati x cena;
               writeln (idbroj:10, sati:10:1, cena:10:2,
               plata:10:2)
        END
END.
```

8.3. Višestruko odabiranje

Jedan način uzastopnog odabiranja u Pascalu je upotreba IF naredbe više puta. Drugi način koristi naredbu CASE. Da bi se pojednostavnilo objašnjenje simbolički ćemo prikazati spomenute naredbe:

```
IF b1 THEN
  S1
```

```
IF b1 THEN
    S1
ELSE
    S2
```

b1 predstavlja Booleove izraze, a Sl i S2 naredbe. Analizirajmo drugi slučaj:

```
IF b1 THEN
    S1
ELSE
    S2
```

Budući da S1 i S2 mogu biti bilo koje Pascal naredbe, mogu biti i IF naredbe. Evo kako to izgleda:

```
IF b1 THEN
      IF b2 THEN
            S3
       ELSE
            S4
ELSE
            S2
```

Kod izvođenja ovakvog niza naredbi najpre se proverava vrednost Booleovog izraza b_1. Ako je vrijednost LAŽ tada se izvrši S2, a ako je b_1 ISTINA izvršava se sledeća IF naredba, pa proverava Booleov izraz b_2.

Ako je b_2 ISTINA izvrši se S3, a ako je LAŽ izvrši se S4. Ovo sve možemo prikazati u tabeli:

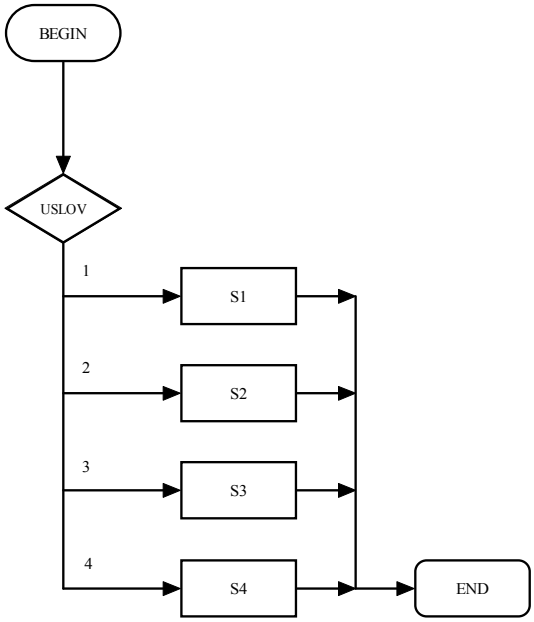
b1	b2	Naredba koja se izvršava
LAŽ	LAŽ	S2
LAŽ	ISTINA	S2
ISTINA	LAŽ	S4
ISTINA	ISTINA	S3

8.4. Naredba CASE

Ponekad možemo naći neki izraz čija će vrednost odrediti koje će se naredbe izvršiti. U tom slučaju koristiće se naredba CASE - puno lakše nego IF.

Ilustrovaćemo naredbu CASE primerom i odgovarajućim dijagramom toka:

```
CASE i of
    1 :S1;
    2: S2;
    3 : S3;
    4 : S4 END
```



Vrednost *i* određuje koja će se naredba izvršiti. Ako je *i*= 1 izvršiće se Sl, ako je *i = 2* S2 itd. U ovom primeru mora vrednost *i* biti u nizu od 1 do 4. Ako je *i* izvan toga niza javlja se greška.

Moguće je označavanje s više od jedne vrednosti. Na primer:

```
CASE j - 3 of
    1,2,3:S1;
    4,5:S2;
    6,7,8:S3
END
```

Ako je vrednost j - 3, 1, 2 ili 3 izvršava se Sl, ako je 4 ili 5 izvršava se S2 itd.

Izraz čija vrednost određuje koja će se naredba izvršiti naziva se *selektor.* Vrednost selektora može biti celobrojnog, znakovnog ili Booleovog tipa. Ne sme biti realnog tipa.

Evo jednog zadatka u kojem će se koristiti naredba CASE.

Potrebno je izvršiti obradu učeničkog testa prema broju bodova, te napraviti distribuciju ocena.

Neka je, na primer, selektor znakovnog tipa upotrebljen za grupisanje po ocenama i to A za odličan, B za vrlo dobar itd. To se može u Pascalu prikazati kao:

```
CASE stepen OF 'A':ocena:=5;
      'B':ocena:=4;
      'C':ocena:=3;
      'D':ocena:=2;
      'F':ocena :=1
END
```

Budući da se zahteva obrada prema broju bodova, sledeća tabela daje grupe s odgovarajućim brojevima bodova kako će se primeniti u programu:

Slovni stepen	Bod	Bod div 10
A	90-100	9, 10
B	80-89	8
C	70-79	7
D	60-69	6
F	0-59	0, 1, 2, 3, 4, 5

Evo sada programa:

```
PROGRAM ocena (input, output);

VAR
    ucenik, bod,
   azbir, bzbir, czbir, dzbir, fzbir : integer;
   ocena : char;
BEGIN
    azbir :=0; bzbir :=0; czbir :0; dzbir:=0; fzbir:=0;
    writeln ('ucenik BR':10, 'bod':10, 'ocena':10);
    writeln;
    WHILE NOT eof (input) DO
        BEGIN
            readln (ucenik, bod);
            CASE bod DIV 10 OF
            0, 1, 2, 3, 4, 5:
            BEGIN
                    ocena:='F'; fzbir:=fzbir + 1
            END;
        6:BEGIN
                    ocena := 'D'; dzbir := dzbir + 1
          END;
        7: BEGIN
                    ocena := 'C'; czbir := czbir + 1
          END;
        8: BEGIN
                   ocena:='B'; bzbir := bzbir + 1
          END;
        9, 10: BEGIN
                    ocena := 'A'; azbir := azbir + 1
          END
        END; {case}
        writeln (ucenik:10, bod:10, ocena:10)
        END;{while}
        writeln;
        writeln ('Distribucija ocena:');
        writeln ('A:':5, azbir:3, 'B:':5, bzbir:3, 'C:':5, czbir:3, 'D:':5,
        dzbir:3, F:':5, fzbir:3)
END.
```

1. Primer za analizu:

Želimo da izračunamo i prikažemo X^Y. Korisnik upisuje broj i eksponent respektivno.

Ulaz: broj i eksponent
Izlaz: rezultat

Algoritam:

1. Učitaj broj i eksponent.
2. Proveri da li je broj veći od 0
3. Ako je X<0 tada prikaži poruku
4. U suprotnom
    - Izračunaj X^Y
    - Prikaži rezultat

Implementacija

```
PROGRAM stepen(input,output);
Var
        broj, eksponent: integer;
        rezultat: real;

BEGIN
        write("Upisi broj i eksponent respektivno: ");
        readln(broj, eksponent);
        if broj<=0 then
                writeln("Broj treba da bude veci od 0 …")
        else begin
                rezultat:=exp(eksponent*ln(broj));
                writeln("Rezultat je: , rezultat:5:2);
        end
END
```

2. Primer za analizu:

Napiši program koji izračunava i prikazuje realne korene (rešenja) jednačine opšteg oblika $\mathbf{ax^2 + bx + c = 0}$

Ulaz: koeficijenti
Izlaz: koreni

Algoritam:

Učitaj koeficijente
Izračunaj diskriminantu (delta)
Proveri diskriminantu

Ako je manja od 0, tada ne postoji realan koren.
Ako je jednaka 0, tada postoji samo jedan koren. Izračunaj ga i prikaži.
U suprotnom, postoje dva korena. Izračunaj ih i prikaži.

Implementacija:

```
PROGRAM koreni(input, output)
Var
        a, b, c: real;
        koren1, koren2: real;
        delta: real;

BEGIN
        write("Upisite koeficijente kvadratne jednacine ax^2+bx+c=0: ");
        readln(a,b,c);
        if a=0 then
                writeln("To nije kvadratna jednacina!")
        else begin
                delta:=sqr(b) – 4*a*c;
                if delta < 0 then
                        writeln("Ne postoji realan koren ove jednacine!")
                else
                        if delta = 0 then begin
                                koren1:= b/2*a;
                                writeln("Postoji samo jedan koren …");
                                writeln("x = ", koren1:4:2);
                        end
                        else begin
                                koren1:= (-b+sqrt(delta))/2*a;
                                koren2:= (-b-sqrt(delta))/2*a;
                                writeln("Koreni su x1= ", koren1:4:2,"i x2= ", koren2:4:2)
                        end;
                end;
        end;
END.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (5)

8. Funkcije i procedure

Kad se gradi jedan složeni sklop kao što je na primer automobil, konstruktori neće početi od najosnovnijih sirovinskih materijala, kao što je ruda gvožđa, ili nafta od koje se prave plastični materijali. Umesto toga, automobil će se sklapati od (jednom) prethodno napravljenih delova, kao što su autogume, akumulatori, motori, automobilski hladnjaci. Svi ti delovi napravljeni su u različitim fabrikama.

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

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

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

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

8.1. Funkcije

Već smo videli kako se koriste standardne funkcije, kao što su kvadriranje (sqr), kvadratni koren (sqrt), odbacivanje članova niza (trunc), zaokruživanje (round), čije su definicije ugrađene u Pascal. Sada ćemo videti kako se definišu vlastite funkcije koje su nam potrebne kod nekog posla, a nisu ugrađene u Pascal.

Funkcija se definiše koristeći funkcijsku deklaraciju. Kao primer definišimo funkciju **fdec** koja daje deo broja iza decimalne tačke. Evo nekoliko primera za fdec:

izraz	**vrednost**
fdec (3.14)	**0.14**
fdec (267.21)	**0.21**
fdec (5.0)	**0.0**

Deklaracija funkcije fdec je:

```
FUNCTION fdec (x: real): real;

BEGIN
   fdec := x - trunc (x)
END
```

Deklaracija funkcije ima istu strukturu kao i program s dva izuzetka:

1) Funkcija počinje s naslovom FUNCTION umesto PROGRAM,

2) Nema točke poslije naredbe END.

Naslov funkcije za fdec je

```
FUNCTION fdec (x: real): real;
```

Rezervisana reč FUNCTION predstavlja funkcijski naslov. Zatim dolazi ime funkcije, fdec, slede u zagradama formalni parametri. Formalni parametri moraju se podudarati sa stvarnim parametrima, koji će se pojaviti kad se funkcija bude koristila. Formalni parametar je deklarisan kao

```
x:real
```

i kaže, da će funkcija uzeti jedan parametar, koji mora biti realni broj. Vrednost stvarnog parametra biće označena sa *x*. Konačno, „:real" na kraju naslova funkcije specificira, da će ova funkcija kao rezultat imati realnu vrednost. Naredba deklaracije funkcije ima samo jednu naredbu

```
fdec = x - trunc (x)
```

koja računa vrednost funkcije i označava je imenom funkcije. Vrednost, koja se dobija računanjem, uvek se obeležava imenom funkcije. Bez obzira koliko postoji naredbi konačno jedna od njih mora označavati vrednost koja će se izračunati, a ta je uvek označena imenom funkcije.

Kako se funkcija koristi u programu?

Neka se, na primer, u programu pojavila naredba koja koristi deklarisanu funkciju fdec:

```
z :=fdec (3.14)
```

Kada se pri izvođenju programa u računaru dođe do imena funkcije fdec organizuje se područje memorije koje se dodeljuje funkciji. To se područje sastoji od dve lokacije, *x* i fdec. Zatim se dodeljuje vrednost stvarnog parametra formalnom parametru *x*,:

```
x := 3.14
```

Sada će naredba biti izvršena

```
fdec := x – trunc (x)
```

i vrednost 0.14 je smeštena u lokaciju nazvanu fdec. Kad je to izvršeno sadržaj lokacije nazvane fdec biće upotrebljen na mestu funkcije u programu sa fdec.

```
z:=fdec
```

Može se reći, da je z := fdec (3.14) ekvivalentno

```
x:=3.14
fdec :=x - trunc (x);
z:=fdec
```

U naredbi funkcijske deklaracije ime funkcije nalazi se na levoj strani dodeljene naredbe i ponaša se kao i svaka druga promenljiva. Međutim, ako se ime funkcije koristi u nekom izrazu ima drugačije značenje!

Već je bilo spomenuto da parametar funkcije može biti i izraz. Na primer:

```
y:=3.14;
z := 2.5 + fdec (2.0 * y + 0.5)
```

ekvivalentno je sa

```
y:=3.14;
```

```
x := 2.0 * y + 0.5;
fdec := x - trunc (x);
z:=2.5+fdec
```

U ovom primeru vrednost 6.78 dodeljena je *x*, a vrednost 0.78 dodeljena je fdec. Vrednost *z* je 3.28.

U programu deklaracije funkcije i procedure dolaze iza deklaracija promenljivih.

Sledeći program pokazuje kako funkcija fdec može biti deklarisana i upotrebljena u programu.

```
PROGRAM fdecimal (input, output);

VAR
   v:real;
FUNCTION fdec (x : real) : real;
BEGIN
   fdec := x - trunc (x)
END; {za fdec}
BEGIN {glavni program)
   WHILE NOT eof (input) DO
      BEGIN
         readln (v);
         writeln (v:10:2, fdec (v):10:2)
      END
END.
```

Izvođenje počinje sa naredbama glavnog programa. Naredba funkcijske deklaracije izvodi se samo onda kad je funkcija dodeljena u glavni program.

U definiciji funkcije ponekad možemo imati više od jednog parametra. Na primer, funkcija koja računa zapreminu kutije; daje njenu dužinu, širinu i visinu. Naslov funkcije za taj slučaj biće

```
FUNCTION zapremina (duzina, sirina, visina: real) : real;
```

Kompletna deklaracija funkcije biće:

```
FUNCTION zapremina (duzina, sirina, visina: real): real;
   {računanje volumena kutije}
BEGIN
   zapremina :=duzina * sirina * visina
END
```

Kada se pozove funkcija *zapremina,* funkciji se u memoriji dodeljuje područje za njene promenljive dužinu, širinu i visinu. Na primer, naredba

```
writeln (zapremina (9.0, 5.0, 3.0): 10:2)
```

ekvivalentna je sa

```
duzina.= 9.0;
sirina:=5.0;
```

```
visina:=3.0;
zapremina :=duzina * sirina * visina;
writeln (zapremina: 10:2)
```

Računar će ispisati rezultat 135.00. Ovo je bio primer vrlo jednostavne funkcije, a sada evo jednog složenijeg zadatka:

izračunati faktorijele za zadati parametar i to

izraz	vrednost
3!	6
4!	24
5!	120
6!	720

Faktorijele računamo tako da uzmemo vrednost promenljive $f = 1$, pa zatim množimo sve do vrednosti n sa korakom povećavanja za jedan.
Program će izgledati ovako:

```
PROGRAM rekurzija (input, output);
VAR n:integer;
FUNCTION fak (n: integer): integer;
{računanje n faktorijel)
VAR
   i, f: integer;
BEGIN
   f:=1;
FOR i:=f TO n DO
      f:=f*i;
      fak:=f
END;
BEGIN
     readln (n); writeln (n:5, fak (n):10)
END.
```

Pretpostavimo da se program fak poziva sledećom naredbom:

```
j:=fak (5)
```

U trenutku poziva fak (5) kada se izvodi program, promenljivama funkcije se dodeljuje potrebno veliko područje memorije. Nakon izvođenja ovog potprograma memorijske lokacije dodeljene potprogramu se oslobađaju. Gornja naredba je ekvivalentna naredbama:

```
n:=5;
```

```
f:=1;
FOR i = 1 TO n DO
    f:=f*i;
fak==f;
j:=fak
```

8.2. Procedure

Procedure su takve strukture kojima se u programu izvršava neki postupak i to jedanput ili više puta prema potrebi. To je vrsta potprograma u Pascalu.

Evo jednog jednostavnog programa u kojem će se koristiti dve procedure.

```
PROGRAM voće (input, output);
PROCEDURE prvo;
   BEGIN
      writeln ('Jabuke')
   END;
PROCEDURE drugo;
   BEGIN
      writeln ('Kruške')
   END;
BEGIN {glavni program}
   prvo;
   drugo;
   prvo
END.
```

Nakon izvođenja ispis će biti

Jabuke

Kruške

Jabuke

U glavnom programu VOĆE prva naredba je bila PRVO, druga naredba DRUGO, a treća naredba PRVO. Sve tri naredbe nazivaju se **procedurne naredbe** (engl. procedure statement), one su-zapravo imena procedura koje su definisane u programu, a počinju rezervisanom reči PROCEDURE.

Prva procedura pozvana je u glavnom programu dva puta, a druga jedanput.

Ovaj primer pokazuje samo princip korišćenja procedure u Pascalu. Sledeći primer pokazaće postupak kojim će se moći uređivati parovi celih brojeva i to tako, da je na prvom mestu manji broj, a na drugom veći.

```
PROCEDURE zamena (VAR x, y : integer);

VAR
```

```
   T: integer;
BEGIN
   T := x;
   x := y;
   y := T
END.
```

Deklaracija procedure dolazi u programu na početku programa posle deklaracije promenljivih.

Procedura u Pascalu, kako je rečeno, počinje rezervisanom reči PROCEDURE iza čega dolazi ime procedure, te u zagradi parametri.

Ime se bira tako da asocira na postupak koji procedura izvršava. Imena procedura moraju biti različita i ne smeju biti ista kao imena bilo koje konstante, promenljive ili tipa koji je korisnik definisao u programu.

Parametri procedure deklarišu se nabrajajući ih sa njihovim tipovima. U datom primeru deklarišu se dva parametra *x* i *y* sa rezervisanom reči VAR, oba celobrojnog tipa.

Neka se sada iskoristi data procedura ZAMENA za uređenje 4 para brojeva i to:

10, 9,
6, 1,
2, 16,
26, 14,

Program će izgledati ovako:

```
PROGRAM parovi (input, output);

VAR

   x, y, I, N : integer;

PROCEDURE zamena (VAR x, y : integer);

VAR
   T: integer;
   BEGIN
      T :=x;
      x := y;
      y := T
   END;
BEGIN {glavni program}
   writeln ('Uredjeni parovi');
   readln (n);
   FOR i:=1§TO N DO
      BEGIN
        read (x, y);
        IF x > y  THEN
        zamena (x, y);
        writeln (x, y)
```

```
        END;
        writeln ('kraj')
END.
```

Nakon izvršenja program će dati ispis

```
UREĐENI  PAROVI
   9          10
   1           6
   2          16
  14          26
    kraj
```

Ista se procedura može iskoristiti za poredak, na primer, grupa od 3 broja. U tom slučaju ZAMENA će se u glavnom programu pozivati tri puta pa će program biti:

```
PROGRAM trojke (input, output);

VAR
   a, b, c, i, n : integer;
PROCEDURE zamena (VAR x, y : integer);
   VAR
      T:integer;
   BEGIN
      T := x;
      x := y;
      y := T
   END;
BEGIN {glavni program}
writeln ('Uredjene trojke');
readln (n);
FOR |:=1 TO n DO
BEGIN
   read (a, b, c);
   IF A> B THEN
      zamena (a, b);
   IF B>C THEN
      zamena (b, c);
   IF A > B THEN
      zamena (a, b);
   writeln (a, b, c)
   END
END.
```

Pretpostavimo da su date trojke

4, 2, 15

8, 1, 0
25, 6, 44
19, 7, 10

Nakon obrade ispis će biti:

UREĐENE TROJKE
2 4 15
0 1 8
6 25 44
7 10 19

Izvršavanje ZAMENA (proml, prom2) obavlja zamenu vrednosti promenljivih proml i prom2 koje su specificirane u procedurnoj naredbi. U proceduri ZAMENA *x* i *y* su formalni parametri.

Posebne promenljive koje će biti korišćene u proceduri date su u procedurnoj naredbi i one izazivaju izvršavanje, a poznate su pod nazivom argumenti procedurne naredbe. U programu TROJKE, A i B su argumenti prve procedurne naredbe (tj. ZAMENA (A, B)), a B i C su argumenti druge procedurne naredbe itd.
Pre izvršavanja procedure parametri su zamenjeni sa odgovarajućim argumentima pojedinačne procedurne naredbe.
Atributi argumenata moraju se slagati sa odgovarajućim parametrima. Moraju biti istoga tipa. Tako navedena procedura ZAMENA može služiti samo za cele brojeve.

U Pascalu postoje dve vrste parametara i to: promenljivi parametri (engl. variable parameters) i parametri vrednosti (engl. value parameters).
Sledeći primeri pokazuju obe vrste:

```
a) PROCEDURE zadatak1  (VAR x, y, z : real)
b) PROCEDURE zadatak2 (n: integer; VAR x:real)
```

Postoje pravila za parametre u proceduri.

Neki parametri služe kao izlazno sredstvo za proceduru.

Parametri koji su rezervisani za izlaz moraju biti deklarisani kao promenljivi parametri. Oni moraju imati rezervisanu reč VAR u svojoj deklaraciji (primer a). S druge strane neki parametri služe samo kao ulazni; oni predaju vrednosti proceduri i obično su deklarisani kao parametri vrednosti (n u primeru b)). Ako parametar služi za oboje, ulaz i izlaz, onda mora biti deklarisan kao promenljivi parametar.

U primeru procedure ZAMENA parametri *x* i *y* su oba ulazni i izlazni parametri, (budući da oni donose vrednosti u proceduru i daju vrednosti iz procedure) pa moraju biti deklarisani kao promenljivi parametri.

Imena parametara u proceduri su ,,lokalna" za tu proceduru. Ona nisu u nikakvoj vezi s imenima koja se koriste izvan procedure. Parametri u proceduri su formalni. U datom primeru procedure ZAMENA *x* i *y* su izabrani bez obzira gde će se procedura koristiti.

U prvom primeru programa PAROVI slučajna je koincidencija da i program ima svoje vlastite *x* i *y*.

Drugi program TROJKE koristi istu proceduru, a nema *x* i *y*. Neke procedure ne trebaju nijedan parametar. U tom slučaju nema deklaracije parametara (primer na početku poglavlja o procedurama!)

Sve vrste definicija i deklaracija što se koriste u glavnom programu (CONST, TYPE i VAR) mogu biti date i u proceduri.

Sve definicije i deklaracije unutar procedure određuju objekte koji su lokalni za tu proceduru. Oni su nedefinisani izvan te procedure i ne mogu se koristiti izvan procedure. Takav je primer promenljiva T u proceduri ZAMENA, tj. ona je lokalna promenljiva.

Izvršenje procedurne naredbe uključuje zamenu parametara sa argumentima. Na primer, u programu TROJKA x i *y* procedure ZAMENA zamenjeni su sa A i B, zatim sa B i C itd.

Promenljivi parametri u proceduri nemaju svoju vlastitu vrednost, ali kad procedura bude pozvana ti će parametri biti zamenjeni sa promenljivama u glavnom programu koje su date kao argumenti poziva. Promenljivi parametri se koriste za vraćanje rezultata iz procedure u glavni program.

Međutim, za parametre vrednosti prilikom izvršavanja procedure procenjuje se odgovarajući argument i njegova vrednost se kopira parametru. Procedura dalje operiše sa tom novom vrednošću. Kad je procedura kompletno izvršena, ta se kopija uništi uz sve ostale tragove procedure.

Bitno je naglasiti, da ta kopija vrednosti nije kopirana nazad u originalnu vrednost argumenta, pre nego što je procedura uništena. To dalje znači, da za parametre vrednosti procedura pribavlja vrednost od argumenta ali je nesposobna da promeni vrednost argumenta. To je razlog zašto se promenljivi parametri koriste za vraćanje rezultata iz procedure, a ne parametri vrednosti.

Argumenti za parametre vrednosti mogu biti i izrazi, a za promenljive parametre samo promenljive.

Na primer,

```
PROCEDURA zamena (x : integer; VAR y : integer)
```

dozvoljava sledeće procedurne naredbe:

```
zamena (5, j)
zamena (x + t, j)
zamena (trunc (x), j) itd.
```

Oblast identifikatora

```
FUNCTION fak (n:integer): integer;          naslov
VAR
i, f :integer;
BEGIN                                       blok
     f:=1;
     FOR i  :=1  TO n DO
          f := f * i;
     fak := f
END
```

Struktura u programu, funkciji ili proceduri je takva, da najpre imamo *naslov,* a zatim niz naredbi koje zovemo blok, na primer.

Deklarisanje promenljivih primenjuje se samo na blok koji sadrži deklaraciju. Značajno je, da se to odnosi samo na promenljive unutar tog bloka.

Deo programa u kojem je to značajno odnosi se na svojstvo identifikatora koje se zove *oblast* (scope) identifikatora. Oblast promenljive je blok u kom je promenljiva deklarisana. U funkciji fak oblast *i* i *f* je blok sadržane deklaracije:

```
VAR
   i, f: integer;
```

Samo u tom bloku vrede *i* i *f*. Razmotrimo sada program koji koristi funkciju fak za ispis brojeva i njihovih faktorijela:

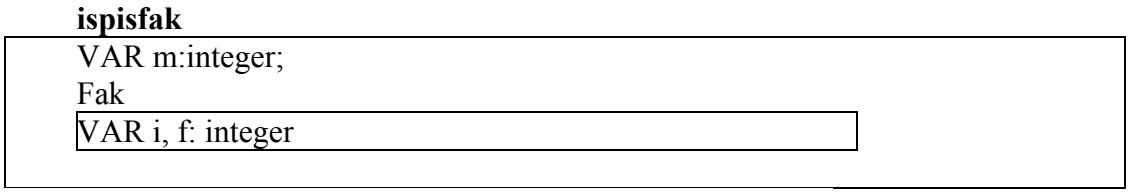
```
PROGRAM ispisfak (output);

VAR
   m: integer;
    FUNCTION fak (n : integer) :
     integer;
    VAR
    i, f: integer;
    BEGIN
       f:=1;
          FOR i:=1 TO n DO
          f:=f*i;
    fak:=f
    END; {za funkciju)
```

```
BEGIN  {glavni  program}
   writeln ('broj':10, 'Faktorijel': 12);
   writeln;
   FOR m:=1 TO 7 DO
      writeln (m:10, fak (m): 10)
END.
```

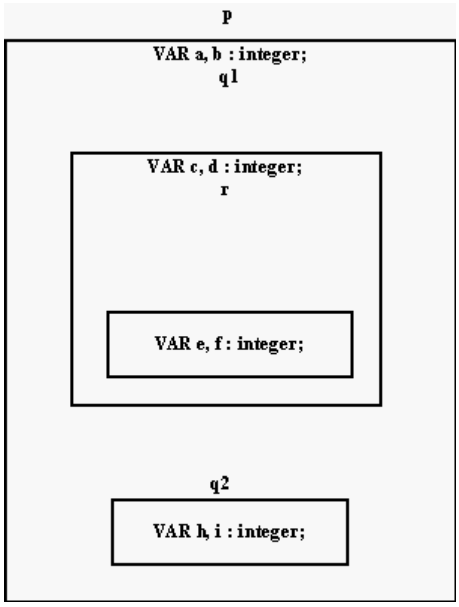
Imamo dva bloka u tom programu. Jedan blok počinje odmah iza naslova programa i proteže se do kraja programa. Drugi blok počinje odmah iza naslova FUNCTION za funkciju faktorijela i proteže se do kraja funkcije fak. Zgodno je nazvati blok koristeći imena programa, funkcije ili procedure, koja se javljaju u naslovu. Zato ćemo se pozivati na blok koji počinje odmah iza naslova programa kao ispis fak i na blok koji počinje odmah iza naslova funkcije kao fak blok. Promenljiva *m* deklarisana je u bloku ispisfak.

Budući da je blok fak unutar bloka ispisfak to se *m* može koristiti i u bloku fak.
Međutim, promenljive *i* i *f* deklarisane su u bloku fak i nisu dostupne spoljašnjim blokovima; u ovom slučaju samo glavnom programu.

To se može ilustrovati blok dijagramom:

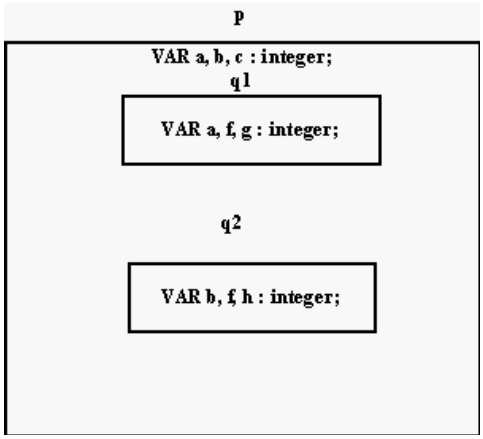


Dijagram pokazuje oblast promenljivih; *m* može biti dostupno unutar spoljašnjeg bloka ili unutrašnjeg bloka; *i* i *f* mogu biti dostupni samo unutrašnjem bloku. Možemo se pomoći ovakvim prikazom smatrajući, da iz bloka fak možemo videti van i videti promenljivu m, ali izvan fak bloka nije moguće videti unutra i opaziti *i* i *f*.
Evo sada jednog složenijeg primera:



U ovom primeru glavni program se zove p. U p su deklarisane dve funkcije ili procedure nazvane ql i q2. Konačno, unutar ql deklarisana je još jedna funkcija ili procedura r. Promenljive *a* i *b* deklarisane su u spoljašnjem bloku glavnog programa p. One su dostupne svim unutrašnjim blokovima. Promenljive deklarisane u spoljašnjem bloku p kažemo da su globalne, jer su one dostupne svim blokovima u programu.

