Vilniaus universitetas
Matematikos ir informatikos fakultetas
Programų sistemų katedra

I N F O R M A T I K A

T u r b o P a s k a l i s 7.0

Doc. V. Undzėno paskaitų konspektai
(skaitoma Fizikos fakulteto 2 kurso studentams)

Literatūra

1. V. Tumasonis. Paskalis ir Turbo Paskalis 7.0. - Vilnius, 1993, 381 psl.
2. A. Vidžiūnas, J.Blonskis. Turbo Paskalis 7.0. Vartotojo vadovas. - Kaunas: Smaltija, 1999, 310 psl.
3. A. I. Marčenko, L. A. Marčenko. Programmirovanije v srede TURBO PASCAL 7.0. - Moskva, “Binom universal”, 1997, 495 psl.
4. A.Vidžiūnas. Delphi 5. Programavimas ir vaizdiniai komponentai. - Kaunas: Smaltija, 2001, 322 psl.

Vilnius - 2001

T U R I N Y S

i. Bendrieji informatikos klausimai

Kas yra informacija? Skaičiai, tekstai, garsai, paveikslai, skulptūros, kt. Informacijos perdavimo būdai: knygos, telefonas, radijas, televizija, kompiuterių tinklai (einama link viskam bendro duomenų perdavimo tinklo). **Problema** - kryptingas informacijos apdorojimas: ją reikia sudėlioti (kaupti) taip, kad galėtume greičiausiai susirasti mums reikalingą informaciją (rūšiavimas pagal autoriaus pavardę, mokslo sritis, bazinius žodžius, kt.). Kompaktiškas saugojimas, greitas perdavimas.

Informatika - mokslas apie kompiuterius ir jų taikymus (mokslas apie informacijos kaupimo ir apdorojimo metodus, informacines technologijas, techninių ir programinių priemonių kūrimą įvairiems reikalams, kt.).

Kompiuterių istorija: jų kartos, universalūs ir specializuoti, skaitmeniniai ir analoginiai.

Kompiuterio struktūra: įrenginių paskirtis, ryšiai tarp jų, kompiuterių architektūra. IBM PC sudėtis, įrenginių charakteristikos.

Informacijos vaizdavimas kompiuteriuose: 2 (dvejetainė) skaičiavimo sistema ir jos vartojimo priežastys. Ryšys tarp 2 skaič. sist. ir 10 skaič. sist. Apie 8 skaič. sist. ir 16 skaič. sist., kodėl jų prireikė. Skaičių vaizdavimas fiksuotu ir slankiu kableliu (byte(1 baitas), shortint(1b), word(2b), integer(2b), longint(4b), real(6b), single(4b), double(8b), extended(10b), comp(8b)). Simboliai ir jų kodavimas (char(1b), string(iki 255b)).

Programinis kompiuterių valdymo principas. Programa - tai komandų seka. Mašininės komandos struktūra.

Algoritmas: kokie veiksmai ir kokia eilės tvarka turi būti atlikti, norint išspręsti kažkokį uždavinį. Nuosekli veiksmų seka.

Programavimo automatizavimas: operacinės sistemos, programavimo kalbos, programų paketai ir t.t.

Komandinis ir grafinis interfeisai: dialogo tarp žmogaus ir kompiuterio palaikymo būdai. Pvz., MS DOS aplinka, NORTON COMMANDER, WINDOWS, LINUX. Dažnai sutinkama santrumpa GUI – Graphic User Interface (grafinis vartotojo interfeisas).

Informatikos standartai: susitarimai laikytis vienodos tvarkos, taisyklių. Pvz., OSI (Open System Interconnection) modelis - tai standartų rinkinys, kurio reikia laikytis, kad skirtingų rūšių kompiuteriai galėtų palaikyti ryšį; protokolai - standartų rinkiniai, kurių laikomasi perduodant duomenis kompiuterių tinklais. ISO - tarptautinė standartizacijos organizacija, yra ir kitos: CEN, ETSI, t.t.

ii. Trumpai apie kompiuterių programinę įrangą. Paskalio programų apdorojimo etapai

Operacinė sistema (OS) - tai gatavų programų rinkinys, padedantis vartotojui prižiūrėti bei eksploatuoti kompiuterį. Kompiuterių gamintojai, programinės įrangos kūrėjai pateikia vartotojams OS užrašytą diske, pastovioje atmintyje, kt. Pvz., MS DOS, jo komponentės BIOS, COMMAND.COM, AUTOEXEC.BAT, "draiveriai" (tvarkyklės), NORTON COMMANDER; WINDOWS; UNIX, LINUX ir t.t. Įvairiems darbams palengvinti yra kuriama pagalbinė programinė įranga (programavimo kalbų transliatoriai, duomenų bazių valdymo sistemos, tekstų rengimo, e-pašto, naršymo po internetą programos, kt.). Čia akcentuosime tik tuos komponentus, kurie reikalingi programuojant Turbo Paskaliu.

Paskalio kalba parašyta programa - tai tekstas iš įvairių simbolių. Pirmas uždavinys žmogui yra perkelti tokios programos tekstą į kompiuterio atmintį. Šiam tikslui kompiuteryje turi būti pagalbinė programa - tekstų redaktorius-tvarkytojas (EDIT, KEDIT, NOTEPAD ar kt.; plačiai žinomas WORD šiems tikslams netinka). Tekstų redaktorių teikiamos paslaugos yra labai plačios: įterpti ar ištrinti simbolį, eilutę, perkelti teksto dalį iš vienos vietos į kitą, kopijuoti, kt.

Kadangi kompiuteris gali veikti tik pagal mašininėmis komandomis išreikštą programą, todėl būtinas antrasis etapas. Jo metu įvesta į kompiuterio atmintį teksto pavidalo Paskalio programa pertvarkoma į mašininių komandų pavidalą. Tą atlieka kita pagalbinė programa - Paskalio kompiliatorius (kompiliatorius yra viena iš transliatorių rūšių). Iš programos **pradinio modulio (.PAS)** gaunamas tarpinis rezultatas - **objektinis modulis**. Jei kompiliatorius aptiko klaidų Paskalio programos pradiniame tekste (modulyje), reikia grįžti prie pirmojo etapo, tekstų redaktoriumi ištaisyti Paskalio programą (pradinį modulį) ir vėl kartoti antrąjį etapą.

Kompiliavimo metu gautas objektinis modulis yra programa mašinine kalba, tačiau dar nevisiškai parengta darbui. Reikalas tas, kad viena Paskalio programa gali susidėti iš kelių modulių, kurių kiekvienas kompiliuojamas atskirai. Reikalingus objektinius modulius surinkti į vientisą programą trečiojo etapo metu imasi ryšių redaktorius (komponuotojas, LINK). Ši pagalbinė programa jau sukuria vykdymui parengtą vartotojo programą mašinine kalba - **vykdomąjį modulį (.EXE)**.

Pertvarkius Paskalio programą į mašininį pavidalą (suformavus programos kodą), ketvirtojo etapo metu belieka ją vykdyti. Netgi kompiuteriui dirbant pagal vartotojo sudarytą programą, į šį procesą gali įsiterpti OS.

TURBO PASCAL (trumpai - TP) yra tokia programavimo sistema (aplinka), kurioje galima atlikti visų anksčiau minėtų pagalbinių programų veiksmus. Šiuo metu plačiai vartojama TURBO PASCAL 7.0 versija. TP - tai platus pagalbinių programų ir kitokių priemonių (simbolių šriftų, monitorių tvarkyklių, procedūrų/funkcijų bibliotekų ir t.t.) rinkinys, užrašytas kompiuterio diske. Prisiminkime: IDE - Integrated Development Environment TURBO PASCAL 7.0 (integruota programavimo aplinka). Paskalio programinėje įrangoje yra, pvz., tokie failai: TURBO.EXE, *.TPU, *.BGI , *.CHR ir kt. (* - tai kažkoks žodelis, pvz., GRAPH, EGAVGA, GOTHIC).

Tai klasikinė programų rengimo technologija.

Programų rengimo technologijos yra tobulinamos. Šiandien jau yra vaizdinio programavimo technologija, įgalinanti pasiekti aukštesnį programuotojų darbo našumą, tačiau reikalaujanti gilesnių žinių apie kompiuterius ir programavimo priemones. Tam tikslui yra *objektinė Paskalio kalba* ir vaizdinė aplinka *Delphi 5*. Joms įsisavinti būtini programavimo Paskalio kalba pagrindai.

Išmokti programavimo principų, rašyti kompiuterių programas Paskalio kalba ir dirbti TURBO PASCAL 7.0 aplinkoje yra šios mokomosios disciplinos uždavinys.

1. ĮVADINĖ DALIS

1.1. Paskalio leksika

Abėcėlė:

1) A - Z, a - z	raidės;
2) 0 - 9	skaitmenys;
3) + - * / = < > [ ] . , ; : ( ) @ $ # { }	specialieji simboliai;
4) <> <= >= := .. (* *) (. .)	simbolių kombinacijos.

Baziniai žodžiai, vardai (identifikatoriai): rašant Paskalio programas vartojami **baziniai žodžiai,** kurie turi apibrėžtą prasmę. Jie yra rezervuoti, t.y. kitiems tikslams (jais vadinti kintamuosius, kt.) jų vartoti negalima. Pavyzdžiui, *begin, end, for, if, real, var* ir t.t.

Vardus pasirenka pats programuotojas, ir jie yra reikalingi pasivadinti kintamiesiems, funkcijoms, vardinėms konstantoms, unikaliam duomenų tipui, procedūroms, funkcijų ir procedūrų bibliotekoms. Reikalavimai vardams:

- pirmas vardo simbolis turi būti raidė, kiti - tik raidės arba skaitmenys;
- viduje neleistini tarpai;
- negali sutapti su baziniu žodžiu;
- ne ilgesnis kaip 63 simboliai;
- viduje leistinas pabraukimo '_ ' simbolis.

Vardų pavyzdžiai: *kaina , prekes_kaina , x3 , a23k , qq , alfa* .
Programoje kiekvienas vardas aprašomas, t.y. nurodoma, kokiam duomeniui ar kitokiam dalykui (pvz., kintamajam, funkcijai, procedūrai, kt.) pavadinti jis yra skirtas.

Bazinius žodžius ir vardus galima rašyti didžiosiomis arba mažosiomis raidėmis. Paskalyje tai neturi reikšmės.

Standartiniai vardai yra tokie, kurių prasmė (reikšmė) iš anksto žinoma. Pvz.:

- *false, true, maxint*	konstantų vardai;
- *boolean, integer, byte, text* ir kt.	nurodo duomenų tipą;
- *input, output*	failų vardai;
- *abs, sin, cos, ln, exp* ir kt.	funkcijų vardai;
- *read, write, clrscr, assign, initgraph* ir kt.	procedūrų vardai.

Skaičiai. Rašant programas arba įvedant duomenis jau dirbančiai programai, dažnai reikia skaičių. Jie gali būti sveikieji ir realieji.

Sveikieji (integer) skaičiai rašomi mums įprastu pavidalu: 5 -38 154.
Jie gali būti užrašomi ir šešioliktaine skaičiavimo sistema, priekyje, kaip šešioliktainių skaičių požymį, naudojant dolerio ženklą $: $3A $FFC $B38.

Realieji skaičiai rašomi su tašku, skiriančiu sveikąją dalį nuo trupmeninės: 2.38 -75.86 25.0 0.37 (nuliai būtini). Kitoks jų užrašymo būdas: 2.38E+0 -7.586E+1 2.5E+1 3.7E-1 4.36294E-28 1.53E+17. Pastarasis būdas įgalina užrašyti labai didelius arba labai mažus skaičius.

Simbolių eilutės rašomos tarp apostrofų (viengubų kabučių). Pvz.:

'VILN' eilutės ilgis - 4 simboliai,
'JON IR PETR' eilutės ilgis - 11 simbolių (tarpas taip pat simbolis)
'1996 kovo 27 d.' eilutės igis - 15 simbolių,
'Firma "SIGMA"' eil. viduje prireikus apostrofo, jis rašomas du kartus.

Komentarai programose. Jie leistini bet kurioje programos vietoje, kur gali būti tarpas. Pvz.: { bet koks tekstas } arba (* bet koks tekstas *) . Kaip matome, komentarams būtini specialūs skliaustai.

Skyryba. Tarp programos elementų (vardų, skaičių, operacijos ženklų, kt.) gali būti bet koks tarpų kiekis. Vienoje programos eilutėje galima rašyti kelis sakinius. Ilgas sakinys gali užimti kelias eilutes. Programas reikia rašyti vaizdžiai, kad jose lengvai galėtume išskirti, pvz., ciklus (kur jų pradžia ir pabaiga), procedūras, kitokius blokus !

1.2. Duomenų tipo sąvoka

Programos apdoroja kažkokius duomenis. Dažniausiai susiduriame su sveikaisiais ir realiaisiais skaičiais, tekstais. Duomenų tipas suprantamas taip: tai visuma reikšmių, kurias gali įsiminti kintamasis ir su kuriomis galima atlikti nustatytus veiksmus.

Duomenų tipas programose nurodomas standartiniais vardais: *integer, real, longint, char, boolean* ir kt. Jei programuotojui prireikia nuosavo (savo sugalvotu pavadinimu) duomenų tipo, jį būtina aprašyti programos TYPE skyriuje. Pvz., tipas *saldainis* aprašomas taip:

```
TYPE  saldainis = ( irisas, karamele, bonbonke, sokoladas );
        sveikasis = integer;
```

Duomenų tipai skirstomi į skaliarinius (*integer, real, char,* kt.) ir struktūrinius (masyvai, simbolių eilutės, aibės, įrašai, objektai).

1.3. Konstantos, kintamieji, žymės

Tai programų elementai. Jie suprantami taip:

konstanta - tai reikšmė, kuri programos bėgyje nekinta (skaičius, simbolių eilutė, loginis dydis). Konstantai saugoti kompiuterio atmintyje skiriama atitinkamo dydžio vieta (kažkiek baitų). Programose jos užrašomos akivaizdžiai arba nurodomos vardu (vardinė konstanta). Vardines konstantas būtina aprašyti, kad žinotume jų reikšmę (tai atliekama programos CONST skyriuje);

kintamasis - visada žymimas vardu ir programos bėgyje gali keisti savo reikšmę. Jis suprantamas kaip tam tikro dydžio atminties sritis, kurioje galima įsiminti nustatyto tipo reikšmę. Atminties srities dydis priklauso nuo kintamojo tipo. Programoje visi kintamieji turi būti aprašyti, nurodytas jų tipas (tai atliekama programos VAR skyriuje);

žymė - sveikasis skaičius (ne didesnis kaip keturženklis) arba vardas, kuris gali būti rašomas prieš sakinį. Jos naudojamos kaip nuorodos GOTO sakiniuose, norint peršokti iš vienos programos vietos į kitą. Žymės aprašomos programos LABEL skyriuje.

1.4. Nuorodos kompiliatoriui

Kiekviena Paskalio programa turi būti išversta į mašininių komandų pavidalą. Šį darbą atlieka kompiliatorius. Kompiliatoriui mes galime nurodyti, kaip kompiliuodamas Paskalio programą jis turi elgtis: į ką kompiliuojamoje programoje kreipti ypatingą dėmesį, į ką nereaguoti, kokias kontrolės priemones įterpti į formuojamą mašininę programą ir t.t.

Programos tekste nuorodos kompiliatoriui gali būti bet kurioje vietoje, ir jos galioja tik toliau po jų vykdomiems programos sakiniams. Nuorodų pavidalas: {$opcijos}. Atkreipkime dėmesį, kad nuorodos kompiliatoriui sintaksiškai mažai skiriasi nuo komentarų programoje {komentarai}. Nuorodų pradžioje rašomas dolerio simbolis $. *Opcija* - tai kažkokia raidė su + ar - ženklu po jos arba be jokio ženklo. Raide nurodoma kontrolės rūšis, o + ar - įtraukti tokią kontrolę į programą ar ne. Nuorodų kompiliatoriui pavyzdys:

```
{$V+,K-,R+}
```

V+ (varying string checking) - programos eigoje bus kontroliuojama, ar simbolių eilučių ilgiai neviršija apbrėžtų ribų;

R+ (range of ...) - bus kontroliuojama, ar vartojant masyvus, jų elementų indeksų reikšmės neišeina už apbrėžtų ribų, kt.

R- - masyvų elementų indeksų reikšmės nebus kontroliuojamos.

Dažniausiai, rašydami programas, nuorodų kompilatoriams nededame. Tačiau reikia turėti galvoje, kad galioja nutylimos nuorodos. Pvz., viena iš nutylimų nuorodų kompiliatoriui yra R-. Plačiau apie nuorodas rasite knygose arba TURBO PASCAL pagalbose (help).

1.5. Programos struktūra

Paskalio programa susideda iš **antraštės** ir **programos bloko**.

Antraštė: PROGRAM vardas ;

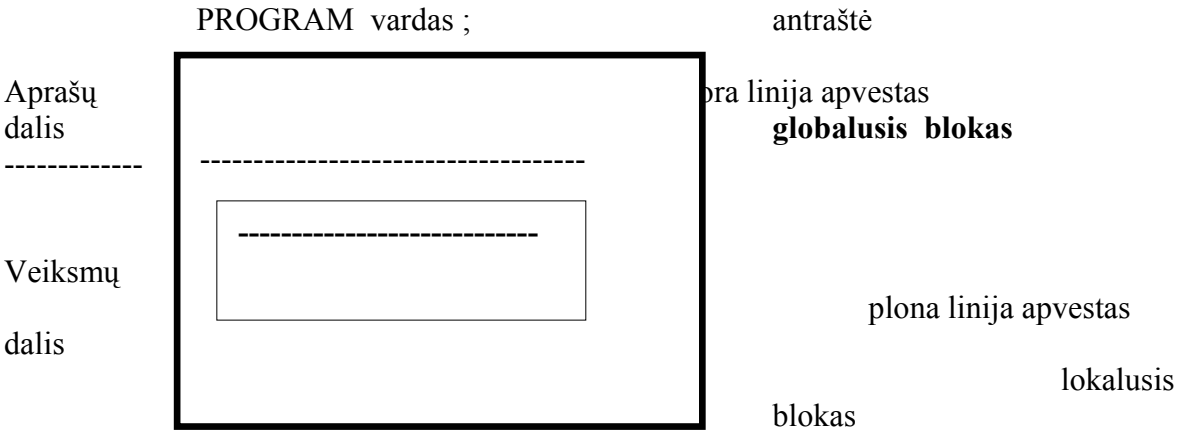
Tuoj po antraštės rekomenduojama įdėti komentarą, kuriame nurodoma ką programa skaičiuoja, kas jos autorius ir kur jį rasti, datą, kt.

Programos blokas susideda iš aprašų dalies ir veiksmų dalies. Aprašų dalies gali ir nebūti.

Programos blokai, priklausomai nuo jų tarpusavio padėties, skirstomi į globaliuosius ir lokaliuosius. Blokas, neįeinantis į jokį kitą bloką, vadinamas **globaliuoju.** Globaliojo bloko viduje esantys kiti blokai (procedūros ir funkcijos) yra **lokalieji**. Globalusis blokas yra pagrindinė programos dalis, ir jis turi būti visada. Lokaliųjų blokų - procedūrų ir funkcijų - programoje gali ir nebūti.

Programos objektai (kintamieji, konstantos, kt.) taip pat yra globalieji ir lokalieji. **Objekto galiojimo sritis** yra tas blokas, kuriame jis yra aprašytas, o taip pat visi tame bloke esantys lokalieji blokai.

Paskalio programos sudėtis schemiškai pavaizduota 1 pav.



1 pav. Paskalio programos sudėties schema.

1.6. Programos bloko struktūra

Po antraštės einantis programos blokas gali turėti septynis skyrius:

PROGRAM vardas ;	programos antraštė
USES Crt, Dos, Graph ;	bibliotekų skyrius
LABEL - - - ;	žymių skyrius
CONST - - - ;	konstantų skyrius
TYPE - - - ;	tipų skyrius
VAR - - - ;	kintamųjų skyrius
PROCEDURE vardas ; - - -	
FUNCTION vardas ; - - -	procedūrų ir funkcijų skyrius
BEGIN	
- - -	sakinių skyrius
sakiniai	
- - -	
END.	

Sakinių skyrius, kurio pradžia nurodoma žodeliu BEGIN, o pabaiga - END, yra būtinas. Kitų skyrių gali ir nebūti.

USES skyrius.

Po bazinio žodžio USES rašomi bibliotekų (bibliotekinių modulių) vardai, kur yra laikomos jūsų programai reikalingų jau gatavų procedūrų ir funkcijų programos.

Yra TURBO PASCAL standartinės bibliotekos. Tačiau programuotojas gali susikurti ir savo asmenines bibliotekas.

Standartinių bibliotekų vardai yra tokie: *SYSTEM, Crt, Dos, Graph*, *Graph3, Overlay, Printer, Turbo3*. Dar yra specializuota biblioteka *Turbo Vision*. Laikui bėgant atsiranda naujų standartinių bibliotekų.

Asmeninėms bibliotekoms vardus duoda pats programuotojas.

USES pavyzdžiai: USES Crt, Graph ; USES Dos, Crt, AsmB ;

Standartinių bibliotekų paskirtys:

SYSTEM. Šios bibliotekos USES skyriuje nurodyti nereikia. Ji reikalinga daugelyje programų, todėl SYSTEM yra automatiškai prieinama. Čia yra sudėti tokie moduliai, kurie būtini kitiems moduliams. Taip pat joje yra standartinių funkcijų ir procedūrų programos: *abs(x), sin(x), sqrt(x), delay(n), length(se), read(...), write(...),* kt.

Crt. Šioje bibliotekoje patalpintos procedūros formuoti vaizdams monitoriuje, valdyti klaviatūrą, garsinę indikaciją, kurti "langus" monitoriuje. Pavyzdžiui:

TextMode(n) - nustato tekstinį ekrano režimą (simbolių kiekį eilutėje);
ClrScr - nuvalo ekraną;
GotoXY(x,y) - perkelia kursorių į *(x,y)* koordinačių vietą ekrane;
Sound(n) - paleidžia *n* Hercų dažnio garsą.

Dos. Čia yra priemonės operacinės sistemos DOS funkcijoms atlikti programos eigoje.

Graph. Čia yra grafikų piešimo monitoriuje procedūros. Jei norime, kad mūsų programa pieštų grafinius vaizdus, būtina pereiti į grafinį darbo režimą, pasitelkus procedūrą

INITGRAPH (ekr, mode, 'C:\TP\BGI');

kur *ekr* - atspindi monitoriaus tipą CGA|VGA|EGA; *mode* - atspindi ekrano skiriamąją gebą VGALO|VGAHI (low, high). Tarp apostrofų nurodomas kelias (katalogų seka), kur diske yra patalpintos monitorių tvarkyklės ("draiveriai"), t.y. Paskalio programinės įrangos failai su plėtiniu .bgi, pvz., ***egavga.bgi***. *ekr* ir *mode* - tai mūsų programoje aprašyti *integer* tipo kintamieji, o CGA, VGA ir kt. - vardinės sveikosios konstantos (jos į mūsų programą "ateina" iš GRAPH bibliotekos).

Nežinant kokio tipo monitorius yra jūsų kompiuteryje, programoje reikia rašyti tokius sakinius:

```
ekr := DETECT; mode := 0;
INITGRAPH( ekr, mode, 'C:\TP\BGI' );
```

DETECT funkcija automatiškai išsiaiškina, kokios rūšies monitorius yra jūsų kompiuteryje, ir INITGRAPH į programą įtrauks atitinkamo monitoriaus tvarkyklę. *mode* kintamasis turi būti įgiįęs reikšmę 0 arba 1 (juo nurodome grafikų išvedimo į monitorių skiriamąją gebą).

Standartinių grafinių procedūrų pavyzdžiai: PutPixel(x,y), Line(x1,y1,x2,y2), Circle(x,y,r), SetColor(...), OutTextXY(...) ir kt.
(plačiau apie tai šiuose konspektuose žiūrėkite Paskalio grafikos priemonių skyriuje)

Printer. Šios bibliotekos procedūros skirtos darbui su spausdintuvais.

Overlay. Tai priemonių rinkinys "persidengiančioms" programoms rašyti. Tokiose programose dėl operatyviosios atminties (RAM) stokos vienu metu nėra laikomos visos reikalingos funkcijos ar procedūros. Vietoje jau "atidirbusios" procedūros į jos vietą operatyvioje atmintyje iš magnetinio disko iškviečiama kita procedūra.

Turbo Vision. Procedūrų biblioteka, kuri naudoja *objektus,* kuriant įvairius meniu (vartotojo interfeisą). *Objektas* - tai toks duomenų tipas (plačiau apie tai šiuose konspektuose žiūrėkite objektinio programavimo skyriuje).

Paprastose programose apsieinama be USES skyriaus (SYSTEM biblioteka naudojama automatiškai).

LABEL skyrius.

Prieš bet kurį Paskalio programos sakinį galima rašyti žymę (angliškai *label* - žymė, etiketė). Jų prireikia tuomet, kai norime GOTO sakiniu peršokti į pažymėtą programos vietą. Žymė - tai skaičius (ne didesnis kaip keturių ženklų) arba vardas ir po jo einantis dvitaškis.

Programos sakinių skyriuje visos pavartotos žymės turi būti apibrėžtos LABEL skyriuje.

Pavyzdys:
```
PROGRAM pvz;
LABEL 385, apci, nz;              žymių skyrius
- - -
BEGIN - - -
GOTO apci; - - -
385 : sakinys ; - - -             sakinių skyrius
apci : sakinys; - - -
GOTO 385; - - -
END.
```

LABEL skyriuje apibrėžta žymė gali būti ir nepanaudota.

CONST skyrius.

Konstanta - tai konkretus skaičius, simbolių eilutė ar kt., ir ji programos bėgyje nesikeičia. Jos gali būti vartojamos įvairiuose sakiniuose.

Jeigu ta pati konstanta vartojama keliuose programos sakiniuose ir yra ilgas jos užrašas, patogiau ją pasivadinti vardu (vardinė konstanta), jam suteikti norimą reikšmę ir vartoti šį vardą, kur bus reikalinga tokia reikšmė. Be to, sumanius pakeisti konstantos reikšmę, tereikės taisyti vienoje programos vietoje, o ne visuose sakiniuose, kur tokia konstanta pavartota.

Konstantų apibrėžimo ir vartojimo pavyzdžiai:
```
PROGRAM konst;
CONST     ep = 4.81532;   sk = 274;
          nsk = -sk;      ds = 53 + sk;
          bruksnys = '----------';
          adresas = 'sauletekio al. 9-342, fizik. fak.';
          taip = true;    ssmax = maxint;     red = raudona;
BEGIN - - -
a := ep * x; - - -                  { a := 4.81532*x; }
b := a + x / ep + ep * c; - - -     { b := a+x/4.81532+4.81532 *c; }
WRITELN ( bruksnys ); - - -         { WRITELN ( '----------'); }
WRITELN ( adresas ); - - -          { WRITELN ( 'sauletekio al....' ) }
END.
```

Yra keletas standartinių konstantų, kurių apibrėžti CONST skyriuje nereikia. Tai MAXINT = 32767, TRUE, FALSE, PI = 3.14159 ir kt. Jei standartinę konstantą apibrėšime CONST skyriuje, tai ji turės tokią reikšmę, kokią mes jai suteiksime. Pvz., CONST MAXINT = 25; Vardinės konstantos reikšmės programos eigoje keisti negalima.

Tipizuotos konstantos apibrėžiamos kartu nurodant ir jos tipą (saugojimo atmintyje būdą). Pavyzdžiui:
```
CONST     a : byte = 25;        { 1 baitas atmintyje }
          b : word = 25;        { 2 baitai atmintyje }
          c : integer = 25;     { 2 baitai atmintyje su ženklu }
```
Tipizuotos konstantos reikšmė programos eigoje gali būti keičiama.

TYPE skyrius.

Šis skyrius reikalingas tuomet, kai programoje norime apsibrėžti savo duomenų tipą. Pavyzdžiui.:
```
TYPE      sveikasis = integer;
          masyvas = array [ 1..8, 1..15 ] of real;
          gele = ( tulpe, roze, gvazdikas, lelija );
```

Taip apibrėžtus duomenų tipų pavadinimus *sveikasis, masyvas, gele* galėsime naudoti VAR skyriuje, aprašant kintamuosius. Pavyzdžiui.:

```
VAR     a, b : sveikasis;
        f, g, h : masyvas;
```

Čia aprašyti *a* ir *b* kintamieji bus *integer* tipo, o *f, g* ir *h* bus dvimačiai realiųjų skaičių masyvai (8 kart po 15 elementų).

VAR skyrius.

Paskalio programose kiekvienas kintamasis turi būti aprašytas VAR skyriuje. Kintamajam apibūdinti vartojami baziniai, duomenų tipą nusakantys žodžiai, kaip *integer, real, string* ir kiti, arba programuotojo apsibrėžti žodžiai TYPE skyriuje. Aprašydami kintamąjį nustatome, kokio tipo duomenims (skaičiams, simboliams, loginėms reikšmėms, kt.) įsiminti jis bus naudojamas, kiek atminities baitų jam skiriama, kokiu pavidalu reikšmės bus įsimenamos (pvz., skaičiams - su ženklu ar be jo, fiksuotu ar slankiu kableliu).

VAR skyriuje aprašant kintamuosius visų pirma rašomi kintamųjų vardai, atskiriant juos kableliu, po to dvitaškis, tipą nurodantis žodis (aprašas) ir kabliataškis. Pavyzdžiui:

```
VAR     a, b, c : integer;
        d : real;
        x, y : array [ -2..5 ]  of  integer;
```

Skiriamos dvi kintamųjų grupės: **skaliariniai ir struktūriniai (sudėtiniai).** Skaliarinis kintamasis gali įsiminti tik vieną reikšmę, o struktūrinis - keletą reikšmių. Pastariesiems priklauso masyvai, simbolių eilutės, aibės, įrašai.

Visų pirma išnagrinėkime skaliarinius - sveikųjų skaičių, realiųjų skaičių, simbolių, loginius, atkarpos, vardinius - kintamuosius.

2. SKALIARINIAI KINTAMIEJI

2.1. Sveikieji kintamieji.

Sveikieji kintamieji gali įsiminti sveikąjį skaičių su ženklu ar be jo (tik teigiamus skaičius). Jiems apibūdinti vartojami tokie baziniai žodžiai: *byte, shortint, word, integer, longint*.

Aprašo pavyzdys:
```
VAR   a, b : byte:
      f, g, h : longint;
```

Kiekvienos rūšies sveikųjų kintamųjų charakteristikos:

tipas	reikšmių diapazonas	atminties dydis (baitais)
byte	0 .. 255	1
shortint	-128 .. 128	1
word	0 .. 65535	2
integer	-32768 .. 32767	2
longint	-2 147 483 648 .. 2 147 483 647	4

Su sveikaisiais kintamaisiais gali būti atliekamos tokios operacijos:

+	- sudėtis;	and	- aritmetinis IR;
-	- atimtis;	or	- aritmetinis ARBA;
*	- daugyba;	not	- aritmetinis NE;
/	- dalyba (rezultatas - realusis);	xor	- sudėtis moduliu 2;
div	- dalyba (rezultatas - sveikasis);	shl	- postūmis kairėn;
mod	- dalybos liekana (rez. ženkl. = dalinamojo ženklui);	shr	- postūmis dešinėn;
		in	- priklauso aibei.

Palyginimo operacijos:

=	- lygu;	<>	- nelygu;
>	- daugiau;	>=	- daugiau arba lygu;
<	- mažiau;	<=	- mažiau arba lygu.

2.2. Realieji kintamieji

Realieji kintamieji apibūdinami baziniais žodžiais *real, single, double, extended, comp.* Jie skiriasi leistinu įsimenamų reikšmių dydžiu ir tikslumu.

Aprašo pavyzdys:
```
VAR   a, b, c : real;
      d, f : double;
      x, y : extended;
```

Kiekvienos rūšies realiųjų kintamųjų charakteristikos:

tipas	reikšmių diapazonas (abs.dydis)	tikslumas (reikšminių deš.poz. kiekis mantisėje)	atminties dydis (baitais)
real	2.9E-39 .. 1.7E+38	12	6
single	1.5E-45 .. 3.4E+38	8	4
double	5.0E-324 ..1.7E+308	16	8
extended	1.0E-4932 .. 1.0E+4932	20	10
comp	10E-18 .. 10E+18	20	8

Pastaba: *comp* kintamieji gali įsiminti tik sveikąsias reikšmes.

Su realiaisiais kintamaisiais leistinos tokios operacijos:

+ - sudėtis; - - atimtis; * - daugyba; / - dalyba;
palyginimo: = , <> , < , <= , > , >= .

Skaičiuojant mišrias išraiškas, t.y. kai jose skiriasi operandų tipai, žemesnio rango operandas pertvarkomas į aukštesniojo pavidalą ir tik po to atliekama operacija (veiksmas). Kintamųjų rangas didėja tokia tvarka: *byte, word, shortint, integer, longint, single, comp, real, double, extended.*

Negalima realiosios reikšmės (kintamojo ar išraiškos rezultato) priskirti sveikajam kintamajam. Aukštesniojo rango sveikojo kintamojo reikšmę (pvz., *longint*) priskiriant žemesnio rango sveikajam kintamajam, galimos klaidos.

2.3. Simbolių kintamieji

Simbolių kintamieji apibūdinami baziniu žodžiu *char* .

Aprašo pavyzdys: VAR a, b : char;

Šio tipo kintamieji gali įsiminti tik po vieną simbolį, nes jiems atmintyje skiriamas tik vienas baitas.

Simbolių kintamuosius galima naudoti palyginimo operacijose. Atliekant jas, yra lyginami simbolių kodai. Pvz., ‘a’ < ‘b’, nes simbolio ‘a’ kodo skaitinė reikšmė yra mažesnė už ‘b’ kodo reikšmę.

Su simboliais dar galima atlikti konkatenacijos (sujungimo) veiksmą, kuris nurodomas pliusu. Pvz., išraiškos ‘a’ + ‘b’ + ‘c’ rezultatas yra ‘abc’. Simbolių kintamajam priskiriant ilgos simbolių eilutės reikšmę, įsimenamas tik jos pirmasis simbolis. Ilgesnes simbolių eilutes gali įsiminti *string* tipo kintamieji (apie juos rašoma toliau).

2.4. Loginiai kintamieji

Loginiai kintamieji apibūdinami žodžiu *boolean*. Jie gali įgyti tik vieną iš dviejų reikšmių: *true* arba *false*.

Aprašo pavyzdys: VAR a, b : boolean;

Šie kintamieji vartojami tik loginėse išraiškose. Loginės išraiškos operandu gali būti palyginimo išraiška, nes jos rezultatas yra loginis dydis. Pavyzdžiui,
p *and* (x<y) *or* q .

Atmintyje loginiam kintamajam skiriamas vienas baitas.

Paskalyje yra šios loginės operacijos: *not, and, or, xor*.

operacija	išraiška	**a** reikšmė	**b** reikšmė	rezultatas
not	not **a**	true		false
		false		true
and	**a** and **b**	true	true	true
		false	true	false
		true	false	false
		false	false	false
or	**a** or **b**	true	true	true
		false	true	true
		true	false	true
		false	false	false
xor	**a** xor **b**	true	true	false
		false	true	true
		true	false	true
		false	false	false

Loginiams kintamiesiems reikšmes galima suteikti priskyrimo sakiniu. Paprasčiausiais READ ir WRITE sakiniais (skirtais *text* tipo failams) loginių reikšmių negalima nei įvesti, nei išvesti.

2.5. Atkarpos kintamieji

Atkarpos kintamieji aprašomi taip:

```
VAR     a, b, c : 1 .. 25;
        x : 20 .. 30;
        r, v : ‘d’ .. ‘m’;
```

Šio pavyzdžio kintamieji a, b, c, x bus svekieji skaičiai, o kintamieji r, v - simbolių tipo. Jie galės įgyti reikšmes tik iš nurodyto intervalo. Simbolių intervalas - tai simboliai jų kodų didėjimo tvarka (raidės abėcėlės tvarka), pradedant ir baigiant nurodytais simboliais.

Dar vienas pavyzdys:

```
CONST   maz = 53;   did = 100;
VAR     a, b : maz .. did;
```

Čia kintamieji a, b galės įgyti sveikųjų skaičių reikšmes tik iš intervalo nuo 53 iki 100.

2.6. Vardiniai kintamieji

Šio tipo kintamieji aprašomi taip:

```
VAR     a,b : (bijunas, tulpe, lelija, gvazdikas);
```

Kintamieji a ir b galės įgyti tik sąraše išvardintas reikšmes: *bijunas, tulpe* ir t.t. Programoje žodžiai *bijunas, tulpe* ir t.t. bus suprantami kaip konkrečios šio tipo kintamųjų reikšmės, o ne kaip kintamųjų vardai.

Kitas pavyzdys:

```
TYPE  gele = (bijunas, tulpe, lelija, gvazdikas);
VAR   a, b, c : gele;
```

Su vardiniais kintamaisiais galima atlikti tik palyginimo operacijas. Pvz., a <= lelija, b > c . Sąrašo pradžioje nurodyta reikšmė laikoma mažiausia, o sąrašo gale - didžiausia.

Vardiniams kintamiesiems yra skirta keletas standartinių funkcijų: *pred, succ, ord*. Pavyzdžiui:

pred(*lelija*) rezultatas bus *tulpe* - sąraše prieš einantis žodis;
succ(*lelija*) rezultatas bus *gvazdikas* - sąraše toliau einantis žodis;
ord(*lelija*) rezultatas bus skaičius 3 - žodžio eilės numeris sąraše.

Vardiniai kintamieji reikšmes gali įgyti pavartojus priskyrimo sakinį. Sakiniuose READ ar WRITE šių kintamųjų vartoti negalima.

Pažintį su struktūriniais (sudėtiniais) kintamaisiais atidėkime vėliau. Kad galėtume greičiau pradėti rašyti paprastas programas, pirma išnagrinėkime Paskalio kalbos sakinius.

3. PASKALIO KALBOS SAKINIAI

Sakiniai rašomi programos bloko sakinių skyriuje, kuris prasideda žodžiu *begin* ir baigiasi *end*. Globaliajame bloke po *end* dedamas taškas, o lokaliajame - kabliataškis.

Paskalyje sakiniai vienas nuo kito skiriami kabliataškiu. Vienoje eilutėje leidžiama rašyti kelis sakinius, o ilgas sakinys gali užimti net kelias eilutes.

Paskalio sakiniai skirstomi į dvi grupes: paprastuosius ir struktūrinius. Paprastųjų grupei priklauso priskyrimo, besalyginio perėjimo (GOTO), kreipimosi į procedūrą (pvz., READ( ), CIRCLE( ) ir kt.) bei tuščiasis sakinys. Struktūriniai sakiniai yra tokie, kurių viduje gali būti kiti sakiniai.Tai sudėtinis, sąlygos ir ciklo sakiniai.