Promenljive *c* i *d* su dostupne ql i r. One nisu dostupne p i q2. Promenljive *e* i *f* mogu se koristiti samo unutar bloka r. Promenljive *h* i *i* mogu se koristiti samo unutar bloka q2. Postoji još jedna situacija koja može zadavati probleme. Evo blok dijagrama pa razmislite:



Pojavljuju se isti identifikatori u više od jednog bloka. To je u Pascalu dozvoljeno, ali nije poželjno. Prvo što treba napomenuti je, da promenljive deklarisane u različitim blokovima nemaju ništa jedna s drugom, osim što imaju isto ime.

Tako *a* deklarisano u p odnosi se na različitu memorijsku lokaciju od *a* koje je u q1; *b* deklarisano u p ima različitu memorijsku lokaciju od *b* deklarisanog u q2 i tako redom.

Zbog toga, naredbe mogu imati različito značenje u zavisnosti od mesta gde se nalaze u programu. Na primer, ako se naredba:

```
f:=25
```

nalazi u ql, tada će se dodeliti 25 promenljivoj *f* deklarisanoj u ql. Ako se ista naredba nalazi u q2 dodeliće se 25 promenljivoj *f* deklarisanoj u q2. Sada razmotrimo naredbu:

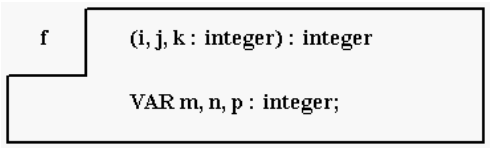
```
a:=100
```

Ako je ta naredba smeštena u p ili q2, dodeliće 100 promenljivoj *a* deklarisanoj u p. Ali šta ako je naredba smeštena u ql? Postoji pravilo za ovaj slučaj, koje kaže: kad je deklarisano više promenljivih sa istim imenom, tada to ime dodeljujemo promenljivoj, koja ima najmanju oblast. Tako gornja naredba a := 100 dodeljuje 100 promenljivoj *a* u ql, budući da ta promenljiva ima manju oblast od *a* deklarisane u p.

Još bi bilo potrebno znati, kako ćemo odrediti blok u kom je ime funkcije ili procedure deklarisano ili formalni parametar deklarisan. Ime funkcije ili procedure je razmatrano kao deklarisano u bloku koji obuhvata definiciju funkcije ili procedure. Formalni parametri s druge strane, razmatrani su kao deklarisani u bloku koji sledi iza naslova funkcije ili procedure. Na primer:

```
FUNCTION f (i, j, k:integer):integer; VAR
   m, n, p:integer; BEG
END
```

Možemo napraviti dijagram oblasti identifikatora:



Formalni parametri su deklarisani unutar bloka koji sledi iza naslova funkcije ili procedure. Ali ime funkcije ili procedure je levo izvan tog bloka, tako da je to deklarisano u bloku, koji obuhvata definiciju funkcije ili procedure. To je potrebno zato da funkcija ili procedura može biti pozvana iz bloka u kojem je deklarisana.

Vidimo u ovom delu, da je moguće dati različitim promenljivama, konstantama, funkcijama i dr. ista imena, uz deklaraciju u različitim blokovima. Ali to ne znači, da ćemo sada pisati programe u kojima mnoge različite stvari imaju ista imena!

Evo jednog programa koji će-pokazati oblast identifikatora. Program pretvara zadatu dužinu izraženu u miljama, jardima, stopama i inčima u inče i ispisuje sumu inča uz ispis učitanih podataka.

```
PROGRAM duzina (input, output);
VAR d1, d2, d3, d4, suma: integer;

PROCEDURE zamena (VAR m, y, f, i : integer; inc: integer);
        BEGIN
             m:= inc DIV (1760*36);
           inc:= inc MOD (1760*36);
             y:= inc DIV 36;
           inc:= inc MOD 36;
             f:= inc DIV 12;
             i:= inc MOD 12
        END;
                                    oblast:
                                    svi identifikatori
                                    zamene i programa

FUNCTION inci (m, y, f, i : integer): integer;
        BEGIN
             inci:= (((m*1760)+ y)*3+f)*12+i
        END;
                                    oblast:
                                    svi identifikatori
                                    inca i programa


BEGIN
   read (d1, d2, d3, d4);
   suma:=inci(d1, d2, d3, d4);
   read (d1, d2, d3, d4);
   suma:= suma + inci(d1, d2, d3, d4);
   zamena(d1, d2, d3, d4, suma);
   writeln('Suma je ', suma, 'inca');
   writeln(d1:5, 'milja', d2:4, 'jardi', d3:3,;
     'stopa', d4:3, 'inca')
END
                                          oblast:
                                          svi identifikatori
                                          programa
```

Mi se pozivamo na promenljive deklarisane u programu kao *globalne.* Promenljive deklarisane u svakoj proceduri ili funkciji, kažemo da su *lokalne* za tu funkciju ili proceduru. Naredbe u .proceduri ili funkciji mogu koristiti globalnu promenljivu uz uslov da lokalna promenljiva nema isto ime. One mogu koristiti vrednost globalnih promenljivih i dodeljuju im nove vrednosti. Ovo

osigurava drugi način u kom podatak može prelaziti u ili iz procedure ili funkcije.

Vrednosti, koje su prešle u funkciju ili proceduru mogu biti dodeljene globalnim promenljivama pre nego što je funkcija ili procedura pozvana.

Procedura ili funkcija može dodeliti vrednosti globalnim promenljivama i one mogu biti dostupne glavnom programu nakon povratka iz procedure ili funkcije. Zbog toga koristimo formalne parametre češće nego globalne promenljive. Ali postoje dve situacije u kojima često upotrebljavamo globalne promenljive:

1. Želimo sačuvati neke vrednosti od jednog poziva funkcije ili procedure do sledećeg. To ne možemo učiniti koristeći lokalne promenljive. Svaki put kad je funkcija ili procedura pozvana njene lokalne promenljive zauzimaju prostor u memoriji samo dotle dok se funkcija ili procedura ne izvrši.
   Znači da nakon izvršenja ne postoje lokalne promenljive, pa je slobodan prostor u memoriji za druge promenljive. Ako se funkcija ili procedura ponovo pozivaju lokalnim promenljivama se dodeljuje novi prostor u memoriji.
2. Ponekad, mnoge od funkcija i procedura u programu manipulišu nekom opštom strukturom podataka, recimo „površina". U tom slučaju „površina" može biti deklarisana kao globalna promenljiva, i sve funkcije i procedure mogu koristiti te globalne promenljive.

Primeri:

1. Napišite funkciju koja vraća recipročnu vrednost svog realnog parametra.

```
FUNCTION recipro (num:real): real;
BEGIN
     If num<>0 then
          recipro:=1/num
END;

a:real; b:integer;
a:=2.0;
a:=recipro(a);       {a=0,5}
a:=recipro(5*a);     {a=0,4}
```

2. Napišite funkciju koja kao parametar koristi veliko slovo i vraća kao vrednost ekvivalentno malo slovo. Ako parametar nije veliko slovo, funkcija treba da vrati svoj parametar.

```
FUNCTION maloslovo
BEGIN
     If (ch>='A') and (ch<='Z') then
          maloslovo:=chr(ord(ch)+ord('a')-ord('A'))
```

```
     else
          maloslovo:=ch;
END;

writeln(maloslovo('2')); {2}
writeln(maloslovo('a')); {a}
writeln(maloslovo('A')); {a}
```

3. Napišite funkciju koja učitava N celobrojnih vrednosti i vraća maksimalnu.

```
FUNCTION maksimum(N:integer):integer;
VAR
     Max, i, num : integer;
BEGIN
     read(num);
     Max:=num;
     For i:=2 to N do BEGIN
          read(num);
          if num > Max then
               Max:=num;
     End;
     Maksimum:= Max;
END;
BEGIN
     writeln('Maksimum je ', maksimum(5):4);
END.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (6)

9. Tipovi podataka

Dva su aspekta programiranja računarom:

1) organizacija podataka, koji će biti obrađivani i
2) naredbe, koje izazivaju obradu podataka.

Do sada je sva pažnja bila koncentrisana na naredbe koje vrše obradu. A sada će se obratiti više pažnje na podatke.

U Pascalu, organizacija podataka je ostvariva poznavajući sistem tipa podataka. Tipovi podataka su, kako je već navedeno:

1) jednostavni,
2) složeni ili strukturirani,
3) pokazivači.

Jednostavni tipovi su oni čije vrednosti ne možemo rastavljati u jednostavnije komponente. *Strukturirani* tipovi su oni čije vrednosti možemo rastavljati u jednostavnije komponente. Tip *pokazivač* su oni podaci čije vrednosti lociraju podatke u memoriji računara.

9.1. Skalarni tipovi podataka

Četiri standardna tipa podataka koji su već bili ranije spomenuti: celobrojni, realni, Booleovi i znakovni su skalarni tipovi podataka.

Pascal dozvoljava programeru da definiše novi tip podataka osim standardnih, čije su definicije ugrađene u jeziku. Možemo upotrebiti definisanje tipa preimenovanjem standardnog tipa:

```
TYPE
    fixed = integer;
    float= real;
    bit = Boolean;
    character=char;
```

Iznosimo jedan primer deklaracija koje koriste prethodno određene tipove podataka:

```
VAR
    i, j, k  :fixed;
    x, y, z :float;
    p, q :bit;
    c :character;
```

Kada Pascal — prevodilac obrađuje te deklaracije doideljuje im tipove onako kako su prethodno određeni, tj. i, j, k mogu biti celobrojni podaci, x, y, z realni itd.

Podaci definicije tipa slede iza definicije konstante i prethode deklaraciji promenljive na početku bloka. Ovo je dobro mesto da rezimiramo redosled u kom se nalaze različite definicije i deklaracije:

— definicija konstante,
— definicija tipa,
— deklaracija promenljive,

— deklaracija funkcije i procedure.

Ako se neka ne koristi, izostavlja se u programu.

Pascal dozvoljava da programer definiše novi skalarni tip podataka unošenjem u popis svih mogućih vrednosti svakog tipa. To su tzv. tipovi podataka koje *definiše korisnik* (engl. User-Defined Scalar Types). Vrednosti upotrebljenog tipa su prikazane identifikatorima. Na primer:

```
TYPE
    dan = (pon, utor, sred, cetv, pet, sub,
    nedj); boja = (crvena, zuta,
    zelena,plava);
```

Ako deklarišemo promenljivu *d* kao

```
VAR
   d:dan
```

tada su moguće vrednosti ponedeljak, utorak itd. Oznaka je d := pon, d := utor itd. Primetimo da standardni Booleov tip može biti definisan

```
Boolean = (false, true)
```

Isti identifikator ne sme biti upotrebljen za vrednosti više od jednog tipa. Tako definicije:

```
rang = (civil, kapetan, vojnik, major,
general); oficir = (kapetan, major, general);
```

ne smeju se obe pojaviti u istom programu.

Na skalarni tip podataka primenjuju se operatori i funkcije. Svi podaci skalarnog tipa su u određenom poretku. Za standardne tipove, red za celobrojne i realne brojeve je uobičajeni numerički poredak. Za Booleove vrednosti vrednost

LAŽ prethodi vrednosti ISTINA. Za znakove postoji posebno uređeni niz (collating sequence) na svakom računaru.

Za tipove podataka koje je korisnik definisao poredak je takav kako su podaci uneseni u popis u samoj definiciji tipa. Tako na primer, vrednosti tipa dan nedelja prethodi ponedeljku, ponedeljak utorku itd.

Dok je poredak definisan za svaki skalarni tip podataka, relacijski operatori

= <> < = > = < >

mogu biti primenjeni na vrednosti svakog skalarnog tipa. (Oba operanda moraju biti istog tipa). Sledeći primeri pokazuju primenu relacijskih operatora na vrednosti skalarnih tipova:

Funkcije pred i succ se primenjuju na vrednosti svih skalarnih tipova osim realnog. Funkcija pred daje vrednost koja odmah prethodi vrednosti njenog stvarnog parametra. Succ daje vrednost koja sledi vrednost njenog stvarnog parametra. Evo nekoliko primera primene pred i succ na celobrojne vrednosti:

izraz	vrednost
pred (5)	**4**
pred (4)	**3**
succ (5)	**6**
succ (6)	**7**

U suštini, za celobrojne pred (i) ima istu vrednost kao i — 1, a succ (i) ima istu vrednost kao i 4- 1. Nisu nam potrebni pred i succ za celobrojne dotle dok dobijamo isti efekt koristeći već odomaćeni izraz i — 1 i i + 1.

Ali za Booleov tip i znakovni nisu definisani aritmetički operatori, pa su pred i succ vrlo korisni. Sledeći primeri pokazuju upotrebu pred i succ za te tipove:

izraz	**vrednost**
succ(ponedeljak)	**utorak**
pred(utorak)	**ponedeljak**
pred(istina)	**laž**
pred('y')	**'x'**
succ('x')	**'y'**

Vrednosti sledećih izraza nisu definisane:

pred (laž)
succ (istina)
pred (nedelja)
succ (kralj)
pred (0)
succ (a)

Vrednosti korisnički zadatih tipova podataka u računaru su numerisani počevši od 0 za prvu vrednost. Na primer, za tip dan:

dan = (ned, pon, utor, sred, cetv, pet, sub)
0 1 2 3 4 5 6

Brojevi se nazivaju redni brojevi odgovarajućih vrednosti. Pascal ima standardnu funkciju **ord,** koja daje redni broj stvarnom parametru:

izraz	**vrednost**
ord (ned)	**0**
ord (pon)	**1**
.	
.	
ord (sub)	**6**

Ord funkcija se može primeniti na standardne tipove Booleov i znakovni, isto tako kao i na korisnički skalarni tip. Za Booleov, ord (laž) je 0, a ord (istina) je 1.

Za znakove, redni brojevi znakova će varirati od jednog računara do drugog. Sledi primer za nekoliko ASCII znakova:

izraz	vrednost
ord ('A')	65
ord ('B')	66
ord ('C')	67
ord ('Z')	90
ord ('0')	48
ord ('9')	57

Za znakovni tip postoji funkcija **chr** koja pretvara jedan redni broj u njemu odgovarajući znak:

izraz	vrednost
chr (65)	'A'
chr (90)	'Z'
chr (48)	'0'
chr (57)	'9'

Jedna od upotreba chr funkcije je: za dobijanje neispisivih upravljačkih znakova. Npr. u ASCII kodu chr (7) je „zvono-znak", koji izaziva da zazvoni zvono na terminalu. Taj znak se koristi za upozorenje korisnika.

```
PROCEDURE alarm;

CONST
       bell = 7; {u ASCII kodu)
VAR
   i:integer;
BEGIN
   FOR i:=1 TO 10 DO
      write(chr(bell))
END
```

Druga važna upotreba funkcija ord i chr je pretvaranja između znakova '0' i '9' u odgovarajuće cele brojeve. Izraz

```
ord (c)-ord ('0')
```

pretvara broj vrednosti c u odgovarajuću celobrojnu vrednost:

izraz	vrednost
ord ('0') – ord ('0')	0
ord ('1') – ord ('0')	1

ord ('2') – ord ('0')	**2**
.	**.**
.	**.**
ord ('9') – ord ('0')	**9**

Za pretvaranje jednog celobrojnog podatka u području 0 — 9 u odgovarajuću cifru (znak) koristimo izraz

```
chr (1 + ord ('0'))
```

koji daje sledeće vrednosti:

izraz	vrednost
chr (0 + ord ('0'))	**'0'**
chr (1 + ord ('0'))	**'1'**
chr (2 + ord ('0'))	**'2'**
.	**.**
.	**.**
chr (9 + ord ('0'))	**'9'**

Evo sada primera jednostavnih programa za funkcije ord i chr.

Prvi ispisuje dane u sedmici sa oznakom broja dana.

```
PROGRAM dani;
TYPE
   dan = (ned, pon, utor, sred, cetv, pet,
sub);
   VAR
   sel:integer;
    d: dan;
BEGIN
   read (sel);
   CASE sel OF
1:writeln (ord (pon), 'Prvi radni dan');
2:writeln (ord (utor), 'Drugi radni dan');
3:writeln (ord (srij), 'Treci radni dan');
4:writeln (ord (četv), 'Cetvrti radni dan');
5:writeln (ord (pet), 'Zadnji radni dan u
sedmici');
6:writeln (ord (sub), ord (ned), 'Dani vikenda')
   END
END.
```

Drugi ispisuje odgovarajuće znakove ASCII koda.

```
PROGRAM ispis; VAR
i:integer; BEGIN
   FOR i =1 TO 10 DO
   writeln (chr (65), (chr
90)) END.
```

Može se lako napisati vlastita funkcija koja će pretvarati brojeve u odgovarajuće vrednosti nekog tipa podataka. Na primer, sledeća funkcija boja pretvara brojeve od 0 do 6 u odgovarajuće boje:

```
FUNCTION  boja (n: integer) :farba;
BEGIN
   CASE n OF
      0: boja := crvena;
      1: boja := narandzasta;
      2: boja •= Zuta;
      3: boja := zelena;
      4: boja := plava;
      5: boja := indigo;
      6:boja:=ljubicasta
   END
END.
```

Za ovu funkciju pretpostavlja se da je u glavnom programu definisan tip „farba", pa .se može koristiti u funkciji koja se nalazi u tom programu. *Farba* predstavlja globalni identifikator.

Skalarni tipovi osim realnog mogu se koristiti u naredbi FOR za kontrolu ponavljanja, zatim u naredbi CASE za vrednost selektora.

Na primer, ako je d promenljiva tipa dan, tada:

```
FOR d:=pon TO pet DO
   S1
```

uslovljava, da će se naredba Sl izvršiti 5 puta. Sledeća naredba će ispisati velika slova i njihove redne brojeve:

```
FOR c:='A' TO 'Z' DO writeln
   (c, ord (c))
```

Sledeća naredba ilustruje upotrebu promenljive *m* za tip šahovske figure, kao selektor u naredbi CASE:

```
CASE m  OF
   pesak   :S1;
   lovac   :S2;
   skakac  :S3;
   top     :S4;
   kraljica:S5;
   kralj   :S6
END
```

Ako je vrednost *m* pesak izvršava se Sl, ako je lovac izvršava se S2 itd.

9.2. Podintervalni tipovi podataka

Za svaki skalarni tip podataka, osim realnog, može se definisati novi tip čije vrednosti su samo neke od vrednosti prethodno definisanog tipa. Na primer:

```
digit = 0..9;
```

r određuje deo područja celobrojnog tipa podataka. Dalje:

```
radndan = pon.. pet;
```

određuje podintervalni tip tipa „dan". Tip čije se vrednosti koriste za određivanje podintervalnog tipa naziva se *pridruženi skalarni tip* (engl.

associated scalare type). Sve operacije koje mogu biti primenjene na pridruženi skalarni tip mogu, takođe, biti primenjene na tip nižeg reda. Jedina razlika između dva tipa je ta, da kada je vrednost dodeljena promenljivoj podintervalnog tipa, tada se proverava da li dodeljena vrednost leži u višem području. Ako to nije, javlja se greška.

Pretpostavimo, da su *i* i *n* deklarisani kako sledi:

```
VAR
    i:integer;
    n :digit
```

Pridruživanje `i := n` je uvek valjano dok je vrednost tipa „digit" takođe i vrednost tipa „integer". Sa druge strane n := i je valjano samo ako vrednost *i* leži u području između 0 — 9. Vrednost tipa integer je takođe vrednost tipa digit, ali samo ako ona leži u području od 0 — 9.

Sve operacije koje se mogu izvršavati na celobrojnim podacima mogu se izvršavati i na digit. Na primer:

```
n:=5;
n := n – 3;
n := 2 + n
```

su valjane (ako se izvršavaju u navedenom redu), jer je uvijek vrednost *n* u području digit.

Međutim u sledećem primeru neće biti tako. Uzmimo da je

```
n:=7;
n := n + 3
```

Sada je *n-u* dodeljena vrednost 7, pa je n + 3 = 10. Kada se ta vrednost pokuša dati n-u, računar javlja grešku.

Evo jednog primera u kojem se koriste podintervalni tipovi:

```
FUNCTION boja (n:0..6): farba;
    BEGIN
      CASE n OF
            0: boja=crvena;
            1: boja=narandzasta;
            2: boja=zuta;
            3: boja=zelena;
            4: boja=plava;
            5: boja=indigo;
            6: boja=ljubicasta
    END
END
```

Podintervalni tip 0..6 u deklaraciji formalnog parametra čini jasnim da poziv funkcije ima smisla jedino onda, ako su vrednosti njenog stvarnog parametra u području.od 0 do 6.

S druge strane i računar proverava (kad je funkcija pozvana) da li je stvarni parametar zaista u tom području, a ako nije, javlja grešku.

U ovom primeru podintervalni tip 0..6 bio je korišćen direktno u deklaraciji formalnog parametra a da nije prethodno imenovan u definiciji tipa. To je uvek moguće.

Tako se može umesto definisanja tipova „dan" i „digit" deklarisati d i n kao:

```
VAR
        d : dan;
        n:digit;
```

ili smo mogli deklarisati d i n direktno sa:

```
VAR
        d:(pon, utor, sred, cetv, pet, sub, ned);
        n: 0.. 9;
```

Na ovaj način postiže se lakše razumevanje identifikatora i može se naslutiti njihova primena u programu. Zatim, ovaj tip identifikatora može se upotrebljavati za deklaraciju formalnih parametara i lokalnih promenljivih u funkciji i proceduri.

9.3. Složeni tipovi podataka

9.3.1. Polja

Često je potrebno memorisati veliki broj podataka kao jednu celinu. U tu svrhu Pascal upotrebljava strukturirani tip podataka nazvan polje (engl. arrays). S obzirom na veličinu polje može biti jednodimenzionalno i višedimenzionalno.

Primer jednodimenzionalnog polja bi bio:

(1) 7
(2) 18
(3) 2
(4) 6
(5) 23

Ono se sastoji od pet članova, čije su vrednosti 7, 18, 2, 6, 23. Da bi polje deklarisali u Pascalu pišemo:

```
TYPE
    lista = ARRAY [1..5] OF integer;
```

U definiciji se javljaju dva tipa podataka.

1) U uglastim zagradama označen je indeks elemenata polja[l.. 5] taj tip zovemo *indeksni* tip (index type).

2) Drugi tip sledi iza reči OF i naziva se *komponentni* tip (component type), koji određuje tip komponenata polja.

Pretpostavimo da smo deklarisali neke promenljive tipa lista:

```
VAR
    a, b : lista;
```

Svaki od identifikatora a i b mora imati memorijsku lokaciju za 5 celobrojnih vrednosti:

a

7
18
2
6
23

Memorisane su samo komponente polja. Indeks vrednosti nije potreban, budući da su komponente memorane (smeštene) na takav način da je moguće adresirati komponentu na osnovu vrednosti indeksa. Memorijska lokacija nazvana a mora imati pet lokacija za svaku celobrojnu vrednost, tj.

a	
	7
	18
	2
	6
	23

Kad memorišemo listu u memoriju možemo koristiti vrednosti sa liste ili memorisati nove vrednosti na listu, a obično radimo oboje. Koristimo nekoliko načina za dodeljivanje memorijskih lokacija koje će držati pojedine komponente liste.

To radimo tako što stavljamo indeks komponente u zagrade posle imena lokacije koja drži listu. Tako a [1] imenuje memorijsku lokaciju, koja drži prvu komponentu vrednosti od a, a [2] određuje memorijsku lokaciju koja drži drugu komponentu itd. To možemo prikazati dijagramom:

a[1]	7
a[2]	18
a[3]	2
a[4]	6
a[5]	23

Promenljiva *a* čija je vrednost unutar polja poznata je kao *promenljiva polja* (engl. array variable). Promenljive 0[1], [2], [3], 0[4] i a[5], čije vrednosti su komponente polja, poznate su kao *indeksirane promenljive* (engl. indexed variables).

Indeksirane promenljive mogu biti upotrebljene u svim slučajevima, koji su dopušteni za promenljive što smo ih ranije prodiskutovali. Na primer, mogu se upotrebiti na levoj strani naredbi. Naredbe:

```
a[1] := 85;
a[2] := 71;
a[5] := 33
```

menjaju vrednost *a* kako sledi:

a	
	85
	18
	71
	6
	33

Indeksirane promenljive se mogu takode koristiti i u izazima. Na primer, ako se naredbe

```
i:= a[1] – a[2];
```

```
j:= 2*a[3];
k:=a[4]*a[5];
```

izvrše, promenljive i, j, k određuju vrednosti

promenljiva	vrednost
i	67
j	142
k	198

Takođe, naredba:

```
writeln (a [1], a [2], a [3], a [5])
```

stvara sledeći ispis:

```
85      18      71      33
```

Indeksni tip može biti bilo koji jednostavni tip podataka osim realnog ili celobrojnog. Vrlo česta upotreba indeksnih tipova su podintervalni tip celobrojnog tipa. Međutim, Booleov tip, znakovni i korisnički određeni skalarni tipovi mogu takode biti upotrebljeni kao indeksni tipovi. Na primer, uzmimo dva prethodna primera:

```
TYPE
   dan — (pon, utr, sred, cetv, pet, sub, ned);
   Sahfig = (pesak, lovac, skakac, top, kraljica, kralj);
```

Sledeća polja koriste te tipove kao indeksne tipove:

```
VAR
    sati:ARRAY [dan] OF real;
    vred: ARRAY [sahfig] OF integer;
```

Indeksirane promenljive koje adresiraju pojedine elemente polja sati su:

```
sati [pon]   sat [utor]   sati [sred]
sati [cetv]   sati [pet] sati [sub]   sati [ned]
```

To možemo koristiti u zapisu broja sati što ih radnik odradi svaki dan u sedmici. Na primer:

```
sati [pon] :=7.4
```

Drugi primer sa šahovskim figurama:

```
vred [pesak] :=1;    vred  [lovac]:=3;
vred [skakac]:=3;    vred  [top]:=5;
vred [kraljica] .= 9;      vred  [kralj] =1000
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (7)

Jednodimenzionalna polja

Evo jednog jednostavnog primera kako se polje može koristiti u obradi. Želimo, na primer, da saberemo 5 elemenata polja. To se može ostvariti primenom naredbe FOR na sledeći način:

```
sum := 0;
FOR  i= 1 TO 5 DO
sum:= sum + a [i];
```

što je ekvivalentno sledećim naredbama:

```
sum := 0;
sum := sum + a[1];
sum := sum + a[2];
sum := sum + a[3];
sum := sum + a[4];
sum := sum + a[5].
```

Primer ilustruje važno svojstvo polja. Indeks ne mora biti konstantan; on može biti bilo koji izraz, koji se može izračunati za vreme izvođenja programa. Na primer, ovu naredbu možemo koristiti više puta:

```
sum = sum + a [i]
```

i to svaki put s drugom vrednošću indeksa.

Evo, još jednog primera. Pretpostavimo, da želimo naći najmanju i najveću komponentu jednog polja. Neka su komponente polja temperaturni zapisi za svaki dan, a želimo naći najvišu i najnižu vrednost za dan. Znači:

```
VAR
    visoka, niska; integer;
    temp: ARRAY [1..24] OF integer;
```

Počećemo s postavljanjem obeju vrednosti, tj. visoke i niske na temp[l]. Zatim, dolazi vrednost temp [2] pa upoređujemo vrednosti visoku i nisku. Kad je nađena komponenta koja je veća od vrednosti visoka, ta vrednost postaje nova vrednost visoka. Kad je nađena manja vrednost od niska, ta komponenta postaje nova vrednost niska:

```
niska:=temp [1];

visoka:=temp [1];

FOR I:=2 TO 24 DO
    IF temp [i] > visoka THEN
    visoka:=temp [i]
    ELSE IF temp [i] < niska THEN
    niska:=temp [i]
```

Polja mogu biti upotrebljena kao stvarni parametri za procedure i funkcije. Na primer:

```
PROCEDURE vreme (VAR niska, visoka: integer;
```

```
                    VAR temp: ARRAY [1..24] OF integer);
VAR
      i: integer;
BEGIN
     niska :=temp [1];
     visoka =temp [1];
     FOR i =2 TO 24 DO
IF temp [i] > visoka THEN
      visoka =temp [i]
ELSE IF temp [i] < niska THEN
      niska:=temp [i]
END
```

Kao primer jedne funkcije sa poljem kao parametrom, napišimo funkciju koja će naći srednju vrednost jednog polja realnih brojeva:

```
FUNCTION sredvr (VARa: ARRAY [1 ..100] OF real;
                                broj: integer): real;
VAR
      i: integer;
      sum:real;
BEGIN
      sum =0.0;
      FOR i:=1 TO broj DO
      sum :=sum + a [i];
      sredvr := sum / broj
END
```

Mnogi operatori se ne mogu primeniti na polja, nego na njihove komponente, tj. na vrednosti indeksiranih promenljivih. Međutim, ako su a i b bilo koja dva polja sa promenljivama istog tipa (isti indeks, isti tip komponenata) može se upotrebiti deklaracija na sledeći način:

```
VAR
      a, b:ARRAY [1..5] OF integer;
```

Tada je naredba

a:=b

ekvivalentna sa pet sledećih naredbi:

```
a [1]:=b [1];
a [2]:=b [2];
a [3]:=b [3];
a [4]:=b [4];
a [5]:=b [5].
```

Ako su pre bile vrednosti

a	
	7
	36
	4

b	
	63
	5
	30

25	1
40	80

sada će biti

a		b	
63		63	
5		5	
30		30	
1		1	
80		80	

Komponente se mogu učitavati i ispisivati iz polja. Možemo koristiti FOR naredbu za učitavanje i ispisivanje komponenata polja toliko koliko nam treba. Na primer, naredba:

```
FOR i:=1 TO 5 DO
read (a [i])
```

učitava pet vrednosti od a[l] do a[5]. Ako su podaci:

4 36 27 88 9

tada je a [1] oznaka za 4, a [2] je 36 itd. Ispis se ostvaruje na isti način:

```
FOR i:= 1 TO 5 DO
    write (a [i])
```

računar će ispisati:

4 36 27 88 9

Ako želimo vertikalni ispis upotrebićemo:

```
FOR i:=1  TO 5 DO
    writeln (a [i])
```

i dobićemo izlaz:

```
 4
36
27
88
 9
```

Ako na više mesta u programu trebamo da učitavamo ili ispisujemo polje napisaćemo proceduru koja će to raditi:

```
PROCEDURE unos (VAR h:ARRAY [1..5] OF integer);

VAR
    i: integer;
BEGIN
    FOR i:=1 TO 5 DO
        read (h [i])
END;
```

```
PROCEDURE ispis (VAR h:ARRAY [1..5] OF integer);
VAR
    i: integer;
BEGIN
    FOR i:=1 TO 5 DO
        write (h [i]:4)
END.
```

Sa procedurom (naredbom)

unos (a)

možemo učitati vrednost polja, a ispis polja sa naredbom:

ispis (a)

U prethodnom odeljku definisan je tip „farba"

```
TYPE
    farba = (crvena, narandzasta, zuta, zelena, plava, indigo, ljubicasta)
```

i napisali smo funkciju boja za prevođenje rednog broja u području od 0 do 6 za odgovarajuće boje. To prevođenje može biti puno efikasnije upotrebom polja.

Deklarišimo polje **t** sa:

```
VAR
    t: ARRAY [0..6] OF farba;
```

njegove komponente su:

t [0] := crvena;
t [1] := narandzasta;
t [2]:=žuta;
t [3] := zelena;
t [4]:=plava;
t [5] := indigo;
t [6] :=ljubičasta

Sada, ako je vrednost **i** bilo koja celobrojna vrednost u nizu od 0 do 6, tada vrednost:

t [i]

odgovara boji prema rednom broju.

Boju možemo prikazati uz pomoć t na dva načina:

1) jedan broj 0 do 6;
2) vrednost tipa *farba.*

Standardna funkcija **ord** prevodi vrednosti od tipa *farba.* u tip 0.. 6. Polje **t** prevodi vrednosti iz tipa 0.. 6 u tip *farba.* Iz tog razloga ponekad se **t** naziva „tablica pretvaranja".

Pakovana polja

Bilo koji tip polja može biti označen rezervisanom rečju PACKED, kao na primer:

PACKED ARRAY [1..5] OF Boolean

Reč PACKED čini da se vrednosti polja smeštaju tako da se što bolje iskoristi memorijski prostor. Često se pakuje po više vrednosti u istu memorijsku lokaciju.

Tipovi polja za koja su vrlo povoljni pakiovani oblici su znakovni i Booleovi (char i Boolean). Često znakovne i Booleove vrednosti uzimaju samo mali deo memorijske lokacije, pa smeštanje više od jedne vrednosti po lokaciji štedi znatan prostor.

Takođe, polja podintervalnog tipa mogu biti efikasno pakovana, budući da računar zahteva manje memorije za smeštaj vrednosti tipa 0 . . 7, kažemo da smešta vrednosti tipa integer.

Pakovana polja mogu biti navedena kao i sva druga polja. Tako ako je **pc** deklarisano sa

```
VAR
    pc : PACKED ARRAY [1.. 5] OF char;
```

tada važe sledeće naredbe:

```
c:=pc [3];
pc [5]:='A'
```

Pascal jezik omogućuje rad sa dve procedure za polja i to *pack* i *unpack* za pakovanje i raspakivanje celog polja odjednom. Pogledajmo kako te procedure rade. Neka su promenljive polja *a* i *z* deklarisane sa:

```
VAR
    a:ARRAY [1..10] OF char;
    z:PACKED ARRAY [1..7] OF char;
```

Tada naredba:

```
pack (a, 3, z)
```

izaziva da vrednosti od a[3] do a[9] budu spakovane u z. Isti rezultat daje i naredba:

```
FOR i==1 TO 7 DO
    z [i]:=a [i] + 2
```

Naredba:

```
unpack (z, a, 3)
```

izaziva da vrednost od *z* bude raspakovana i njene komponente budu smeštene u a[3] do a[9]. Isti rezultat dobijamo i sa:

```
FOR i:=1 TO 7 DO
    a [i+2]:=z [i]
```

Pascal dozvoljava i upotrebu pakovanih polja znakova. Mogu se koristiti konstante niza, kao na primer ,,zdravo", predstavlja u Pascalu konstantu tipa pakovanog polja i može se označiti kao:

```
PACKED ARRAY [1..6] OF char
```

Pakovana polja znakova su jedini tip polja kojima su pridružene vrednosti. Za sve ostale tipove polja, vrednosti konstante moraju biti označene komponentama.

Pretpostavimo da imamo jedno polje deklarisano sa:

```
ime: PACKED ARRAY [1..10] OF char;
```

Možemo koristiti niz konstanti da označimo vrednosti ime. Ali moramo biti pažljivi - operator pridruživanja samo radi onda kada su polja na levoj i desnoj strani istog tipa. Zbog toga svaka konstanta niza koja označava vrednosti od ime mora biti sastavljena od 10 znakova. Ako vrednost, koju želimo da označimo

ima manje od 10 znakova, tada moramo ispuniti ostatak do 10 znakova dodavanjem praznina („blenkova”). Na primer:

```
ime:='Ivan       '
ime:='Jadranka   '
ime:='Davor      '
ime:='Dunja      '
ime :='Anastazija'
```

U Pascalu razmatramo dva specijalna slučaja pakovanih polja znakova.

Prvo: relacijski operatori mogu se primeniti na pakovana polja znakova istog tipa i možemo ih upoređivati po abecedi. Tako važe sledeći Booleovi izrazi i svi imaju vrednost ISTINA:

```
Ivan > Dunja

Davor < Marko

Ante < Aranka
```

Drugo: Kada se promenljiva tog tipa (znakovna pakovana polja) pojavi u WRITE ili WRITELN naredbi ispiše se celo polje. To ćemo koristiti kad moramo napisati naredbe kao:

```
writeln ('Iznos je:', iznos : 6:2)
```

Evo konstante niza:

```
'Iznos je:'
```

koja je polje tipa:

```
PACKED ARRAY [1..11] OF char
```

Taj niz mogao se deklarisati kao:

```
poruka: PACKED ARRAY [1..11] OF char;
```

i označimo:

```
poruka:='Iznos je:'
```

tada dve naredbe:

```
writeln (poruka, iznos: 6:2) i
writeln ('Iznos je:', iznos: 6:2)
```

daju isti ispis.

Višedimenzionalna polja

Do sada razmatrana polja su bila jednodimenzionalna, gledajući vertikalno ili horizontalno. Međutim, u Pascalu postoje i višedimenzionalna polja, na primer, dvodimenzionalno polje deklariše se kao:

```
t: ARRAY [1..4, 1..3] OF integer;
```

Polje t je polje sa 4 reda i 3 kolone (stupca) (označeni indeksi u zagradi), na primer:

t	1	2	3
1	35	24	11
2	18	82	90
3	63	71	51
4	20	44	7

Komponente polja određene su indeksom reda i kolone, na primer, 90 je u redu 2 kolone 3, t [2,3]. Uopšteno, može se dvodimenzionalno polje označiti t [i, j].

Ako je polje trodimenzionalno označavamo ga, na primer:

b: ARRAY [1..30, 1..5, 1..4] OF integer;

Možemo to lakše sebi predstaviti knjigom, prvi indeks određuje stranicu u knjizi, drugi određuje red na stranici, a treći određuje kolonu na toj stranici, na primer b [7, 2, 3].

Zapisi

Polja (arrays) imaju dve istaknute karakteristike:

1) *Sve komponente jednog polja su podaci istog tipa.*

2) *Indeks, koji se koristi za adresiranje pojedine komponente polja može biti izračunat za vreme izvršavanja programa.*

Zapis (engl. record) je suprotan polju u odnosu na obe karakteristike. Komponente zapisa (poznate kao fields) mogu biti od različitih tipova podataka i obično jesu. Identifikatori polja s kojima se komponente polja adresiraju ne mogu biti izračunati za vreme izvršavanja, već moraju biti odredeni u trenutku pisanja programa.

Zapis je skup podataka. Podaci u jednom zapisu mogu biti različitih tipova.

Tako se, na primer, mogu napraviti školski zapisi, zapisi o zaposlenosti, medicinski zapisi itd. Svaki od tih zapisa sadrži u sebi više različitih tipova podataka.

Sledeći primer ilustruje definisanje zapisa:

```
TYPE
    vreme = RECORD
                nebo: (oblacno, maloobl, vedro);
                padavine: (kisa, sneg, slana);
                niski,
                visoki: integer
END;
```

Ovde je vreme zapis. Identifikatori nebo, padavine, niski, visoki identifikatori su polja. Oni su potrebni da ukažu na komponente vrednosti tipa vreme. Na primer, pretpostavimo da su u, v i w promenljive tipa vreme:

```
VAR
    u, v, w: vreme;
```

Možemo smatrati, da je w ime područja memorije koja ima četiri lokacije, po jednu za svaku komponentu vrednosti tipa vreme:

oblačno
kiša
50
75

Možemo pojedinim komponentama dodeliti oznaku w - tako dobijemo oznaku polja (engl. field designator) i uz to imena lokacija u kojima su odgovarajuće komponente:

w. nebo	oblačno
w. oborine	kiša

w.niski	50
w.visoki	75

Oznake polja mogu biti korišćene isto tako kao i sve promenljive. Na primer:

w. nebo:=maloobl

mijenjajući sadržaj w. nebo u maloobl,

writeln (w. niski, w. visoki)

stvara ispis:

50 75

zatim:

i:=w. visoki -- w. niski označava vrednost 25.

Primetimo da je oznaka polja w.nebo slična indeksiranoj promenljivoj a[3].

Promenljiva zapisa w odgovara promenljivoj polja *a*. Oznaka polja nebo odgovara indeksu 3. Ali ne zaboravimo razliku: indeksirane promenljive u polju imaju isti tip podataka, a ovde kod zapisa uopšteno imaju različit tip podataka. Indeks indeksirane promenljive može biti izračunat tokom izvršavanja programa. Međutim, identifikator polja od jedne oznake polja mora biti specificiran u napisanom programu.

A sada evo nekoliko tipičnih primera za definiciju zapisa (record type):

```
zaposleni = RECORD
              matbroj: PACKED ARRAV [1 ..9] OF char;
              ime,
              adresa: PACKED ARRAY [1 ..40] char;
              otplata: real;
              bracst: (samac, ozenjen, razveden);
              godrod: integer
            END;
auto = RECORD
              napravljen,
              model,
              tip,
              boja: PACKED ARRAV [1 ..10] OF char;
              tezina: integer
            END;

knjiga = RECORD
              naslov: PACKED ARRAV [1 ..80] OF char;
              autor,
              izdavac,
              grad: PACKED ARRAY [1 ..20] OF char;
              godina: integer;
              cena: real
            END;
datum = RECORD
```

```
        mesec: (jan, feb, mar, apr, maj, jun, jul, aug, sep, oct, nov,
                dec);
        dan:      (pon, ut, sred, cetv, pet, sub, ned);
        danmes:   1..31;
        godina: integer
    END;
```

Moguće je da u okviru jednog zapisa koristimo drugi, ugnježdeni zapis (engl. nested records). Na primer, definišimo. zapis tipa *dan:*

```
dan = RECORD
      dt: datum;
      wx: vreme
  END;
```

Vrednost tipa *dan* će imati vrednosti polja tipa *dan* i *vreme.* Sve dotle dok su datum i vreme tipa zapisa, biće i zapis *dan* tipa zapisa. Na primer, deklarišimo:

```
danas:dan;
```

Tada su danas.dt i danas.wx promenljive tipa datum i vreme. Moramo dodeliti vrednostima danas.dt i danas.wx jedan dodatni identifikator:

```
danas. dt. mesec
danas. dt. dan
danas. dt. danmes
danas. dt. godina
danas. wx. nebo
danas. wx. padavine
danas. wx. nisko
danas. wx. visoko
```

Primer:

```
danas. dt. mesec:= april;
danas. dt. dan := petak;
danas. dt. danmes := 3;
danas. dt. godina := 1981;
danas. wx. nebo := oblacno;
danas. wx. padavine := kisa;
danas. wx. niski=55;
danas. wx. visoki=75;
```

Možemo smatrati da je *danas* deo memorije:

danas.dt.mesec	april
danas.dt.dan	petak
danas.dt.danmes	3
danas.dt.godina	1981

danas.wx.nebo	oblacno
danas.wx.padavine	kisa
danas.wx.niski	55
danas.wx.visoki	75

Zapisi i polja mogu biti *ugnježdeni (umreženi), na primer:*

```
osoba:RECORD
    ime:RECORD prvi, srednji,
    zadnji: PACKED ARRAY [1 ..20] OF char END;
adr:RECORD
    ulica,
    grad: PACKED ARRAV [1..40] OF char;
    drzava: PACKED ARRAY [1..2] OF char;
    kontin: PACKED ARRAY [1..5] OF char
    END
END;
```

Pisanje identifikatora je dosta glomazno, pogotovo kad su polja jednog zapisa takođe zapisi. Drugi problem se javlja kada potražimo različita polja istog zapisa, na primer

```
izvestaj: ARRAY [1..7] OF vreme;
```

Ako se zatraže polja zapisa izvestaj [i] računar će obraditi sadržaj lokacije u memoriji, zapisa, i to redom kako idu vrednosti *i*.

Na primer:

```
izvestaj [i].nebo
izvestaj [i].padavine
izvestaj [i].niski
izvestaj [ij.visoki
```

U navedenom primeru računar će prvi put dodeliti nebo, zatim padavine, pa niski i konačno visoki. Znači, trebalo bi na neki način računaru dati na znanje koja nam lokacija zapisa treba, pa da je on pamti sve dotle dok nam je potrebno.

Naredba WITH to omogućuje. Evo primera:

```
WITH w DO
    BEGIN
        nebo := oblacno;
        padavine:=kisa;
        niski =55;
        visoki =75
    END.
```

To je ekvivalentno:,

```
w.  nebo :=ob!acno;
w.  padavine := kisa;
w.  niski  :=55;
w.  visoki :=75
```

Naredba WITH primenjena na izvestaj [i] daće:

```
WITH  izvestaj [i] DO
      BEGIN
            nebo :=oblacno;
            padavine := kisa;
            niski :=55;
            visokh=75
      END
```

U naredbi WITH dozvoljeno je više od jedne promenljive zapisa, na primer:

```
WITH danas, dt, wx DO
  BEGIN
     mesec := april;
     dan := petak;
     danmeseca := 3;
     godina := 1981;
     nebo := oblacno;
     padavine := kisa;
     niski:=55;
     visoki :=75
  END
```

Oblast važenja identifikatora polja određena je zapisom u kome se nalazi. Ako se isti identifikator koristi u nekoliko umreženih zapisa, tada je njegova oblast zapis u kome se nalazi. To znači, da se isti identifikator može upotrebiti u različitim zapisima, na primer:

```
VAR
     u:RECORD
               a,
               b:integer
          END;
     v:RECORD
             a,
             b: real
          END
```

Nema poteškoća za razlikovanje identifikatora različitih zapisa:

```
u. a := 50;

u. b := 20;

v. a :=4.2;

v. b :=7.8
```

Celobrojne vrednosti promenljive zapisa označene su u poljima s oznakom *u;* realne s v. Iste naredbe mogu biti napisane kao:

```
WITH u DO
```

```
  BEGIN
        a := 50;
        b =20
  END;
WITH v DO
  BEGIN
        a:= 4.2;
        b:= 7.8
  END
```

Moguće je, takođe, i korišćenje istih identifikatora za imena promenljivih i identiflkatore polja:

```
VAR
    a, b: Boolean;
    u: RECORD
        a, b:integer
        END;
    v:RECORD
        a, b:real
        END;
```

Sa ovom deklaracijom sledeće naredbe su valjane:

```
a:= true;          b:= false;
u. a:= 50;         u. b := 20;
v. a:= 4.2;        v. b. := 7.8
```

Ove naredbe se mogu i ovako napisati:

```
 a:= true;          b:= false;
WITH u DO
BEGIN
        a:=50;   b = 20
END;
WITH v DO
BEGIN
        a:=4.2;   b:=7.8
 END
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (8)

Pokazivači

Promenljive koje su do sada spominjane bile su statičke; tj. deklarisane su u programu i egzistiraju dok se ne završi izvođenje bloka u kojem su deklarisane. Međutim, vrednosti tih promenljivih mogu biti promenjene nakon izvršenja programa. Nemamo naredbu koja bi mogla stvoriti novu statičku promenljivu ili uništiti staru.

Poteškoće sa statičnim promenljivama su u tome što je nemoguće odrediti odjednom koliko će nam biti potrebno memorije. Želimo da pišemo programe na takav način da oni mogu stvarati promenljive kad su one potrebne i ukloniti ih kad više nisu potrebne. Računar može opet iskoristiti „staru" memorijsku lokaciju za novo stvorenu promenljivu. Promenljive koje su stvorene i uklonjene kad je program izvršen nazivaju se dinamičke promenljive.

Dinamičke promenljive nisu se nalazile u deklaraciji promenljive, nema identifikatora za njih. Rutine koje stvaraju dinamičke promenljive odnose se na adresu memorijske lokacije, koja je dodeljena. Tako se onda adresa naziva **pokazivač** (engl. pointer). Za rad sa dinamičkim promenljivama potrebne su nam dve stvari: 1) tipovi podataka čije vrednosti su pokazivači, i 2) mehanizam za upućivanje na memorijsku lokaciju koja je označena vrednošću pokazivača.

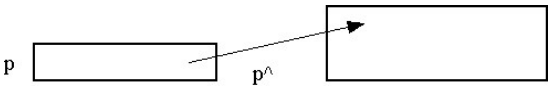
Označavanje tipa pokazivača je na primer:

```
^ integer
^ real.
```

Primer:

```
p: ^ integer
```

znači, da je vrednost *p* „pokazivač" u promenljivoj tipa integer. Dalje, oznaka *p* označava neku celobrojnu promenljivu, na primer, p^:=25, ili i:=p^ označava da će vrednost *i* postati vrednost *p*. Ovo se može slikovito pokazati, na primeru sa p i p^



Standardna procedura new stvara novu promenljivu. Na primer: new (p)

stvara novu celobrojnu (integer) promenljivu i dodeljuje joj adresu p. Naredba:

```
p^:=5
```

dodeljuje vrednost 5 novoj promenljivoj.

Ako nam određena promenljiva nije više potrebna u programu, pomoću poziva procedure *dispose* oslobađa se memorijska lokacija pridružena promenljivoj, a promenljiva više ne postoji.

Da bi označili da se promenljiva neće više menjati koristimo rezervisanu reč NIL.

```
p:=NIL
```

Izraz p^ je besmislen kad je vrednost p NIL.

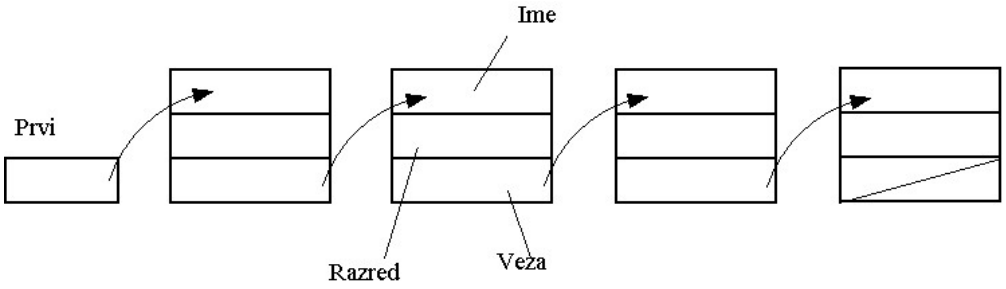
Pokazivači za promenljive tipa integer i real se retko koriste. Često se koriste za zapise. Pretpostavimo da želimo da napravimo popis razreda kao listu zapisa u memoriji računara; po jedan zapis za svakog učenika. Želimo da imamo fleksibilnu listu da bi mogli izbrisati zapise učenika koji napuštaju razred i dodavati nove zapise za nove učenike. Da bi to postigli koristimo povezanu listu. Evo primera:

```
ucenik:RECORD
        ime: PACKED ARRAY [f ..20] OF char ;
        razred: integer ;
        veza: ^ ucenik
        END;
```

Vrednost vezanog dela zapisa je pokazivač na sledeći zapis u listi. Za zadnji zapis u listi, vrednost veznog dela zapisa je NIL. Možemo koristiti promenljivu prvi deklarisanu kao:

```
prvi: ^ ucenik
```

da bi označili da pokazivač pokazuje (point to) na prvi zapis vezane liste, što pokazuje slika:

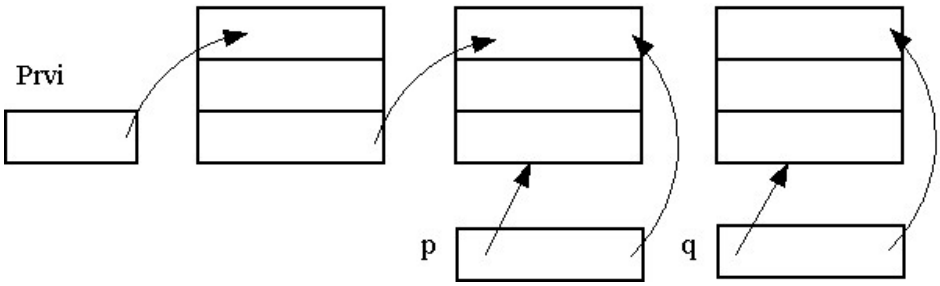


Zadnji lik u dijagramu ima dijagonalu što označava da zadnji zapis veznog dela zapisa ima vrednost NIL.
Evo sad jednog primera kojim će se napraviti „povezana lista" učeničkih zapisa:

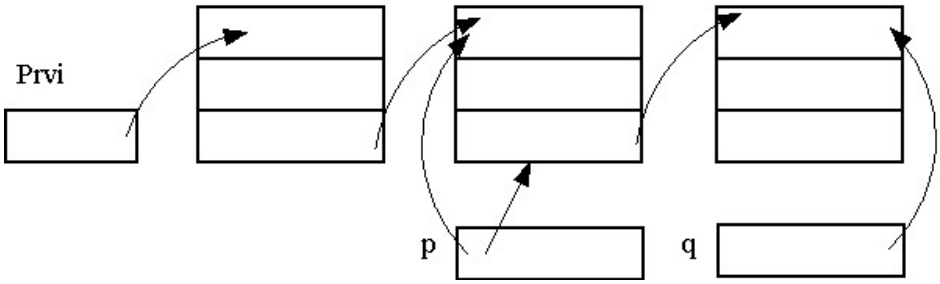
```
PROCEDURE razred (prvi: ^ ucenik);

VAR
      p, q:^ucenik; i:integer;
BEGIN
      new (p); {stvaranje naslova}
      prvi := p;
      WHILE NOT eof (input) DO
      BEGIN
         new(q)
         p^.veza:=q
         p:=q
         FOR i:=1 TO 20 DO
                  Read (p^.ime[i]);
         Readln
      END
      P^.veza:=NIL;
END.
```

Ilustracija za nekoliko naredbi iz ovog programa:

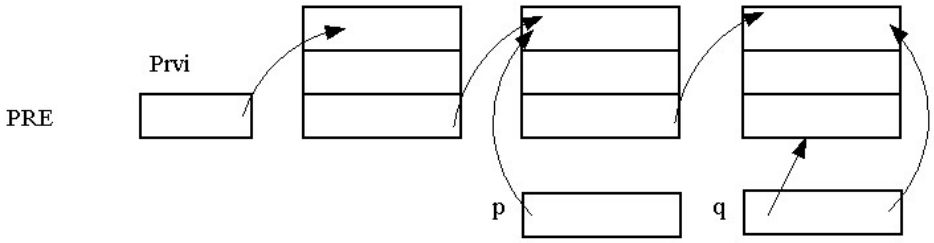
1) new(q) je nastajanje novog zapisa:

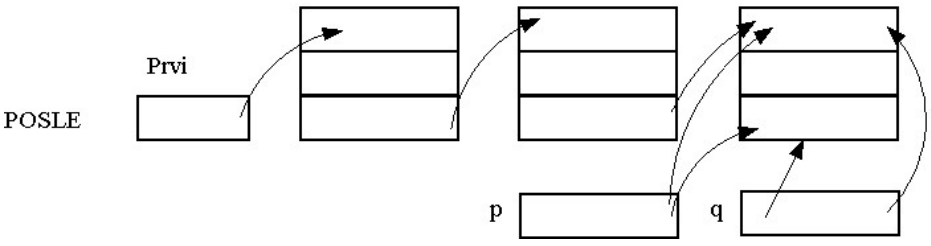


2) p^veza:=q puni se vezno polje (u zapisu na koji ukazuje p) sa sadržajem od q.



3) p:=q saržaj od p se prebacuje u q tako da se može ponovo koristiti.





Datoteke

Skup vrednosti smeštenih u pomoćnu memoriju poznat je pod nazivom datoteka. Postoji više vrsta datoteka: *sekvencijalne (engl.* sequential)*, direktne* (engl. random) i *indeksno-sekvencijalne.*

Čitanje sekvencijalne datoteke moguće je samo onim redom kojim su vrednosti bile smeštene u pomoćnu memoriju. Vrednosti u direktnoj datoteci (direct-access file) moguće je direktno pročitati, prema želji programera. Naime, direktan pristup je metoda kojom se može pročitati pojedinačni zapis iz datoteke (sa diska) bez potrebe da se pristupa bilo kom drugom zapisu. Indeksno-sekvencijalna datoteka ima brži pristup od sekvencijalne i poseduje *zapis ključeva (indeks)* i *zapis podataka,* pa kod čitanju imamo dve akcije, prvo čitamo zapis ključeva, pa onda traženi zapis

podataka.

Standardni Pascal ima samo sekvencijalnu datoteku, dok nove verzije Pascala uvode i druge tipove.

Računar ne manipuliše direktno sa vrednostima koje su smeštene u pomoćnu memoriju. Vrednosti moraju biti prenesene u glavnu (radnu) memoriju. Evo primera za datoteku celobrojnog tipa koja je na pomoćnoj memoriji (disku):

6 2 9 8 1 7 3 5 4

možemo zamisliti da je vrednost 9 vidljiva kroz prozor u glavnoj memoriji i trenutno dostupna za obradu u centralnom procesoru.

Datoteka se deklariše kao