3.1. Priskyrimo sakinys

Priskyrimo sakinio bendrasis pavidalas yra

kintamasis := išraiška;

Juo “liepiama” kompiuteriui visų pirma apskaičiuoti išraišką, o po to gautą rezultatą priskirti kintamajam.

Keletas pastabų:

- išraiškoje pavartoti kintamieji jau anksčiau turi būti įgiję reikšmes;
- kintamojo, kuriam priskiriamas išraiškos rezultatas, tipas turi sutapti su išraiškos rezultato tipu. Leistinas tik toks neatitikimas: sveikąjį skaičių (rezultatą) galima priskirti realiajam kintamajam. Jei išraiškos rezultatas yra sveikasis aukštesnio rango skaičius ir jis yra priskiriamas žemesnio rango sveikajam kintamajam, galimos klaidos (kompiliatorius to nekontroliuoja).

Priskyrimo sakinių pavyzdžiai:

```
VAR     a, b : integer;
        c, d : byte;
        x, y : real;
        p : boolean;
        r : char;
BEGIN   a := 5;
        b := a mod 3 + a;
        c := b;                 { galimos klaidos }
        x := b;
        y := b + 3.61;          { išraiškos rezultato tipas  real }
        x := (a+b)/2 + sin(y);
        r := ‘k’;
        p := true;
        a := x;                 { neleistinas sakinys }
        x := r;                 { neleistinas sakinys }
END.
```

3.2. GOTO sakinys

Programose sakiniai atliekami nuosekliai vienas po kito. Prireikus keisti jų atlikimo tvarką, rašomas GOTO sakinys. Šis sakinys vartojamas norint besalygiškai peršokti į reikiamą programos vietą. Tokia vieta programoje turi būti pažymėta, t.y. prieš sakinį turi būti žymė. Žymės aprašomos programos LABEL skyriuje. Šio sakinio bendrasis pavidalas yra

GOTO žymė;

Jame *žymė* - tai skaičius arba vardas. Skaičiuje turi būti ne daugiau kaip keturi skaitmenys.

Pavyzdys:

```
- - -
LABEL  38, abc;
```

```
- - -
BEGIN  - - -
38 : sakinys;  - - -
GOTO  abc;  - - -
sakinys; - - -
GOTO 38; - - -
abc : sakinys; - - -
END.
```

Žymės galiojimo sritis yra tik tas programos blokas, kuriame ji aprašyta.

3.3. Kreipimosi į procedūras sakiniai

Procedūra - tai programos blokas (standartinis arba programuotojo asmeninis), atliekantis tam tikrus veiksmus. Ji turi vardą, kuriuo į ją kreipiamasi. Po vardo skliaustuose dar gali būti (bet nebūtinai) parametrai, kuriais perduodamos procedūros darbui reikalingos reikšmės. Apie pačių procedūrų sudarymą rašoma toliau (kitame skyriuje).

Dažniausiai yra naudojamos standartinės procedūros, kurios laikomos bibliotekose SYSTEM, DOS, CRT, GRAPH ir kitose. Programose vartojant bibliotekose saugomas procedūras (ir funkcijas), būtina USES skyriuje nurodyti bibliotekų vardus (išskyrus SYSTEM). Pavyzdžiui:

```
PROGRAM  uzdavinys;
USES  graph, crt;
- - -
BEGIN
- - - kreipiniai į procedūras - - -
- - - ir kitokie sakiniai - - -
END.
```

Keletas kreipimosi į standartines procedūras pavyzdžių:

```
ClrScr;              - šia procedūra "nuvalomas ekranas" (Clear Screen);
WRITE(a, b, c);      - į ekraną išvedmos  a, b ir c  kintamųjų reikšmės;
READ(x, y);          - klaviatūra įvedamos  x ir y  kintamųjų reikšmės;
CIRCLE(x, y, s);     - brėžiamas  s  spindulio apskritimas, kurio centro
                       koordinatės  yra ekrano taške (x,y).
```

Pradedančiajam programuotojui visų pirma reikia žinoti paprasčiausias duomenų įvedimo ir išvedimo procedūras: READ, READLN, WRITE ir WRITELN. Išnagrinėkime jas čia. Vėliau, nagrinėdami failus ir jų tipus, prie įvedimo ir išvedimo procedūrų dar sugrįšime.

Įvedimo procedūros READ ir READLN. Paprasčiausiu atveju, kai programoje nėra jokių papildomų nurodymų, šios procedūros prašo įvesti duomenis klaviatūra. Jų pavidalas yra

```
READ (x1, x2, ... , xn );        READLN (x1, x2, ... , xn);
```

Čia x1, x2 ir t.t. yra kintamųjų vardai, kurių reikšmes norime įvesti. Kompiuteris lauks tol, kol neįvesime procedūroje nurodyto reikšmių kiekio. Vienoje eilutėje renkant keletą skaičių, jie atskiriami tarpais. Surinkus daugiau skaičių nei reikalauja įvedimo procedūra, likusius paims kita READ ar READLN procedūra, jei prieš tai nebus nuorodos pradėti įvedimą nuo naujos eilutės pradžios. Į naują eilutę visada pereinama įvykdžius READLN procedūrą.

Įvedimo procedūrose kintamųjų gali ir nebūti: READ; READLN; Tuomet kompiuteris lauks, kol paspausite ENTER klavišą.

Norint įvesti *char* tipo kintamojo reikšmę, vartokime atskirą įvedimo sakinį, prieš tai perėję į eilutės pradžią.

Išvedimo procedūros WRITE ir WRITELN. Paprasčiausiu atveju jomis skaičiavimo rezultatai išvedami į ekraną. Jų pavidalas yra

```
WRITE (x1, x2, ... , xn);        WRITELN (x1, x2, ... , xn);
```

Čia x1, x2 ir t.t. yra kintamųjų vardai, kurių reikšmes norime išvesti. Išvedimo procedūrose be kintamųjų dar gali būti konstantos ir išraiškos. Pavyzdžiui,

```
WRITE ( 'Rezultatas =' , x1 , '---' , x2 , '--ALFA=' , 25.38 );
```

Nauja išvedimo procedūra rezultatus išveda į tą pačią eilutę kaip ir anksčiau vykdyta procedūra. Kai eilutė užsipildo, automatiškai pereinama į naują eilutę. Į naują eilutę visada pereinama užbaigus WRITELN procedūrą. Pastaroji gali ir neturėti parametrų: WRITELN;

Formatuotas išvedimas. Išvedant rezultatus, realieji skaičiai atvaizduojami slankaus kablelio pavidalu. Norint išvesti reikšmę mums įprastu pavidalu, galima paskirti bendrąjį pozicijų kiekį visam skaičiui ir pozicijų kiekį trupmeninei daliai. Pavyzdžiui:

```
VAR   a : real;
      b : double;
BEGIN
 a := 25;        b := 16.85;
 WRITELN ( a, b );                  {2.50000000000E+01   1.68500000000E+02}
 WRITELN ( a:8:2 , b:11:3 );              {  ^^^25.00^^^^^16.850 ,   čia ^ reiškia tarpą  }
END.
```

Realiajam skaičiui nurodžius tik vieną formato skaičių, reikšmę išves slankaus kablelio pavidalu.

Išvedamo rezultato formatą galima nurodyti ir sveikiesiems skaičiams bei simbolių eilutėms. Pavyzdžiui:

```
VAR     d : integer;
        g : longint:
BEGIN
        d := 38:          g := 852;
        WRITELN ( d:4 , g:6 , 'abc':5 );      { ^^38^^^852^^abc ,   čia ^  reiškia tarpą }
END.
```

3.4. Tuščiasis sakinys

Parašius tik kabliataškį, turėsime tuščiąjį sakinį. Pavyzdžui:

... a := 23.5; **;** **;** b := a - 1.23; ...

žymė : **;**

Prieš tuščiąjį sakinį gali būti žymė.

3.5. Sudėtinis sakinys

Sudėtinis sakinys - tai sakinių grupė, kurios priekyje rašomas žodelis BEGIN, o pabaigoje - END. Pavyzdžiui:

BEGIN read (x, y); a:= 23.5; b := a*x - 1.23; ... writeln(x, b); END;

Po sakinio, esančio prieš pat END, kabliataškis nebūtinas. Sudėtinis sakinys traktuojamas kaip vienetas ir jį programoje galima naudoti ten, kur gali būti vienas elementarus sakinys. Dažniausiai jų prireikia sąlygos (IF, CASE) ir ciklo (FOR, WHILE) sakiniuose.

3.6. Sąlygos sakinys IF

IF sakinys įgalina keisti nuoseklią programos sakinių atlikimo tvarką, atsižvelgiant į tikrinamas sąlygas. Jų pavidalas:

```
IF sąlygos_išraiška THEN sakinys_1 ELSE sakinys_2;
        arba
IF sąlygos_išraiška THEN sakinys_1;
```

Juose "sąlygos_išraiška" - tai palyginimo arba/ir loginė išraiška.

Vykdant IF sakinį, visų pirma apskaičiuojama "sąlygos_išraiška". Jei gautas rezultatas yra *true* , toliau atliekamas tik po THEN esantis "sakinys_1". Priešingu atveju (*false*) atliekamas tik po ELSE esantis "sakinys_2". Jei IF sakinyje ELSE konstrukcijos nėra, esant "sąlygos_išraiškos" rezultatui *false* , iš karto pereinama prie kito po IF einančio programos sakinio.

Atkreipkime dėmesį, kad prieš ELSE kabliataškis nerašomas, nes dar nepabaigtas IF sakinys.

Tiek po THEN, tiek po ELSE gali būti tik vienas sakinys. Todėl šiose vietose, esant reikalui, dažnai vartojamas sudėtinis sakinys.

IF sakiniai gali būti vienas kito viduje.

Sakinių pavyzdžiai:

```
IF  a > b  THEN  x :=  a - b    ELSE  x :=  a*b + 3;
IF  (x < 5) and (x > 2) THEN  BEGIN r := (x*x + y*y); writeln(x, y, r) END;
  IF  raide = ‘a’
        THEN  IF  numeris = 25
                  THEN  BEGIN suma := kain*kiek; writeln(numeris) END
                  ELSE  IF  numeris = 73
                            THEN  writeln(‘klaida’, numeris)
                            ELSE    begin  x :=  b + c;  y := c - d  end;
```

3.7. Varianto sakinys CASE

Jei IF sakinys leidžia išsišakoti programai tik į dvi dalis, tai CASE sakinys duoda galimybę suprogramuoti norimo kiekio išsišakojimus. Pastarojo pavidalas:

```
CASE  išraiška  OF
      reikšmių_sąrašas_1 :  sakinys_1;
      reikšmių_sąrašas_2 :  sakinys_2;
      - - -
      reikšmių_sąrašas_N :  sakinys_N;
      ELSE  sakinys;
END;
```

Po CASE žodžio gali būti tik skaliarinė, diskrečią reikšmę (sveikąjį skaičių arba simbolį) duodanti išraiška. “Reikšmių_sąrašai’ - tai diskrečios reikšmės, atskirtos viena nuo kitos kableliu.

Vykdant CASE sakinį, visų pirma apskaičiuojama “išraiška”. Po to vykdomas tas sakinys, prieš kurį esančiame “reikšmių_sąraše” yra apskaičiuotoji reikšmė. Jei apskaičiuotosios reikšmės nėra nė viename sąraše, vykdomas po ELSE esantis sakinys.

ELSE konstrukcijos CASE sakinyje gali ir nebūti. Tuomet, nesant apskaičiuotosios reikšmės nė viename sąraše, vykdomas po CASE sakinio einantis kitas programos sakinys.

CASE sakinių pavyzdžiai:

```
 VAR  k, n : integer;
 begin
      read(k);  - - -
      CASE   k + 2*n   OF
        1, 4, 9, 3 : y := a - b;
        2, 15      : BEGIN  writeln(a, b);  read(y);  c  := b - y   END;
        24 .. 61   : z := a+b/c;          { reikšmių sąrašas nurodytas atkarpa }
        ELSE    BEGIN  write(‘klaida, kartokite’); read(n)   END;
      END;
```

Dirbant su simboliais, pvz., VAR raide: char; , “reikšmių_sąrašai” atrodytų, pavyzdžiui, taip:

```
      READ (raide);
      CASE   raide  OF
        ‘a’, ‘d’, ‘k’  :  sakinys;
        ‘n’ .. ‘s’     :  sakinys;        { reikšmių sąrašas nurodytas atkarpa }
      END;
```

3.8. Ciklo sakiniai

Paskalyje yra trijų rūšių ciklo sakiniai: FOR, REPEAT ir WHILE.

FOR ciklas. Jo pavidalai:

```
FOR  ciklo_kintamasis := prad._išr.  TO  gal._išr.  DO  sakinys;
FOR  ciklo_kintamasis := prad._išr.  DOWNTO  gal._išr.  DO  sakinys;
```

Šiuose sakiniuose “cilko_kintamasis” - tai sveikojo, atkarpos arba vardinio tipo kintamasis. “Prad._išr.” ir “gal._išr.” rezultatai taip pat turi būti vieno iš ką tik paminėto tipo. Kai

“ciklo_kintamasis” yra sveikojo tipo, vykdant ciklą jo reikšmė kinta dydžiu +1 (sakinyje su TO) arba -1 (sakinyje su DOWNTO). Turint atkarpos arba vardinio tipo “ciklo_kintamąjį”, ciklo metu imamos iš eilės einančios (TO - tolesnė, DOWNTO - prieš einanti) jų reikšmės. Po žodelio DO rašomas tik vienas paprastas arba sudėtinis sakinys.

FOR ciklas vykdomas taip. Visų pirma apskaičiuojamos “prad._išr.” ir “gal._išr.”, o jų rezultatai įsimenami darbiniuose atminties laukuose. Po to “prad._išr.” rezultatas priskiriamas “ciklo_kintamajam”. Toliau tikrinama, ar “ciklo_kintamojo” reikšmė nėra didesnė (cikle su TO) už “gal._išr.” rezultatą. Jei reikšmė mažesnė, vykdomas po DO esantis sakinys ir, įvykdžius jį, “ciklo_kintamojo” reikšmė padidinama vienetu. Jei reikšmė didesnė, ciklas užbaigiamas. Analogiškai vykdomas ir ciklas su DOWNTO.

FOR ciklų pavyzdžiai:

```
VAR  n : integer;  ...
FOR  n := 3  TO  15  DO  WRITELN( n,  n*n );
FOR  n := 26  DOWNTO  1  DO
                    BEGIN  ... sakiniai  ... END;

VAR  r : (‘c’ .. ‘t’); ...
FOR  r := ‘g’  TO  ‘m’  DO  sakinys;
FOR  r := ‘s’  DOWNTO  ‘b’  DO  sakinys;

TYPE  spalv = (juod, zal, mel, gelt, raud, balt);
VAR  x : spalv;
VAR  x := zal  TO  raud  DO  sakinys;
VAR  x := balt  DOWNTO  juod  DO  sakinys;
```

Pastabos dėl FOR ciklų: po DO vykdomais sakiniais (ciklo kūne) nedera keisti “ciklo_kintamojo” reikšmės; ciklą galima nutraukti GOTO sakiniu “pabėgant” iš ciklo kūno; ciklui pasibaigus, “ciklo_kintamojo” reikšmė yra neapibrėžta (už ją negarantuojama); FOR ciklas gali būti neatliktas nė karto.

REPEAT ciklas. Jo pavidalas:

```
REPEAT  ... sakiniai ...  UNTIL  išraiška;
```

Po UNTIL turi būti loginį rezultatą duodanti išraiška (palyginimo arba loginė).
Kol “išraiškos” rezultatas yra *false* , tol ciklas yra kartojamas.
Pavyzdžiai:

```
VAR  a : real;  - - -
a := 0;
REPEAT  ...  a := a + 0.1;  ...  UNTIL  a = 2.5;

VAR  x : char;  - - -
REPEAT  ...  READLN(x);  ...  UNTIL  x > ‘t’;
```

WHILE ciklas. Jo pavidalas:

```
WHILE  išraiška  DO  sakinys;
```

Po WHILE turi būti loginį rezultatą duodanti išraiška (palyginimo arba loginė). Kol “išraiškos” rezultatas yra *true* , tol ciklas yra kartojamas.

Pavyzdžiai:

```
VAR  n : integer;
       x : real;  - - -
n:= 0;  WHILE  n < 25  DO
                  BEGIN  n := n + 1; WRITELN(n, n*n); END;
x := 0.6;  WHILE  abs(x) > 1E-5  DO  sakinys;
```

3.9. Skyrybos taisyklės

Baigę nagrinėti pagrindinius Paskalio sakinius, padrykime keletą pastabų skyrybos klausimais:

1. Kabliataškis (;) nerašomas po žodžių USES, LABEL, TYPE, CONST, VAR, UNIT .
2. Kabliataškis nerašomas po BEGIN ir gali būti nerašomas prieš END.
3. Sąlygos sakiniuose kabliataškis nerašomas po IF, CASE, THEN ir nerašomas prieš ELSE.
4. Ciklo sakiniuose kabliataškis nerašomas po FOR, REPEAT, WHILE, DO ir gali būti nerašomas prieš UNTIL.

3.10. Paprastų programų pavyzdžiai

Prieš rašant programą visų pirma reikia sudaryti sprendžiamo uždavinio algoritmą, t.y. planą, kokius veiksmus ir kokia eilės tvarka numatome atlikti. Žemiau pateiktuose pavyzdžiuose, kur veiksmų atlikimo tvarka yra aiški iš uždavinio sąlygos, algoritmo neaprašome. Kai kuriuose iš jų, programavimo niuansams suprasti, algoritmai aiškinami plačiau.

1 pavyzdys.

Išspausdinti sveikųjų skaičių nuo 1 iki N žingsniu 1 kvadratų, kūbų ir kvadratinių šaknų lentelę. N reikšmę įvesti klaviatūra.

```
PROGRAM  laipsniai ;
        {programa skaičių laipsnių lentelei išvesti;
         autorius – P.Jonaitis, tel. 222222;   data - 2001 06 26}
USES  crt ;
VAR   n, j, kv, kub : integer ;
        sakn : real ;
BEGIN
   CLRSCR ;            {standartine procedūra CLRSCR nuvalomas ekranas}
   WRITELN ( 'Iveskite sveikaji skaiciu N ' ) ;
   READLN ( n ) ;      {įvedame skaičių, iki kurio skaičiuosime laipsnius}
                       {toliau išvedame lenteles antraste}
   WRITELN ( '-------------------------------------------------------------' ) ;
   WRITELN ( '  Skaicius      Kvadratas      Kūbas      Kv.šaknis       ' ) ;
   WRITELN ( '-------------------------------------------------------------' ) ;
                       {toliau skaičiuojame ir išvedame rezultatus}
   FOR  j := 1  TO  n  DO
        BEGIN
          kv := j * j;
          kub := kv * j ;
          sakn := SQRT ( j ) ;
          WRITELN ( j:5,  kv:12,  kub:12,  sakn:15:2 ) ;
        END ;
   WRITELN ( '-------------------------------------------------------------' ) ;
END .
```

2 pavyzdys.