```
TYPE
    podaci = FILE OF integer; VAR
    f:podaci;
```

Primetimo, kao kod polja, da je specificiran tip komponenata koje mogu biti smeštene u datoteku. S druge strane veličina datoteke nije zadata. Veličina varira u zavisnosti od broja vrednosti koje su stvarno smeštene u datoteku.

Ako je f datoteka promenljive, tada f^ predstavlja prozor kroz koji se može videti jedna vrednost. f^ tretiramo kao promenljivu. Vrednost dobijamo iz datoteke koristeći f^ u izrazu

i:=f^

i prvi korak u prenosu vrednosti u datoteku je označen f :

f^ := i

Obično f^ zovemo baferom, budući da bafer označava jedan deo glavne memorije koja čuva vrednost učitanu iz pomoćne memorije (ili jednu koja se upisuje u pomoćnu memoriju). Možemo nacrtati prikaz datoteke f i njenog bafera f kao:

6 2 9 8 1 7 3 5 4

f^

Datoteke čiji se identifikatori pojavljuju kao parametri u naslovu programa su spoljašnje datoteke. To su datoteke koje postoje pre izvođenja određenog programa. Datoteke čiji se identifikatori ne pojavljuju kao parametri u naredbi program su *lokalne datoteke* (engl. local files). Program koristi lokalne datoteke za privremeno smeštanje podataka.

U Pascalu postoji nekoliko standardnih procedura za učitavanje i pisanje na datoteku. To su procedure **reset**, **rewrite**, **get** i **put**

Proceduralna naredba: reset (f)

pozicionira prozor f^ na početak datoteke:

6 2 9 8 1 7 3 5 4

f^

Vrednost f^ je sada 6 - prva vrednost u datoteci. Možemo napisati:

i:=f^

gde je sada i vrednosti 6. Proceduralna naredba:

get (f)

pomera prozor na sledeću poziciju:

6 2 9 8 1 7 3 5 4

f^

Sada naredba i := f^ određuje vrednost 2. Izvršavajući ponovo:

get (f)

opet se prozor pomera za jedno mesto dalje:

6 2 9 8 1 7 3 5 4

f^

itd.

Izraz:

eof (f)

je LAŽ sve dotle dok se prozor datoteke pomera po komponentama datoteke. Kad dođe na kraj datoteke vrednost eof (f) postaje ISTINA. Za naš primer:

6 2 9 8 1 7 3 5 4 eof(f) je LAŽ

6 2 9 8 1 7 3 5 4 □ eof(f) je ISTINA

Sledeći primeri daju opšti oblik naredbi za učitavanje i obradu vrednosti u datoteci :

```
reset (f);
WHILE NOT eof (f) DO
BEGIN

    {naredbe koje koriste vrednosti f^}
  get (f)
END.
```

Na primer, želimo da saberemo vrednosti u datoteci celobrojnih podataka:

```
sum :=0
 reset (f);
WHILE  NOT eof (f) DO
    BEGIN
        sum :=sum + f^;
        get (f)
     END.
```

U datoteku možemo upisati novu vrednost samo onda kada je vrednost eof ISTINA - tada se prozor pomakao iza zadnje vrednosti. Upisivanje nove vrednosti u datoteku obeležavamo s f^ (novu vrednost) i izvršavamo put *(f)*. Na primer, želimo dodati vrednost 5

f^:= 5

daće:

6 2 9 8 1 7 3 5 4 5

f^

Izvođenjem: put (f) pomera se prozor napred za jednu poziciju, znači:

```
6    2    9    8    1    7    3    5    4    5
                                                f^
```

Mada možemo dodavati vrednosti na kraj već postojeće datoteke, uobičajeno je da se počne sa praznom datotekom. Naredba:

```
rewrite (f)
```

označava jednu praznu datoteku za datoteku promenljive f. Posle te naredbe možemo videti datoteku kao

```
f^
```

Vrednosti eof (f) je ISTINA pa možemo početi sa upisivanjem vrednosti u datoteku. Na primer, ako se izvrši:

```
f^ =8;
put (f);
f^ := 6;
put(f);
f^:= 9;
put(f);
```

datoteka će izgledati:

```
8        6        9
                          f^
```

Kao sledeći primer upisivanja u datoteke, sledeće naredbe smeštaju vrednosti 1 do 100 u datoteku:

```
rewrite (f);
FOR i:=1 TO 100 DO
   BEGIN
      f^:=i;
      put(f)
   END.
```

Postoje često slučajevi kada je potrebno da u jednoj datoteci promenimo podatke, na primer, mesečno. Uzmimo primer datoteke za izveštaje kupaca.

Datoteku za izveštaje kupaca zovemo glavna datoteka, a datoteku koja zadržava informacije o trgovinsko-prodajnim akcijama datoteka poslovna. Nama su potrebni sadržaji poslovne datoteke, da bi promenili vlasnikovu datoteku.

Specificirajmo na primer da se vlasnikova datoteka od prošlog meseca naziva „stara" datoteka koja, uz poslovnu, čini „novu" vlasnikovu datoteku, a u sledećem mesecu će biti „stara". Pretpostavimo, da zapisi u vlasnikovoj datoteci imaju sledeći oblik:

```
glavna = RECORD
            kljuc: PACKED ARRAY [1..9] OF char;
            stalnipod: PACKED ARRAY [1..80] OF char;
            prompod: PACKED ARRAY [1..80] OF char
            END;
```

Identitet osobe (ili drugi entitet) određuje polje kljuc. Dalje, uzeli smo samo još dva polja: stalnih podataka,

koji se neće menjati i drugo koje će se mesečno menjati. U praksi imaćemo naravno više polja, koja će se menjati i koja se neće menjati.

Postoje tri vrste akcija za rad s datotekama: možemo *dodavati novi zapis* datoteci, možemo *menjati zapis* koji je u datoteci ili možemo *izbrisati zapis* koji je u datoteci. Pokažimo to:

tiprada = (dodati, menjati, brisati)

Neka zapisi u poslovnoj datoteci imaju sledeći oblik:

```
rad = RECORD
            kljuc: PACKED ARRAY [1 ..9] OF char; vrsta: tiprada;
      stalnipod: PACKED ARRAY [1..80] OF char;
      prompod: PACKED ARRAY [1 ..80] OF char
      END.
```

Vrsta akcije određuje koje će se od naznačenih polja koristiti. Ako dodajemo novi zapis, koriste se oba *stalnipod* i *prompod.* Ako menjamo već postojeći zapis, koristimo samo *prompod.* Ako brišemo zapis, ne koristimo nijedno od ovih polja.

Pretpostavimo sada da imamo tri datoteke: staru, novu i poslovnu:

stara, nova: FILE OF glavna; poslovna: FILE OF rad;

Zapisi u staroj i poslovnoj datoteci moraju biti poređani u „rastućem" redu u skladu s njihovim kljuc-poljima. Program uređuje novu datoteku čiji su zapisi u „rastućem" poretku u skladu s njihovim kljuc-poljima.

Ukratko program za menjanje datoteke sastoji se od:

```
reset (stara);
reset (poslovna);
rewrite (nova);
krajdat: = eof (stara) OR eof (poslovna);
WHILE NOT krajdat DO
      uporediobradi; WHILE NOT eof (stara) DO

      iducaglavna;

WHILE NOT eof (poslovna) DO
       dodavanje
```

Pogledajmo proceduru za upoređivanje koja određuje proces koji će izvršiti:

```
PROCEDURE uporediobradi; BEGIN
      IF stara ^. kljuc < poslovna ^. kljuc THEN
                  iducanova
      ELSE IF stara ^ . kljuc = poslovna ^. kljuc THEN
                  promilibrisi
      ELSE
                  dodavanje
END.
```

Ova procedura upoređuje ključeve zapisa tekućih datoteka stara i poslovna, pa su moguća tri slučaja:

1. Ključ tekućeg zapisa iz stare datoteke je manji od ključa tekućeg zapisa iz poslovne datoteke. Budući da su datoteke u „rastućem" poretku, nijedan tekući poslovni zapis, niti bilo koji sledeći, ne može imati isti ključ kao tekući stari zapis. Zato nema akcije koja se može izvršiti na tekućem starom zapisu. Zato se poziva iduća (nova) da upiše

tekući stari zapis u novu datoteku i dohvati sledeći stari zapis.

2. Ključevi tekućeg zapisa stare i poslovne datoteke su isti. Imamo slaganje poslovnog zapisa s odgovarajućim starim zapisom. Jedine moguće operacije su menjanje i brisanje; ne možemo dodati zapis s istim ključem kao što je već postojeći zapis u datoteci. Pozivamo proceduru *promilibrisi:*

3. Ključ tekućeg, starog zapisa je veći od ključa tekućeg poslovnog zapisa. Akcija mora odrediti novi zapis koji će se dodati u novu datoteku; pozivamo proceduru dodavanje.

```
PROCEDURE iducaglavna;

BEGIN
      iducaglavna ^ :=stara ^;
      put (iducaglavna);
      get (stara);
      krajdat := eof (stara)
END
```

Procedura *promilibrsii* menja ili briše tekući stari zapis. Ako je poslovni zapis pogrešno zahtevan poziva se jedna dodatna, procedura *greska,* da bi informisala korisnika o pogrešnoj akciji i da da novi poslovni zapis:

```
PROCEDURE promilibrisi;
BEGIN
CASE poslovna ^.vrsta OF
   dodavanje:greska;
   zam:BEGIN
            stara ^. prompod := poslovna^.
           prompod;
           get (poslovna);
           krajdat:=eof (poslovna)
           END;
           brisi BEGIN
           get (stara);
           get (poslovna);
           krajdat:=eof (stara) OR eof (poslovna)
           END
   END {CASE}
END.
```

Procedura dodavanje koristi podatke u poslovnom zapisu da doda novi zapis u novu datoteku:

```
PROCEDURE dodavanje;

BEGIN
           IF poslovna ^.vrsta < > dodavanje THEN
                    greska
      ELSE
      BEGIN
      nova ^.kljuc := poslovna ^. kljuc;
      nova ^. stalnipod := poslovna ^. stalnipod;
      nova ^. prompod := poslovna ^. prompod;
```

```
        put (nova);
        get (poslovna);
        krajdat:=eof (poslovna)
        END
    END.
```

Procedura *greska* opominje na jednu pogrešnu akciju i daje novi poslovni zapis:

```
PROCEDURE greska;
BEGIN
        writeln ('POGRESNA AKCIJA', poslovna ^.kljuc);
        get (poslovna); krajdat := eof (poslovna)
 END.
```

Datoteke znakova

Datoteke znakova se obično dele u redove. Tekst može biti definisan:

```
text=FILE OF char;
```

Standardne datoteke *input* i *output* su tipa *text.*

Kraj svakog reda označava se sa posebnim upravljačkim znakom (najčešće eoln znak). Ovaj separator redova možemo u primeru predstaviti pomoću vertikalne crte.

Neka je tekstualna datoteka sastavljena od tri reda;

```
37245.11        28.23
12345.61        10.77
23856.21        73.45
```

to možemo predstaviti kao:

```
37245.11  28.23  |  23856.21  7345  |  12345.61  10.77
```

U programima se za *detekciju kraja reda* koristi funkcija eoln (end of line). Izraz eoln je ISTINA kada je prozor datoteke f iznad linijskog separatora, a LAŽ u suprotnom.

Standardne procedure

Budući da se tekstualne datoteke vrlo često koriste, Pascal daje dodatne standardne procedure za rad s njima. To su **read, readln, write, writeln** procedure. Neka je c znak promenljive a f datoteka promenljive. Tada je:

```
read (f, c)
```

ekvivalentno s

```
e:=f^;
get (f)
```

i

```
write (f, c)
```

ekvivalentno s

```
f^ := c;
put (f)
```

Procedura readln pomera prozor datoteke na znak koji sledi idući linijski separator. Tako:

```
readln (f)
```

je ekvivalentno s

```
WHILE NOT eoln (f) DO
    get (f);
    get (f)
```

Skupovi

Skup je grupa vrednosti. Na primer, skup sastavljen od vrednosti 4, 1, 5, 8 obeležava se u Pascalu:

```
[4, 1, 5, 8]
```

Vrednosti koje pripadaju skupu nazivaju se *elementi.* Tip skupa može biti deklarisan:

```
slovoskup = SET OF 'A'.. 'Z'
```

Tip ' A'.. 'Z' nazivamo osnovni tip i to je tip elemenata skupa. Pretpostavimo da smo deklarisali:

```
s: slovoskup;
```

tada su neke od mogućih vrednosti s:

```
['A']   ['A', 'C']   [T, 'N', 'F', 'R']   [ ]
```

Primetimo da je [ ] prazan skup! Razmotrimo sledeće definicije

```
osnboje = (crvena, zuta, plava); osnskup = SET OF osnboje;
```

Moguće vrednost tipa *osnskup* su:

```
[ ]   [crvena]   [zuta]   [plava]   [crvena, zuta] [crvena, plava]   [zuta, plava]   [crvena, zuta, plava]
```

Promenljive tipa *osnskup* moraju imati jednu od tih osam vrednosti.

Osnovni tip mora biti jednostavan tip osim realnog. Svaki računar ima određena ograničenja za vrednosti tipa skup, a to su:

1. Ograničenje broja elemenata skupa.

2. Samo celobrojni elementi koji pripadaju određenom podintervalnom tipu mogu biti elementi skupa. Na primer, ako je podintervalni tip 0.. 58, tada su 1.. 5, 10.. 50, i 0.. 58 dozvoljeni osnovni tipovi, ali - 1.. 3 i 98 .. 100 nisu. Takođe je dozvoljen skup [25, 40], ali [-2,5] i [59] nisu.

3. Ako znak nije dozvoljen kao osnovni tip, tada samo znakovne vrednosti koje pripadaju nekim uzetim podskupovima znaka mogu biti elementi skupova.

Operacije na skupovima

Pretpostavimo da promenljive *i* i *j* imaju vrednosti 8 i 5. Tada „graditelj" skupa (engl. set constructor)

```
[i, j, i+j, i-j, i *j, i div j]
```

ima vrednosti

```
8, 5, 13, 3, 40, 1.
```

Lista elemenata može sadržavati podskupove isto tako kao i individualne vrednosti. Tako

```
[1..5]
```

je ekvivalentno

[1, 2, 3, 4, 5]

i

['A', 'C', .. 'F','L','W'..'Z']

je ekvivalentno

['A', 'C', 'D', 'E', 'F, 'L', 'W, 'X', 'Y', *'Z'*]

Imamo tri operacije koje možemo primeniti na skupove da bi generisali nove skupove. To su unija, presek i razlika, koji se u Pascalu obeležavaju operatorima + , * , - . Oni su definisani, kad su 5 i *t* promenljive istog tipa skupa:

s + t	Unija.s i t, sastoji se od elemenata koji pripadaju s, t, ili i s i t.
s x t	Presek s i t, sastoji se od elemenata koji pripadaju i jednom i drugom (s i t).
s - t	Razlika s i t, sastoji se od elemenata koji pripadaju s a ne pripadaju t.

Relacijski operatori mogu se primeniti na skupove istog tipa skupa:

= < >

< = > =

IN

s = t	ISTINA ako je s jednako t; (što znači da s i t imaju iste elemente).
s < > t	ISTINA ako s nije jednak t; (s i t nemaju iste elemente).
s < = t	ISTINA ako je s podskup od t; (ako je svaki element od s takođe i element od t).
s > = t	ISTINA ako je s nadskup od t; (ako je svaki element od t takođe i element od s).
i IN s	ISTINA ako je / jedan element od s.

Skupovi se često upotrebljavaju kao zamena za Booleove operatore. Na primer:

(c = ' + ') OR (c = '-') OR (c = 'x') OR (c='/')

možemo napisati:

c IN ['+', '-', '*', '/'] ili drugi primer:

('A' < = c) **AND** (c < = 'Z') ekvivalentan izraz je ostvaren upotrebom skupova:

c IN ['A'..'Z']

U kompleksnim programima može biti više uslova čija ISTINA ili LAŽ može uticati na izvođenje programa. Određeni delovi programa mogu ,,zapisati" te pojedinačne uslove kad su ISTINA ili LAŽ, što će uticati na daljnji tok izvršavanja programa.

Pretpostavimo da imamo zaseban program sa šest uslova a, b, c, d, e i f. Definišimo tip podataka čije su vrednosti tih šest uslova:

uslov = (a, b, c, d, e, f);

Koristićemo skup t za zapis uslova koji su istiniti: t:SET OF uslov;

Na početku programa označimo:

t:=[ ]

ako je dalje uslov a istinit biće: t:=t + [a]

Pretpostavimo dalje da je u drugom delu programa uslov a lažan. Tada se mora a udaljiti iz skupa:

t:=t-[a]

Ako je vrednost t[a, c, f] pre izvršenja te naredbe, kasnije će biti [c, f].

Sada pretpostavimo da želimo da se izvrši Sl naredba, pod ovim uslovima:

a i c su ISTINA

f je LAŽ

Prvo moramo eliminisati b i d:

t * [a, c, f]

zatim:

t * [a, c, f] = [a, c]

Sledeća naredba kontroliše izvršenje Sl:

IF t * [a, c, f] = [a, c] THEN

Sl

Evo sada jednog programa koji koristi i skupove uz ostale već poznate tipove podataka. Skup čine znakovi u programu. Pokušajte da analizirate program!

```
PROGRAM ispitivanje (input, output); VAR
   c:char; special: SET OF char;
   trazenje: Boolean; BEGIN
       special :=['+','-', '*', '/', '=', '<', '>', '(', ')',':',';',',','[',']';
      trazenje = true;
      read (c);
      WHILE trazenje DO
      BEGIN
      WHILE c = '' DO {preskok praznina}
                read (c);
                IF c IN ['A'..'Z'] THEN {rezervisana rec ili identifikator)
                BEGIN
                REPEAT
                write (c);
                read (c)
                UNTIL NOT (c IN ['A'.-'Z', '0'..'9']);
                writeln
                END
            ELSE IF c IN ['0'..'9'] THEN {celi}
            BEGIN
            REPEAT
```

```
                write (c);
                read (c)
            UNTIL NOT (c IN ['O'..'9']);
            writeln
            END
        ELSE IF c IN special THEN {znak}
        BEGIN
        REPEAT
        write (c);
        read (c)
            UNTIL NOT (c IN special);
            writeln
            END
        ELSE IF c = '.' THEN
        BEGIN
        writeln (c);
        trazenje := false
        END
        ELSE {'otpaci'}
        BEGIN
            writeln ('Strani znak:', c);
            read (c)
            END
    END {while}
END. {ispitivanja}
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (9)

Brojni sistemi

Paralelno sa razvojem pisma, razvijali su se i znakovi za prikaz brojeva. Potreba stvaranja naziva i znakova za veće brojeve bila je prva okolnost koja je prisilila čoveka na traženje sistemskih postupaka. Na primer, brojevi 1, 2, 3, 4 mogli bi se označavati sa I, II, III, IIII, ali je ovakav sistem nemoguće zadržati za velike brojeve. Zbog toga razvijeni su brojni sistemi, tj. načini označavanja brojeva nizovima znakova - cifri.

Postoje različiti sistemi, a danas je u upotrebi tzv. aditivno-multiplikativni sistem koji su u Evropu preneli Arapi, a razvijen je u Indiji. U tom sistemu možemo po volji veliki broj napisati pomoću svega nekoliko različitih cifara (najmanje dve). Svaka cifra tog sistema ima svoju brojnu i mesnu (pozicionu) vrednost. Takav sistem se zato naziva i težinski ili pozicioni. Krajnje leva cifra ima najveću težinu, a krajnje desna cifra najmanju. Zbog toga se krajnje leva cifra zove najznačajnijom cifrom, a krajnje desna cifra najmanje značajnom cifrom. Broj upotrebljenih cifara određuje osnovu (bazu) sistema. Opšti prikaz broja **R** u težinskom sistemu je:

$$\mathbf{R = d_n d_{n-1} ... d_2 d_1 d_0 . d_{-1} d_{-2} ... d_{-(m-1)} d_{-m} =}$$

$$\mathbf{= d_n B_n + d_{n-1} B_{n-1} + ... + d_2 B_2 + d_1 B_1 + d_0 B_0 + d_{-1} B_{-1} + d_{-2} B_{-2} + ... + d_{-(m-1)} B_{-(m-1)} d_{-m} B_{-m}}$$

gde je $\mathbf{d_i}$ odgovarajuća cifra ($\mathbf{d_i}$. **(B-1)**), a **B** osnova sistema.

Danas je uobičajen težinski sistem sa osnovom 10. Razlog je anatomske prirode: čovek ima deset prstiju koje je koristio kao pomoćno sredstvo prilikom računanja. Zapravo, sistem sa osnovom 12 bio bi praktičniji (deljivost bez ostatka sa 2, 3, 4, 6), ali bi prelaz na njega izazvao velike probleme. Zanimljivo je i to da su Vavilonci upotrebljavali sistem sa osnovom 60, čije tragove nalazimo kod mera za ugao i vreme. Računari koriste binarni brojni sistem, tj. sistem sa osnovom 2. Takav sistem je najjednostavniji jer zahteva svega dve cifre (0 i 1), a to znači i jednostavne elektronske sklopove za prikaz tih cifara. U računarstvu se upotrebljavaju i sistemi sa osnovom 8 i 16, prvenstveno zbog lakog pretvaranja između njih i binarnog sistema, pa se ponekad koriste za skraćeni prikaz binarnih brojeva.

Dekadni sistem

Dekadni sistem ima osnovu 10 i koristi sledeće cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Svaka cifra dekadnog broja ima svoju težinu koja je stepen broja 10 (10^i). Pritom je eksponent (i) ceo broj, a njegova vrednost određena je položajem cifre u broju.

Primer

$43 = 4*10^1 + 3*10^0$

$444 = 4*10^2 + 4*10^1 + 4*10^0$

$12.5 = 1*10^1 + 2*10^0 + 5*10^{-1}$

Može nas zanimati koliko različitih brojeva možemo da prikažemo brojem koji ima n cifara (na primer, kod kalkulatora i računara n je ograničeno). Tada govorimo o **kapacitetu (K) broja sa n cifara**: $\mathbf{K = B^n}$, gde je **B** osnova brojnog sistema. Dakle, kapacitet je broj koji nam kaže koliko različitih brojeva možemo prikazati sa n cifara, ako je zadata osnova sistema. Najveći broj **M** koji možemo prikazati sa n cifara je za jedan manji od kapaciteta, tj.: $\mathbf{M = B^n - 1 = K - 1}$.

Primer
Sa 4 cifre u dekadnom sistemu možemo prikazati $10^4 = 10000$ različitih brojeva, a najveći je 10000 - 1 = 9999.

Binarni sistem

Cifre binarnog sistema su **0** i **1**, a njegova osnova **B = 2**. Binarna cifra zove se **bit** (skraćeno od engleskog izraza **Bi**nary digi**t**). Ukupni kapacitet **K** binarnog broja sa n bita je $\mathbf{K = 2^n}$, a najveći broj M koji možemo prikazati je $\mathbf{M = 2^n - 1 = K - 1}$.

Primer
Sa 8 bita možemo prikazati 2^8=256 različitih brojeva, najveći je 255 (11111111_2), a najmanji je 0 (00000000_2).

Pretvaranje binarnog broja u dekadni

Kao i kod dekadnog sistema radi se o težinskom sistemu, dakle važi:

$101101_2 = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 32 + 0 + 8 + 4 + 0 + 1 = 45_{10}$

Na taj način možemo bilo koji binarni broj pretvoriti u dekadni. Kod dekadnog broja obično ne označavamo osnovu sistema, ali, ako se radi o nekoj drugoj osnovi, moramo je označiti kao u prethodnom primeru.

Primer
$11001_2 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 25_{10}$

Primer
$1.111_2 = 1*2^0 + 1*2^{-1} + 1*2^{-2} + 1*2^{-3} = 1 + 0.5 + 0.25 + 0.125 = 1.875_{10}$

Primer
Može li broj 1020 pripadati binarnom sistemu? Ne može. Zašto? Za svaku cifru d mora da važi d <= (B-1). Budući da je B = 2, za cifru 2 broja 1020 ne važi 2 <= 1.

Pretvaranje dekadnog broja u binarni

Pretvaranje prirodnog dekadnog broja u binarni može se opisati sledećim postupkom:
1. Podeli dekadni broj sa 2.
2. Zapiši ostatak deljenja (0 ili 1).
3. Dobijeni količnik (celobrojni deo) podeli sa 2.
4. Zapiši ostatak deljenja.
5. Ako količnik nije 0 vrati se na tačku 3.

Ostaci deljenja koje smo zapisivali predstavljaju traženi binarni broj koji treba čitati obrnuto, tj. zadnja dobijena cifra je najznačajnija cifra, a prva dobijena cifra je najmanje značajna cifra.

Primer
Pretvoriti dekadni broj 43 u binarni.

```
43 : 2 = 21 -- ostatak 1
21 : 2 = 10 -- ostatak 1
```

```
10 : 2 = 5 -- ostatak 0
5 : 2 = 2 -- ostatak 1
2 : 2 = 1 -- ostatak 0
1 : 2 = 0 -- ostatak 1
```

Prema tome, dobije se $43_{10} = 101011_2$

Primer
Napravite tablicu dekadnih brojeva od 0 do 18 i njihovih binarnih ekvivalenata. Uočite kako se broji u binarnom sistemu.

Dekadni brojevi manji od 1 pretvaraju se u binarne brojeve primenom sledećeg postupka:
1. Pomnoži dekadni broj sa 2.
2. Ako je dobijeni broj veći od 1 iza tačke u binarnom broju piše se 1.
3. Ako je dobijeni broj manji od 1 iza tačke u binarnom broju piše se 0. Postupak se ponavlja sa delom umnoška iza decimalne tačke s time da se 0 ili 1 dopisuje već napisanim brojevima (sa desne strane).

Primer
Pretvoriti dekadni broj 0.625 u binarni.

0.625*2 = 1.250 -- beležimo 1
0.250*2 = 0.500 -- beležimo 0
0.500*2 = 1.000 -- beležimo 1

$0.625_{10} = 0.101_2$

Ispravnost pretvaranja može se proveriti tako da se dobijeni binarni broj ponovo pretvori u dekadni:

$0.101_2 = 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0.625_{10}$

Ako imamo realni dekadni broj veći od 1, možemo ga pretvoriti u binarni broj tako da pretvorimo posebno celobrojni deo, a posebno deo iza decimalne tačke, a dobijene binarne brojeve saberemo.

Primer
Pretvoriti 43.625 u binarni broj.

Od pre imamo: $43_{10} = 101011_2$ i $0.625_{10} = 0.101_2$. Dakle, $43.625_{10} = 101011.101_2$.

Sabiranje binarnih brojeva

Sabiranje binarnih brojeva može se naučiti imajući u vidu sledeća pravila za sabiranje dva bita:

0+0=0
0+1=1
1+0=1
1+1=0 i prenos 1

Prenos se prenosi u sledeću kolonu.

Primer

```
001101 kontrola: 13
```

```
+100101 +37
------ --
110010 50
```

Primer

```
1011011 91
+1011010 + 90
-------- ----
10110101 181
```

Dekadno	**Binarno**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010

Oktalni sistem

Oktalni sistem ima osnovu **8** i koristi sledeće cifre: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**. Kapacitet (**K**) **n** oktalnih cifara je $\mathbf{K = 8^n}$, a najveći broj (**M**) koji možemo prikazati sa **n** cifara je $\mathbf{M = 8^n - 1 = K - 1}$. U informatici se oktalni sistem koristi za skraćeni prikaz binarnih brojeva.

Primer
Sa dve oktalne cifre možemo prikazati $8^2 = 64$ različita broja, a najveći je $8^2 - 1 = 63$ (77_8).

Pretvaranje oktalnog broja u dekadni

Pretvaranje se radi na sličan način kao i u slučaju binarnog broja, što pokazuje sledeći primer.

Primer
Pretvoriti oktalne brojeve 37, 142 i 364 u dekadne.
$37_8 = 3*8^1 + 7*8^0 = 24 + 7 = 31_{10}$
$142_8 = 1*8^2 + 4*8^1 + 2*8^0 = 64 + 32 + 2 = 98_{10}$
$364_8 = 3*8^2 + 6*8^1 + 4*8^0 = 192 + 48 + 4 = 244_{10}$

Pretvaranje oktalnog broja u binarni i binarnog u oktalni

Ovo je vrlo jednostavno pretvaranje i zbog toga se oktalni sistem koristi za skraćeni prikaz binarnih brojeva. Svaku oktalnu cifru treba prikazati sa tri bita i obrnuto.

Primer
Pretvoriti oktalni broj 76543 u binarni.

```
7 6 5 4 3
111 110 101 100 011
```

Dakle, $76543_8 = 111110101100011_2$.

Primer
Pretvoriti binarni broj 1101101111 oktalni.
Potrebno je rastaviti binarni broj u grupe od po tri bita počevši sa desne strane. Ako na kraju nedostaju cifre, treba dodati jednu ili dve nule sa leve strane. Svaku grupu od tri bita treba zameniti jednom oktalnom cifrom.

$1101101111_2 = 001\ 101\ 101\ 111_2 = 1557_8$

Pretvaranje dekadnog broja u oktalni

Primenjuje se istii algoritam kao i u slučaju dekadno-binarnog pretvaranja, s razlikom da se deli sa 8.

Primer
Pretvoriti dekadni broj 127 u oktalni.

```
127 : 8 = 15 -- ostaje 7
15 : 8 = 1 -- ostaje 7
1 : 8 = 0 -- ostaje 1
```

Dakle, $127_{10} = 177_8$.

Heksadecimalni sistem

Heksadecimalni sistem ima osnovu **16** i koristi cifre **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **A**, **B**, **C**, **D**, **E**, **F**. Vidimo da heksadecimalni sistem koristi slova A - F za dekadne ekvivalente 10 - 15. Sa **n** heksadecimalnih cifri možemo da prikažemo $\mathbf{K = 16^n}$ različitih brojeva, a najveći je $\mathbf{M = 16^n - 1 = K - 1}$. U informatici se heksadecimalnim sistemom služimo za skraćeni prikaz binarnih brojeva.

Pretvaranje heksadecimalnog broja u dekadni

Pretvaranje se vrši kao i kod binarnog i oktalnog sistema, što ilustruje sledeći primer.

Primer
Pretvoriti heksadecimalne brojeve 23, 3B i 1AF u dekadne.

$23_{16} = 2*16^1 + 3*16^0 = 32 + 3 = 35_{10}$
$3B_{16} = 3*16^1 + B*16^0 = 3*16^1 + 11*16^0 = 48 + 11 = 59_{10}$
$1AF_{16} = 1*16^2 + A*16^1 + F*16^0 = 1*16^2 + 10*16^1 + 15*16^0 = 256 + 160 + 15 = 431_{10}$

Pretvaranje dekadnog broja u heksadecimalni

Pretvaranje celog dekadnog broja u heksadecimalni vrši se deljenjem sa 16, slično kao i kod pretvaranja u binarni i oktalni sistem.

Primer
Pretvoriti dekadni broj 127 u heksadecimalni.

```
127 : 16 = 7 -- ostaje 15 (F)
7 : 16 = 0 -- ostaje 7
```

Dakle, $127_{10} = 7F_{16}$.

Pretvaranje heksadecimalnog broja u binarni i obratno

Pretvaranje je slično kao u slučaju oktalnog broja, ali se radi sa grupom od četiri bita. Svakoj heksadecimalnoj cifri odgovaraju četiri binarne cifre (bita).

Primer
Napraviti tablicu dekadnih brojeva od 0 do 16 i njihovih binarnih, oktalnih i heksadecimalnih ekvivalenata.

Dekadski	**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	20	10

Primer
Pretvoriti heksadecimalni broj AF3 u binarni

$A_{16} = 10_{10} = 1010_2$ $F_{16} = 15_{10} = 1111_2$ $3_{16} = 0011_2$

$AF3_{16} = 101011110011_2$

Primer
Pretvoriti binarni broj 1110110011 u heksadecimalni.
Prvo treba podeliti binarni broj u grupe od po četiri bita, počevši sa desne strane. Kako zadnja grupa sadrži samo dva bita, treba je dopuniti na četiri bita dodavanjem dve nule sa leve strane. Svaku grupu od četiri bita treba prikazati jednom heksadecimalnom cifrom.

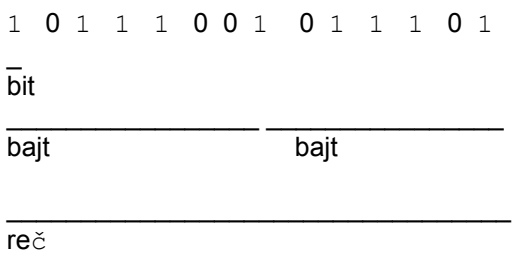
$1110110011_2 = 0011\ 1011\ 0011 = 3B3_{16}$

Prikaz brojeva i znakova u računaru

Za smeštanje (memorisanje) brojeva u računaru služi elektronski sklop koji se naziva **bistabil (flip-flop)**. Naziv bistabil dolazi otuda što takav sklop ima dva stabilna stanja. Jedno stanje (na primer, niskog napona) odgovara znaku 0, a drugo stanje (na primer, visokog napona) odgovara znaku 1. Prema tome bistabil je sklop koji može da zapamti cifre 1 ili 0 (jedan bit). Kako se binarni broj sastoji od više cifara (bitova) za prikaz broja moramo upotrebiti nekoliko bistabila. Takva grupa bistabila čini **registar**. Registri su sastavni deo svih delova računara. Broj bistabila u registru nekog računara određuje njegovu **dužinu**. Dužina većine registara u nekom računaru je određena dužinom reči računara. **Reč** je količina informacija koju računar može da obradi u jednoj operaciji, smesti u memoriju, odnosno uzme iz memorije. Najčešće dužine reči (pa prema tome i registara) su 8, 16, 32 i 64 bita, a kod personalnih računara danas je uobičajena dužina reči od 32 bita.

Grupa od 8 bitova obično se naziva **bajt** (engl. *byte*). Jedan bajt se sastoji od osam bitova. Bit se skraćeno označava s **b**, a bajt s **B**, pa važi **1B = 8b**. Dakle, personalni računari imaju dužinu reči od 4B ili 32b.

Sledeća slika prikazuje registar dužine 16 bita (2 bajta). Svaki bistabil simbolički je prikazan kao jedan kvadratić u koji je zapisan jedan bit. Informacija smeštena u registru čini jednu reč. Reč može predstavljati broj, znak, kôd neke instrukcije i sl. Kako je sadržaj jednog bistabila 0 ili 1, a registar se u ovom slučaju sastoji od 16 bistabila, možemo jednom rečju predstaviti 2_{16} različitih objekata, na primer, 2_{16} različitih brojeva ($2_{16} = 65536$).



Slika 1.2.1

Za smeštanje podataka i programa u računaru služi memorija računara koja se može predstaviti kao skup registara o čemu će više reči biti u narednom poglavlju. Za sada nas zanima u kom obliku se podaci zapisuju u memoriju računara. Osnovni tipovi podataka koji se upisuju u memoriju računara su brojevi (prirodni, celi i realni) i znakovi (slova, cifre, znakovi interpunkcije i sl.).

Prikaz prirodnih brojeva

Prirodni brojevi se zapisuju u memoriju računara slično kao što bi ih zapisivali na papir. Najvažnija razlika je u tome što je broj bitova u računaru koji imamo na raspolaganju za prikaz broja ograničen. Broj bita za prikaz broja nije proizvoljan i može biti jednak dužini reči, ali i duplo manji (polureč) ili duplo veći (dvostruka reč). Šta ćemo odabrati, zavisi od naše procene veličina brojeva koji će se u našim proračunima pojaviti.

Primer

Na raspolaganju za prikaz broja imamo jedan bajt. Kako će u memoriji računara biti prikazan dekadni broj 8?

$8_{10} = 1000_2$.

U memoriji računara biće zapisano `00001000`. Važno je uočiti da smo napisali i nule sa leve strane što je uobičajeno kada se prikazuje sadržaj nekog dela memorije. Na taj način se vidi koliko je bita određeno za prikaz broja, te iznos svakog bita.

Primer
Na raspolaganju za prikaz broja imamo dva bajta. Kako će u memoriji računara biti prikazan binarni broj 11011? Koji je najveći, a koji najmanji dekadni broj koji možemo prikazati sa dva bajta?

Broj 11011_2 će sa dva bajta biti prikazan kao `0000000000011011`.
Najmanji broj je `0000000000000000` ($=0_{10}$), a najveći `1111111111111111` ($=2_{16}-1=65535_{10}$)

Primer
Na raspolaganju za prikaz prirodnog broja imamo dva bajta. Kako će u memoriji računara biti prikazan broj $1F4B_{16}$?

Odgovor glasi: `0001111101001011`. Uočite praktičnost prikaza stanja dva bajta pomoću heksadecimalnog sistema.

Ovakav način prikaza prirodnih brojeva u memoriji računara naziva se **prirodni binarni kôd**.

Jedna od posledica ograničenog broja bitova za prikaz brojeva u računaru je i pojava **prenosa** (carry) kod aritmetičkih operacija. Naime, rezultat neke aritmetičke operacije može zauzimati više bita nego što imamo na raspolaganju. Ako se to dogodi rezultat aritmetičke operacije nije tačan (jer nedostaju bitovi najveće težine) i tada treba registrovati pojavu greške prilikom izvođenja aritmetičke operacije.

Primer
Za prikaz brojeva u računaru na raspolaganju je jedan bajt. Saberite binarne brojeve 10101010 i 1000000.

```
 10101010
+10000000
----------
100101010
```

Vidimo da rezultat zauzima 9 bita. Deveti bit biće „odsečen" i izgledaće da je rezultat `00101010`, što je pogrešno. Zato u sklopu za sabiranje postoji i deveti bit (ako se radi o sklopu koji može sabirati 8-bitne brojeve), koji služi za kontrolu ispravnosti dobijenog rezultata. Ako je deveti bit nula, rezultat je ispravan, a, ako je jednak jedinici rezultat nije ispravan jer je došlo do prenosa. U našem slučaju deveti bit je jednak jedinici, što znači da rezultat `00101010` nije tačan.

Prikaz celih brojeva

Negativne brojeve prikazujemo dodajući znak minus (-) ispred apsolutne vrednosti broja. Međutim, računar upotrebljava binarni sistem zato jer je sagrađen od elektronskih sklopova koji imaju samo dva stanja (na primer, nizak i visoki izlazni napon), koja predstavljaju znak 0 ili znak 1. Prema tome, umesto znakova plus i minus moramo koristiti znakove 0 ili 1.

U računaru za prikaz nekog broja imamo na raspolaganju određen broj cifara (bitova). Na primer, za prikaz nekog broja možemo imati na raspolaganju memorijsku lokaciju dužine 4 bita. Ako želimo da prikažemo i cele brojeve moramo jedan **bit** odvojiti **za predznak**. Za predznak se odvaja krajnji levi bit. Ako je on 0, to znači da je broj pozitivan, a ako je on 1, to znači da se radi o negativnom broju. Na primer, 0001_2 bi bio broj $+1_{10}$, a 1001_2 bi bio broj -1_{10}. Takav način prikazivanja negativnih brojeva je vrlo jednostavan, ali je pritom postupak sabiranja i oduzimanja relativno komplikovan. Osim toga postoje dve nule (+0 i -0).

Zbog navedenih razloga primenjuje se često tehnika **dvojnog komplementa**. Bit za predznak u tehnici dvojnog komplementa interpretira se kao binarno mesto sa odgovarajućim težinskim faktorom, ali sa negativnim predznakom.

Primer
Broj 1011_2 prikazan u registru od 4 bita tehnikom dvojnog komplementa shvatamo ovako:

$1011_2 = -1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = -8 + 0 + 2 + 1 = -5_{10}$.

Dvojni komplement nekog binarnog broja dobija se tako što se primeni sledeći postupak:

1. Dopuniti broj čiji dvojni komplement tražimo na broj bita koji imamao na raspolaganju za prikaz broja dodajući nule sa leve strane.
2. U dobijenom broju zameniti nule sa jedinicama i obratno.
3. Dodati 1.
4. Ako se pojavi prenos koji bi zahtevao dodatni bit on se zanemaruje.

Primer
Na raspolaganju za prikaz broja je 8 bita. Prikazati dekadni broj -5 tehnikom dvojnog komplementa.

Broj 101_2 (5_{10}) dopunimo sa nulama i dobijemo: `00000101`.
Zamenimo nule i jedinice i dobijemo: `11111010`.
Dodamo 1:

```
11111010
+00000001
----------
11111011
```

Dakle, $-5 = 11111011_2$.

Primer
Naći dvojni komplement binarnog broja 00000000.

Primenom gornjeg postupka dobijamo (1)00000000. Jedinica nastala prenosom prilikom sabiranja bila bi deveti bit i nju odbacujemo. Dvojni komplement broja 00000000 je 00000000.

Možemo postaviti sledeće pitanje: Kako znamo da li je u registar upisan negativan broj ili pozitivan broj koji počinje sa jedinicom? Radi se o dogovoru. Moramo uvek naglasiti da radimo sa celim brojevima u tehnici dvojnog komplementa ili da radimo samo sa prirodnim brojevima. Drugim rečima, moramo znati šta sadrži neka memorijska lokacija (registar): prirodni broj, celi broj, realni broj, znakove u ASCII kodu itd..

Primer
Imamo na raspolaganju 4 bita za prikaz broja. Koliko možemo prikazati brojeva ako prikazujemo samo

pozitivne (prirodne) brojeve, a koliko brojeva možemo prikazati ako prikazujemo i negativne brojeve tehnikom dvojnog komplementa?

Ako prikazujemo samo pozitivne brojeve možemo prikazati 16 (2_4) različitih brojeva (od 0000_2 do 1111_2). Ako odvojimo jedan bit za predznak možemo prikazati 8 pozitivnih i 8 negativnih brojeva, dakle ukupno, takođe, 16 brojeva što pokazuje tabela.

Binarno	**Dekadno**
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Prednost upotrebe dvojnog komplementa za zapis celih brojeva je u činjenici da se **oduzimanje binarnih brojeva** svodi na sabiranje vrednosti dvojnog komplementa broja koji treba da se oduzme. Pritom treba zanemariti eventualni dodatni bit koji nastaje prenosom kod sabiranja.

Primer
$0101_2 - 0010_2 = ?$ ($5_{10} - 2_{10} = ?$)

Prvo se nađe dvojni komplement broja 0010. To je 1101+1=1110.

```
0101
+1110
-----
10011
```

Peti bit nastao prenosom treba zanemariti. Rezultat je 0011_2 (3_{10}).

Prikaz realnih brojeva

Realni brojevi prikazuju se u dekadnom sistemu tako da tačka odvaja celi deo od decimalnog dela. Na primer, 12.343, 0.000233, 112000.1 su realni dekadni brojevi. Ovakav način prikaza (poznat i kao **zapis sa nepokretnom tačkom**) nepraktičan je za jako velike ili jako male brojeve. U tom slučaju koristi se **eksponencijalni prikaz** realnog broja (poznat i **zapis sa pokretnom tačkom**). Na primer, masa elektrona vrlo je mala i iznosi $9.109*10^{-31}$ kg, a njegovo naelektrisanje $1.602*10^{-19}$ C. Brzinu svetlosti, takođe, praktičnije je prikazati u eksponencijalnom obliku prikaza jer se radi o velikom broju ($3*10^8$ m/s). Zapis tog oblika sastoji se od mantise, osnove i eksponenta. Na primer, broj 15.825 mogli bi u obliku sa pokretnom tačkom zapisati ovako:

mantisa eksponent osnova

```
15.825 = 15.825 * 10^0   15.825      0          10
       = 0.15825 * 10^2   0.15825    2          10
       = 15825 * 10^-3    15825      -3         10
```

Vidi se da se na ovaj način broj može zapisati na mnogo načina. Ako se postavi ograničenje na mantisu takvo da se ona uvek nalazi u području:

B^{-1} <= |mantisa| < 1,

gde je B osnova brojnog sistema, govorimo o **normiranom prikazu**. Dakle, u normiranom prikazu tačka se postavlja ispred najznačajnijeg broja koji nije nula, što pokazuju sledeći primeri:

decimalni broj	**mantisa**	**eksponent**
15.825	0.15825	2
0.054	0.54	-1
1234	-0.1234	4
0.0	0.0	0
0.00343	-0.343	-2

U dekadnom brojnom sistemu mantisa se nalazi između 0.1 i 1. Naravno, izuzetak je broj nula. U binarnom brojnom sistemu (B = 2, $2^{-1} = 0.5$) za mantisu važi: 0.5 <= |mantisa| < 1.

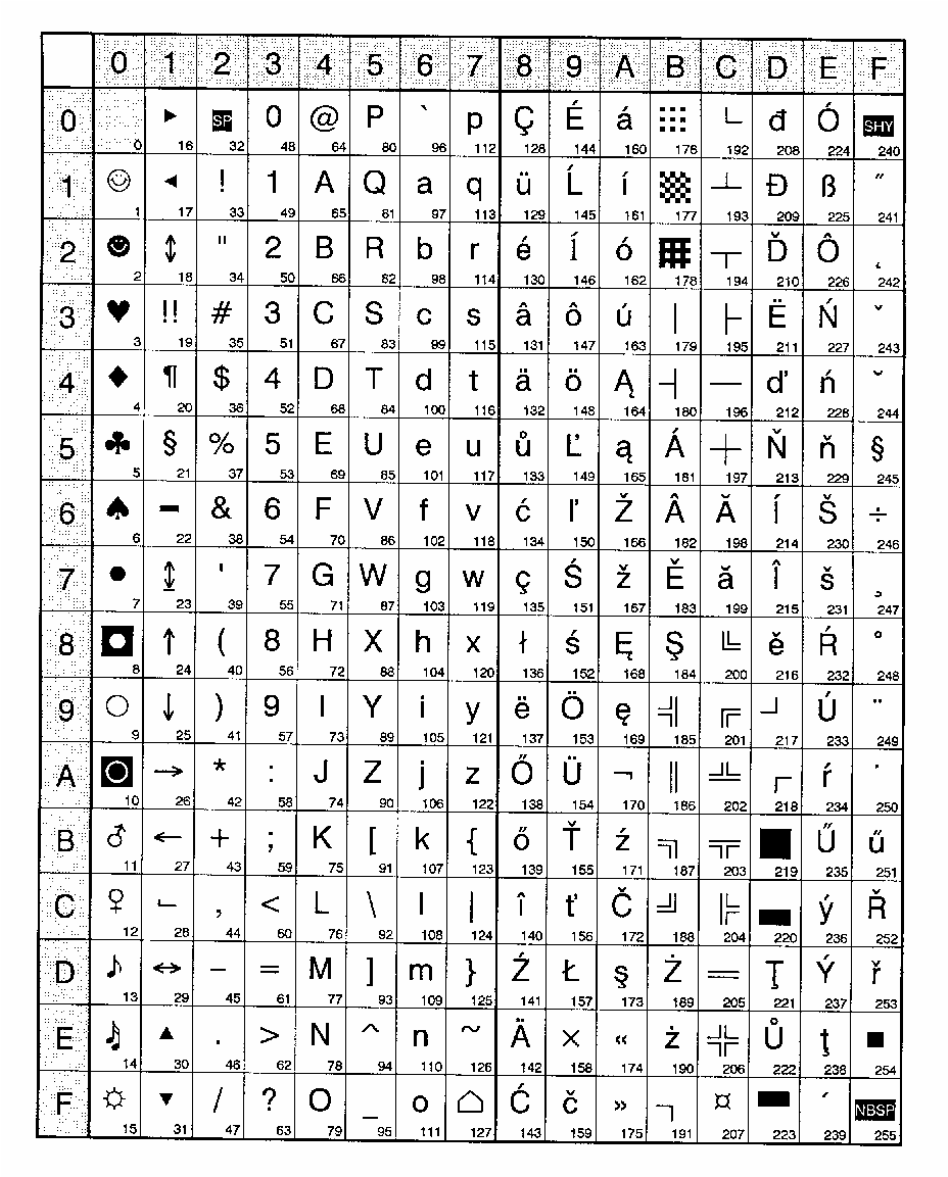
U računaru se realni brojevi prikazuju u zapisu sa pokretnom tačkom. Takav zapis može zauzimati jednu ili dve reči (4 ili 8B). Od toga jedan deo (manji) zauzima eksponent, a drugi deo mantisa. Nula kojom započinje mantisa u normiranom prikazu se ne zapisuje. Mantisa i eksponent mogu biti u istom kodu (na primer, dvojni komplement), ali ne moraju. Detaljan opis ovakvog zapisa daje na primer standard ANSI/IEEE Std 754-1985 čiji opis premašuje okvire ovog teksta.

Prikaz nebrojčanih veličina u računaru

Osim sa brojevima, računari moraju raditi i sa slovima i drugim znakovima. Njih u memoriju računara ne možemo zapisati u izvornom obliku, već samo koristeći unapred dogovorenu kombinaciju binarnih cifri za svaki znak. Takva kombinacija bitova naziva se **kôd** određenog znaka.

Da bi se omogućila razmena podataka između računara potrebno je imati standardizovan kôd koji će svi upotrebljavati i razumeti. Danas je u širokoj upotrebi **ASCII** (*American Standards Code for Information Interchange*). To je osmo-bitni kôd (kôd čija je dužjina 8 bita), koji omogućuje prikaz velikih i malih slova, specijalnih znakova (na primer, *, +, =, ?, $, %, itd.), te upravljačkih znakova (na primer, početak poruke, kraj poruke, novi red, itd.). Ukupno je sa osam bita moguće prikazati 256 (2_8=256) različitih znakova. Međutim, prvih 128 znakova je zaista standardizovano, a preostalih 128 nije jedinstveno standardizovano. Razlog tome je što je originalni ASCII koristio 7 bita. Dodatnih 128 kodova za novih 128 znakova dobijeno je dodavanjem jednog bita, te je dobijen tzv. proširenI skup znakova. Kompanija IBM koristi neke od dodatnih 128 kodova za prikaz slova koja su specifična za različite evropske zemlje. Naime, ne treba zaboraviti da je reč o američkom standardu, koji ne vodi računa o specifičnostima drugih zemalja.

Sledeća slika prikazuje ASCII kôd sa tzv. kodnom stranicom IBM 852 (Latin II), kod koje se među gornjih 128 kodova nalaze kodovi za slova slovenskih jezika, pa i srpskog.



Slika 1.2.2 Kodna stranica Latin II (IBM 852)

U gornjoj tabeli prikazani su dekadni ekvivalenti binarnih kodova. Na primer, kôd slova A zapravo je `01000001`, što je zgodnije prikazati skraćeno heksadecimalno kao 41, ili dekadno 65.

Postoji i drugi način za prikazivanje naših slova (popularno zvani YUSCII kôd), koji je stariji od kodne stranice 852, a naše znakove smešta unutar prvih 128 znakova, žrtvujući pri tome neke važne znakove.

Rešenje prikazuje sledeća tabela. Ovaj kôd se danas retko koristi. U MS DOS operativnom sistemu prevladava IBM 852 kôd.

U tabeli su prikazani i kodovi koje za prikaz naših znakova koristi firma Microsoft u operativnom sistemu Windows, čime se dodatno komplikuje problem naših slova.

Naš znak YUSCII IBM 852 Microsoft 1250

Č	94_{10} (^)	172_{10}	200_{10}
Ć	93_{10} ( ])	143_{10}	198_{10}
Đ	92_{10} (\ )	209_{10}	208_{10}
Š	91_{10} ([ )	230_{10}	138_{10}
Ž	64_{10} (@)	166_{10}	142_{10}
č	126_{10} (~)	159_{10}	232_{10}
ć	125_{10} (})	134_{10}	230_{10}
đ	124_{10} (|)	208_{10}	240_{10}
š	123_{10} ({)	231_{10}	154_{10}
ž	96_{10} (`)	167_{10}	158_{10}

Poteškoće sa našim znakovima mogu nastupiti prilikom sortiranja reči po abecedi. Naime, kod sortiranja se koristi činjenica da numeričke vrednosti kodova (dakle kodovi shvaćeni kao binarni brojevi) odgovaraju redosledu slova u abecedi. Na primer, kôd slova a manji je od koda slova b, a taj je opet manji od koda slova c, itd. Međutim, to važi samo za slova američke abecede koja zauzimaju kodove 65 - 90 (slova A - Z) i 97 - 122 (slova a - z). Zbog toga program koji ne vodi računa o specifičnostima naših slova neće dobro sortirati, na primer, prezimena sa našim slovima, iako će sortiranje engleskih prezimena raditi bez greške.

Sad sledi listing programa u Turbo Pascalu koji radi pretvaranja iz jednog brojnog sistema u drugi (decimalnog u binarni i obrnuto; decimalnog u oktalni i obrnuto itd.).

```
PROGRAM ConversionMenu;
{ Conversion Menu 1.1              }
{ Copyright (C) 1997 H†kon Stordahl }

{ E-mail  : stordahl@usa.net           }
{ Homepage: http://stordahl.home.ml.org/ }

USES Crt, CONVUNIT;

PROCEDURE SetUpScreen;

BEGIN
  ClrScr;
  WriteLn('Conversion Menu 1.0');
  WriteLn('Written by H†kon Stordahl 03.06.96');
  WriteLn;
  WriteLn('1. Binary to decimal');
  WriteLn('2. Decimal to binary');
  WriteLn('3. Hexadecimal to decimal');
  WriteLn('4. Decimal to hexadecimal');
  WriteLn('5. Binary to hexadecimal');
  WriteLn('6. Hexamdecimal to binary');
  WriteLn('7. Octal to decimal');
  WriteLn('8. Decimal to octal');
  WriteLn('9. Exit');
END;

VAR
```

```
  Ch : CHAR;
  St : STRING;
  Num: LONGINT;

BEGIN { main program }
  SetUpScreen;
  REPEAT
    Ch := ReadKey;
    SetUpScreen;
    GotoXY(1, 14);
    CASE Ch OF
      #49:              { 1 }
      BEGIN
        Write('Enter binary number: ');
        ReadLn(St);
        WriteLn(St, ' in binary is equal to ', BIN2DEC(St), ' in decimal.');
      END;
      #50:              { 2 }
      BEGIN
        Write('Enter decimal number: ');
        ReadLn(Num);
        WriteLn(Num, ' in decimal is equal to ', DEC2BIN(Num), ' in binary.');
      END;
      #51:              { 3 }
      BEGIN
        Write('Enter hexadecimal number: ');
        ReadLn(St);
        WriteLn(St, ' in hexadecimal is equal to ', HEX2DEC(St), ' in decimal.');
      END;
      #52:              { 4 }
      BEGIN
        Write('Enter decimal number: ');
        ReadLn(Num);
        WriteLn(Num, ' in decimal is equal to ', DEC2HEX(Num), ' in hexadecimal.');
      END;
      #53:              { 5 }
      BEGIN
        Write('Enter binary number: ');
        ReadLn(St);
        WriteLn(St, ' in binary is equal to ', BIN2HEX(St), ' in hexadecimal.');
      END;
      #54:              { 6 }
      BEGIN
        Write('Enter hexadecimal number: ');
        ReadLn(St);
        WriteLn(St, ' in hexadecimal is equal to ', HEX2BIN(St), ' in binary.');
      END;
      #55:
      BEGIN
        Write('Enter octal number: ');
        ReadLn(St);
```

```
      WriteLn(St, ' in octal is equal to ', OCT2DEC(St), ' in decimal.');
    END;
    #56:
    BEGIN
      Write('Enter decimal number: ');
      ReadLn(Num);
      WriteLn(Num, ' in decimal is equal to ', DEC2OCT(Num), ' in octal.');
    END;
  END;
 UNTIL Ch = #57;       { 9 }
 ClrScr;
END.  { main program }
```