Apskaičiuoti sumą $S = \sum_{k=1}^{N} (1 / k) = 1/1 + 1/2 + 1/3 + \ldots + 1/N$, ir faktorialą $N! = 1 * 2 * 3 * \ldots * N$, kur $N > 0$. N reikšmę įvesti klaviatūra.

```
PROGRAM  suma_ir_faktorialas ;
        {programa skaičiuoja sumą ir faktorialą pagal pateiktas formules;
         autorius – J.Petraitis;  tel.  222222;  data  2001 06 26 }
USES  crt ;
VAR   k, n : integer ;
        suma : real ;
        fakt : longint ;
BEGIN
   CLRSCR ;
   WRITELN ('Įveskite sveikąjį skaičių N: ') ;
   READLN ( n ) ;                                  {įvedame skaičių N}
        {toliau pasirengiam kintamuosius suma ir fakt sumai ir faktorialui
         kaupti}
   suma := 0 ;
   fakt := 1 ;
   FOR  k := 1  TO  n  DO
        BEGIN
           suma := suma + 1 / k ;
           fakt := fakt * k ;
        END ;
                {toliau išvedame skaičiavimo rezultatus}
   WRITELN ( 'Kai  N =  ' , n ) ;
   WRITELN ( 'Suma =  ' , suma:10:5 ) ;
   WRITELN ( 'Faktorialas =  ' , fakt ) ;
END .
```

3 pavyzdys.

Apskaičiuoti funkciją

$$F(x) = \begin{cases} x + (x+1)^2 + (x+2)^3, & \text{kai } x < 1, \\ 1 + 1/x + 1/(x^2) + 1/(x^3) + \ldots + 1/(x^{10}), & \text{kai } 1 <= x <= 3, \\ 1 + \ln(x), & \text{kai } x > 3. \end{cases}$$

Čia užrašas *a^b* reiškia *a* kėlimą laipsniu *b* (Paskalyje nėra kėlimo laipsniu operacijos).
x reikšmę įvesti klaviatūra.

```
PROGRAM    issisakojusi_funkcija;
          { programa išsišakojančios funkcijos reikšmei apskaičiuoti;
            UAB "Puikūs programuotojai", tel. 222222 }
USES  crt ;
VAR    x,  f  : real ;
          k : integer ;
BEGIN
    CLRSCR ;
    WRITELN ( 'Įveskite x reikšmę: ' ) ;
    READLN ( x ) ;
    IF  x < 1
          THEN    f :=  x + (x+1) * (x+1) + (x+2) * (x+2) * (x+2) ;
    IF  ( 1 <= x ) and ( x <= 3 )
          THEN   BEGIN
                       f := 1 ;
                       a := 1 ;
                       FOR  k := 1  TO  10  DO
                            BEGIN
                               a := a / x ;
                               f := f + a ;
                            END ;
                 END ;
   IF  x  > 3
          THEN    f := 1 + ln ( x ) ;
                       {toliau išvedame skaičiavimo rezultatus}
   WRITELN ( 'Kai  x = ', x:6:2 ) ;
   WRITELN ( 'Funkcijos reikšmė = ', f:8:3 );
END .
```

4 pavyzdys.

Trigonometrinės funkcijos *cos (x)* , kai *x* išreiškiamas radianais, reikšmė randama sumuojant tokios eilutės narius:

cos (x) = 1 – (x^2) / (2!) + (x^4) / (4!) – (x^6) / (6!) + …

Čia užrašas *x^2* reiškia *x* kėlimą laipsniu *2* .

Nežiūrint to, kad Paskalyje yra *cos(x)* standartinė funkcija, kosinusui apskaičiuoti sudarykime programą patys. *x* reikšmę įveskime klaviatūra, o kosinusą skaičiuokime tikslumu ε = 1E-5.

Algoritmas. Duotos eilutės, kurios narius sumuojant gaunama cos(x) reikšmė, k-tojo nario bendra išraiška yra

a_k = ((-1)^k) * (x^(2k)) / ((2k)!), kur k = 0, 1, 2, …

Taigi, reikia apskaičiuoti sumą

cos(x) = $S = a_0 + a_1 + a_2 + a_3 + \ldots$

Čia sumuojamų narių kiekis iš anksto nėra žinomas. Skaičiavimai nutraukiami tada, kai tik eilinis (paskutinis) sumos narys absoliutiniu dydžiu pasidaro mažesnis už ε .

Suma ir faktorialas skaičiuojami rekurentiškai, kaip ir 2 pavyzdyje. Primename, kad nulio faktorialas lygus vienetui (0! = 1).

```
PROGRAM  kosinusas ;
      { programa kosinusui apskaičiuoti;  autorius M.Pupytė;  tel. 222222 }
USES  crt ;
VAR  a, eps, s, x, y : real ;
      f, k, n, z : integer ;
BEGIN
   CLRSCR ;
   WRITELN ( 'Įveskite  x  reikšmę ' ) ;
   READLN ( x ) ;
   WRITELN ( 'Įveskite epsilon  reikšmę skaičiavimo tikslumui nurodyti ' ) ;
   READLN ( eps ) ;
      {toliau pasiruošiam kintamųjų pradines reikšmes: z – ženklui, k – eilutės
       nario  numeriui, s – narių sumai kaupti}
   z := 1 ;   s := 1 ;   k := 0 ;
   REPEAT
      z := -z ;                      { kaitaliojamas  z  ženklas }
      k := k + 1 ;                   { padidinamas eilutės nario numeris }
      f := 1 ;   y := 1 ;
      FOR  n := 1  TO  2*k  DO
         BEGIN
                  f := f * n ;       { skaičiuojam  2k  faktorialą }
                  y := y * x ;       { keliam  x  laipsniu  2k }
         END ;
      a := z * y / f ;               {gaunama eilinio sumos nario reikšmė}
      s := s + a ;                   {kaupiama narių suma}
   UNTIL  abs ( a ) < eps ;
                                     { išvedame skaičiavimo rezultatus }
   WRITELN ( 'Kai  x = ',  x:6:2 ) ;
   WRITELN ( 'Mūsų pačių apskaičiuota  cos(x)  reikšmė = ', s:10:6 ) ;
   WRITELN ( 'Standartinės funkcijos cos(x) reikšmė = ', cos(x):10:6 ) ;
END .
```

4. STRUKTŪRINIAI (SUDĖTINIAI) KINTAMIEJI

Šios rūšies kintamiesiems priklauso *masyvai, simbolių eilutės, aibės* ir *įrašai*. Dar yra taip vadinamieji *objektai*, kurių čia nenagrinėsime (apie juos žiūrėkite objektinio programavimo skyriuje).

4.1. Masyvai.

Vienodo tipo duomenis galima jungti į masyvus. Tokiu būdu krūvai duomenų suteikiamas tik vienas vardas, o jiems saugoti - vientisas atminties laukas. Atskiras masyvo elementas nurodomas po masyvo vardo stačiakampiuose skliaustuose rašant indeksus. Indeksu gali būti sveikoji konstanta, sv. kintamasis arba sv. reikšmę duodanti išraiška.

Programose masyvai aprašomi, pavyzdžiui, taip:

```
CONST     a = 1;  b = 15;
VAR       mas : array [ a .. b ]  of  integer;
          r, t : array [ -5 .. 8 ]  of  real;
          zod : array [ 1 .. 10 ]  of  char;
          x : array [ 1 .. 6, 5 .. 25 ]  of  byte;
          y : array [ 1 .. 6 ]  of  array [ 5 .. 25 ]  of  byte;
          z : array [ a .. b, -3 .. 7]  of  double;
```

Masyvai gali būti didesnio nei dviejų matavimų kiekio.

Atmintyje vienam masyvo elementui skiriama tiek baitų, kiek reikalauja nurodytas tipas. Visi elementai išdėstomi tokia eilės tvarka, kad pirmiau auga kairėje stovintis indeksas. Pvz., z[1, -3], z[2, -3], ..., z[15, -3], z[1, -2], z[2, -2], ..., z[15, -2], z[1, -1], z[2, -1], ...

Veiksmai atliekami tik su atskirais masyvo elementais. Yra tik dvi išimtys, kai operuojama iš karto su visu masyvu:

```
r := t;          { r  it  t  turi būti visiškai vienodi masyvai }
WRITE( x );   { x  turi būti  char tipo elementų masyvas }
```

Darbo su masyvais pavyzdžiai:

```
VAR   a : array[1..20]  of real;
      b : array[1..5, 1..8]  of  integer;
      j, m, n :  integer;
      sa :  real;
      sb:  longint;
- - -          { masyvo išvalymas }
FOR  j :=  1  TO  20  DO  a[j] :=  0;  { vienmačiam  masyvui a viengubas ciklas}
FOR  m  := 1  TO  5  DO                { dvimačiam  masyvui  b dvigubas ciklas}
  FOR  n  :=  1  TO  8  DO    b[m, n] :=  0;
          { masyvo  elementų  reikšmių  įvedimas }
FOR  j :=  1  TO  20  DO    READ ( a[j] );
FOR  m  := 1  TO  5  DO
  FOR  n  :=  1  TO  8  DO    READ ( b[m, n] );
          { masyvo elementų sumos radimas }
sa := 0;   FOR  j :=  1  TO  20  DO    sa := sa + a[j];
sb := 0;   FOR  m :=  1  TO  5  DO
             FOR  n :=  1  TO  8  DO    sb := sb + b[m, n];
          { masyvo  elementų  reikšmių  išvedimas }
FOR  j := 1  TO  20  DO
  BEGIN      IF  ((j - 1) mod 5) = 0  THEN  WRITELN;
             WRITE ( a[j] : 12 : 3 );
  END;              { čia išveda po penkias reikšmes į vieną eilutę }
```

```
FOR  m := 1  TO  5  DO
  BEGIN       WRITELN;
              FOR  n := 1  TO  8  DO  WRITE ( b[m, n] : 7 );
  END;        { čia išveda penkiose eilutėse po aštuonias reikšmes }
```

Pastaba: panaudojus nuorodą kompiliatoriui {$R+}, po jos einančiuose programos sakiniuose būtų kontroliuojama, ar masyvų elementų indeksai neišeina už apsibrėžtų ribų.

Masyvo elementais taip pat gali būti toliau nagrinėjamos simbolių eilutės, aibės arba įrašai. Pavyzdžiui, VAR a : array [1..8] of string;

4.2. Simbolių eilutės

Simbolių eilutes toliau vadinsime tiesiog eilutėmis. Dirbant su tekstine informacija, galima naudoti *char* tipo masyvus. Simbolių masyvai dažnai yra nepatogūs, nes tuomet būtina naudoti indeksuotus kintamuosius, kontroliuoti indeksų reikšmes, kt. Patogiau yra vartoti eilutes, kurių ilgis tampa lygus priskirtų (įvestų) simbolių kiekiui. Eilutės aprašomos *string* žodžiu. Pavyzdžiai:

```
VAR   a : string [ 19 ];
        b : string;              { nutylimas maksimalus ilgis yra 255 }
```

Po *string* esančiu skaičiumi nurodoma, kokio maksimalaus ilgio eilutę kintamasis galės įsiminti. Priskiriant (įvedant) jam ilgesnę reikšmę, paskutiniai simboliai bus atmetami.

Eilutės atskiras elementas gali būti nurodomas indeksu, kaip ir masyvų atveju. Pavyzdžiui, a[4], a[15] yra aukščiau aprašytos *a* eilutės 4 ir 15 elementai. Atkreipkime dėmesį į tai, kad nuliniame eilutės elemente (jam skirtame atminties baite) yra įsimenamas ne kažkokio simbolio kodas, o skaičius - einamasis eilutės ilgis. Pavyzdžiui, įvykdžius priskyrimo sakinį (arba įvedimo sakinį READ)

```
a := ‘antanas’;
```

turėsime a[0] = 7 (skaitinė reikšmė), a[1] = ‘a’, a[2] = ‘n’ ir t.t. Išvedimo sakiniu, pavyzdžiui, WRITE(a) bus išvesti tik 7 simboliai.

Eilutės ir jų atskiri elementai gali būti naudojami konkatenacijos (+) ir palyginimo (=, <, >, <>, <=, >=) operacijose. Pavyzdžiui,

```
a := ‘kose’;
b := ‘skani’ + ‘  ‘ + a;  { b = ‘skani kose’ }
```

Palyginant dvi eilutes, yra lyginami jų simbolių kodai. Eilutėse simboliai imami iš kairės į dešinę. Kai tik randami nelygūs simboliai, nustatoma, kuri eilutė yra didesnė. Pavyzdžiui,

‘A’ < ‘B’ tokio palyginimo rezultatas yra *true*,
‘B’ < ‘b’ - *true* ,
‘ab_’ > ‘ab’ - *true* ,
‘2’ < ‘12’ - *false* , nes, imant pirmuosius simbolius, ‘2’ > ‘1’ .

Pastebėkime, kad lotyniškų raidžių kodai didėja, imant jas abėcėlės tvarka. Mažųjų raidžių kodai yra didesni už didžiųjų raidžių kodus.

Įvedant eilučių reikšmes READ arba READLN sakiniais, kiekvienai eilutei reikia vartoti atskirą READLN sakinį.

Turbo Paskalio 7.0 versijoje pradėtos naudoti **ASCIIZ tipo eilutės**. Tai simbolių masyvai, tačiau atkreipkime dėmesį į tai, kad šitokio masyvo apraše apatinė indekso reikšmė lygi nuliui. Pavyzdžiui,

```
VAR   a : array[0..6] of char;
```

ASCIIZ eilutės maksimalus ilgis gali būti 65535 simboliai, kai tuo tarpu STRING eilučių - tik 255 simboliai.

Taip aprašytam masyvui, kitaip vadinamam ASCIIZ eilute, reikšmę įvesti galima READ(a); READLN(a); sakiniais, o išvesti - WRITE(a); WRITELN(a); sakiniais. Įvedus trumpesnę reikšmę, paskutinių nepanaudotų tokios eilutės elementų baitai užpildomi nuliais (visi bitai lygūs nuliui). Išvedami tik simboliai iki nulinio elemento. Tokiu būdu, ASCIIZ eilutės skiriasi nuo STRING eilučių tuo, kad, priskyrus reikšmę, jų ilgis užfiksuojamas ne skaičiumi eilutės nuliniame baite, o pirmą kartą sutiktu baitu, kuriame yra kodas lygus nuliui.

Tačiau pastebėkime, kad ASCIIZ eilutei priskyrimo sakiniu galima priskirti tik tokio ilgio reikšmę, koks nurodytas jos apraše (mūsų pavyzdžio **a** eilutei - 7 simboliai). Šiam trūkumui apeiti yra įvestas standartinis *pchar* duomenų tipas:

```
VAR  x : pchar;
```

Šio tipo kintamajam priskyrimo sakiniu galima priskirti ASCIIZ eilutės reikšmę ir bet kokios simbolių eilutės konstantos reikšmę. READ sakiniais jam reikšmės įvesti negalima, o išvesti WRITE sakiniais - galima.

```
VAR   a: array[0..24561] of char;
      x : pchar;
- - -
      x := a;                 { kintamasis x  tapo kintamojo a  kopija , }
      READ ( a);              { tačiau su kitokiomis galimybėmis        }
      WRITE ( x );  - - -
```

Šiuo atveju x[0] = a[0] , x[1] = a[1] ir t.t. Čia WRITE(x); sakiniu bus išvedama ASCIIZ eilutės *a* reikšmė.

Darbui su *pchar* tipo kintamaisiais Paskalyje 7.0 yra modulių biblioteka STRINGS. Pavyzdžiui, su funkcija *strlen (x)* galima sužinoti, kokio ilgio reikšmę turi įgiję *pchar* tipo kintamasis. Dalis funkcijų ir procedūrų yra skirta kopijuoti *string* tipo kintamojo reikšmei į *pchar* tipo kintamąjį ir atvirkščiai.

pchar tipo kintamiesiems negalima naudoti konkatenacijos (sujungimo) operacijos, kas yra leistina *string* ir *char* tipo reikšmėms. Įdomu tai, kad *pchar* tipo kintamasis gali būti sudėties ar atimties operacijoje su sveikuoju skaičiumi (word). Skaičiumi nurodytu simbolių kiekiu yra paslenkama eilutės pradžia, atitinkamai pakeičiant eilutės ilgį.

Pavyzdžiui:

```
VAR   x : pchar;  - - -
x := 'abcdefghj';        { eilutės  x  ilgis 9 simboliai, reikšmė - 'abcdefghj' }
x := x + 5;              { eilutės  x  ilgis tapo 4 simboliai, reikšmė - 'fghj' }
x :=  x - 2;             { eilutės  x ilgis tapo 6 simboliai, reikšmė - 'defghj' }
```

ASCIIZ eilučių įdėja yra perimta iš Turbo C++ kalbos.

4.3. Aibės

Tai rinkinys elementų, kurie kažkokiu požiūriu tarpusavyje yra susiję. Į tokį rinkinį žiūrima kaip į sudėtinį vienetą. Aibių elementais turi būti skaliariniai dydžiai. Jos aprašomos žodeliais *set of* . Pavyzdžiui,

```
VAR  a : set of  1 .. 58;
     b : set of  'd' .. 'p';
     c : set of  char;
```

Čia *a* yra aibės tipo kintamasis, kurio elementais gali būti skaičiai tik iš nurodyto intervalo. Analogiškai *b* aibėje gali būti tik raidės iš nurodyto intervalo. *c* aibėje gali būti bet kokie simboliai. Vartojant TYPE, aibės gali būti aprašomos taip:

```
TYPE  aibe = set of 5 ..25;
VAR  k, m, w : aibe;
```

Išraiškose aibės rašomos tarp stačiakampių skliaustų, išvardinant juose elementus. Pavyzdžiai:

```
VAR   pirmsk, lygsk : set of  5 .. 20;
      balses, mykia, ttt  : set of  'f' .. 't';
- - -
pirmsk :=  [ 5, 7, 11, 13, 17, 19 ];
lygsk :=  [ 6, 8, 10, 12, 14, 16, 18, 20 ];
balses : = [ 'i', 'o' ];
mykia :=  [ 'l', 'm', 'n', 'r' ];
ttt := [];                          { tuščia aibė }
```

Aibėje gali būti ne daugiau kaip 256 elementai. Kontrolei vartojama opcija {$R+}.

Operacijos su aibėmis:

+ - sujungimas (sąjunga), * - persidengimas (sankirta),
- - skirtumas, IN - priklausymas aibei,
= , <> , <= , >= - palyginimo operacijos.

operandai ir operacija			rezultatas
['a' , ' b']	+	['m' , 'p' , 'x']	['a' , 'b' , 'm' , 'p' , 'x']
[1 , 3 , 4 , 7]	+	[2 , 4 , 8]	[1 , 2 , 3 , 4 , 7 , 8]
[1 , 3 , 4 , 7]	-	[2 , 4 , 7]	[1 , 3]
[0 .. 9]	-	[0 .. 5]	[6 .. 9]
[0 , 2 , 3 , 5 , 8 , 9]	*	[0 .. 5]	[0 , 2 , 3 , 5]

aibių palyginimas	rezultatas bus ***true*** , kai
a = b	aibės turi tuos pačius elementus
a <> b	aibių visi elementai nevienodi
a <= b	visi **a** aibės elementai yra aibėje **b**
a >= b	visi **b** aibės elementai yra aibėje **a**
c IN **a**	c elementas yra aibėje **a**

Aibių vartojimo pavyzdys:

```
VAR   skirtukai, balses : set of char;
          a : char;   - - -
skirtukai := [ ',' , '?' , '-' , '.' ];   balses := [ 'a' , 'e' , 'i' , 'o' , 'u' ];
READLN ( a );
IF  a IN skirtukai  THEN  WRITELN ( 'skyryba' );
IF  a IN (balses + skirtukai + [ 'm' , 'n' ] )  THEN  WRITELN (  … ); - - -
```

4.4. Įrašai

Masyvai, eilutės, aibės yra skirti vienodo tipo duomenims. Tačiau dažnai tenka charakterizuoti objektus duomenimis, kurių tipai skiriasi. Pavyzdžiui, žmogaus anketa. Joje pavardė - simbolių eilutė, gimimo metai - sveikasis skaičius, alga - realusis skaičius ir t.t. Tokiems duomenų rinkiniams programoje atspindėti yra vartojami įrašų tipo (RECORD) kintamieji.