Modul (unit) koji je potreban da bi prethodni program mogao da radi.

```
PROGRAM ConversionMenu;
{ Conversion Menu 1.1              }
{ Copyright (C) 1997 H†kon Stordahl }

{ E-mail  : stordahl@usa.net           }
{ Homepage: http://stordahl.home.ml.org/ }

USES Crt, CONVUNIT;

PROCEDURE SetUpScreen;

BEGIN
 ClrScr;
 WriteLn('Conversion Menu 1.0');
 WriteLn('Written by H†kon Stordahl 03.06.96');
 WriteLn;
 WriteLn('1. Binary to decimal');
 WriteLn('2. Decimal to binary');
 WriteLn('3. Hexadecimal to decimal');
 WriteLn('4. Decimal to hexadecimal');
 WriteLn('5. Binary to hexadecimal');
 WriteLn('6. Hexamdecimal to binary');
 WriteLn('7. Octal to decimal');
 WriteLn('8. Decimal to octal');
 WriteLn('9. Exit');
END;

VAR
 Ch : CHAR;
 St : STRING;
 Num: LONGINT;

BEGIN { main program }
 SetUpScreen;
 REPEAT
   Ch := ReadKey;
```

```
SetUpScreen;
GotoXY(1, 14);
CASE Ch OF
  #49:              { 1 }
  BEGIN
    Write('Enter binary number: ');
    ReadLn(St);
    WriteLn(St, ' in binary is equal to ', BIN2DEC(St), ' in decimal.');
  END;
  #50:              { 2 }
  BEGIN
    Write('Enter decimal number: ');
    ReadLn(Num);
    WriteLn(Num, ' in decimal is equal to ', DEC2BIN(Num), ' in binary.');
  END;
  #51:              { 3 }
  BEGIN
    Write('Enter hexadecimal number: ');
    ReadLn(St);
    WriteLn(St, ' in hexadecimal is equal to ', HEX2DEC(St), ' in decimal.');
  END;
  #52:              { 4 }
  BEGIN
    Write('Enter decimal number: ');
    ReadLn(Num);
    WriteLn(Num, ' in decimal is equal to ', DEC2HEX(Num), ' in hexadecimal.');
  END;
  #53:              { 5 }
  BEGIN
    Write('Enter binary number: ');
    ReadLn(St);
    WriteLn(St, ' in binary is equal to ', BIN2HEX(St), ' in hexadecimal.');
  END;
  #54:              { 6 }
  BEGIN
    Write('Enter hexadecimal number: ');
    ReadLn(St);
    WriteLn(St, ' in hexadecimal is equal to ', HEX2BIN(St), ' in binary.');
  END;
  #55:
  BEGIN
    Write('Enter octal number: ');
    ReadLn(St);
    WriteLn(St, ' in octal is equal to ', OCT2DEC(St), ' in decimal.');
  END;
  #56:
  BEGIN
    Write('Enter decimal number: ');
    ReadLn(Num);
    WriteLn(Num, ' in decimal is equal to ', DEC2OCT(Num), ' in octal.');
  END;
```

```
    END;
  UNTIL Ch = #57;        { 9 }
  ClrScr;
END.  { main program }
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu (XX)

Grafika - Procedure i funkcije

Grafika visoke rezolucije odnosi se na drugačiji način prikazivanja „stvari” na ekranu. Postoje dva ekranska režima u Pascalu: tekstualni i grafički.

Tekstualni režim se nekada mnogo više koristio (vreme DOS-a). Prilikom izvršavanja programa, računar je automatski koristio tekstualni režim (pošto Turbo Pascal 7.0 radi pod MS-DOS-om). Iz tog razloga, tekstualni režim je veoma pogodan u fazi razvoja programa kada ima dosta testiranja. Tekstualni režim deli ekran aproksimativno na 25 redova (sa leva na desno) i 80 kolona (odozgo na dole). "Ćelija" je pravougaonik gde se presecaju jedan red i jedna kolona. Ćelije su organizovane slično policama u prodavnicama obuće, jedino što u ćeliju ne možete da stavite svoje cipele nego samo jedan ASCII znak.

Sve komande koje se odnose na tekstualni režim nalaze se u modulu (jedinici ili junitu) Crt u Pascalovoj biblioteci ugrađenih funkcija I procedura. Niže je navedena lista svih procedura i funkcija koje se nalaze u Crt.

AssignCrt

Pridružuje tekstualnu datoteku Crt prozoru.

ClrEol

Briše sve znakove od tekuće pozicije pokazivača do kraja reda (linije).

ClrScr

Briše ekran i vraća pokazivač u gornji levi ugao ekrana.

Delay

Kašnjenje za navedeni broj milisekundi.

DelLine

Deletes the line containing the cursor.

GotoXY

Moves the cursor to the given coordinates within the virtual screen.

HighVideo

Selects high-intensity characters.

InsLine

Inserts an empty line at the cursor position.

KeyPressed

Determines if a key has been pressed on the keyboard.

LowVideo

Selects low-intensity characters.

NormVideo

Selects the original text attribute read from the cursor location at startup.

NoSound

Turns off the computer's internal speaker.

ReadKey

Reads a character from the keyboard.

Sound

Starts the internal speaker.

TextBackground

Selects the background color.

TextColor

Selects the foreground character color.

TextMode

Selects a specific text mode.

WhereX

Returns the X coordinate of the current cursor location.

WhereY

Returns the Y coordinate of the current cursor location.

Window

Defines a text window on the screen.

Graphics mode, on the other hand, use an entirely different software and hardware. The software consists of screen drivers contained in "BGI" files provided by Borland. In graphics mode, the screen is organized into tiny dots called "pixels" (which stands for "picture elements"). The pixels are labelled by x and y coordinates with the top, left-most pixel being (0,0). The total number of pixels on the screen depends on the driver that the computer is using. Graphics mode offers a more flexible way of presenting images on the screen; by placing together many coloured dots, one can create a very realistic-looking image. All of the procedures and functions that relate to graphics are found in the Graph unit.

Procedura **InitGraph** - Inicira grafički režim rada video adaptera. Zaglavlje procedure:

```
Procedure InitGraph(var Driver,Mode: Integer; Path: String);
```

Ovde je Driver - promenljiva tipa Integer koja određuje tip grafičkog drajvera; Mode - promenljiva istog tipa koja definiše režim rada grafičkog adaptera; Path - izraz tipa String koji sadrži ime datoteke drajvera i putanju do nje.

U trenutku poziva procedure na nekom od diskova računara mora se nalaziti datoteka koja sadrži potreban grafički drajver. Procedura učitava taj drajver u radnu memoriju i prevodi adapter u grafički režim rada. Tip drajvera mora da odgovara tipu grafičkog adaptera. Za navođenje tipa drajvera u modulu predviđene su sledeće konstante:

```
const

Detect=0;{Režim autodetekcije}

CGA=1;

MCGA=2;

EGA=3;

EGA64=4;

EGAMono=5;
```

```
IBM8514=6;

HercMono=7;

ATT400=8;

VGA=9;

PC3270=10;
```

Većina adaptera može da radi u više različitih režima. Zbog toga, da bi naveli adapteru potrebni režim rada, koristimo promenljivu Mode, čije vrednosti mogu biti sledeće konstante:

```
const
{ Adapter CGA : }
CGACO = 0;        {Niska rezolucija, paleta    0}
CGAC1 = 1;        {Niska rezolucija, paleta    1}
CGAC2 = 2;        {Niska rezolucija, paleta    2}
CGAC3 = 3;        {Niska rezolucija, paleta    3}
CGAHi = 4;        {Visoka rezolucija }
{Adapter MCGA:}
MCGACO = 0;       {Emulacija CGACO}
MCGAC1 = 1;       {Emulacija CGAC1}
MCGAC2 = 2;       {Emulacija CGAC2}
MCGAC3 = 3;       {Emulacija CGAC3}
MCGAMed = 4;      {CGAHi}
MCGAHi = 5;       {640x480}
{Adapter EGA :}
EGALo = 0;        {640x200, 16 boja}
EGAHi = 1;        {640x350, 16 boja}
EGAMonoHi = 3;    {640x350, 2 boje}
{Adapterы HGC i   HGC+:}
HercMonoHi = 0;   {720x348}
{AdapterATT400:}
ATT400CO = 0;     {Analogno režimu CGACO}
ATT400C1 = 1;     (Analogno režimu CGAC1}
ATT400C2 = 2;     {Analogno režimu CGAC2}
ATT400C3 = 3;     {Analogno režimu CGAC3}
```

```
ATT400Med = 4;    {Analogno režimu CGAHi}
ATT400H1 = 5;     {640x400, 2 boje}

{Adapter VGA:}

VGALo = 0;          {640x200}

VGAMed = 1;          {640x350}

VGAHi = 2;          {640x480}

PC3270H1 = 0;        {Analogno HercMonoHi}

{Adapter 1VM8514}

IBM8514LO =0;      {640x480, 256 boja}

IBM8514H1 = 1;     {1024x768, 256 boja}
```

Neka se, na primer, drajver CGA.BGI nalazi u direktorijumu TP\BGI na disku C i postavlja režim rada 320x200 sa paletom 2. Tada procedura ima oblik:

```
Uses Graph;

var

Driver, Mode : Integer;

begin

Driver := CGA;{Drajver}

Mode := CGAC2;{Režim rada}

InitGraph(Driver, Mode,' C:\TP\BGI') ;

.......
```

Ako je tip adaptera nepoznat ili ako je program predviđen za rad s svakim adapterom, koristi se procedura sa autodetekcijom tipa drajvera:

```
Driver := Detect;

InitGraph(Driver, Mode, 'C:\TP\BGI');
```

Posle ovoga uspostavlja se grafički režim rada ekrana, a pri izlazu iz procedure promenljive Driver i Mode sadrže celobrojne vrednosti, određene tipom drajvera i režimom njegovog rada. Pri tome za adaptere koji su sposobni da rade u više režima, bira se stariji režim, tj. onaj kodiran maksimalnom cifrom. Tako, pri radu sa CGA -adapterom procedura sa vrednošću Driver = Detect vraća u promenljivoj Driver vrednost 1 (CGA) a za Mode - vrednost 4 (CGAHi), a u slučaju adaptera VGA vraća se Driver = 9 (VGA) i Mode = 2 (VGAHi).

Funkcija GraphResult - Vraća vrednost tipa Integer, u kojoj je kodiran rezultat poslednjeg obraćanja grafičkoj proceduri. Ako nije pronađena greška, vrednost funkcije biće nula, u suprotnom slučaju – negativan broj, koji ima sledeće značenje:

const

grOk = 0;{Nema greške}

grInitGraph =-1;{Nije inicijalizovan grafički režim}

grNotDetected =-2;{Nije određen tip drajvera}

grFileNotFind =-3;{Nije nađen grafički drajver}

grInvalidDriver =-4;{Nepravilan tip drajvera}

grNoLoadMem =- 5;{Nema memorije za smeštanje drajvera}

grNoScanMem = - 6;{Nema memorije za skeniranje oblasti}

grNoFloodMem =- 7;{Nema memorije za popunjavanje oblasti}

grFontNotFound = -8;{Nije nađena datoteka sa fontom}

grNoFontMem =- 9;{Nema memorije za smeštanje fonta}

grInvalidMode =-10;{Nepravilan grafički režim}

grError =-11;{Opšta greška}

grIOError =-12;{Greška ulaza/izlaza}

grInvalidFont =-13;{Nepravilan format fonta}

grInvalidFontNum=-14; {Nepravilan broj fonta}

Funkcия GraphErrorMsg.

Vraća vrednost tipa String, tj. po navedenom kodu greške se daje odgovarajuće tekstualno objašnjenje. Zaglavlje funkcije:

```
Function GraphErrorMsg(Code: Integer): String;
```

Ovde je Code - kôd greške, koji vraća funkcija GraphResult.

Na primer, tipičan redosled operatora za inicijalizaciju grafičkog režima sa autodetekcijom tipa drajvera i postavkom maksimalne rezolucije ima sledeći oblik vid:

```
var

Driver, Mode, Error:Integer;

begin

Driver := Detect;{Autodetekcija drajvera}

InitGraph(Driver, Mode,' ');{Inicira grafiku}

Error := GraphResult;{Daje rezultat}

if Error <> grOk then{Provera greške}

begin{Greška u proceduri iniciajalizacije}

WriteLn(GraphErrorMsg(Error));{Dajemo obaveštenje}

.......

end

else{Nema grešaka}

.......
```

Čest uzrok nastajanja greški pri obraćanju proceduri InitGraph je nepravilno navođenje mesta nalaženja datoteke sa drajverom grafičkog adaptera (na primer, datoteka CGA.BGI za adapter CGA). Postavka lokacije drajvera postiže se navođenjem putanje do neophodne datoteke pri pozivu procedur eInitGraph.

Ako je, na primer, drajver u poddirektorijumu DRIVERS direktorijuma PASCAL na disku D, tada je potrebno koristiti sledeći poziv:

```
InitGraph(Driver, Mode, 'd:\Pascal\Drivers');
```

Napomena: U svim sledećim primerima procedura InitGraph poziva se sa parametrom Driver u obliku praznog reda. Takav oblik obraćanja biće korektan samo u slučaju kada se potrebna datoteka grafičkog drajvera nalazi u tekućem direktorijumu.

Procedura CloseGraph.

Okončava rad adaptera u grafičkom režimu i ponovo uspostavlja tekstualni režim rada ekrana. Zaglavlje:

```
Procedure CloseGraph;
```

Procedura RestoreCRTMode.

Služi za kratkotrajan povratak u tekstualni režim. Za razliku od procedure CloseGraph ne potire (anulira) postavljene parametre grafičkog režima i ne oslobađa memoriju, predviđenu za smeštanje grafičkog drajvera. Zaglavlje:

```
Procedure RestoreCRTMode;
```

Funkcия GetGraphMode.

Vraća vrednost tipa Integer, u kojoj je sadržan kôd postavljenog režima rada grafičkog adaptera. Zaglavlje:

```
Function GetGraphMode: Integer;
```

Procedura SetGraphMode.

Postavlja novi grafički režim rada adaptera. Zaglavlje:

```
Procedure SetGraphMode(Mode: Integer);
```

Ovde je Mode - kôd režima koji se postavlja.

Sledeći programa ilustruje prelaz iz grafičkog režima u tekstualni i obratno:

```
Uses Graph;
var .
Driver, Mode, Error : Integer;
begin
{Iniciramo grafički režim}
Driver := Detect;
InitGraph(Driver, Mode, '');
Error := GraphResult; {Pamitimo rezultat}
i£ Error <> grOk then {Provera greške}
WriteLn(GraphErrorMsg(Error)) {Jeste greška}
else
begin {Nema greške}
WriteLn (' Ovo je grafički režim');
WriteLn ('Pritisnite "Enter"...':20);
ReadLn;
{Prelaz u tekstualni režim}
RestoreCRTMode;
WriteLn (' a ovo tekstualni...');
ReadLn;
{Povratak u grafički režim}
SetGraphMode (GetGraphMode);
WriteLn ('Opet grafički režim...');
```

```
ReadLn;

CloseGraph

end

end.
```

Procedura DetectGraph.

Vraća tip drajvera i režim njegovog rada. Zaglavlje:

```
Procedure DetectGraph(var Driver,Mode: Integer);
```

Ovde je Driver - tip drajvera; Mode - režim rada.

Za razliku od funkcije GetGraphMode ova procedura vraća u promenljivoj Mode maksimalan mogući broj grafičkog režima za dati adapter.

Funkcия GetDriverName.

Vraća vrednost tipa String, koja sadrži ime učitanog grafičkog drajvera. Zaglavlje:

```
Function GetDriverName: String;
```

Funkcия GetMaxMode.

Vraća vrednost tipa Integer, koja sadrži broj mogućih režima rada adaptera. Zaglavlje:

```
Function GetMaxMode: Integer;
```

Funkcija GetModeName.

Vraća vrednost tipa String, koja sadrrži rezoluciju ekrana i ime režima rada adaptera prema njegovom broju. Zaglavlje:

```
Function GetModName(ModNumber: Integer): String;
```

Ovde je ModNumber - broj režima.

Sledeći program posle inicijalizacije grafičkog režima ispisuje na ekran red, koji sadrži ime učitanog drajvera, a takođe i sve moguće režime njegovog rada.

```
Uses Graph;

var

a,b: Integer;

begin

a := Detect;

InitGraph(a, b, '');

WriteLn(GetDriverName);

for a := 0 to GetMaxMode do

WriteLn(GetModeName(a):10);

ReadLn;

CloseGraph

end.
```

Procedura GetModeRange.

Vraća dijapazon mogućih režima rada datog grafičkogo adaptera. Zaglavlje ima oblik:

```
Procedure GetModeRange(Drv: Integer; var Min, Max: Integer);
```

Ovde je Drv - tip adaptera; Min - promenljiva tipa Integer, koja vraća donju moguću vrednost numeracije režima rada; Mah - promenljiva tog istog tipa, samo gornju vrednost.

Ako je zadata nepravilna vrednost parametra Drv, procedura vraća u obe promenljive vrednost -1. Pre obraćanja proceduri moguće je da se ne uspostavi grafički režim rada ekrana. Sledeći program ispisujet na ekranu nazive svih adaptera i dijapazon mogućih numeracija režima njiihovog rada.

```
Uses Graph;
var
D,L,H: Integer;
const
N: array [1..11] of String [8] =
('CGA ', 'MCGA ', 'EGA ',
'EGA64 ', 'EGAMono ', `VM8514 ',
'HercMono', 'ATT400 ', 'VGA ',
'PC3270 ', 'Greska ');
begin
WriteLn('Adapter Min. Maks.');
for D := 1 to 11 do
begin
GetModeRange(D, L, H);
WriteLn(N[D], L:7, H:10)
end
end.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (10)

Koordinate, prozori, stranice

Mnoge grafičke procedure i funkcije koriste pokazivač tekuće pozicije na ekranu koji je za razliku od tekstualnog kursora nevidljiv. Položaj tog pokazivača, kao i svala koordinata na grafičkom ekranu zadaje se relativno, tj. u odnosu na levi gornji ugao ekrana, koji u tom smislu ima koordinate 0,0. Na taj način, horizontalna koordinata ekrana raste sa leva na desno, a vertikalna – odozgo na dole.

Funkcije GetMaxX i GetMaxY

Vraćaju vrednost tipa Word, koja sadrži maksimalne koordinate ekrana u tekućem režimu rada, respektivno po horizontali i vertikali. Na primer:

```
Uses Graph;

var

a,b: Integer;

begin

a := Detect; InitGraph(a, b, '');

WriteLn(GetMaxX, GetMaxY:5);

ReadLn;

CloseGraph

end.
```

Funkcije GetX и GetY.

Vraćaju vrednost tipa Integer, koja sadrži tekuće koordinate pokazivača, respektivno po horizontali i vertikali. Координате se određuju u odnosu na levi gornji ugao prozora, ili ako prozor nije definisan, onda ekrana.

Procedura SetViewPort.

Postavlja pravougaoni prozor na grafičkom ekranu. Zaglavlje:

```
Procedure SetViewPort(XI,Y1,X2,Y2: Integer; ClipOn: Boolean);
```

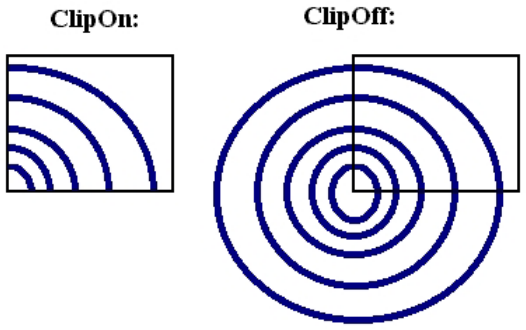
Ovde su X1...Y2 – koordinate levog gornjeg ugla (XI,Y1) i desnog donjeg (X2,Y2) ugla prozora; ClipOn – izraz tipa Boolean, koji određuje „odsecanje" elemenata prikaza koji ne mogu da stanu u prozor.

Koordinate prozora uvek se zadaju u odnosu na levi gornji ugao ekrana. Ako parametar ClipOn ima vrednost True, elementi prikaza, koji ne mogu da stanu u prozor, odsecaju se, u suprotnom

slučaju granice prozora se ignorišu. Za upravljanje tim parametrom u modulu se mogu koristiti konstante:

```
const

ClipOn = True; {Uključi odsecanje}

ClipOff = False; {Ne uključuj odsecanje}
```

Sledeći primer ilustruje uticaj parametra ClipOn. Program pravi dva pravougaona prozora sa različitim vrednostima parametra i smešta u njih nekoliko kružnica. Zbog bolje preglednosti prozori su oivičeni ramovima (slika 14.1).



Slika 1. Odsecanje prikaza u prozoru

```
Uses Graph,CRT;

var

x,y,e: Integer;

xll,yll,xl2,yl2, {Koordinate 1-og prozora}

x21,x22, {Levi gornji ugao 2-og}

R, {Početni radius}

k: Integer;

begin

DirectVideo := False {Blokiramo direktan pristup video memoriji u modulu CRT}

{Iniciramo grafički režim}

x := Detect; InitGraph(x, y, '');

{Proveravamo rezultat}

e := GraphResult; if e <> grOk then
```

```
WriteLn(GraphErrorMsg (e) ) {Greška}
else
begin {Nema greške}
{Izračunavamo koordinate prema rezoluciji ekrana}
x11:=GetMaxX div 60;
x12:=GetMaxX div 3;
y11:=GetMaxY div 4; y12:=2*y11;
R:=(x12-x11) div 4; x21:=x12*2;
x22:=x21+x12-x11;
{Crtamo prozore}
WriteLnt'ClipOn:':10,'ClipOff:':40);
Rectangle(x11, y11, x12, y12); Rectangle(x21, y11 x22, y12);
{Postavljamo prvi prozor i crtamo četiri kružnice}
SetViewPort(x11, y11, x12, y12, ClipOn);
for k := 1 to 4 do
Circle(0,y11,R*k);
{Postavljamo drugi prozor i crtamo kružnice}
SetViewPort(x21, y11, x22, y12, ClipOff);
for k := 1 to 4 do
Circle(0,y11,R*k);
{Čekamo na aktiviranje nekog tastera na tastaturi}
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Procedura GetViewSettings.

Vraća koordinate i vrstu odsecanja tekućeg grafičkog prozora. Zaglavlje:

```
Procedure GetViewSettings(var ViewInfo: ViewPortType);
```

Ovde je ViewInfo – promenljiva tipa ViewPortType. Ovaj tip u modulu Graph se određuje na sledeći način:

```
type

ViewPortType = record

x1,y1,x2,y2: Integer; {Koordinate prozora}

Clip : Boolean {Vrsta odsecanja}

end ;
```

Procedura MoveTo.

Postavlja novi tekući položaj pokazivača. Zaglavlje:

```
Procedure MoveTo(X,Y: integer);
```

Ovde su X, Y – nove koordinate pokazivača, respektivno po horizontali i vertikali.

Koordinate se određuju u odnosu na levi gornji ugao prozora, a ako on nije postavljen, tada u odnosu na levi gornji ugao ekrana.

Procedura MoveRel.

Postavlja novi položaj pokazivača u relativnim koordinatama.

```
Procedure MoveRel(DX,DY: Integer);
```

Ovde su DX.DY- priraštaji novih koordinata pokazivača respektivno po horizontali i vertikali.

Priraštaji se zadaju u odnosu na položaj koji je pokazivač imao u trenutku pozivanja procedure.

Procedura ClearDevice.

Briše grafički ekran. Posle pozivanja ove procedure pokazivač se nalazi u gornjem levom uglu ekrana, a sam ekran obojen je bojom koja je zadata procedurom SetBkColor. Zaglavlje:

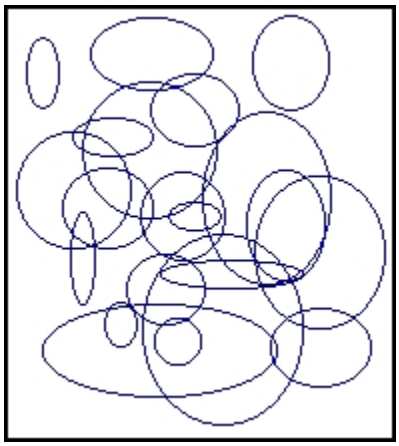
```
Procedure ClearDevice;
```

Procedura ClearViewPort.

Briše grafički prozor, a ako prozor nije u tom trenutku definisan – ceo ekran. Pri brisanju prozor se boji bojom koja je označena brojem 0 u tekućoj paleti. Pokazivač se premešta u levi gornji ugao prozora. Zaglavlje:

```
Procedure ClearViewPort;
```

U sledećem programu na ekranu se obrazuje prozor koji se zatim ispunjava kružnicama na slučajan način (slika 2.). Posle pritiska na bilo koji taster prozor se briše. Za izlazak iz programa pritisnite taster Enter.



Slika 2. Prozor sa kružnicama generisanim na slučajan način

```
Uses CRT,Graph;
var
x1,y1,x2,y2,Err: Integer;
begin
{Iniciramo grafički režim}
xl := Detect; InitGraph(xl,x2,'');
Err := GraphResult; if ErrogrOk then
WriteLn(GraphErrorMsg(Err))
else
begin
{Određujemo koordinate prozora prema rezoluciji ekrana}
x1 := GetMaxX div 4,-y1 := GetMaxY div 4;
x2 := 3*x1; y2 := 3*y1;
{Formiramo prozor}
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
```

```
{Ispunjavamo prozor „slučajnim” kružnicama}
repeat
Circle(Random(Ge tMaxX),Random(Ge tMaxX)
Random(GetMaxX div 5))
until KeyPressed;
{Brišemo prozor i čekamo da bude pritisnut taster Enter}
ClearViewPort;
OutTextXY(0,0,'Press Enter...1);
ReadLn;
CloseGraph
end
end.
```

Procedura GetAspectRatio.

Vraća dva broja koja omogućavaju da se odredi odnos strana ekrana. Zaglavlje:

```
Procedure GetAspectRatio(var X,Y: Word);
```

Ovde su X, Y – promenljive tipa Word. Vrednosti vraćene u tim promenljivama omogućavaju da izračunamo odnos strana grafičkog ekrana u pikselima. Izračunati koeficijent može se iskoristiti pri stvaranju geometrijskih slika, kao što su kružnice, kvadrati itd. Na primer, ako želite da napravite kvadrat sa dužinom stranice od L piksela po vertikali, morate da koristite operatore

```
GetAspectRatio (Xasp, Yasp);
Rectangle(x1, y1, x1+L*round (Yasp/Xasp), y1+L);
```

a ako L određuje dužinu kvadrata po horizontali, tada se koristi operator

```
Rectangle (x1,y1,x1+L,y1+L*round(Xasp/Yasp));
```

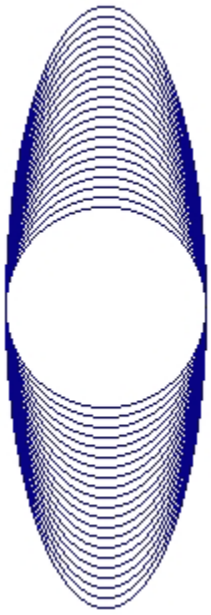
Procedura SetAspectRatio.

Postavlja koeficijent razmeravanja odnosa strana grafičkog ekrana. Zaglavlje:

```
Procedure SetAspectRatio(X,Y: Word);
```

Ovde su X, Y- uspostavljeni odnosi strana.

Sledeći program stvara 20 kružnica sa različitim odnosima (razmerama) strana ekrana (slika 3.).



Slika 3. Kružnice pri raznim odnosima strana ekrana

```
Uses Graph,CRT;
const
R =.50;
dx = 1000;
var
d,m,e,k : Integer;
Xasp,Yasp: Word;
begin
d := Detect;
InitGraph(d, m,.'');
e : = GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
```

```
GetAspectRatio(Xasp, Yasp);
for k := 0 to 20 do
begin
SetAspectRatio(Xasp+k*dx,Yasp);
Circle(GetMaxX div 2,GetMaxY div 2,R)
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Procedura SetActivePage.

Postavlja za aaktivnu navedenu stranicu video memorije. Zaglavlje:

```
Procedure SetActivePage(PageNum: Word);
```

Ovde je PageNum – broj stranice.

Procedura može da se koristi samo sa adapterima koji podržavaju višestranični rad (EGA, VGA itd.). Praktično procedura preadresira grafički izlaz u drugu oblast video memorije, međutim, prikaz teksta pomoću Write/WriteLn uvek se obavlja samo na stranicu koja je vidljiva u datom trenutku (aktivna stranica može biti nevidljiva). Numeracija stranica počinje od nule.

Procedura SetVisualPage.

Stvara vidljivom stranicu sa navedenim brojem. Poziva se:

```
Procedure SetVisualPAge(PageNum: Word);
```

Ovde je PageNum – broj stranice.

Procedura može da se koristi samo sa adapterima koji podržavaju višestranični rad (EGA, VGA itd.). Numeracija stranica počinje od nule.

Sledeći program počinje prvo da crta kvadrat u vidljivoj stranici video memorije i kružnicu u nevidljivoj. Posle pritiska na taster Enter dolazi do smene vidljivih straniva video memorije.

```
Uses Graph;
var
d,m,e: Integer;
```

```
s : String;
begin
d := Detect; InitGraph(d, m, '');
e := GraphResult; if e <> grOk then
WriteLn (GraphErrorMsg(e))
else {Nema greške. Proveravamo da li drajver podržava višestranični rad video memorije}
if d in [HercMono,EGA,EGA64,MCGA,VGA] then
begin {Koristimo višestranični režim}
if d<>HercMono then
SetGraphMode(m-1);
{Crtamo u vidljivoj stranici}
Rectangle(10,10,GetMaxX div 2,GetMaxY div 2);
OutTextXY(0,0,'Page 0. Press Enter...');
{Crtamo u nevidljivoj}
SetActivePage (1);
Circle(GetMaxX div 2, GetMaxY div 2, 100);
OutTextXY(0,GetMaxY-10,'Page 1. Press Enter...');
{Prikazujemo stranice; Čekamo na pritisak na taster Enter}
ReadLn;
{Promena stanja stranica}
SetVisualPage(1);
ReadLn;
{Promena stanja stranica}
SetVisualPage (0);
ReadLn;
CloseGraph
```

```
end
else
begin {Drajver ne podržava višestranični režim}
s := GetDriverName; CloseGraph;
WriteLn('Adapter',s,' koristi samo 1 stranicu')
end
end.
```

Obratite pažnju na operator

```
if doHercMono then
SetGraphMode(m-1);
```

Pomoću njega garantovano postavljamo višestranični režim rada kod adaptera EGA, MCGA, VGA. Kako smo već rekli, posle inicijalizacije grafike sa Driver=Detect uspostavlja se režim sa maksimalnim brojem; nabrojani adapteri u tom režimu mogu da rade samo sa jednom grafičkom, da bi obezbedili rad sa dve stranice, treba smanjiti brojčanu oznaku režima.

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (12)

Linije i tačke

Procedura PutPixel.

Iscrtava tačku zadate boje na navedenim koordinatama. Zaglavlje:

```
Procedure PutPixel(X,Y: Integer; Color: Word);
```

Ovde su X, Y- koordinate tačke; Color – boja tačke.

Koordinate se zadaju relativno u odnosu na levi gornji ugao prozora ili, ako prozor nije postavljen, u odnosu na gornji levi ugao ekrana.

Sledeći program periodično iscrtava na ekranu „zvezdano nebo” i zatim ga gasi. Za izlazak iz programa pritisnite bilo koji taster na tastaturi.

```
Uses CRT, Graph;

type

PixelType = record

x, y : Integer; end;

const

N = 5000; {Broj „zvezda”}

var

d,r,e,k: Integer;

x1,y1,x2,y2: Integer;

a: array [1..N] of PixelType; {Koordinate}

begin

{Inicijalizacija grafičkog režima}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e<>grOk then

WriteLn(GraphErrorMsg(e))

else
```

```
begin
{Kreiranje prozora u centru ekrana }
x1 := GetMaxX div 4;
y1 := GetMaxY div 4;
x2 := 3*x1;
y2 := 3*y1;
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Kreiranje i memorisanje koordinata svih „zvezda” }
for k := 1 to N do with a[k] do begin
x := Random(x2-x1);
y := Random(y2-y1)
end;
{Petlja (ciklus) crtanja}
repeat
for k := 1 to N do
with a[k] do {„Paljenje zvezde”}
PutPixel(x,y,white);
if not KeyPressed then
for k := N downto 1 do with a[k] do {„Gašenje zvezde”}
PutPixel(x,y,black)
until KeyPressed;
while KeyPressed do k := ord(ReadKey);
CloseGraph
end;
end.
```

Funkcija GetPixel.

Vraća vrednost tipa Word, koja sadrži boju piksela sa navedenim kordinatama. Zaglavlje:

```
Function GetPixel(X,Y: Integer): Word;
```

Ovde su X, Y – koordinate piksela.

Procedura Line.

Iscrtava liniju sa navedenim koordinatama početka i kraja. Zaglavlje:

```
Procedure Line(X1,Y1,X2,Y2: Integer);
```

Ovde su XL. .Yl – koordinate početka (Xl, Y1) i kraja (X2, Y2) linije.

Linija se iscrtava tekućim stilom i tekućom bojom. U sledećem primeru u centru ekrana se kreira prozor u kojem se zatim iscrtavaju linije na slučajan način. Za izlazak iz programa pritisnite bilo koji taster na tastaturi.

```
Uses CRT, Graph;

var

d,r,e : Integer;

x1,y1,x2,y2: Integer;

begin

{Inicijalizacija grafičkog režima }

d := Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

{Kreiranje prozora u centru ekrana }

x1 := GetMaxX div 4;

y1 := GetMaxY div 4;

x2 := 3*x1;
```

```
y2 := 3*y1;
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Petlja za crtanje slučajnih linija }
repeat
SetColor(succ(Random(16))); {Slučajna boja }
Line(Random(x2-x1), Random(y2-y1),
Random(x2-x1), Random(y2-y1))
until KeyPressed;
if ReadKey=#0 then d:= ord(ReadKey);
CloseGraph
end
end.
```

Procedura LineTo.

Iscrtava liniju od tekućeg položaja pokazivača do njegovog položaja zadatog novim koordinatama. Zaglavlje:

```
Procedure LineTo(X,Y: Integer);
```

Ovde su X, Y – koordinate novog položaja pokazivača, tj. Coordinate drugog kraja linije.

Procedura LineRel.

Iscrtava liniju od tekućeg položaja pokazivača do položaja zadatog priraštajima njegovih koordinata. Zaglavlje:

```
Procedure LineRel (DX, DY: Integer);
```

Ovde su DX, DY- priraštaji koordinata novog položaja pokazivača. U procedurama LineTo i LineRel linija se iscrtava tekućim stilom i tekućom bojom.

Procedura SetLineStyle.

Postavlja novi stil iscrtavanja linija. Zaglavlje:

```
Procedure SetLineStyle(Type,Pattern,Thick: Word)
```

Ovde su Type, Pattern, Thick – odgovarajući tip, mustra (šablon) i debljina linije. Tip linije može se zadati pomoću jedne od sledećih konstanti:

```
const

SolidLn= 0; {Puna linija}

DottedLn= 1; {Tačkasta linija}

CenterLn= 2; {Linija crta-tačka}

DashedLn= 3; {Crtičasta linija}

UserBitLn= 4; {Mustru linije definiše korisnik }
```

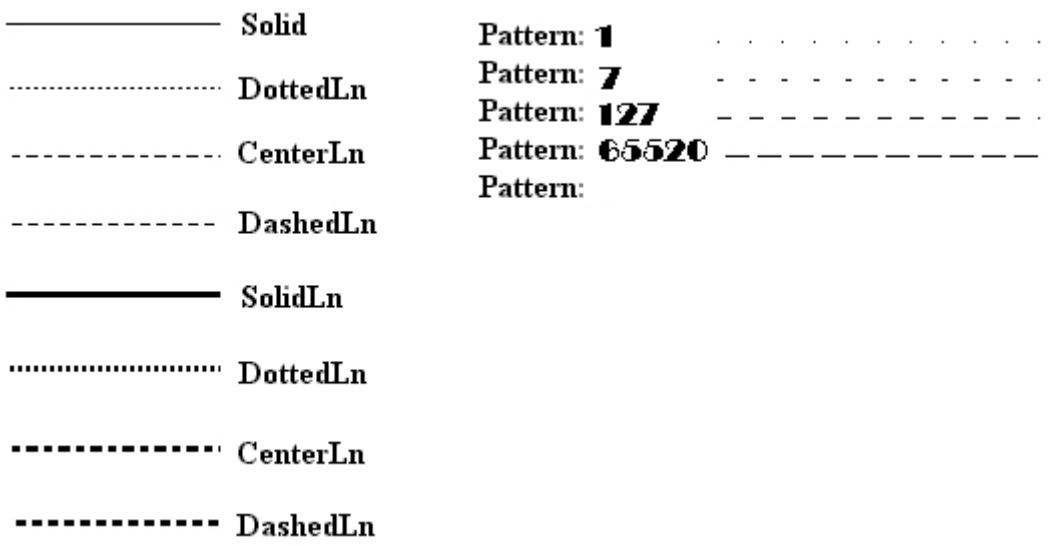
Parametar Pattern uzima se u obzir samo za linije čiji tip definiše korisnik (tj. u slučaju kada je Type = UserBitLn). Pri tome dva bajta parametra Pattern određuju mustru linije: svaki postavlja na jedinicu bit te reči koja odgovara pikselu u liniji, nula bit – pikselu koji ne svetli. Na taj način, parametar Pattern zadaje odsečak linije dužine 16 piksela. Ta mustra se periodički ponavlja po celoj dužini linije.

Parametar Thick može da ima jednu od dve vrednosti:

```
const

NormWidth = 1; {Debljina jedan piksel}

ThickWidth = 3; {Debljina tri piksela}
```

Uočite da se stil linije postavljen procedurom (tekući stil) koristi pri kreiranju pravougaonika, mnogougaonika i drugih geometrijskih figura.

U sledećem primeru se prikazuju linije svih standardnih stilova, zatim se uvodi mustra i iscrtava linija koja koristi tu mustru (slika. 1.). Za izlazak iz programa upišite nulu.



slika 1. Vrste linija

```
Uses CRT, Graph;
const
style: array [0..4] of String [9] = (
'SolidLn ', 'DottedLn ', 'CenterLn 'DashedLn', 'UserBitLn');
var
d,r,e,i,j,dx,dy: Integer;
p: Word;
begin
{Inicijalizacija grafičkog režima}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn (GraphErrorMsg(e))
else
begin
{Izračunavanje pomaka linije}
dx := GetMaxX div 6;
dy := GetMaxY div 10;
```

```
{Iscrtavanje standardnih linija}
for j := 0 to 1 do {Za dve debljine}
begin
for i := 0 to 3 do {Četiri tipa linija}
begin
SetLineStyle(i, 0, j*2+1);
Line(0,(i+j*4+l)*dy,dx,(i+j*4+l)*dy);
OutTextXY(dx+10, (i+j*4+l)*dy,style [i])
end
end;
{Uvođenje muste i crtanje linije}
 j := 0;
dy := (GetMaxY+1) div 25;
repeat
OutTextXY(320,j*dy,'Pattern: ');
GotoXY(50,j+1);
ReadLn(p); if p <> 0 then
begin
SetLineStyle(UserBitLn,p,NormWidth);
Line(440,j*dy+4, 600, j*dy+4);
inc(j)
end
until p = 0;
CloseGraph
end
end.
```

Procedura GetLineSettings.

Vraća tekući stil linije. Zaglavlje:

```
Procedure GetLineSettings(var StyleInfo: LineSettingsType)
```

Ovde je StyleInfo – promenljiva tipa LineSettingsType kojom se vraća tekući stil linije.

Tip LineSettingsType definiše se u modulu Graph na sledeći način:

```
type

LineSettingsType = record

LineStyle: Word; {Tip linije}

Pattern : Word; {Mustra}

Thickness: Word {Debljina}

end;
```

Procedura SetWriteMode.

Postavlja način uzajamnog dejstva linija koje se ponovo iscrtavaju sa onim što već postoji na ekranu. Zaglavlje:

```
Procedure SetWriteMode(Mode);
```

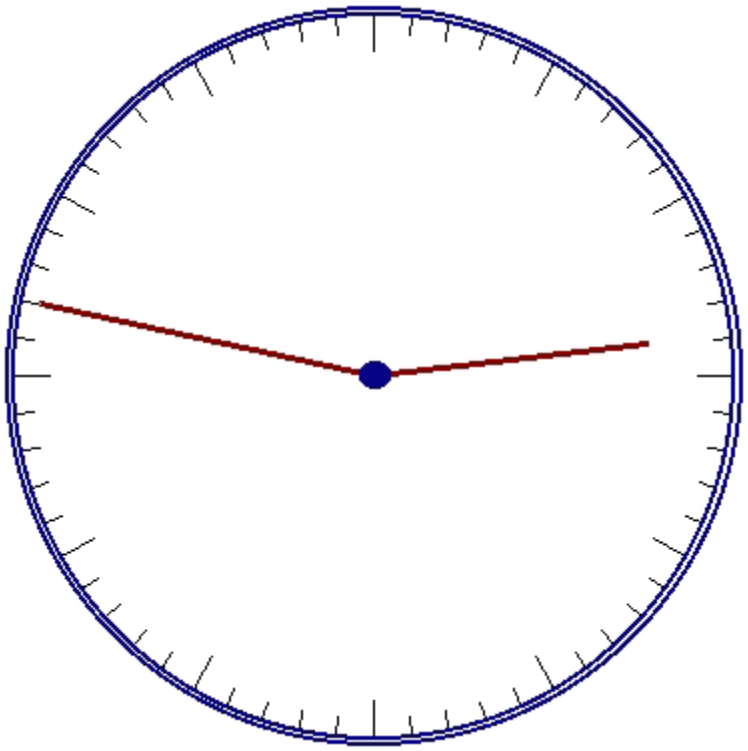
Ovde je Mode – izraz tipa Integer, koji zadaje način uzajamnog dejstva linija koje se iscrtavaju i postojećeg prikaza.

Ako parametar Mode ima vrednost 0, linije koje se iscrtavaju superponiraju se na postojeći prikaza na običan način (za one koji znaju asembler, realizuje se preko instrukcije MOV mikroprocesora). Ako je vrednost 1, superponiranje se ostvaruje primenom logičke operacije XOR (isključivo ILI): u tačkama preseka linija koje se iscrtavaju sa postojećim prikazom na ekranu osvetljenost piksela se invertuje (pretvara u suprotnu vrednost), tako da dva uzastopna iscrtavanja jedne te iste linije ne menjaju izgled ekrana.

Režim postavljen procedurom SetWriteMode, primenjuje se na procedure Drawpoly, Line, LineRel, LineTo i Rectangle. Za zadavanje parametra Mode mogu se koristiti sledeće konstante modula:

```
const

CopyPut = 0;{Superponiranje operacijom MOV}

XORPut = 1;{Superponiranje operacijom XOR}
```

U sledećem primeru na ekranu se simulira rad klasičnog časovnika (slika 2.). Da bi se lakše uočilo ono o čemu je ovde bilo reči, u simulaciji je ritam časovnika usporen 600 puta (uočite operator Delay (100)). Po želji, možete vrlo lako program poboljšati vezivanjem njegovog pokazivanja vremena sa sistemskim satom i dodavanjem sekundare. Za izlazak iz programa pritisnite bilo koji taster na tastaturi.



Slika 2. Časovnik

```
Uses Graph, CRT;

var

d,r,r1,r2,rr,k,

x1,y1,x2,y2,x01,y01: Integer;

Xasp,Yasp : Word;

begin

{Inicijalizacija grafičkog režima}

d := detect; InitGraph(d, r, '');

k := GraphResult; if k <> grOK then

WriteLn(GraphErrorMSG(k))
```

```
else
begin
{Određivanje odnosa strana i razmere ekrana}
x1 := GetMaxX div 2;
y1 := GetMaxY div 2;
GetAspectRatio(Xasp, Yasp);
{Izračunavanje prečnika}
r:= round(3*GetMaxY*Yasp/8/Xasp);
r1 := round(0.9*r); {Podeok za sate}
r2 := round(0.95*r); {Podeok za minute}
{Iscrtavanje časovnika}
Circle(x1,y1,r); {Prva (unutrašnja) kružnica}
Circle(x1,y1,round(1.02*r) ); {Druga kružnica}
for k := 0 to 59 do {Podeoci časovnika}
begin
if k mod 5=0 then
rr := r1 {Podeok za sate}
else
rr : = r2; {Podeok za minute}
{Određivanje koordinata kraja podeoka}
x0l := x1+Round(rr*sin(2*pi*k/60));
y0l := y1-Round(rr*Xasp*cos(2*pi*k/60)/Yasp);
x2 := x1+Round(r*sin(2*pi*k/60));
y2 := y1-Round(r*Xasp*cos(2*pi*k/60)/Yasp);
Line(x01,y01,x2,y2) {Crtanje podeoka}
end;
```

```
{Priprema iscrtavanja kazaljki}
SetWriteMode(XORPut);
SetLineStyle(SolidLn,0,ThickWidth);
{Brojač minuta u jednom satu}
{k = minuti}
r := 0;
{Petlja za crtanje kazaljki}
repeat
for k := 0 to 59 do if not KeyPressed then begin
{Koordinate male kazaljke - sati}
x2 := x1+Round(0.85*r1*sin(2*pi*r/60/12));
y2 := y1-Round(0.85*r1*Xasp*cos(2*pi*r/60/12)/Yasp);
{Koordinate velike kazaljke - minuti}
x01 := x1+Round(r2*sin(2*pi*k/60));
y01 := y1-Round(r2*Xasp*cos(2*pi*k/60)/Yasp);
{Crtanje kazaljki}
Line(x1,y1,x2,y2);
Line(x1,y1,x01,y01) ;
Delay(100); {Za simulaciju realnog režima rada potrebno je postaviti zadršku 60000}
{Za uklanjanje kazaljki sa ekrana iscrtavamo ih još jednom!}
Line(x1,y1,x01,y01);
Line(x1,y1,x2,y2);
{Povećavamo i korigujemo brojač minuta u satu }
inc(r); if r=12*60 then
r := 0
end
```

```
until KeyPressed;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (13)

Mnogougaonici

Procedura Rectangle.

Iscrtava pravougaonik sa navedenim koordinatama temena. Zaglavlje:

```
Procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

Ovde su X1... Y2 - koordinate levog gornjeg (X1, Y1) i desnog donjeg (X2, Y2) temena pravougaonika. Pravougaonik se iscrtava korišćenjem tekuće boje I tekućeg stila linije.

U sledećem primeru na ekranu se iscrtava 10 pravougaonika ugnježdenih jedan u drugi.

```
Uses Graph, CRT;

var

d,r,e,xl,yl, x2,y2,dx,dy: Integer;

begin

{Inicijalizacija grafičkog režima}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e <> grOK then

WriteLn(GraphErrorMsg(e))

else

begin

{Određivanje priraštaja stranica}

dx := GetMaxX div 20;

dy := GetMaxY div 20;
```

```
{Crtanje ugnježdenih pravougaonika }

for d := 0 to 9 do

Rectangle(d*dx,d*dy,GetMaxX-d*dx,GetMaxY-d*dy);

if ReadKey=#0 then d := ord(ReadKey);

CloseGraph

end

end.
```

Procedura DrawPoly.

Iscrtava proizvoljno izlomljenu liniju, zadatu koordinatama tačaka loma (pregiba).

```
Procedure DrawPoly(N: Word; var Points)
```

Ovde je N – broj tačaka loma, uključujući obe krajnje tačke; Points – promenljiva tipa PointType, koja sadrži koordinate tačaka loma.

Koordinate tačaka loma zadaju se parom vrednosti tipa Word: prva vrednost određuje horizontalnu, druga – vertikalnu koordinatu. Za koordinate tačaka loma možemo koristiti sledeće definicije u modulu type:

```
type

PointType = record

x, y : Word

end;
```

Pri iscrtavanju koristi se tekuća boja i tekući stil linije. Evo kako se pomoću ove procedure može na ekranu nacrtati grafik sinusoide:

```
Uses Graph;

const

N = 100; {Broj tačaka grafikona }

var
```

```
d, r, e: Integer;
m : array [O..N+1] of PointType; k : Word;
begin
{Inicijalizacija grafičkog režima }
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Izračunavanje koordinata grafikona }
for k := 0 to N do with m[k] do
begin
x := trunc(k*GetMaxX/N);
y := trunc(GetMaxY*(-sin(2*Pi*k/N)+1)/2)
end;
{Zatvaramo grafik prave linije}
m[succ(N)].x := m[0] .x;
m[succ(n)].y := m[0] .y;
DrawPoly(N + 2, m);
ReadLn;
CloseGraph
end
end.
```

U primeru se za provlačenje prave linije koristi „zatvaranje” loma (pregiba) – tj. prva i poslednja koordinata njenih tačaka pregiba se poklapaju.

Uočite da na broj tačaka loma N – izraz tipa Word, u samoj proceduri se stavlja ograničenje vezano za konačnu veličinu korišćenog bafera. U to se možete uveriti tako što ćete u prethodnom primeru promeniti vrednost N: za N=678 grafik se neće iscrtati na ekranu, a funkcija GraphResult će vratiti vrednost -6 (nema dovoljno memorije za **просмотра областей). U tom smislu, za ovaj program vrednost praga za broj tačaka loma je 679. U isto vreme kod programa

```
Uses Graph;

const

N=510; {Granična vrednost pri kojoj je još uvek vidljiva dijagonalna linija }

var

d,k: Integer;

Coo: array [1..N] of PointType;

begin

d := Detect; InitGraph(d,k,' ') ;

for k := 1 to N do with Coo[k] do

if odd(k) then

begin

X := 0;

Y := 0

end

else

begin

X := GetMaxX;
```

```
Y := GetMaxY

end;

DrawPoly(N,Coo);

ReadLn;

CloseGraph

end.
```

Ta vrednost je jednaka 510. U ovom programu lom se zadaje u obliku višestrukog superponiranja jedne na drugu dijagonalnih linija.

Lukovi, kružnice, elipse

Procedura Circle.

Iscrtava kružnicu. Zaglavlje:

```
Procedure Circle(X,Y: Integer; R: Word);
```

Ovde su X, Y- κoordinate centra; R – poluprečnik (radius) u pikselima.

Kružnica se crta u tekućoj boji. Debljina linije je definisana tekućim stilom, oblik linije je uvek SolidLn. Procedura iscrtava pravilnu kružnicu uzimajući u obzir promenu linearne razmere poluprečnika u zavisnosti od njegovog pravca u odnosu na stranice grafičkog ekrana, tj. uzimanjem u obzir koeficijenta GetAspectRatio. U vezi sa tim parametar R određuje broj piksela u horizontalnom pravcu.

U sledećem primeru u centru ekrana obrazuje se prozor, koji se postepeno ispunjava slučajnim kružnicama. Za izlazak iz programa pritisnite bilo koji taster na tastaturi.

```
Uses Graph, CRT;

var

d,r,e,x,y: Integer;

begin.

{Inicijalizacija grafičkog režima }
```

```
d i= Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Stvaranje prozora u centru ekrana}
x := GetMaxX div 4;
y := GetMaxY div 4;
Rectangle(x,y,3*x,3*y);
SetViewPort(x+1,y+1,3*x-1,3*y-1,ClipOn);
{Petlja za crtanje slučajnih kružnica}
repeat
SetColor(succ(Random(white))); {Slučajna boja}
SetLineStyle(0,0,2*Random(2)+1); { i stil linije}
x := Random(GetMaxX); {Slučajni položaj}
y := Random(GetMaxY); {centra kružnice}
Circle(x,y,Random(GetMaxY div 4));
until KeyPressed;
if ReadKey=#0 then x := ord(ReadKey);
CloseGraph
end
end.
```

Procedura Arc.

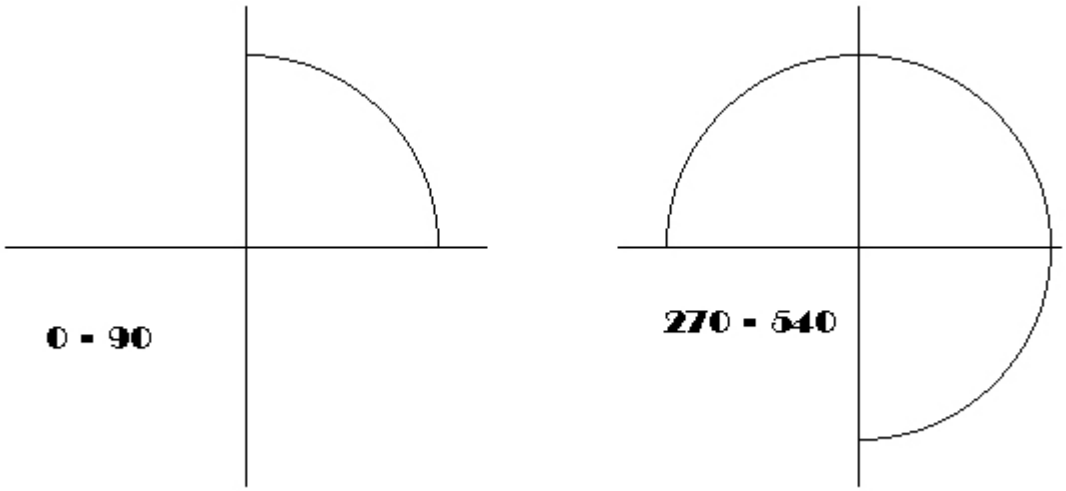
Iscrtava luk. Zaglavlje:

```
Procedure Arc(X,Y: Integer; BegA,EndA,R: Word);
```

Ovde su X, Y – koordinate centra; BegA, EndA – početni i završni ugao luka respektivno; R - poluprečnik.

Uglovi se računaju u smeru suprotnom od kretanja kazaljke na satu i izražavaju se u stepenima. Nulti ugao odgovara horizontalnom pravcu vektora sa leva na desno. Ako je zadata vrednost početnog ugla 0 i završnog - 359, biće iscrtana puna kružnica. Pri crtanju luka koriste se oni isti odnosi što se tiče linije i poluprečnika kao i kod procedure Circle.

Evo kako izgledaju dva luka, jedan sa uglovima 0 i 90, a drugi 270 i 540 stepeni (slika 1.):



Slika 1. Ilustracija procedure Arc

Sledeći program stvara taj prikaz:

```
Uses Graph, CRT;

var

d, r, e : Integer;

Xasp,Yasp: Word;

begin

{Inicijalizacija graičkog režima }
```

```
d := Detect;
InitGraphtd, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
GetAspectRatio(Xasp,Yasp);
{R = 1/5 od vertikalne *razmere *dimenzije ekrana }
r := round(Yasp*GetMaxY/5/XAsp);
d := GetMaxX div 2; {Pomak drugog grafikona }
e : = GetMaxY div 2; {Položaj horizontalne ose }
{Pravljenje levog grafikona }
Line (0,e,5*r div 2,e); {Horizontalna osa }
Line (5*r div 4,e div 2,5*r div 4,3*e div 2) ;
Arc (5*r div 4,e,0,90,R); {Luk}
OutTextXY(0,e+e div 8,'0 - 90'); {Natpis}
{Desni grafikon}
Line (d,e,d+5*r div 2,e);
Line (d+5*r div 4,e div 2, d+5*r div 4,3*e div 2);
Arc (d+5*r div 4,e,270,540,R);
OutTextXY(d,e+e div 8,'270 - 540');
{Čekanje na pritisak na bilo koji taster }
if ReadKey=#0 then d := ord(ReadKey);
```

```
CloseGraph

end

end.
```

Procedura GetArcCoords.

Vraća koordinate tri tačke: centra, početka i kraja luka. Zaglavlje:

```
Procedure GetArcCoords(var Coords: ArcCoordsType);
```

Ovde je Coords – promenljiva tipa ArcCoordsType, u kojoj procedura vraća koordinate centra, početka i kraja luka.

Tip ArcCoordsType je definisan u modulu Graph na sledeći način:

```
type

ArcCoordsType = record

X,Y : Integer; {Koopdinate centra}

Xstart,Ystart: Integer; {Početak luka}

Xend,Yend : Integer; {Kraj luka}

end;
```

Zajedničko korišćenje procedura Arc i GetArcCoords omogućava iscrtavanje spajanja dveju pravih pomoću luka. Obratite pažnju na korekciju dužine poluprečnika u sledećem primeru, u kojem se iscrtava pravougaonik sa zaobljenim temenima.

```
Uses Graph,CRT;

const

RadX = 50; {Horizontalni poluprečnik}

lx = 400; {Širina}

ly = 100; {Visina}

var
```

```
d,r,e: Integer;

coo : ArcCoordsType;

x1,y1: Integer;

xa,ya: Word;

RadY : Integer; {Vertikalni polupre?nik  }

begin

{Inicijalizacija grafi?kog režima}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e <> grOK then

WriteLn(GraphErrorMsg(e))

else

begin

GetAspectRatio(xa,ya) ; {Izra?unavanje odnosa strana }

{Izra?unavanje vertikalnog radiusa i položaja slike s obzirom na odnos
strana ekrana }

RadY := round (RadX *( xa /ya) );

x1 := (GetMaxX-lx) div 2;

y1 := (GetMaxY-2*RadY-ly) div 2;

{Iscrtavanje slike}

Line (x1,y1,x1+lx,y1); {Gornja horizontalna}

Arc (x1+lx,y1+RadY,0,90,RadX) ; {Zaobljenje}

GetArcCoords(coo);

with coo do
```

```
begin

Line(Xstart,Ystart,Xstart,Ystart+ly);

{Desna vertikala}

Arc(Xstart-RadX,Ystart+ly,270,0,RadX);

GetArcCoords (coo);

Line(Xstart,Ystart,Xstart-lx,Ystart);

{Donja horizontala}

Arc(Xstart-lx,Ystart-RadY,180,270,RadX);

GetArcCoords(coo);

Line(Xstart,Ystart,Xstart,Ystart-ly);

Arc(Xstart+RadX,Ystart-ly,90,180,RadX)

end ;

if ReadKey=#0 then d := ord(ReadKey);

CloseGraph

end

end.
```

Procedura Ellipse.

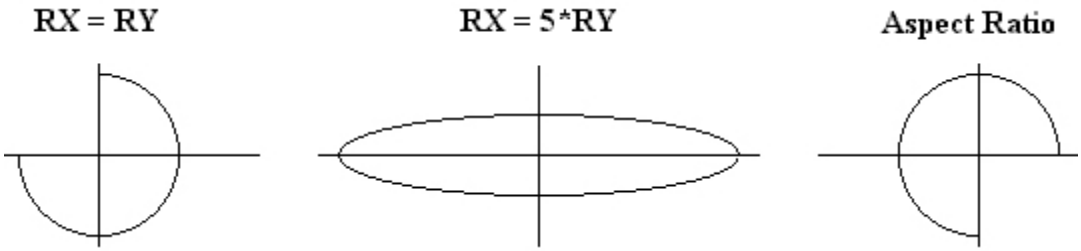
Iscrtava elipsu (eliptični luk). Zaglavlje:

```
Procedure Ellipse(X,Y: Integer; BegA,EndA,RX,RY: Word);
```

Ovde su X, Y – koordinate centra; BegA, EndA – početni i krajnji ugao luka respektivno; RX, RY- horizontalni i vertikalni poluprečnik elipse u pikselima.

Pri iscrtavanju luka elipse koriste se isti odnosi što se tiče linija kao i u proceduri Circle, i isti odnosi što se tiče uglova kao i u proceduri Arc. Ako poluprečnike usaglašavamo uzimanjem u obzir koeficijenta razmere ekrana GetAspectRatio, tada će biti nacrtana pravilna kružnica.