Įrašų aprašymo pavyzdžiai:

```
VAR   anketa : RECORD
                    pavarde : string [15];
                    gimetai :  word;
                    alga : array [ 1 .. 12 ] of real;
                    pareigos :  string [25];
                END;
      data :  RECORD
                    metai :  1900 .. 2000;
                    menuo : (sausis, vasaris, …. , gruodis);
                    diena :  1 .. 31;
                END;
```

Programoje į atskirus įrašų elementus kreipiamasi jų sudėtiniais vardais:
anketa.pavarde , anketa.alga[7] , data.menuo ir t.t.

Operacijos su įrašais. Vienam įrašui galima priskirti kito įrašo reikšmes, jei jie abu yra tapatūs. Pvz., *airasas := birasas;* Įrašo vardas dar gali būti naudojamas tipizuotų ir netipizuotų failų (apie juos vėliau) įvedimo/išvedimo sakiniuose. Šiaip su įrašų elementais elgiamasi kaip su įprastais kintamaisiais.

Galimybė WITH. Kad įrašų elementams kažkurioje programos dalyje nereikėtų naudoti patikslintų vardų, rašomas WITH sakinys, jame nurodomas įrašo vardas ir tolau leistina naudoti tik elementų vardus. Pavyzdžiui,

```
                                                    tai ekvivalentiška
WITH  anketa  DO                          |
 BEGIN        pavarde := 'Jonaitis';      |      anketa.pavarde := 'Jonaitis';
              gimetai := 1967;            |      anketa.gimetai := 1967;
              alga [7] := 1250;           |      anketa.alga [7] := 1250;
 END;                                     |
```

Variantiniai įrašai. Įrašo viduje panaudojus konstrukciją CASE, galima turėti keletą panašių įrašų, besiskiriančių tik kažkuriuo elementu. Pavyzdžiui,

```
TYPE          variras  =  RECORD
                             a : real;
                             b : integer;
                             CASE  x : integer  OF
                             1, 3, 6 : ( r : real);
                             4, 2 : ( );                  { tuščias elementas }
                             8, 15 : ( zz : boolean );
                           END;
VAR    abc : variras;
```

Kai kintamasis abc.x programos bėgyje įgis, pvz., reikšmę 15, turėsime įrašą su abc.a , abc.b ir abc.zz elementais. Įgijus reikšmę 4, turėsime įrašą tik su dviem elementais (nes CASE konstrukcijoje reikšmę 4 atitinka tuščias elementas).

CASE konstrukcija gali būti tik įrašo pabaigoje.

Pateiksime dar vieną variantinio įrašo pavyzdį, kai CASE konstrukcijos valdantysis kintamasis yra kitokio tipo:

```
TYPE     vtk  =  ( jonas, petras, antanas );
VAR    kuku  =  RECORD
                    X : char;
                    Y : array [ 1 .. 25 ] of real;
                    Z : string[25];
                    CASE  k : vtk  OF
                          petras : ( a : array[ 1.. 6 ] of char );
                          antanas , jonas : ( b : string [12] );
                END;
```

4.5. Programų su struktūriniais kintamaisiais pavyzdžiai

1 pavyzdys.

Turime 50 skaičių masyvą. Klaviatūra įveskime juos į kompiuterio atmintį, o po to apskaičiuokime įvestų skaičių vidurkį bei raskime didžiausią ir mažiausią skaičius.

Rezultatus išveskime į monitorių taip: visų pirma išveskime visą masyvą po 10 skaičių kiekvienoje eilutėje, o po to vidurkį, didžiausią ir mažiausią skaičius.

Algoritmas.

Ieškant didžiausio elemento, visų pirma daroma prielaida, kad pirmas masyvo elementas (arba bet kuris kitas masyvo elementas) yra didžiausias. Jis įsimenamas atskirame kintamajame. Po to šio atskiro kintamojo reikšmė lyginama su visais masyvo elementais. Kai tik randamas dar didesnis masyvo elementas, atskirame kintamajame įsimenama jo reikšmė.

Analogiškai randamas ir mažiausias elementas.

```
PROGRAM  skaiciu_masyvo_apdorojimas;
        { programa apskaičiuoja skaičių masyvo vidurkį, randa didžiausią ir
         mažiausią masyvo elementus; autorius J.Jonaitis;  tel. 222222 }
USES        crt;
VAR         mas : array[ 1 .. 50 ] of real;
        did, maz, vid : real;
            j : integer;
begin
        ClrScr;
        WRITELN ( 'Įveskite 50 skaičių' );
        FOR  j := 1  TO  50  DO   READ ( mas[j] );
        vid := 0;                       { skaičių sumai kaupti }
        did := mas[1];                  { prielaida dėl didžiausio elemento }
        maz := mas[1];                          { prielaida dėl mažiausio
elemento }
        FOR j := 1  TO  50  DO
          begin     vid := vid + mas[j];              { kaupiama suma }
                    IF mas[j] > did  THEN  did := mas[j];
                    IF mas[j] < maz  THEN maz := mas[j];
          end;
        vid := vid / 50;        { gaunamas skaičių vidurkis }
        ClrScr;
        WRITELN ( 'Skaičiu masyvas' );
        FOR j := 1  TO  50  DO
          begin     IF  (j mod 10) = 0  THEN  WRITELN;
                    WRITE ( mas[j]:7:2 );
          end;
        WRITELN;
        WRITELN ( 'Skaičių vidurkis: ',  vid:9:3 );
        WRITELN ( 'Didžiausias skaičius: ', did:8:2 );
        WRITELN ( 'Mažiausias skaičius: ', maz:8:2 );
    end.
```

2 pavyzdys.

Tarkime, turime ne didesnį kaip 100 skaičių masyvą. Įveskime masyvo elementus ir išveskime juos į monitorių surūšiuotus (sudėliotus) didėjimo tvarka. Prieš vedant masyvo elementus visų pirma įvedamas skaičius, nurodantis masyvo dydį (elementų jame kiekį).

Algoritmas.

Masyvo elementų rūšiavimui yra visa eilė algoritmų. Imkime vieną iš paprastesnių – burbulo algoritmą (skaičiavimo trukmės požiūriu ne visais atvejais jis yra optimalus). Rūšiuojant lyginami du gretimi masyvo elementai ir, jeigu reikia, jie sukeičiami vietomis. Tokiu būdu "prabėgus" visą masyvą, didžiausias elementas perkeliamas į masyvo galą. Po to viskas daroma iš naujo, ir į antrą vietą nuo masyvo galo perkeliamas antras pagal dydį masyvo elementas ir t.t. Tam programoje reikia ciklo cikle.

```
PROGRAM  masyvo_elementu_rusiavimas;
USES  crt;
VAR   a : array[ 1 .. 100 ] of real;
      t : real;
      k, m, n : integer;
begin
      ClrScr;
      WRITELN ( 'Įveskite skaičių, kiek masyve yra elementų' );
      READLN ( k );
      WRITELN ( 'Įveskite masyvo elementus. Jų kiekis: ', k:3 );
      FOR  m := 1  TO  k  DO  READ ( a[m] );
                  { toliau rūšiuojame masyvo a elementus didėjimo
tvarka}
      FOR  m := 1  TO  k-1  DO
         FOR  n := 1  TO  k-m  DO
            IF  a[n] > a[n+1]  THEN
                              begin   t := a[n];
                                      a[n] := a[n+1];
                                      a[n+1] := t;
                              end;
                  { į monitorių išvedamas surūšiuotas skaičių masyvas }
      FOR  m := 1  TO  k  DO    WRITE ( a[m]:8:2 );
end.
```

Šios programos IF sakinyje pakeitus palyginimo ženklą priešingu, masyvo elementai būtų rūšiuojami mažėjimo tvarka.

3 pavyzdys.

Turime 6 x 8 matavimų matricą (dvimatį masyvą). Rasti jos didžiausią elementą ir į monitorių išvesti visus to stulpelio ir tos eilutės elementus, kuriuose yra didžiausias matricos elementas.

Algoritmas.

Matricos didžiausias elementas ieškomas panašiai, kaip ir vienmačio masyvo atveju (žiūr. 1 pavyzdį). Skirtumas toks, kad matricos atveju reikia ciklo cikle.

Čia mums rūpi ne pati didžiausio elemento reikšmė, bet jo vieta matricoje, t.y. indeksai. Todėl pradžioje, papildomam kintamajam suteikus kurio nors matricos elemento reikšmę (prielaida, kad jis yra didžiausias), būtina užfiksuoti ir jo indeksus. Tikrinimo metu radus didesnį elementą, užfiksuojami ir jo indeksai.

```
PROGRAM  matrica;
USES crt;
VAR   mtr : array [ 1..6, 1..8 ] of real;
      t : real;
      st, eil, m, n : integer;
begin
      ClrScr;
                  { įvedame matricos elementus}
      WRITELN ( 'Įveskite 6 x 8 matavimų matricos elementus' );
      FOR m := 1 TO  8  DO
         FOR n := 1 TO  6  DO   READ ( mtr[n,m] );
                  {ieškome didžiausio elemento matricoje mtr }
      t := mtr[1,1];
      st := 1;
      eil := 1;
      FOR m := 1  TO  8  DO
         FOR n := 1  TO  6  DO
            IF  mtr[n,m] > t  THEN  begin
                                       t := mtr[n,m];
                                       st := n;
                                       eil := m;
                                    end;
            {į monitorių išvedame atitinkamų stulpelio ir eilutės elementus}
      WRITELN ( 'Stulpelio elementai' );
      FOR m := 1  TO  8  DO   WRITE ( mtr[eil, m]:8:2 );
      WRITELN ( 'Eilutės elementai' );
      FOR n := 1  TO  6  DO    WRITE ( mtr[n, st]:8:2 );
end.
```

4 pavyzdys.

Sudarykime programą, kuri apskaičiuotų, kiek įvesto teksto eilutėje iš viso yra žodžių, ir keli iš jų prasideda raide ‘s’. Paprastumo dėlei laikykime, kad tekste yra tik mažosios lotynišos raidės.

Algoritmas.

Kadangi tarp žodžių yra koks nors skyrybos ženklas, o žodis prasideda raide, tai pakanka tikrinti porą greta stovinčių simbolių. Jei pirmas poros simbolis yra ne raidė, o antras – raidė, tuomet turime žodžio pradžią. Užfiksavus žodį, tikrinama, ar jis prasideda raide ‘s’. Simbolis yra raidė, jei jis yra daugiau arba lygu ‘a’ ir mažiau arba lygu ‘z’.

```
PROGRAM  zodziai_eilutese;
USES  crt;
VAR   tekst : string;
      zod, ks, j, n : integer;

Begin
      ClrScr;
      WRITELN ( ‘Įveskite eilutę teksto’ );
      READLN ( tekst );
      n := length ( tekst );    {nustatėm, kiek simbolių yra įvestoje teksto
eilutėje}
      zod := 0;                 {zod – iš viso žodžių}
      ks := 0;                  {ks – keli žodžiai prasideda raide s}
            {ar eilutė neprasideda žodžiu, t.y. ar pirmas simbolis raidė}
      IF   (tekst[1] >= ‘a’) and (tekst[1] <= ‘z’)  THEN
                  begin   zod := zod + 1;
                        IF  tekst[1] = ‘s’  THEN  ks := ks + 1;
                  end;
            {tikrinamos simbolių poros, kaip žodžio pradžios požymis}
      FOR  j := 2  TO  n-1  DO
         IF  (not ((tekst[j] >= ‘a’) and (tekst[j] <= ‘z’))) and
             ((tekst[j+1] >= ‘a’) and (tekst[j+1] <= ‘z’))   THEN
                  begin   zod := zod + 1;
                        IF  tekst[j+1] = ‘s’  THEN  ks := ks + 1;
                  end;
      WRITELN ( ‘Įvestoje teksto eilutėje iš viso yra ‘, zod, ‘ žodžių’ );
      WRITELN ( ks, ‘ žodžių prasideda raide s’ );
end.
```

5. PROCEDŪROS IR FUNKCIJOS

Turbo Paskalyje yra didelis standartinių (gatavų) procedūrų ir funkcijų rinkinys. Jos yra laikomos bibliotekiniuose moduliuose SYSTEM, CRT, GRAPH ir kt. Tam Turbo Paskalio programinėje įrangoje yra failai, pvz., CRT.TPU, GRAPH.TPU, kt. Programų USES skyriuje turime nurodyti bibliotekinių modulių vardus, kurių procedūras ir funkcijas žadame naudoti. Pvz., USES Crt, Graph;

Pateikime programos fragmentą, kuriame naudojamos standartinės procedūros ir funkcijos:

```
- - -    USES Crt, Graph; - - -
         BEGIN - - -
                ClrScr;                              {prodedūra iš CRT }
                READ ( a, b );                       {procedūra iš SYSTEM }
                x := sin(a) + exp(b);                {funkcijos iš SYSTEM }
                WRITELN ('rezultatas:  ' , x );      {procedūra iš SYSTEM }
                INITGRAPH (d, m 'c:\tp\bgi' );       {procedūra iš GRAPH }
                CIRCLE ( x, y, r );      - - -       {procedūra iš GRAPH }
         END.
```

Prisiminkime, kad SYSTEM bibliotekinio modulio USES skyriuje nurodyti nereikia. Jame esančios procedūros ir funkcijos visada yra prieinamos.

Tačiau Turbo Paskalio autoriai prikurti procedūrų ir funkcijų visiems gyvenimo atvejams negali. Todėl vartotojai (programuotojai) tik jiems reikalingas procedūras ir funkcijas turi susikurti patys. Toliau nagrinėsime klausimus, kaip galima sudaryti nuosavas procedūras ir funkcijas.

Apskritai, **procedūros ir funkcijos yra modulinio programavimo technologijos pagrindas.** Kiekvieną sudėtingą uždavinį tenka skaidyti į funkcionaliai užbaigtas dalis, t.y. modulius, ir programoje juos apipavidalinti kaip procedūras ar funkcijas. Tai įgalina programavimo darbą paskirstyti keliems žmonėms, naudoti aukštesnį programavimo technologijos lygį.

Paskalio procedūrų ir funkcijų programų tekstai rašomi po CONST, TYPE, VAR, kt. skyrių prieš pat pagrindinį programos sakinių skyrių. Programose į procedūrą ar funkciją kreipiamasi nurodant jos vardą ir faktinius parametrus (duomenis, reikalingus jos darbui). Kreipinyje parametrų gali ir nebūti.

Kreipinys į procedūrą - tai atskiras sakinys. Pvz.,

```
PROCVARDIS ( parametrai );
```

Kreipiniai į funkcijas gali būti tik išraiškose.

5.1. Procedūros

Sintaksiškai jų sudėtis yra tokia:

```
PROCEDURE  vardas ( … parametrai … );
  USES … ;
  LABEL … ;
  CONST … ;
  TYPE … ;
  VAR … ;
  BEGIN
  - - - sakiniai - - -
  END;                                   { po END - kabliataškis }
```

Konstantos, tipai, kintamieji, kt. aprašyti procedūroje galioja tik procedūros ribose. Tai lokalūs objektai. Prisiminkime, kad joje galios ir procedūrą gaubiančios programos elementai. Procedūrų lokaliems kintamiesiems vieta atmintyje paskiriama tik po kreipimosi į procedūrą. Procedūrai baigus darbą, ši atmintis išlaisvinama ir gali būti panaudota kitų procedūrų ar funkcijų darbe.

Procedūros antraštėje išvardinant parametrus, nurodomi ir jų tipai. Vieni parametrai naudojami perduoti duomenims į procedūrą iš kreipinio į ją, o kiti - perduoti procedūroje apskaičiuotas reikšmes į kreipinį. Todėl yra du parametrų nurodymo pavidalai:

1) PROCEDURE abc (x, y : real); Šiuo atveju kreipinių atitinkamų parametrų reikšmės bus perduodamos procedūros parametrams *x* ir *y*. Tarkime yra toks kreipinys: *abc(r, t);* Tuomet r --> x , t --> y (tik tokia kryptim).

2) PROCEDURE VTD (VAR a, b : integer); Prieš parametrus užrašius VAR, tokie parametrai gali ir priimti kreipinių parametrų reikšmes, ir procedūroje apskaičiuotas reikšmes perduoti kreipinių parametrams. Jei turime kreipinį *vtd(m, n);* , tai šiuo atveju reikšmių perdavimo kryptys m <--> a , n <--> b .

Procedūros antraštės pavyzdys su įvairiais parametrais:

```
PROCEDURE kuku ( a : integer; b,c : real; VAR m, n : char;
                                  p : byte; VAR x, y, z : boolean );
```

Kreipiniuose į *kuku* procedūrą turės būti toks pat atitinkamų tipų parametrų kiekis.

Kreipinių į procedūras parametrais (kitaip dar juos vadina faktiškaisiais parametrais) gali būti kintamieji, konstantos arba išraiškos. Jei kreipinio parametras yra konstanta arba išraiška, tai juos atitinkantys procedūros parametrai (kitaip juos dar vadina formaliaisiais parametrais) turi būti nurodyti pradedant VAR žodeliu (juk apsakaičiuotos reikšmės negalima perduoti konstantai ar išraiškai).

Faktiškųjų ir formaliųjų parametrų tipai būtinai turi būti vienodi.

Procedūros pavyzdys Nr. 1.

Užduoties sąlyga: Skaičiavimo metu išvedant rezultatus į ekraną, galima nustatyti norimą simbolių spalvą ir fono spalvą kiekvienai eilutei. Sudarykime procedūrą simbolių ir fono spalvos nustatymui.

```
PROGRAM procpav1;
  USES crt;
  VAR fonsp, simbsp : byte;
    PROCEDURE fonsimbsp ( f, s :byte );
    BEGIN
       Textbackground ( f );                    {standartinė procedūra}
       Textcolor ( s );                  {standartinė procedūra}
    END;
  BEGIN     ClsScr;                     {standartinė procedūra}
            - - -
            fonsp := 1;   simbsp := 4;
            fonsimbsp ( fonsp, simbsp );
            WRITELN ( 'raudoni simboliai melyname fone' );
            - - -
            fonsimbsp ( 2, 0 );
            WRITELN ( 'juodi simboliai zaliame fone' );
            - - -
            WRITELN ( 'taip pat juodi simb. zaliam fone' );
            END.
```

Procedūros pavyzdys Nr. 2.

Užduoties sąlyga: Reikia įvesti N skaičių masyvą (N < 100). Rasti jo didžiausią ir mažiausią elementus (1 procedūra), o taip pat apskaičiuoti, kiek elementų yra didesnių už elementų vidurkį ir kiek yra neigiamų elementų.

```
PROGRAM procpav2;
USES crt;
TYPE mas = array [ 1 .. 100 ] of real;
VAR a: mas;
       did, maz : real;
       m, n, kdv, kn : integer;
  PROCEDURE didmazel ( x : mas; k : integer; VAR de, me : real );
       VAR j : integer;
       BEGIN          de : = x [1];  me : = x [1];
```

```
                    FOR  j : = 1 to k  DO
                      BEGIN  IF  de < x [j]  THEN  de : = x [j];
                                 IF  me > x [j]  THEN  me : = x [j];
                      END;
        END;                                    {procedūros didmazel pabaiga}
    PROCEDURE dvn (x : mas; k : integer; VAR kdv, kn :integer );
        VAR  j : integer;
             vid : real;
        BEGIN
             vid : = 0;
             FOR  j : = 1 to k  DO   vid : = vid + x [j];
             vid : = vid / k;            {suradome elementų vidurkį}
             kdv : = 0;   kn : = 0;
             FOR  j : = 1 to k  DO
               BEGIN       IF  x[j] > vid  THEN  kdv : = kdv + 1;
                           IF  x[j] < 0      THEN  kn : = kn + 1;
               END;
        END;                                    {procedūros dvn pabaiga}
BEGIN
        ClrScr;
        WRITELN ('Kiek masyvo elementų norite įvesti?');
        READLN ( n );
        WRITELN ('Įveskite  ' , n , ' masyvo elementų');
        FOR  m : = 1 to n  DO  READLN ( a[m] );
        didmazel ( a , n , did, maz );
        dvn ( a , n , kdv , kn );
        ClrScr;
        WRITELN ( 'Masyvo mažiausias elementas:  ' , maz );
        WRITELN ( 'Masyvo didžiausias elementas:  ' , did );
        WRITELN ( 'Didesnių už vidurkį elementų kiekis:  ', kdv );
        WRITELN ( 'Neigiamų elementų kiekis:  ' , kn );
END.
```

5.2. Funkcijos

Sintaksiškai jų sudėtis yra tokia:

```
FUNCTION  funkcvardas ( … parametrai … ): tipas ;
  USES … ;
  LABEL … ;
  CONST … ;
  TYPE … ;
  VAR … ;
  BEGIN  - - -
       - - -  sakiniai  - - -
       funkcvardas : = apskaič.reikšmė;
  END;
```

Funkcijos sakinių skyriuje turi būti sakinys, kuriuo funkcijos vardui priskiriama apskaičiuota reikšmė. Baigus darbą, pastaroji ir yra funkcijos rezultatas.

Funkcijos ypatybė yra ta, kad ji apskaičiuoja tik vieną skaliarinę reikšmę, ir todėl jos *tipas* nurodomas FUNCTION sakinyje. Parametrai dažniausiai aprašomi be VAR žodelio (tačiau jis leistinas). Šiaip funkcijų parametrų aprašymas yra toks pat kaip ir procedūrų, ir su tuo susijusios taisyklės yra vienodos.

Kreipiniai į funkcijas dažniausiai vartojami įvairiose išraiškose.

Visos kitos funkcijų sąvybės, kaip objektų galiojimo sritis, atminties skyrimas lokaliems kintamiesiems, kt. yra yra tokios pačios kaip ir procedūrų.

Funkcijos pavyzdys.

Užduoties sąlyga. Sudarykime X kėlimo Y laipsniu funkciją. Matematinė išraiška yra tokia: X**Y = EXP (Y * LN (X)) .

```
PROGRAM  kelimlaipsn;
  USES  crt;
  VAR  a, b, c : real;
  FUNCTION  laipsnis ( x, y : real ) : real;
        BEGIN
              laipsnis : =      exp ( y * ln ( x ) );
        END;
  BEGIN       - - -
        a : = 2.83;  b: = 3.47;
        c : = 23.47 + laipsnis ( a, b );  - - -
        WRITELN ( b, laipsnis ( 6.25 , 3.6178 ), c );
  END.
```

FUNCTION sakinyje aprašant parametrus su žodeliu VAR, įmanoma sudaryti tokias funkcijas, kurios be apskaičiuojamos vienintelės reikšmės keistų ir kreipinio parametrų (argumentų) reikšmes, t.y. jos gali elgtis kaip funkcijos ir procedūros vienu metu.

5.3. Procedūrų/funkcijų parametrų reikšmių perdavimo klausimai

Kreipinio į proc./f-ją parametrų (faktinių parametrų) reikšmių perdavimas proc./f-jų parametrams (formaliesiems parametrams) Turbo Paskalyje vyksta tokiais būdais:

1) kai **proc./f-jos antraštėje (PROCEDURE arba FUNCTION sakinyje) prieš parametrus nėra žodelių VAR arba CONST**:

   ```
   PROCEDURE  abc ( p1, p2 : integer; p3, p4 : real ); … END;
   abc ( a, b, c, d );          {kreipinys į  abc  procedūrą}
   ```

   Kreipinio parametrų *a, b, c, d* reikšmės persiunčiamos į tas atminties vietas, kurios, pradėjus procedūrai darbą, buvo paskirtos pastarosios parametrams *p1, p2, p3, p4*. (a, b, c, d → p1, p2, p3, p4).

   Šiuo atveju failas negali būti parametru.

2) kai **proc./f-jos antraštėje prieš parametrus yra VAR** žodelis:

   ```
   PROCEDURE  abc ( VAR p1, p2 : byte; VAR p3 : real ); … END;
   abc ( a, b, c );
   ```

   Kreipinio parametrams a, b, c paskirtų atminties sričių adresai, procedūrai pradedant darbą, perduodami pastarąjai, ir procedūros parametrams p1, p2, p3 paskiriamos tos pačios atminties sritys kaip ir kreipinio parametrams a, b, c . (a, b, c ←→ p1, p2, p3).

   Šiuo atveju kreipinio parametrais negali būti konstantos ir išraiškos, pvz., abc (a , 2*a + 3 , 16.74) . Failas gali būti parametru !!!

3) kai **proc./f-jos antraštėje prieš parametrus yra CONST** žodelis:

   ```
   PROCEDURE  abc ( CONST  p1, p2 : real );
   abc ( a, b );
   ```

   Kreipinio parametrų a, b adresai perduodami procedūrai. Procedūros parametrai p1, p2 gali paimti a, b reikšmes, bet pakeisti jų negali.

   Kreipinio parametru gali būti kintamasis, išraiška, konstanta. Failas negali būti parametru.

Procedūros parametrui, aprašytam su CONST, programoje negalima priskirti reikšmės. Tokio parametro negalima naudoti kreipiniuose į kitas procedūras/funkcijas.

Palyginkime šį parametrų perdavimo būdą su 1-uoju variantu, kai procedūroje prieš parametrus nėra nei VAR, nei CONST. 1-ojo varianto atveju parametrams procedūros viduje galima priskirti reikšmes ir kt. Parametrus-konstantas (lyginant su 1-uoju variantu) patogu naudoti masyvų, įrašų ir pan. atvejais. Nereikia persiuntinėti daug reikšmių, o pakanka perduoti tik struktūrinio (sudėtinio) tipo konstantos adresą. Pateiksime keletą struktūrinio tipo konstantų aprašymo pavyzdžių:

a) masyvas-konstanta
```
CONST  mas : array [ 1 .. 4 ] of real = ( 0.25, 3.72, -63.457, 42.1 );
```
b) įrašas-konstanta
```
TYPE  ras = record
                a : real;  b : integer; c : char;
            end;
CONST  x : ras = ( a : 25.31; b : 45; c : 'G' );
```
c) aibė-konstanta
```
TYPE  abe = set of  0 .. 14;
 CONST  x : abe = [ 2, 3, 8, 11 ];
```

4) kai **netipizuoti parametrai procedūroje**:
```
PROCEDURE  paty ( VAR  p1, p2 ;  p3 : real );
paty ( a, b, c );
```
Šiuo atveju procedūrai perduodami kreipinio parametrų *a, b* (netipizuotų) adresai, o parametro *c* (tipizuoto) - reikšmė.

Tačiau procedūros viduje netipizuotiems parametrams būtina suteikti tipą. Tai atliekama ypatingu būdu, kurį pailiustruosime pavyzdžiu:
```
PROGRAM   rusis;
  TYPE  mxm = array [ 1 .. 300 ] of byte;
  VAR  a : array [ 1 .. 5 ] of byte;
       b : array [ 1 .. 27 ] of char;
  PROCEDURE  tre ( VAR m; k : word );
       - - -
       maz := mxm (m) [j];    - - -
       mxm ( m) [n] := mxm (m) [j]; - - -
  END;
BEGIN  - - -  tre ( a, 5 ); - - -          {tvarkys byte tipo masyvą}
        - - -  tre ( b, 27 ); - - -        {tvarkys char tipo masyvą}
END.
```
tre procedūra tinka įvairaus ilgio masyvams (dažniausiai - vienodo tipo masyvams; jei masyvų elementai atminty užima tiek pat baitų, pvz., *byte, char, shortint*, galima taikyti ir skirtingo tipo masyvams).

5) kai **procedūroje yra parametrai - atvirieji masyvai**:
```
PROCEDURE amfora ( a : array of integer );
amfora ( x );                    { pvz., VAR  x : array [ 1 .. 25 ] of integer;}
amfora ( y );                    { pvz., VAR  y : array [ 5 .. 17 ] of integer;}
```
Po kreipimosi į procedūrą jos viduje standartinėmis funkcijomis *Low(a)* ir *High(a)* galima gauti parametro-masyvo indeksų kitimo rėžius
(pvz., s := 0; FOR j := Low(a) to High(a) DO s := s + a[j];).

Lyginant su netipizuotų parametrų atveju, ten indeksų apatinė riba turėjo būti vienoda (lygi 1) visiems masyvams. Čia abi indeksų ribos gali būti skirtingos.

6) kai **procedūroje yra parametrai - atvirosios eilutės**:
```
PROCEDURE abba ( VAR  ss : openstring );
abba ( x );          { VAR x : string[27];  x ilgis gali būti įvairus}
```

Jei PROCEDURE sakinyje prieš parametrą nėra VAR, tuomet nesvarbu, ar rašysime *openstring* ar *string*.
Po kreipimosi į procedūrą jos viduje standartine funkcija *Length (ss)* galima sužinoti parametrų-eilučių ilgį.
{$P+} - kompiliatoriaus nuoroda dėl atviruųjų eilučių. Panaudojus pastarąją, nesvarbu ką rašysime *openstring* ar *string*.
Tas pats bus, jei

```
{$P+}
PROCEDURE  abba ( VAR  ss : string ); - - -
```

7) kai parametrai - procedūros/funkcijos:
Procedūros ar funkcijos antraštėje formaliu parametru gali būti procedūra arba funkcija. Pvz.,
PROCEDURE aaa(VAR x:real; PROCEDURE qwe(VAR p,q:real));
FUNCTION bbb(a,b:real; FUNCTION trg(d,f:integer):real);
Kreipiniuose į tokias proc./f-jas kaip parametras rašomas tik proc. arba f-jos vardas. Pvz., aaa(x, pp1); arba bbb(x,y,fff) . Šiuose pavyzdžiuose *pp1* - procedūra su dviem parametrais, *fff* - funkcija taip pat su dviem parametrais.

5.4. Nuorodos kompiliatoriui {$I failas} vartojimas

Tarkime, kad kažkokia procedūra/funkcija yra reikalinga keliose skirtingose programose. Kad kiekvienoje iš tų programų nereikėtų vis iš naujo klaviatūra surinkinėti tokios pačios procedūros/funkcijos teksto, galima surinkti proc./f-ją tik vieną kartą ir užrašyti magn. diskų *.txt* faile. Į tas programas, kurioms reikia tos proc./f-jos, reikės įdėti tik atitinkamą nuorodą, t.y. {$I failo_vardas}. Pavyzdys:

```
PROGRAM aaa;
USES - - - ;
TYPE - - - ;
VAR - - - ;
{$I c:\katal\proce2.pas}        {į šią vietą kompiliatorius įstatys  c:  disko
                                  katal  katalogo failo  proce2.pas  turinį}
BEGIN  - - -
- - -
END.
```

Kompiliatoriaus nuoroda {$I failas} galima įterpti ir USES, CONST, TYPE, VAR skyrius, užrašytus atskiruose failuose. Tačiau šios nuorodos negalima vartoti po BEGIN sakinių skyriuje.

Atkreipkime dėmesį į tai, kad Paskalyje yra ir nuorodos {$I-} bei {$I+} įvedimo/išvedimo kontrolei. Nepainiokime jų su nuoroda {$I failas}.

5.5. UNIT moduliai (vartotojo bibliotekiniai moduliai)

Savo sukurtas proc./f-jas vartotojas gali pasidėti į asmeninę biblioteką, t.y. susikurti asmeninį bibliotekinį modulį. Tai duoda galimybę naudotis tokio bibliotekinio modulio procedūromis ir funkcijomis visose programose, kur to prireikia.

Prisiminkime, kad įvairios standartinės proc./f-jos laikomos CRT, GRAPH ir kt. bibliotekose. Prireikus šių bibliotekų, programose, tai nurodome USES skyriuje. Analogiškai reikia elgtis ir su asmeninėmis bibliotekomis. Pvz., USES Crt, Graph, Asmbib, Bbilt3;

Kuriamojo bibliotekinio modulio (UNIT modulio) sudėtis yra tokia:

```
UNIT  vardas;
INTERFACE
        USES - - - ;                    { Tai globalūs elementai,     }
        CONST - - - ;                   { kurie galios programose,    }
        TYPE - - - ;                    { naudojančiose bibl.modulį }
        VAR - - - ;
        proc./f-jos1 antraštė su parametrais;
        proc./f-jos2 antraštė su parametrais;
        - - -
        proc./f-josN antraštė su parametrais;
IMPLEMENTATION
        USES - - - ;                    { Tai lokalūs elementai,    }
        CONST - - - ;                   {  kurie galioja tik bibl.    }
        TYPE - - - ;                    {  modulio proc./f-jose     }
        VAR - - - ;
        PROCEDURE vardas( --- );  - - -  END;
        FUNCTION vardas ( --- ):tipas; - - - END;

        Proc./f-jos1 antraštė be parametrų;
                BEGIN  - - -  END;
        Proc./f-jos2 antraštė be parametrų;
                BEGIN  - - -  END;
        - - -
        Proc./f-jos antraštė be parametrų;
                BEGIN  - - -  END;
BEGIN  - - -
- - - sakiniai - - -                    {  inicializavimo dalis  }
END.
```

UNIT modulio inicializavimo dalyje rašomi tokie sakiniai, kuriais priskiriamos pradinės reikšmės ryšio (INTERFACE) dalyje aprašytiems globaliems kintamiesiems. Inicializavimo dalyje sakinių gali ir nebūti, bet BEGIN ir END turi išlikti.

UNIT modulis užrašomas į failą *vardas.pas*, kurio vardas turi būti toks pat kaip ir esantis po UNIT sakinio.

UNIT modulis kompiliuojamas (apdorojamas Turbo Pascal kompiliatoriumi) panašiai, kaip ir Paskalio programos. Rezultate gaunamas TPU (Turbo Pascal Unit) tipo failas, t.y. *vardas.tpu* . Atkreipkime dėmesį į tai, kad UNIT modulis yra tik atitinkamai sutvarkytų procedūrų ir funkcijų saugykla. USES sakinyje nurodyti UNIT moduliai paimami ir įtraukiami į vartotojo programą ir, kompiliuojant pastarąją, bus gaunamas pilnas vartotojo programos .EXE failas (UNIT moduliai į vartotojo programą įtraukiamos statiškai, t.y. visam programos vykdymo laikui).

Kompiliuodami UNIT modulį atkreipkime dėmesį, kad TURBO PASCAL aplinkos COMPILE meniu opcija DESTINATION turėtų reikšmę DISK, o ne MEMORY. Sukompiliuotas modulis turi būti užrašytas diske, o ne pasilikti operatyvioje atmintyje.

UNIT modulio sudarymo pavyzdys:

Sąlyga.

Sudarykime modulį, kuriame būtų:

1) procedūra išvesti tekstui pageidaujamoje ekrano vietoje;
2) procedūra išvesti tekstui pageidaujamoje ekrano vietoje ir nurodyto dydžio bei spalvos lange.

```
UNIT  Ekranas;
INTERFACE
        USES  Crt;
        VAR   Fonas : byte;
        PROCEDURE  Viet ( x1, y1: byte; tks : string );
        PROCEDURE  Lang ( x1, y1, x2, y2 : byte; tks : string );
IMPLEMENTATION
        PROCEDURE   Viet;
                BEGIN           Goto ( x1, y1 );
                                Write ( tks );
                END;
        PROCEDURE  Lang;
                BEGIN           Window ( x1, y1, x2, y2 );
                                TextBackGround ( fonas );
                                ClrScr;          {vykdant ClrScr kartu nudažomas langas}
                                Write ( tks );
                                Window ( 1, 1, 80, 25 );
                                TextBackGround ( 0 );
                END;
BEGIN
        Fonas := 1;                     {mėlynai}
END.
```

Šiame pavyzdyje *fonas* yra globalus kintamasis. Jis galios visose programose, kuriose bus USES *Ekranas*;

Toliau parašykime programą, kurioje būtų naudojamos mūsų sudaryto UNIT modulio *Ekranas* procedūros.

```
PROGRAM  pvz;
        USES  Crt, Ekranas;
        VAR  x : string;
        BEGIN           - - -
                        Viet ( 25, 8,  'spausdins 8 eilutę nuo 25 pozicijos' );
                        x := 'Naujas tekstas';
                        Lang ( 15, 4, 40, 7, x );       {spausdins mėlyname lange}
                        Fonas := Red;                   { red - raudona spalva        }
                        Lang ( 30, 6, 45, 8, x );       {spausdins raudonam lange }
                        - - -
        END.
```

Šioje programoje naudojami vardai *Fonas* ir *Red ,* nors jie nėra čia aprašyti. *Fonas* yra "atėjęs" iš mūsų sudaryto UNIT modulio *Ekranas* kaip globalus kintamasis, o *Red* - kaip Paskalio standartinio modulio *Crt* globalus elementas (vardinė konstanta).

5.6. Dinamiškai susijusios bibliotekos (DLL)

DLL (angl. DLL – Dynamically Linked Libraries) gali būti naudojamos tik šiose Turbo Paskalio (TP) versijose: BP.EXE, BPC.EXE ir BPW.EXE (ši -Windows'e). Yra skirtingos TP versijos, galinčios dirbti tik atitinkamoje aplinkoje (operacinėje sistemoje):

1. TP versija, veikianti MS DOS aplinkoje, esant realiam procesoriaus darbo režimui (TURBO.EXE, TPC.EXE).
2. TP versija, veikianti MS DOS aplinkoje, esant apsaugotam (protected) procesoriaus darbo režimui (BP.EXE, BPC.EXE).
3. TP versija, veikianti Windows aplinkoje (BPW.EXE).

Šiose TP versijose atitinkamai suformuojami arba naudojami tokio tipo failai:

1) .EXE (EXEcutable moduliai, gaunami sukompiliavus programas), .TPU (Turbo Pascal Unit moduliai), TURBO.TPL (Turbo Pascal Library - pagrindinė standartinių Paskalio modulių biblioteka);
2) .EXE, .TPU, .TPP, .TPW, .DLL, TPP.TPL (TPP - Turbo Pascal Protected);
3) .EXE, .TPU, .TPP, .TPW, .DLL, TPW.TPL (TPW - Turbo Pascal Windows).

Pirmiausiai DLL buvo pradėtos naudoti Windows aplinkoje, ir tik vėliau jos buvo įdiegtos MS DOS aplinkoje.

DLL vertingiausias bruožas yra tas, kad jas, įkrovus į operatyviąją atmintį, vienu metu gali naudoti kelios taikomosios programos. Nežiūrint to, kad MS DOS aplinkoje DLL naudojamos dirbant procesoriui apsaugotuoju režimu, jos yra visiškai suderintos su Windows aplinkos DLL. DLL gali naudoti įvairiomis programavimo kalbomis (pvz., Paskaliu, C++, kt.) rašomos programos. Kitaip tariant, kitomis programavimo kalbomis sukurtas DLL gali vartoti Paskalio programos ir atvirkščiai.

Paskalio kalbos požiūriu DLL yra UNIT bibliotekinių modulių tęsinys (patobulinimas), nežiūrint, kad DLL ir turi kai kuriuos apribojimus, palyginus su UNIT moduliais.