U sledećem programu iscrtavaju se tri elipsasta luka (slika 2.) sa različitim odnosima poluprečnika. Uočite da što je veća rezolucija ekrana to je odnos strana bliži jedinici i usled toga se prvi grafikon manje razlikuje od trećeg.



Slika 2. Elipsasti lukovi

```
Uses Graph, CRT;

var

d,r,e: Integer;

xa,ya: Word;

begin

{Inicijalizacija grafičkog režima}

d := Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOK then

WriteLn(GraphErrorMsg(e))

else

begin

{Prvi grafikon}

OutTextXY(5 0,4 0,'RX = RY'); {Natpis}

Line (0,100,160,100); {Osa X}

Line (80,55,80,145); {Osa Y}

Ellipse (80,100,180,90,40,40);
```

```
{Drugi grafikon}
OutTextXY(260,40,'RX = 5*RY');
Line (190,100,410,100);
Line (300,55,300,145);
Ellipse (300,100,0,359,100,20);
{Treći grafikon}
OutTextXY(465,40,'Aspect Ratio');
Line (440,100,600,100);
Line (520,55,520,145);
GetAspectRatio(xa, ya);
Ellipse (520,100,0,270,40,round(40*(xa/ya)));
if ReadKey=#0 then
 d := ord(ReadKey);
CloseGraph
end
end.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (14)

Boje i palete

Procedura SetColor.

Postavlja tekuću boju za linije i simbole koji se iscrtavaju. Zaglavlje:

```
Procedure SetColor(Color: Word);
```

Ovde je Color – tekuća boja.

Funkcija GetColor. Vraća vrednost tipa Word, koja sadrži kôd tekuće boje. Zaglavlje:

```
Function GetColor: Word;
```

Funkcija GetMaxColor.

Vraća vrednost tipa Word, koja sadrži maksimalni mogući kôd boje, koji može da se koristi kod pozivanja SetColor. Zaglavlje:

```
Function GetMaxColor: Word;
```

Procedura SetBkColor.

Postavlja boju pozadine. Zaglavlje:

```
Procedure SetBkColor(Color: Word);
```

Ovde je Color – boja pozadine.

Za razliku od tekstualnog režima, kod koga boja pozadine može biti samo tamna nijansa, u grafičkom režimu to može biti bilo koja boja. Postavljanje nove boje pozadine brzo menja boju grafičkog ekrana.

Sledeći program demonstrira rad sa procedurom SetBkColor. Program iscrtava deset ugnježdenih pravougaonika a potom se ciklično menja boja pozadine. Za izlazak iz programa dovoljno je pritisnuti bilo koji taster na tastaturi.

```
Uses Graph, CRT;

const
```

```
NC: array [0..15] of String [12] =
('Black','Blue','Green','Cyan','Red','Magenta',
 ' Brown','LightGray','DarkGray','LightBlue',
 'LightGreen1,'LightCyan1,'LightRed',
'LightMagenta','Yellow','White');
var
d, r, e, k, color, dx, dy: Integer;
begin
{Inicijalizacija grafike}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Ispis teksta u centru ekrana}
OutTextXY(200,GetMaxY div 2,'BACKGROUND COLOR');
dx := GetMaxX div 30; {Povećanje dužine}
dy := GetMaxY div 25; {Povećanje visine}
for k := 0 to 9 do{Iscrtavanje 10 pravougaonika}
Rectangle(k*dx,k*dy,GetMaxX-k*dx,GetMaxY-k*dy);
color := black; {Početna boja pozadine}
repeat {Petlja za promenu boje}
SetBkColor(color) ;
```

```
SetFillStyle(0,Color);

Bar(345,GetMaxY div 2,440,GetMaxY div 2+8);

OutTextXY(345,GetMaxY div 2,NC[color]);

delay(1000);

inc(color);

if color > White then

color := Black until KeyPressed;

if ReadKey=#0 then

k := ord(ReadKey);

CloseGraph

end

end.
```

Funkcija GetBkColor.

Vraća vrednost tipa Word, koja sadrži tekuću boju pozadine. Zaglavlje:

```
Function GetBkColor: Word;
```

Procedura SetPalette.

Zamenjuje jednu od boja palete novom bojom. Zaglavlje:

```
Procedure SetPalette(N: Word; Color: ShortInt);
```

Ovde je N – broj boje u paleti; Color – broj novopostavljene boje.

Data procedura radi samo sa adapterima tipa EGA ili VGA. Ne može da se koristi sa IBM8514 ili VGA verzijom koja podržava 256 boja – kod tih adaptera se koristi posebna namenska procedura SetRGBPalette). Prvobitni raspored boja u paleti EGA/VGA odgovarao je redosledu konstanti koje ih opisiju Black,....White, tj. boja sa indeksom 0 - crnoj, 1 - plavoj, 2 - zelenoj itd. Posle pozivanja procedure svi delovi prikaza koji koriste boju sa indeksom N iz palete boja, dobijaju boju Color. Na primer, ako primenimo operator

```
SetPalette(2,White);
```

to će boja sa indeksom 2 (prvobitno to je Cyan) biti zamenjena belom

Sledeći program iscrtava na ekranu niz pravih raznih boja a zatim na slučajan način menja boje palete.

```
Uses Graph, CRT;

var

d,r,e,N,k,color: Integer;

Palette : PaletteTyper;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e <> grOK then

WriteLn(GraphErrorMsg(e))

else

begin

{Biramo debelu neprekidnu liniju}

SetLineStyle(SolidLn, 0, ThickWidth);

GetPalette(Palette) ; {Tekuća paleta}

for Color := 0 to Palette.Size-1 do

begin

SetColor(Color);

Line(GetMaxX div 3,Color*10,2*GetMaxX div 3,Color*10)

end;
```

```
{Menjamo paletu i čekamo na reakciju korisnika}

while not KeyPressed do

for e := 0 to Palette.Size-1 do

SetPalette(e,Random(Palette.Size));

if ReadKey=#0 then d := ord(ReadKey);

CloseGraph

end

end.
```

Procedura GetPalette.

Vraća veličinu i boje tekuće palete. Zaglavlje:

```
Procedure GetPalette(var PaletteInfo: PaletteType);
```

Ovde je PaletteInfo – promenljiva tipa PaletteType, koja vraća veličinu i boje palete.

U modulu Graph definisana je konstanta

```
const

MaxColors =15;
```

i tip

```
type

PaletteType = record

Size : Word; {Broj boja u paleti}

Colors : array [0..MaxColors] of ShortInt

{Numeracija boja palete}

end;
```

Pomoću sledećeg programa možemo prikazati na ekranu brojeve svih mogućih boja tekuće palete.

```
Uses Graph;

var

Palette: PaletteType;

d,r,e,k: Integer;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

GetPalette(Palette); {Dobijanje palete}

CloseGraph; {Vraćanje u tekstualni režim}

with Palette do {Ispisivanje brojeva boja}

for k := 0 to pred(Size) do

Write(Colors[k]:5);

end

end.
```

Procedura SetAllPalette.

Istovremeno menja nekoliko boja palete. Zaglavlje:

```
Procedure SetAllPalette(var Palette);
```

Parametar Palette u zaglavlju procedure opisan je kao netipizirani parametar. Prvi bajt tog parametra mora da sadrži dužinu N palete, ostali N bajtova – broj ponovo postavljenih boja u dijapazonu od -1 do MaxColors. Kôd -1 označava da se odgovarajuća boja polazne palete ne menja.

U sledećem programu vrši se istovremena promena svih boja palete.

```
Uses Graph, CRT;

var

Palette: array [0..MaxColors] of Shortint;

d,r,e,k: Integer;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

{Biramo debele neprekidne linije}

SetLineStyle(SolidLn, 0, ThickWidth);

{Iscrtavamo linije u svim dostupnim bojama}

for k := 1 to GetMaxColor do

begin

SetColor(k);

Line(GetMaxX div 3,k*10,2*GetMaxX div 3,k*10)

end;
```

```
Palette[0] := MaxColors; {Veličina palete}

repeat {Petlja za menjanje palete}

for k := 1 to MaxColors do

Palette[k] := Random(succ(MaxCoLors));

SetAllPalette(Palette)

until KeyPressed;

if ReadKey=#0 then k := ord(ReadKey);

CloseGraph

end

end.
```

Funkcija GetPaletteSize.

Vraća vrednost tipa Integer, koja sadrži veličinu palete (maksimalni broj dostupnih boja). Zaglavlje:

```
Function GetPaletteSize: Integer;
```

Procedura GetDefaultPalette.

Vraća strukturu palete, postavljenu po zahtevu (u režimu samopodešavanja). Zaglavlje:

```
Procedure GetDefaultPalette(var Palette: PaletteType);
```

Ovde je Palette – promenljiva tipa PaletteType (vidi proceduru GetPalette), u kojoj se vraća veličina i boje palete.

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (15)

Procedura SetFillStyle.

Postavlja stil (tip i boju) ispune. Zaglavlje:

```
Procedure SetFillStyle(Fill,Color: Word);
```

Ovde je Fill – tip ispune; Color – boja ispune.

Pomoću ispune možemo prekriti bilo koji fragment prikaza sa periodično ponavljajućim uzorkom. Za navođenje tipa ispune koriste se sledeće unapred definisane konstante:

```
const
EmptyFill = 0;{Ispuna nijansom (nema uzorka)}
SolidFill = 1;{Neprekidna ispuna}
LineFill = 2;{Ispuna -------}
LtSlashFill = 3;{Ispuna ///////}
SlashFill = 4;{Ispuna podebljanim ///}
BkSlashFill = 5;{Ispuna podebljanim \\\}
LtBkSlashFill = 6;{Ispuna \\\\\\\}
HatchFill = 7;{Ispuna +++++++}
XHatchFill = 8;{Ispuna xxxxxxx}
InterleaveFill= 9;{Ispuna pravougaone rešetke}
WideDotFill = 10;{Ispuna retkim tačkama}
CloseDotFill = 11;{Ispuna čestim tačkama}
UserFill = 12;{Uzorak definiše korisnik}
```

Program u sledećem primeru demonstrira sve standardne tipove ispune.

```
Uses Graph, CRT;
var
d,r,e,k,j,x,y: Integer;
```

```
begin
{Inicijalizacija grafike}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
x := GetMaxX div 6;{Položaj grafika}
y := GetMaxY div 5;{na ekranu}
for j := 0 to 2 do{Dva reda}
for k := 0 to 3 do{po četiri kvadrata}
begin
Rectangle((k+1)*x,(j+1)*y,(k+2)*x,(j+2)*y);
SetFillStyle(k+j*4,j+1);
Bar((k+1)*x+1,(j+1)*y+1,(k+2)*x-1,(j+2)*y-1)
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Ako parametar Fill ima vrednost 12 (UserFill), to crtež uzorka je određen programski preko procedure SetFillPattern.

Procedura SetFillPattern.

Postavljanje mustre i boje šrafure. Zaglavlje:

```
Procedure SetFillPattern(Pattern: FillPatternType;Color: Word);
```

Ovde je Pattern – izraz tipa FillPatternType; postavlja mustru crteža za Fill - UserFill u proceduri SetFillStyle; Color – boja ispune.

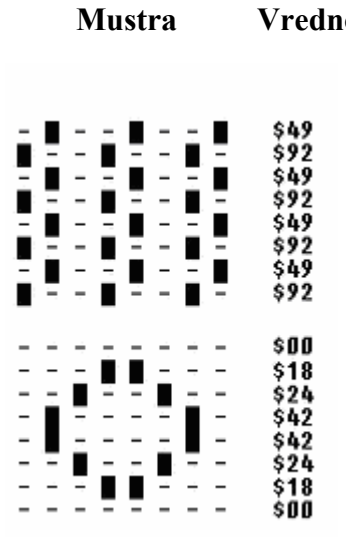
Mustra crteža zadaje se u vidu matrice od 8x8 piksela i može se predstaviti pomoću niza od 8 bajtova sledećeg tipa:

```
type

FillPatternType = array [1..8] of Byte;
```

Svaki razred bilo kog bajta kontroliše osvetljenost piksela, pri čemu prvi bajt određuje 8 piksela prvog reda ekrana, drugi bajt - 8 piksela drugog reda itd..

Na slici 1. prikazan je primer dve mustre popune. Na crtežu crtice označavaju neosvetljene piksele, a pravougaonici osvetljene. Za svakih 8 piksela daje se heksadekadni kôd odgovarajućeg bajta.

Sledeći program popunjava tim mustrama dve pravougaone oblasti na ekranu.



Slika 1. Uzorci ispune i nihovi kodovi

```
Uses Graph, CRT;

const

pattl: FillPatternType= ($49,$92,$49,$92,$49,$92,$49,$92);

patt2: FillPatternType= ($00,$18,$24,$42,$42,$24,$18,$00);

var

d,r,e: Integer;

begin  {Inicijalizacija grafike}

d := Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOk then
```

```
WriteLn(GraphErrorMsg(e))
else
begin
if d=CGA then
SetGraphMode (0) ; {Postavljamo boju za CGA}
SetFillStyle(UserFill,White);
{Levi gornji kvadrat}
SetFillPattern(Patt1,1);
Bar(0,0,GetMaxX div 2, GetMaxY div 2);
{Desni donji kvadrat}
SetFillPattern(Patt2,2);
Bar(GetMaxX div 2,GetMaxY div 2,GetMaxX,GetMaxY);
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Ako je prilikom pozivanja procedure naveden nedozvoljeni kôd boje, poziv procedure se ignoriše i pamti se ranije postavljena mustra popune. Specijalno, ako je u prethodnom primeru obrišemo operator

```
if d=CGA then
SetGraphMode(0);
```

koji postavlja režim rada sa bojom CGA -kartice, biće prikazana dva jednaka pravougaonika, jer naredba

```
SetFillPattern(patt2, 2);
```

ima nedozvoljeni kôd boje za taj režim rada i ignoriše se. Međutim, ovo se ne odnosi na proceduru SetFillStyle za vrednosti parametra Fill u opsegu od 0 do 11: program će normalno raditi i u režimu visoke rezolucije CGA-kartice, pri čemu se sve boje palete, osim boje pozadine zamenjuju belom.

Procedura GetFillPattern.

Vraća mustru ispune, koja je ranije postavljena procedurom SetFillPattern. Zaglavlje:

```
Procedure GetFillPattern(var Pattern: FillPatternType);
```

Ovde je Pattern – promenljiva tipa FillPatternType, kojom se vraća mustra popune.

Ako program ne postavlja mustru popune pomoću procedure SetFillPattern, niz Pattern se popunjava bajtovima vrednosti 255 ($FF).

Procedura GetFillSettings.

Vraća tekući stil popune. Zaglavlje:

```
Procedure GetFillSettings(var PattInfo: FillSettingsType);
```

Ovde je PattInfo – promenljiva tipa FillSettingsType, kojom se vraća tekući stil popune.

U modulu Graph definisan je tip:

```
type

FillSettingsType = record

Pattern: Word; {Образец}

Color : Word {Цвет}

end;
```

Polja Pattern i Color u tom zapisu imaju ista značenja kao i analogni parametri kod pozivanja procedure SetFillStyle.

Procedura SetRGBPalette.

Postavlja gamu boja pri radu sa monitorom IBM 8514 и VGA karticom. Zaglavlje:

```
Procedure SetRGBPalette(ColNum,RedVal, GreenVal,BlueVal:Integer);
```

Ovde je ColNum – broj boje; RedVal, GreenVal, BlueVal – izrazi tipa Integer, koji postavljaju intenzitet respektivno crvene, zelene i plave komponente boje.

Ta procedura radi samo sa monitorom IBM 8514, kao i sa karticom VGA, i koristi video memoriju kapaciteta 256 Kb. U prvom slučaju parametar ColNum zadaje vrednost u opsegu 0...255, u drugom u opsegu 0...15. Za postavljanje intenziteta koristi se 6 starijih bita mlađeg bajta svakog od parametara RedVal, GreenVal, BlueVal.

U sledećem primeru u centru ekrana iscrtava se pravougaonik bele boje, posle čega se boja menja na slučajan način pomoću procedure SetRGBPalette. Za izlazak iz programa potrebno je pritisnuti bilo koji taster na tastaturi.

```
Uses Graph,CRT;

var

Driver, Mode, Err, xl, yl: Integer;

begin

{Inicijalizacija grafike}

Driver := Detect;

InitGraph(Driver, Mode, '');

Err := GraphResult;

if ErroO then

WriteLn(GraphErrorMsg(Err))

else if Driver in [IBM8514, VGA] then

begin

{Iscrtavamo pravougaonik u centru ekrana}

x1 := GetMaxX div 4;

y1 := GetMaxY div 4;

SetColor(lS);

Bar(x1,y1,3*x1,3*y1);

{Menjamo belu boju u slučajnu}

while not KeyPressed do

SetRGBPalette(15,Random(256),Random(256),Random(256));

CloseGraph

end

else

begin
```

```
CloseGraph; .

WriteLn('Adapter ne podrzava ' , 'RGB-rezim upravljanja bojama')

end

end.
```

Procedura FloodFill.

Popunjava proizvoljnu zatvorenu figuru koristeći tekući stil popune (mustra i boja). Zaglavlje:

```
Procedure FloodFill(X,Y: Integer; Border: Word);
```

Ovde su X, Y- koordinate bilo koje tačke unutar zatvorene figure; Border – boja granične linije.

Ako figura nije zatvorena, popuna se „razliva" po celom ekranu.

Treba napomenuti da se u proceduri upotrebljeni algoritam provere granice zatvorene figure ne odlikuje savršenstvom. Konkretno, ako se iscrtava uzastopce dva prazna reda, popunjavanje se prekida. Ta situacija obično nastaje pri popunjavanju malih figura pomoću tipa LtSlashFill. U Borlandovom uputstvu za Turbo Pascal preporučuju, ako je to moguće, da se umesto procedure FloodFill koristi procedura FillPoly (popunjavanje pravougaonika).

Sledeći program demonstrira popunjavanje slučajnih kružnica. Prvo se u centru ekrana crta prozor, u kojem se popunjava mali pravougaonik. Deo pravougaonika ostaje nepopunjen, u šta se možete uveriti jer program prekida rad i čeka da bude pritisnut taster Enter. Zatim se obavlja popunjavanje slučajnih kružnica sve dok se ne pritisne neki od tastera na tastaturi. Uočite da se pravougaonik popunjava u potpunosti ako se umesto tipa LtSlashFill (kosa šrafura linijama normalne debljine) koristi SlashFill (šrafura podebljanim linijama).

```
Uses Graph, CRT;

var

d, r, e, x, y, c : Integer;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult;
```

```
if e <> grOk then . . WriteLn(GraphErrorMsg(e))
else
begin
{Kreiranje pravougaonog prozora}
x := GetMaxX div 4;
y. := GetMaxY div 4;
Rectangle(x,y,3*x,3*y);
SetViewPort(x+1,y+1, 3*x-1,3*y-1,ClipOn);
{Demonstriranje ispune malog pravougaonika}
SetPillStyle(LtSlashFill,GetMaxColor);
Rectangle(0,0,8,20); FloodFill(1,1,GetMaxColor);
OutTextXY(10,25,'Press Enter...');
ReadLn; {Čekamo da bude pritisnut Enter}
{Iscrtavamo kružnice sve dok ne bude pritisnut neki taster }
repeat
{Na slučajan način određujemoi stil ispune}
SetFillStyle(Random(12),Random(GetMaxColor+1));
{Zadajemo koordinatu centra i boju kružnice}
x := Random (GetMaxX div 2);
y := Random (GetMaxY div 2);
c := Random (succ(GetMaxColor));
SetColor(c);
{Iscrtavamo i bojimo kružnicu}
Circle(x, y, Random(GetMaxY div 5));
FloodFill (x, y, c)
until KeyPressed;
```

```
if ReadKey=#0 then

x := ord(ReadKey);

CloseGraph

end

end.
```

Процедура Bar.

Заполняет прямоугольную область экрана. Заголовок:

```
Procedure Bar(X1,Y1,X2,Y2: Integer);
```

Здесь XJ...Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов закрашиваемой области.

Процедура закрашивает (но не обводит) прямоугольник текущим образцом узора и текущим цветом, которые устанавливаются процедурой SetFillStyle.

Следующая программа дает красивые цветовые эффекты (закраска случайных прямоугольников).

```
Uses Graph, CRT;

var

d, r, e : Integer;

begin

{Inicijalizacija grafike}

d : = Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

{Kreiramo prozor u centru ekrana}

d := GetMaxX div 4;

r := GetMaxY div 4; Rectangle(d,r,3*d,3*r);

SetViewPort(d+1,r+1,3*d-1,3*r-1,ClipOn);
```

```
{Petlja za iscrtavanje i bojenje slučajnih mnogougaonika}
repeat
SetFillStyle(Random(12),Random(succ(GetMaxColor)));
Bar(Random(Ge tMaxX),Random(Ge tMaxY),
  Random(Ge tMaxX),Random(Ge tMaxY));
until KeyPressed;
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Pripremio Dragan Marković

Uvod u programiranje u Turbo Pascalu 7 (16)

Procedura Bar3D.

Iscrtava trodimenzionalni prikaz paralelopipeda i ukrašava njegovu prednju stranu. Zaglavlje:

```
Procedure Bar3D (X1,Y1,X2,Y2,Depth: Integer; Top: Boolean);
```

Ovde su X1... Y2 – κoopdinate levog gornjeg (X1, Y1) i desnog donjeg (X2, Y2) ugla prednje strane; Depth – treća dimenzija trodimenzionalnog prikaza („dubina") u pikselima; Top – način prikaza gornje strane.

Ako parametar Top ima vrednost True, gornja strana paralelopipeda se iscrtava, u suprotnom – ne iscrtava se (ta varijanta se koristi za prikazivanje dva paralelopipeda koji su jedan na drugom, pogledaj sledeći primer). Za vrednost tog parametra može se iskoristiti jedna od sledećih konstanti, definisanih u modulu Graph:

```
const

TopOn = True;

TopOff = False;
```

Pri iscrtavanje koristi se tekući stil linije (SetLineStyle) i tekuća boja (SetColor). Prednja strana se popunjava tekućim stilom popune (SetFillStyle).

Procedura se obično koristi za pravljenje stubičastih dijagrama. Treba uzeti u obzir da je paralelopiped „prozračan", tj. kroz njegove neobojene strane mogu se videti drugi elementi prikaza.

Sledeći program ilustruje različite aspekte primene procedure Bar3D.

```
Uses Graph,CRT;

var

d, r, e: Integer;

begin

{Inicijalizacija grafike}

d := Detect;

Ini-tGraph(d, r, ' ') ;
```

```
e := GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Stubac sa gornjom granicom:}
Bar3D (80, 100, 120, 180, 15, TopOn);
{Stubac bez gornje granice:}
Bar3D (150, 150, 190, 180, 15, TopOff);
{Taj stubac "stoji" na sledećem i proziran je:}
Bar3D (230, 50, 250, 150, 15, TopOn);
Bar3D (220, 150, 260, 180, 15, TopOn);
Bar3D (300, 150, 340, 180, 15, TopOff);
SetLineStyle(3,0,1);
SetColor(Yellow);
SetFillStyle(LtSlashFill,Yellow);
Bar3D (300, 50, 340, 150, 15, TopOn);
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph;
end
end.
```

Procedura Fill Poly.

Ograđuje linijom i boji zatvoreni poligon. Zaglavlje:

```
Procedure FillPoly(N: Word; var Coords);
```

Ovde je N – broj temena zatvorenog poligona (mnogougla); Coords – promenljiva tipa PointType, koja sadrži koordinate temena.

Koordinate temena zadaju se parom vrednosti tipa Integer: prva vrednost određuje horizontalnu, druga – vertikalnu koordinatu. Za njih se može iskoristiti sledeći u modulu definisani tip:

```
type

PointType = record

x, y : Integer

end;
```

Stil i boja linije konture zadaju se procedurama SetLineStyle i SetColor, tip i boja ispune - procedurom SetFillStyle.

U sledećem primeru na ekranu se iscrtavaju mnogouglovi slučajno obojeni.

```
Uses Graph, CRT;

var

d, r, e: Integer;

p : array [1..6] of PointType; n, k : Word;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, ' ') ;

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

{Kreiramo prozor u centru ekrana}

d := GetMaxX div 4;

r := GetMaxY div 4;

Rectangle(d,r,3*d,3*r);

SetViewPort(d+l,r+l,3*d-l,3*r-l,ClipOn);

{Petlja za iscrtavanje slučajno obojenih mnogougaonika}

repeat
```

```
{Biramo slučajnu boju i uzorak)
SetFillStyle(Random(12),Random(succ(GetMaxColor)));
SetColor (Random(succ(GetMaxColor)));
{Dodeljujemo slučajne koordinate}
n := Random (4) + 3 ; for k := 1 to n do with p[k] do
begin
x := Random (GetMaxX div 2);
y := Random (GetMaxY div 2)
end;
FillPoly (n, p) {Iscrtavamo i bojimo}
until KeyPressed;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Procedura FillEllipse.

Ograđuje linijom i boji elipsu. Zaglavlje:

```
Procedure FillEllipse(X,Y,RX,RY: Integer);
```

Ovde su X, Y – koordinate centra; RX, RY- horizontalni i vertikalni radius elipse u pikselima.

Elipsa se ograđuje linijom, zadatom procedurama SetLineStyle i SetColor, i ispunjava se korišćenjem parametara postavljenih procedurom SetFillStyle.

Procedura Sector.

Iscrtava i ispunjava elipsasti segment. Zaglavlje: Procedure Sector(X,Y: Integer; BegA,EndA,RX,RY: Word);

Ovde su BegA, EndA – odgovarajući početni i krajnji ugao elipsastog segmenta. Ostali parametri su analogni parametrima procedure FillEllipse.

U sledećem programu na ekranu se iscrtavaju elipse i sektori obojeni bojama po slučajnom izboru. Za izlazak iz programa pritisnite bilo koji taster.

```
Uses Graph, CRT;

var

d, r, e : Integer;

begin

{Inicijalizacija grafike}

d := Detect; InitGraph(d, r, '');

e := GraphResult; if e <> grOk then

WriteLn(GraphErrorMsg(e))

else

begin

{Kreiramo prozor u centru ekrana}

d := GetMaxX div 4;

r := GetMaxY div 4;

Rectangle(d,r,3*d,3*r);

SetViewPort(d+1,r+1,3*d-1,3*r-1,ClipOn);

{Petlja iscrtavanja}

repeat

SetFillStyle(Random(12), Random(succ(GetMaxColor)));

SetColor (Random(succ(GetMaxColor)));

Sector(Random(GetMaxX div),Random(GetMaxY div 2),

Random(360),Random(360),Random(GetMaxX div 5),

Random(GetMaxY div 5));

FillEl.lipse (Random (GetMaxX div 2),
```

```
Random(GetMaxY div 2),Random(GetMaxX div 5),
Random(GetMaxY div 5))
until KeyPressed;
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

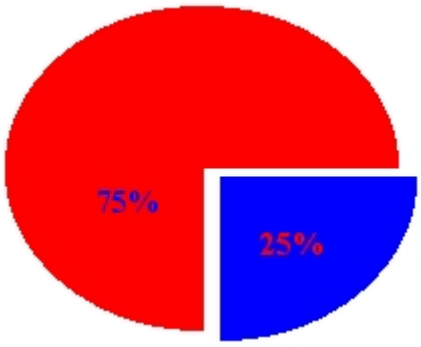
Procedura PieSlice.

Iscrtava i ispunjava segment kružnice. Zaglavlje:

```
Procedure PieSlice(X,Y: Integer; BegA,EndA,R: Word);
```

Za razliku od procedure Sector, navodi se samo jedan horizontalni poluprečnik R, ostali parametri su analogni parametrima procedure Sector.

Segment se oivičava linijom zadatom procedurama SetLineStyle i SetColor, i ispunjava pomoću parametara definisanih procedurom SetFillStyle. Proceduru je zgodno koristiti pri konstruisanju kružnih dijagrama, kao, na primer, u sledećem programu (slika 1.).



Slika 1. Ilustracija procedure PieSlice

```
Uses Graph, CRT;
var
d, r, e : Integer;
begin
```

```
{Inicijalizacija grafičkog režima}
d := Detect;
InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Iscrtavanje malog segmenta}
SetFillStyle(WideDotFill, White);
PieSlice(GetMaxX div 2+5,GetMaxY div 2+4,270,360,100);
{Iscrtavanje velikog segmenta}
SetFillStyle (SolidFill, Red);
PieSlice (GetMaxX div 2,GetMaxY div 2, 0,270,100).;
{Ispisivanje naslova}
OutTextXY (GetMaxX div 2+90,GetMaxY div 2+70, '25%');
OutTextXY(GetMaxX div 2-50,GetMaxY div 2-20, '75%');
{Čekamo da bude pritisnut bilo koji taster}
if ReadKey=#0 then d := ord(ReadKey);
Close,Graph
end
end.
```

Pripremio Dragan Marković