DLL skiriamieji bruožai:

- UNIT moduliai į kiekvieną juos naudojančią taikomąją programą įtraukiami statiškai kompiliavimo metu, o DLL moduliai iškviečiami į operatyviąją atmintį tik taikomosios programos vykdymo metu. Tai įgalina kelioms vienu metu dirbančioms taikomosioms programoms laikyti operatyviojoje atmintyje tik vieną DLL modulio egzempliorių;
- DLL modulių sukompiliuotos programos (jų kodai) ir kiti reikalingi resursai, skirtingai nei UNIT modulių atveju, nėra įtraukiamos (sujungiamos) į juos naudojančią taikomąją programą, o yra laikomi atskirame .DLL tipo faile. Pastarasis failas turi būti prieinamas taikomosios programos darbo metu. DLL viduje esančios procedūros ir funkcijos su jas naudojančia taikomąja programa susiejamos dinamiškai;
- DLL siauresnės galimybės, lyginant su UNIT, pasireiškia tuo, kad DLL'uose nėra globaliųjų konstantų, tipų, kintamųjų, kt. (UNIT moduliuose jie aprašomi INTERFACE skyriuje), o yra tik globaliai galiojančios procedūros ir funkcijos. DLL'uose aprašyti kintamieji pasiekiami tik kreipusis į jų procedūras ar funkcijas;
- kompiliuojant vartotojo programą, DLL panaudojamos automatiškai. UNIT moduliai paimami ir kompiliuojami kartu su visa vartotojo programa. DLL moduliai kompiliuojami atskirai nuo vartotojo programų.

DLL bibliotekos sudarymas. Pailiustruokime tai pavyzdžiu:

Užduotis. Sudarykime dinaminę biblioteką apdoroti vektoriams (vienmačiams masyvams).

```
LIBRARY  vektor;              { bibliotekos antraštė }
USES Globkint, Crt;           { Globkint  yra asmeninė UNIT tipo biblioteka }
PROCEDURE  vektspausd ( vekt : Tvekt; n : byte );
                                   { tipas Tvekt  ateina iš Globkint UNIT'o }
        VAR  j : byte;
        BEGIN       WRITELN ( 'vektoriaus elementai:' );
                    FOR  j := 1  TO  n  DO  WRITE ( vekt[j] : 8 );
                    WRITELN;
        END;
PROCEDURE  VekDidEl ( vekt : Tvekt; n : byte ); EXPORT;
        VAR  mak : integer;
             j : byte;
        BEGIN       mak := vekt[1];
                    FOR  j := 1  TO  n  DO
                          IF  vekt[j] > mak  THEN  mak := vekt[j];
                    ClrScr;
                    Vektspausd ( vekt, n );
                    WRITELN ( 'Maks. El. : ', mak );
        END;
PROCEDURE  VekMazEl ( vekt : Tvekt; n : byte ); EXPORT;
        - - -
        END;
PROCEDURE  VekInvers (vekt : Tvekt; n : byte ); EXPORT;
        - - -
        END;
EXPORTS                       { atkreipkim dėmasį: EXPORTS su S gale }
VekDidEl   index 1;
VekMazEl  index 2;
VekInvers  index 3;

BEGIN       - - -
        { inicializacijos dalis. Jos gali ir nebūti }
END.
```

Atkreipkime dėmesį į EXPORT; sakinį, einantį po procedūros ar funkcijos antraštės, ir į EXPORTS sakinį.

EXPORT sakiniu liepiama suformuoti tolimąjį (far) kreipimosi į proc./f-ją būdą, kai proc./f-jos adresui nurodyti naudojami keturi baitai, ir parengia tokią proc./f-ją eksportavimui, suformuojant jai specialius įėjimo ir išėjimo taškus. Tačiau proc./f-ja nebus eksportuojama, jei jos vardo nenurodysime po EXPORTS sakinio.

EXPORTS sakinyje po proc./f-jos vardo dar nustatomas jos eilės numeris bibliotekoje (index n). Matysime, kad iš DLL bibliotekos proc./f-ją galima bus išsikviesti, nurodant jos eilės numerį.

LIBRARY inicializacijos dalies sakiniai, kaip ir UNIT modulių atveju, atliekami tik vieną kartą, tačiau ne vartotojo programos paleidimo metu, o įkraunant DLL biblioteką į operatyviąją atmintį (juk DLL gali naudoti kelios programos). Kad vartotojo programa žinotų, ar DLL biblioteka jau įkrauta į operatyviąją atmintį, kiekvienai DLL sukuriamas vartojimo skaitliukas. Jis parodo, kiek skirtingų programų naudoja tą DLL. Naujai vartotojo programai prireikus DLL, tas skaitliukas padidinamas vienetu. Kai tik šio skaitliuko reikšmė pasidaro lygi nuliui, DLL pašalinama iš operatyvios atminties. DLL įkraunama į operatyviąją atmintį, prireikus jos pirmą kartą.

Procedūrų/funkcijų importas iš DLL. Programos parašytos *Borland Pascal with Objects* kalba (2 ir 3 TP versijos, žiūr. šio paragrafo pradžią), gali kviestis proc./f-jas iš DLL trim būdais:

1) nurodant tikrąjį proc./f-jos vardą;
2) kviečiamai proc./f-jai suteikiant naują vardą;
3) naudojant proc./f-jos eilės numerį bibliotekoje.

Proc./f-jų, kurias reikia importuoti iš DLL, antraštės nurodomos atitinkamose vietose vartotojo programoje (po aprašų) arba UNIT modulyje (*interface* ir *implementation* skyriuose). Tam būtina:

- po proc./f-jos antraštės rašyti *external* (UNIT'e *implementation* skyriuje);
- po *external* nurodyti DLL vardą;
- naudoti kompiliatoriaus nuorodą {$F+} arba *far* kreipimosi į proc./f-jas modelį (*far;* sakinys rašomas po proc./f-jos antraštės).

Proc./f-jos importas iš DLL tikruoju vardu. Pvz., mūsų turėto pavyzdžio DLL bibliotekos *vektor* procedūrai *VekDidEl* reikės rašyti taip:

```
PROCEDURE  VekDidEl;  EXTERNAL 'vektor';
```

Proc./f-jos importas iš DLL, suteikiant proc./f-jai naują vardą. Pvz.,

```
PROCEDURE VekDidEl;  EXTERNAL 'vektor' NAME 'naujvard';
```

Vartotojo programoje į šią proc. reikės kreiptis ne vardu *VekDidEl* , o vardu *naujvard* .

Proc./f-jos importas iš DLL, naudojant proc./f-jos eilės numerį bibliotekoje. Pvz.,

```
PROCEDURE VekDidEl; EXTERNAL 'vektor' INDEX 1;
```

Naudojant numerius, pagreitinamas proc./f-jų įkrovimas į operatyviąją atmintį.

UNIT modulis, importuojantis DLL proc./f-jas. Importuojantys UNIT moduliai dažniausiai naudojami pagerinti programos struktūriškumui ir vėlesnei jos priežiūrai. Tokiuose moduliuose dažniausiai surašomi importuojamų proc./f-jų vardai ir tipai bei konstantos, reikalingi ryšiui su DLL. Importo moduliai nėra būtini, jie tik palengvina DLL naudojančių programų kūrimą.

Sudarykime UNIT modulį, kuris importuoja procedūras iš mūsų turėto pavyzdžio DLL bibliotekos VEKTOR.

```
UNIT  impvz;
INTERFACE
        USES  globkint;
        PROCEDURE VekDidEl ( vekt : Tvekt; n : byte );
        PROCEDURE  VekMazEl ( vekt : Tvekt; n : byte );
        PROCEDURE  VekInvers ( vekt : Tvekt; n : byte );
IMPLEMENTATION
        PROCEDURE  VekDidEl;  EXTERNAL 'vektor' INDEX 1;
        PROCEDURE  VekMazEl; EXTERNAL 'vektor' INDEX 2;
        PROCEDURE  VekInvers;  EXTERNAL 'vektor' INDEX 3;
END.
```

Šį UNIT modulį, o tuo pačiu ir VEKTOR.DLL, naudojančios programos pavyzdys:

```
PROGRAM  kuku;
USES  Crt, Globkint, Impvz;
VAR   a : Tvekt;                    { tipas Tvekt ateina iš Globkint UNIT'o }
        k, n : byte;
BEGIN          - - -
        VekDidEl ( a, n );      - - -
        VekMazEl ( a, n ); - - -
END.
```

Vartotojo programoje, nenaudojant imprtuojančio UNIT modulio *impvz*, DLL procedūras galima importuoti, pavyzdžiui, taip:

```
PROGRAM  kuku;
USES  Crt, Globkint;
```

```
VAR  a : Tvekt;                          { tipas Tvekt ateina iš Globkint UNIT'o }
        k, n : byte;
PROCEDURE  VekDidEl  ( a : Tvekt; j : byte ); EXTERNAL 'vekt';
PROCEDURE  VekMazEl ( a : Tvekt; j : byte ); EXTERNAL 'vekt'
                                                        NAME 'mazas';
BEGIN  - - -
        VekDidEl ( a, n );       - - -
        Mazas ( a, n ); - - -
END.
```

6. DUOMENŲ ĮVEDIMAS/IŠVEDIMAS. FAILAI

6.1. Bendrieji klausimai

Failai (bylos, rinkmenos) - tai duomenų rinkiniai (tekstai, skaičiai, vaizdai, garsai) išoriniuose kompiuterio įrenginiuose. Pvz., duomenys į kompiuterio atmintį gali būti įvedami klaviatūra, iš disko, kompiuterių tinklo ryšio kanalu, mikrofonu, vaizdo kamera, kt. Iš kompiuterio atminties duomenys (skaičiavimo rezultatai) dažniausiai išspausdinami, išvedami į monitorių (vaizduoklį), magnetinį diską, garsiakalbį, perduodami kompiuterių tinklo ryšio kanalu.

Failai egzistuoja atskirai nuo programų. Sukurti failai saugomi įvairiose informacijos laikmenose (diskeliuose, kompaktiniuose diskuose, popieriuje). Norint jau sukurtą failą skaityti, kompiuterio programoje turi būti nurodyta, kokiu įrengimu tai norima atlikti. Kuomet failas yra kuriamas (rašomas), kompiuterio programoje turi būti nurodyta, kuriuo įrenginiu (atitinkamai laikmenoje) tai norima daryti. Taigi, kompiuterio programose turi būti kintamieji, kuriais būtų pažymėti (atstovaujami) failai. Paskalio kalboje tokie kintamieji dažniausiai aprašomi baziniu žodžiu *file*, po kurio gali eiti ir kiti baziniai žodžiai. Jie dažniausiai aprašomi taip:

```
VAR  s : file  of  integer;                    {sveikųjų skaičių failas}
       r : file  of  real;                     {realiųjų skaičių failas}
       t : file  of  char;                     {simbolių failas}
       m : file of array[1..25] of  longint;   {failas, kurio elementas yra
                                                masyvas}
       x : file;                               {toks failas naudojamas įvesti
                                                ar išvesti nurodytą baitų kiekį}
```

Dirbdami kompiuteriu, duomenis dažniausiai įvedame klaviatūra, o išvedame į monitorių ar spausdintuvą. Perduodami duomenys yra *char* tipo (įvedam ir išvedam tekstą mums suprantamais simboliais). Todėl Paskalyje paprastumo ir aiškumo dėlei yra numatyti tekstiniai failai, kurie aprašomi taip:

```
VAR  t : text;
```

Programoje aprašytas kintamasis-failas turi būti susiejamas su įrenginiu, kuriuo bus skaitomas ar užrašomas realus duomenų failas. Tam yra sakinys

```
ASSIGN (kintamasis-failas, įrenginys);
```

Juo, kaip ir priskyrimo sakiniu, kintamajam-failui priskiriamas įrenginys, su kuriuo bus dirbama. Pvz.,

```
ASSIGN (a, 'LPT1');                    {LPT1 yra spausdintuvo vardas}
ASSIGN (b, 'c:\destytojas1\jonas.dd'); {bus dirbama su nurodytu failu}
ASSIGN (c, kintamasis-string);         {darbas priklausys nuo
                                        kintamajam-string įvestos
                                        reikšmės, pvz., 'a:\kuku.sk'}
```

Šiuose pavyzdžiuose *a, b* ir *c* yra kintamųjų-failų vardai.

Prieš duomenų rašymą ar skaitymą, failai turi būti atidaromi, o baigus darbą - uždaromi. Failo atidarymo prasmė yra ta, kad, pvz., išvedant duomenis į diską, diske visų pirma turi būti parengta sritis kuriamam failui (kataloge įrašomas failo vardas, paskiriama jam vieta diske), programai atmintyje papildomai paskiriami laukai, kurie vadinami įvedimo/išvedimo buferiais. Uždarant failą, pvz., diske užrašomas failo pabaigos požymis (jei failas buvo kuriamas), išlaisvinama įvedimo/išvedimo buferiams skirta atminties sritis.

Failai atidaromi sakiniais

```
REWRITE (a);    - kuriamam failui, kai bus rašoma į jį (WRITE),
RESET (b);      - failui, iš kurio bus skaitoma (READ).
```

o uždaromi sakiniu

```
CLOSE (a);
```

Šiuose pavyzdžiuoe *a* ir *b* yra kintamųjų-failų vardai.

Taigi, programose su failais turi būti tokia skinių eilė:

```
ASSIGN, ...,    REWRITE, …,    WRITE, …,    CLOSE
                RESET          READ
```

kt.

Programoje aprašytiems failams naudojami įvedimo/išvedimo sakiniai, kuriuose turi būti nurodytas kintamasis-failas. Pvz.,

READ (a, … kintamieji …);
READLN (aa, … kintamieji …);
WRITE (b, … kintamieji …);
WRITELN (bb, … kintamieji …);
BLOCKREAD (c, kintamasis, baitų-kiekis);
BLOCKWRITE (d, kintamasis, baitų-kiekis); ir t.t.

Čia *a*, *aa, b, bb, c* ir *d* yra kintamieji-failai.

Pastebėkime, kad lig šiol mūsų rašytose programose READ, READLN, WRITE, WRITELN sakiniuose kintamųjų-failų nenurodydavome. Iš tikrųjų juose yra naudojami standartiniai kintamieji-failai *input* ir *output*, kurių programose nereikia nei aprašyti, nei nurodyti įvedimo/išvedimo sakiniuose. Jie yra *text* tipo failai. Pvz., sakiniai

READ (input, … kintamieji …); ir READ (… kintamieji …);

duoda tokį patį rezultatą. Analogiškai yra su

WRITE (output, … kintamieji …); ir WRITE (… kintamieji …);

sakiniais.

Paskalyje yra trys failų rūšys:

- tekstiniai failai (VAR a : text;);
- tipizuoti failai (VAR b : file of integer;);
- netipizuoti failai (VAR c : file;).

Išnagrinėkime juos plačiau.

6.2. Tekstiniai failai

Paskalio programose tekstiniai failai aprašomi baziniu žodžiu *text*:

```
VAR  a, b, c : text;          {čia aprašėme tris failus - a, b ir c}
```

Tekstiniai failai susideda iš simbolių eilučių, kurių ilgis gali būti nuo 0 iki 255 simbolių. Juk klaviatūra duomenis įvedame eilutėmis, o į monitorių ar spausdintuvą duomenis išvedame taip pat eilutėmis.

Kadangi eilučių ilgis gali būti įvairus, būtina kažkaip nurodyti jų pabaigą. Tam naudojami du kompiuterių valdantieji simboliai, kurie vadinami CR (Carriage Return, kodas - 13) ir LF (Line Feed, kodas - 10). Pvz., vedant duomenis klaviatūra ir paspaudus klavišą ENTER, įvestos simbolių eilutės gale užrašomi minėti CR ir LF simboliai. Pagal juos kompiuterio operacinė sistema supranta, kad baigėm vesti eilutę. Išvedant duomenis į monitorių, eilutėje išvedama simbolis po simbolio. Kai tik ateina CR ir LF simboliai, pereinama į kitą monitoriaus eilutę. Jeigu tekstinį failą rašome į diską, jame užrašomi ir CR bei LF simboliai. Skaitydami nuo disko tekstinį failą, eilutės pabaigą galime užfiksuoti standartine funkcija EOLN(kintamasis-failas). Pastaroji funkcija tikrina, ar gautas simbolis nėra CR ar LF. Kai gaunami šie simboliai, EOLN( ) funkcijos reikšmė tampa *true*. Šios funkcijos vartojimo pavyzdžiai:

```
VAR  a : text;
       s : char;
WHILE  not EOLN(a)  DO  BEGIN … READ(a, s);  … END;
REPEAT … READ(a, s); … UNTIL  EOLN(a);
```

Atkreipkime dėmesį, kad magnetiniame diske patalpinto tekstinio failo gale yra užrašomas specialus valdantysis simbolis (jo kodas - 26). Skaitant tokius failus, jų pabaigą užfiksuoti galima standartine funkcija EOF(kintamasis-failas).

Tekstiniams failams naudojami tokie įvedimo/išvedimo sakiniai:

READ ir READLN
WRITE ir WRITELN

Vykdant WRITELN sakinį, baigus išvesti jame nurodytų kintamųjų reikšmes, išvedami ir valdantieji CR bei LF simboliai, t.y. pereinama prie naujos eilutės. Vykdant READLN sakinį, visų pirma įvedamos jame nurodytų kintamųjų reikšmės, o po to dar praleidžiami įvesti simboliai, kol

sutinkami CR ir LF. Po to naujas READ ar READLN sakinys duomenis ims nuo naujos eilutės pradžios.

Tekstinių failų svarbi savybė yra ta, kad įvedant/išvedant kintamųjų reikšmes automatiškai atliekama duomenų transformacija. Pvz., *integer* tipo kintamajam norint įvesti reikšmę 123 (atmintyje šio tipo kintamajam skirtuose dviejuose baituose ji bus saugoma kaip dvejetainis fiksuoto kablelio pavidalo skaičius), mes turėsime įvesti tris simbolius - '1', '2' ir '3'. Kiekvieno iš jų kodas užima po baitą. Taigi, trijuose baituose esančius simbolių kodus reikia transformuoti į dviejuose baituose užrašomą dvejetainį fiksuoto kablelio skaičių. Analogiška priešingos krypties transformacija atliekama išvedimo metu.

Priminsime, kad Paskalyje yra du standartiniai kintamieji-failai *input* ir *output*, kurių programose nereikia nei aprašyti, nei nurodyti įvedimo/išvedimo sakiniuose. Jie yra *text* tipo failai. Atitinkamai duomenys įvedami klaviatūra, o išvedami į monitorių.

Tekstinių failų naudojimo pavyzdžiai:

1 pavyzdys.

Sudarykime programą, kuri skaičiavimo rezultatus rašytų į diską. Failą iš diskų galėsime skaityti kai tik prireiks ir galėsime atsispausdinti operacinės sistemos priemonėmis.

```
PROGRAM  isvedamfaila;
VAR  a : text;
        b : string;
        x, y, z : real;
        m, n : integer;
BEGIN
        WRITELN ('iveskite failo varda, kur diske rasysit rezultatus');
                                {išvedimui naudoja standartinį output failą}
        READLN (b); {įvedamui naudoja standartinį input failą}
        ASSIGN (a, b);
        REWRITE (a);          …
        WRITE (a, x, m, n);    {išvedimui naudojamas aprašytas a failas}
        WRITELN (a, '***', y, '---', z);          …
END.
```

Komentaras šiai programai: pvz., jeigu READLN(b) sakiniu įvesime tokią simbolių seką *a:\kuku.dd* , tuomet mūsų skaičiavimo rezultatai bus užrašomi lankstaus diskelio *a:* faile *kuku.dd* .

Pavyzdys 2.

Tarkim, tekstų redaktoriumi (pvz., KEDIT, NOTEPAD, kt.) turime sukūrę duomenų failą diske. Norime, kad mūsų programa skaitytų šiuos duomenis ir apdorotų.

Sukurto failo *c:\studentai.dmn* eilutėse yra tokie duomenys:

```
pavardė-1
kursas,  grupės nr.,  pažymių vidurkis,  stipendijos dydis
pavardė-2
kursas,  grupės nr.,  pažymių vidurkis,  stipendijos dydis
…
pavardė-N
kursas,  grupės nr.,  pažymių vidurkis,  stipendijos dydis
```

Sudarykime programą, kuri skaitytų *c:\studentai.dmn* failą ir išvestų į monitorių 2 kurso 1 grupės studentų duomenis.

```
PROGRAM  skaitomfaila;
VAR  fa : text;
        xx, pavarde : string;
        kursas, grupe : integer;
        pazym,  stip : real;
BEGIN
        WRITELN ('iveskite failo varda, kur yra duomenys apie studentus');
```

```
        READLN (xx);
        ASSIGN (fa, xx);
        RESET (fa);
        ClrScr;
        WRITELN (' 2 kurso 1 grupes studentu duomenys');
        WRITELN ('-------------------------------------------');
        WHILE  not EOF(fa)  DO
          BEGIN
            READLN (fa, pavarde);
            READLN (fa, kursas, grupe, pazym, stip);
            …
            IF  (kurs = 2) and (grupe = 1)
              THEN  WRITELN (pavarde:20, pazym:5:2, stip:10:2);
          END;
END.
```

Dirbant su tekstiniais failais labai praverčia kai kurios standartinės procedūros/funkcijos.

Funkcija EOLN( ) tikrina eilutės pabaigą. Jei jos reikšmė *true*, reiškia yra eilutės pabaiga.

Funkcija EOF( ) - tikrina failo galą. Jei jos reikšmė *true*, reiškia pasiektas failo galas.

Procedūra APPEND ( ) atidaro joje nurodytą jau egzistuojantį failą, į kurį toliau bus tęsiamas duomenų išvedimas. Naudojama tik duomenų išvedimui.

Procedūra REWRITE ( ) visada sukuria naują failą ir duomenys išvedami nuo jo pradžios.

Procedūra FLUSH ( ) į išorinį įrengimą išveda informaciją, sukauptą tekstiniam failui skirtame buferyje. Buferio turinys automatiškai išvedamas tik jį visiškai užpildžius.

Procedūra SetTextBuf (kintamasis-failas, kintamasis [, baitų kiekis]) įgalina nustatyti norimo dydžio (baitais) įvedimo/išvedimo buferį nurodyto kintamojo vietoje. Programoje ji turi būti anksčiau nei RESET, REWRITE ar APPEND procedūra.

Procedūra RESET ( ), kaip jau žinome, atidaro skaitymui pasirinktą failą.

6.3. Tipizuoti failai

Tipizuoti failai gali būti tik diskuose. Kokiu pavidalu duomenys saugomi kompiuterio operatyviojoje atmintyje, tokiu pačiu pavidalu jie užrašomi į diską. Įvedimo/išvedimo metu jokia duomenų transformacija neatliekama. Taip pat duomenys nėra grupuojami į eilutes (palyginkime su tekstiniais failais) ir todėl tipizuotiems failams WRITELN ir READLN sakiniai neleistini.

Tipizuotas failas susideda iš vienodų elementų. Elemento tipas (*integer*, *real*, kt.) gali būti bet koks, kokio reikia jūsų sprendžiamam uždaviniui. Pvz.:

```
…   TYPE  kr = array[1..25] of real;
    VAR  a : file of  kr;          {a failo elementas yra realiųjų skaičių masyvas}
         x : kr;
         b : file of  byte;        {b failo elementas yra byte tipo skaičiai}
         y : byte;
    BEGIN …
      ASSIGN (a, …);
      ASSIGN (b, …);
      REWRITE (a);
      RESET (b);              …
      WRITE ( a, x);
      READ (b, y);            …
      CLOSE (a);  CLOSE (b);
    END.
```

Atkreipkime dėmesį į kintamuosius *x* ir *y*. Failo tipą turi atitikti tipas kintamųjų, kurių reikšmes norime įvesti/išvesti.

Vykdant mūsų parašytą programą, kompiutrio operacinė sistema pastoviai seka, kelintas tipizuoto failo elementas yra įvedamas/išvedamas. Atidarant failą RESET ar REWRITE sakiniu, nustatomas elemento numeris lygus 0. Įvykdžius eilinį programos įvedimo/išvedimo sakinį konkrečiam tipizuotam failui, failo elemento numeris padidinamas vienetu (operacinė sistema tam pasiskiria kintamąjį), t.y. pasiruošiama kitam elementui apdoroti. Kelintas tipizuoto failo elementas yra paruoštas apdoroti, galima sužinoti standartine funkcija

FILEPOS (kintamasis-failas).

Norint sužinoti, kiek apskritai elementų yra tekstiniame faile, naudojama standartinė funkcija

FILESIZE (kintamasis-failas).

Vartojant standartinę procedūrą

SEEK (kintamasis-failas, elemento-numeris);

galima nusistatyti norimą tipizuoto failo elemento numerį. Būtent tas elementas po to bus įvedamas/išvedamas. Tai suteikia diske esančio tipizuoto failo elementų tiesioginio išrinkimo galimybę (prisiminkime darbą su programos viduje esančio masyvo elementais).

Programos su tipizuotu failu pavyzdys.

Sudarykime akademinės grupės studentų anketų failą diske. Tarkime, kad anketoje turi būti pavardė, specialybės pavadinimas, paskutinės sesijos egzaminų pažymiai ir stipendijos dydis.

```
PROGRAM  StudentuAnketos;
USES  Crt;
TYPE  anketa = RECORD
                    pavard : string[15];
                    spec : string[10];
                    paz : array[1..5] of byte;
                    stip : real;
                END;
VAR   a : anketa;
      f : file of anketa;
      j : integer;
BEGIN
      ClrScr;
      ASSIGN ( f, 'c:\fizika\anketos.dm' );
      REWRITE ( f );
      WRITELN( 'darbui nutraukti, vietoje pavardes iveskite *' );
      REPEAT
        WRITE ( 'iveskite pavarde: ' );   READLN ( a.pavard );
        IF  a.pavard <> '*'  THEN
            BEGIN
               WRITE ( 'iveskite specialybe: ' );  READLN ( a.spec );
               WRITE ( 'iveskite pazymius: ' );
                    FOR  J := 1 TO 5 DO  READ ( a.paz[j] );  READLN;
               WRITE ( 'iveskite stipendija: ' );  READLN ( a.stip );
               WRITE ( f, a );       {užrašome į diską įvestą anketą}
            END;
        WRITELN;  {prieš naują anketą monitoriuje - tuščia eilutė}
      UNTIL  a.pavard = '*' ;
      CLOSE ( f );
END.
```

Kalbant apie tipizuotus failus atkreiptinas dėmesys į *file of char* tipo failą. Juo įvedami/išvedami simboliai. Tačiau mūsų anksčiau nagrinėti tekstiniai failai (*text*) taip pat įveda/išveda tik simbolius. Tarp *text* tipo ir *file of char* tipo failų yra tokie esminiai skirtumai:

1) programoje naudojant *text* failus, įvedimo ir išvedimo metu automatiškai atliekama duomenų transformacija.

Pvz., *integer* tipo kintamajam norint įvesti reikšmę 123 (atmintyje šio tipo kintamajam skirtuose dviejuose baituose ji bus saugoma kaip dvejetainis fiksuoto kablelio pavidalo skaičius), mes klaviatūra turėsime įvesti tris simbolius - '1', '2' ir '3'. Kiekvieno iš jų kodas užima po baitą. Taigi, trijuose baituose esančius simbolių kodus reikia transformuoti į dviejuose baituose užrašomą dvejetainį fiksuoto kablelio skaičių.

Naudojant *file of char* failą jokia duomenų transformacija neatliekama. Pastarieji failai naudojami tik *char* tipo kintamųjų reikšmėms įvesti/išvesti;

2) *text* failams neleistinas tiesioginio išrinkimo metodas (pvz., standartinė funkcija *seek(n)*). *File of char* failams tai yra leistina;

3) naudojant *text* failą, galima įvesti/išvesti tik tam skaliarinio tipo kintamųjų (*integer, real*, *string*, kt.) reikšmes, tačiau tai neleistina visam masyvui, įrašui (jų elementams - leistina). *File of char* failu galima įvesti/išvesti visą masyvą ar įrašą;

4) t*ext* failai turi eilutės požymį (pvz., standartinė funcija *eoln(f)*), *file of char* failų įrašai nedalinami į eilutes;

5) t*ext* failams leistini sakiniai READLN ir WRITELN, *file of char* - ne.

6.4. Netipizuoti failai

Netipizuotas failas traktuojamas kaip baitų visuma. Kokio tipo duomenys laikomi tuose baituose turi žinoti programuotojas. Šiuo atveju akcentuojama tai, kad norima kaip galima greičiau įvesti/išvesti norimą baitų kiekį. Netipizuoti failai aprašomi taip:

```
VAR  ntf : text;
```

Darbui su jais naudojami tokie įvvedimo/išvedimo sakiniai (procedūros):

```
BLOCKWRITE ( ntf, kintamasis, baitų-kiekis );
BLOCKREAD ( ntf, kintamasis, baitų-kiekis );
```

Vykdant BLOCKWRITE ( ), nuo nurodytam kintamajam paskirtos atminties pradžios imamas nurodytas baitų kiekis ir išvedama. Analogiškai, BLOCKREAD ( ) procedūra skaito nuo disko ar kt. įrengimo nurodytą baitų kiekį ir perduoda į operatyviosios atminties vietą, paskirtą nurodytam kintamajam.

Pavyzdys:

```
…
VAR  f  file;
        a : array[1..250] of real;
BEGIN
        ASSIGN ( f, 'c:\kuku.dd' );
        REWRITE ( f );  …
        BLOCKWRITE ( f, a, 900 );   …
        CLOSE ( f );
END.
```

Šiame pavyzdyje *a* masyvas atmintyje užima 1500 baitų. Išvedimo sakiniu perdavėme tik 900 baitų.

Turėkime galvoje, kad įvedant/išvedant daug atminties užimančius kintamuosius, duomenys perduodami standartinio dydžio porcijomis po 128 baitą, o ne iš karto visas nurodytas baitų kiekis. Tačiau perduodamos porcijos dydį gali nurodyti pats programuotojas sakiniuose REWRITE ( ) ir RESET ( ):

```
REWRITE ( netipizuotas-failas, porcijos-dydis-baitais );
RESET ( netipizuotas-failas, porcijos-dydis-baitais );
```

Pastebėkime, kad toks šių sakinių pavidalas leistinas tik netipizuotiems failams.

6.5. Dar šis tas apie įvedimą/išvedimą

Duomenims įvesti/išvesti Paskalyje dar yra specialios paskirties masyvai:

PORT [porto-numeris], PORTW [porto-numeris].

Pavyzdys.

```
…
USES  Crt;
VAR  a : array [1..100] of byte;              { byte tipo elementas atmintyje užima
                                                1 baitą}
       b : array [1..250] of integer;         { integer tipo elementas  atmintyje
                                                užima 2 baitus - žodį }
BEGIN  …
FOR  j := 1  TO  100  DO   PORT [ $1BE ] := a[j];
{šiuo ciklu į portą, kurio šešioliktainis adresas 1BE, bus išvesta 100 baitų}
FOR  j :=  1  TO  250  DO
       BEGIN
              DELAY ( 5 );
               b[j] := PORTW [ $10E ];
       END;
```

{čia iš porto, kurio šešioliktainis adresas $10E, bus įvesta 250 žodžių (dvibaičių); Crt bibliotekos procedūra DELAY ( ), pristabdanti programos darbą nurodytą milisekundžių kiekį, įdėta tam, kad į portą būtų kreipiamasi mažesniu dažnumu, nes jis gali nespėti dirbti greičiau}

Pastaba: Turbo Paskalio 7.0 aplinkoje šią informaciją galima rasti tokiu keliu: spausti klavišą F1, po to GLOSSARY ir DIRECT PORT ACCESS.

7. EKRANINĖ GRAFIKA

Paskalio kalboje yra grafikų piešimo monitoriuje priemonės. Standartinėje bibliotekoje GRAPH yra daug šiems tikslams skirtų standartinių procedūrų ir funkcijų. Tai programinės priemonės. Kad geriau suprastume šių programinių priemonių reikalingumą ir paskirtį, visų pirma susipažinkime su grafikų piešimo aparatinėmis priemonėmis.

7.1. Aparatinės grafikos priemonės

Monitorius dirba kaip ir įprastas televizorius. Ekrane vaizdas kartojamas 25 kadrų per sekundę dažniu. Vaizdas kadre susideda iš taškų, vadinamų pikseliais (angl. *pixel - picture element,* paveikslėlio elementas). Pikselių kiekis eilutėje ir eilučių kiekis ekrane priklauso nuo monitoriaus tipo (markės). Dažniausiai jis būna nuo 320x200 iki 1024x768. Specialios paskirties ekranuose (laikraščių ar žurnalų leidybos, architektūros, dizaino, kt. uždaviniams) pikselių kiekis gali būti dar didesnis. Kuo daugiau pikselių, tuo geresnė vaizdo kokybė, tačiau vaizdui suformuoti reikia daugiau laiko.

Vaizdo adapteris (videoadapteris) - tai sudėtinga elektroninė plokštė su savo mikroprocesoriumi ir atmintimi, įstatyta į kompiuterio pagrindinės plokštės jungtį. Kai kuriuose kompiuteriuose vaizdo adapterio elementai ir jungimo su monitoriumi lizdas būna sumontuoti pagrindinėje plokštėje.

Skiriamos dvi pagrindinės vaizdo adapterių dalys: kineskopo kontroleris (CRT - Cathode Ray Tube controller) ir vaizdo atmintis (videoatmintis).

Priklausomai nuo panaudoto kineskopo buvo sukurti įvairūs vaizdo adapteriai:

MDA - Monochrome Display Adapter;
CGA - Color Graphics Adapter, 320x200 (jo numeris - 1);
EGA - Enchanced Graphics Adapter, 640x350 (numeris - 3);
VGA - Virtual Graphics Array, 640x480 (numeris - 9); ir kt.

Vaizdo atmintis reikalinga įsiminti informacijai apie kiekvieną pikselį: spalvai, ryškumui, mirksėjimui. Vieno pikselio informacijai užrašyti specialios (aukštos) kokybės monitoriuose skiriama net iki 3 baitų. Vaizdo atmintis turi būti tokio dydžio, kad joje tilptų bent vieno vaizdo kadro pikselių informacija. Šiandieną dažniausiai ji būna iki 8 Mbaitų talpos, kur gali tilpti 2, 4 ar daugiau kadrų. To reikia, pvz., žiūrint multifilmus, kad mažiau mirgėtų vaizdas ir jis negadintų akių. Ekrane rodomas vienas kadras, o kai reikia - pereinama prie kito. Kol vieno kadro pikselių informaciją vaizdo plokštė automatiškai skaito iš vaizdo atminties (skanuoja) ir rodo ekrane, į kitam kadrui skirtą vaizdo atmintį kompiuteris siunčia naujo formuojamo kadro pikselių informaciją. Taip vyksta du lygiagretūs procesai.

7.2. Programinės grafikos priemonės

Vaizdui monitoriaus ekrane gauti reikia "uždegti" atitinkamą pikselių grupę. Taip išvedami ne tik grafikai, bet ir simboliai (skaitmenys, raidės, kt.). Simbolių vaizdai yra suformuoti 8x8 dydžio pikselių matricose. Šitokios matricos išvedinėjamos monitoriui dirbant taip vadinamuoju tekstiniu režimu.

Grafikams programuoti ir išvesti į ekraną *Borland Int.* firma Paskaliui sukūrė specialią programinę įrangą: procedūrų ir funkcijų biblioteką GRAPH, tvarkyklių ("draiverių") rinkinį darbui su įvairių rūšių vaizdo adapteriais, simbolių šriftų rinkinį tekstams išvesti, kuomet kompiuteris dirba grafiniu režimu.

Bibliotekoje GRAPH (tai UNIT modulis) yra apie 80 procedūrų ir funkcijų vaizdams formuoti ir pagalbiniams veiksmams atlikti. Turbo Pascal programinėje įrangoje ji laikoma faile GRAPH.TPU.

Monitorių tvarkyklės ("draiveriai") yra sukurtos visiems plačiausiai naudojamiems vaizdo adapteriams. Paskalio programa turi dirbti normaliai, nesvarbu koks monitorius yra jūsų kompiuteryje. Tvarkyklė - tai įrengimą valdanti programa (šiuo atveju programa, padedanti išvesti

duomenis į monitorių). Turbo Pascal programinėje įrangoje jie yra laikomi BGI (Borland Graphic Interface) katalogo failuose HERC.BGI, CGA.BGI, EGAVGA.BGI ir kt.

Netrukus matysime, kad norint Paskalio programomis piešti grafikus, būtina kompiuterį pervesti iš tekstinio režimo į grafinį režimą. Dirbant grafiniu režimu tekstai (simboliai) į monitorių gali būti išvedami norimu šriftu ("fontu"). Pastebėkime, kad ir šiuo atveju simboliai išvedami 8x8 dyžio pikselių matricomis, kaip ir dirbant tekstiniu režimu. Šriftai Paskalio programose gali būti nurodomi specialiais žodžiais arba numeriais, o simbolių pikselių informacijos matricos laikomos atitinkamuose failuose (.CHR):

šrifto pavadinimas	numeris	failas
DefaultFont	0	
TriplexFont	1	TRIP.CHR
SmallFont	2	LITT.CHR
SansSerifFont	3	SANS.CHR
GothicFont	4	GOTH.CHR

Jeigu programoje nenurodome, kokio šrifto simboliais išvesime tekstus, automatiškai naudojamas *DefaultFont*. Šriftui nurodyti yra procedūra

SetTextStyle (šriftas, išvedimo-kryptis, simbolių-dydis);

7.3. Išvedimas į monitorių tekstiniu ir grafiniu režimais

Dažniausiai Paskalio programos skaitinę ir tekstinę informaciją išveda į monitorių, kompiuteriui dirbant tekstiniu režimu. Savo ruožtu tekstinis režimas gali būti dvejopas:

tekstinio režimo nustatymo procedūra	išvedama į ekraną
TextMode (1);	25 eilutės po 40 simbolių
TextMode (2);	25 eilutės po 80 simbolių

Šios procedūros yra CRT bibliotekoje. Nesant nurodymų (pagal nutylėjimą) veikia TextMode (2);

Išvedami į monitorių simboliai gali būti paryškinti (tai nustatoma procedūra HighVideo;), normalaus ryškumo (NormVideo;) ir susilpninto ryškumo (LowVideo;). Šios ryškumo procedūros programoje rašomos anksčiau nei išvedimo sakiniai WRITE ar WRITELN. Vienas WRITE sakinys gali išvesti rezultatus į monitorių vienokiu ryškumu, kitas - kitokiu, jei prieš juos bus skirtingos ryškumo procedūros.

Norint Paskalio programa piešti grafikus monitoriuje, prieš tai būtina pervesti kompiuterį į grafinį darbo režimą (inicijuoti grafiką). Tam yra procedūra

InitGraph(monitoriaus-tvarkyklės-tipas,išvedimo-režimas, tvarkyklių-laikymo-vieta);

Ši procedūra ir daugelis vėliau šiame paragrafe minimų yra laikomos GRAPH bibliotekoje.

Prieš nustatant grafinį režimą, mes turime žinoti, koks monitorius ir vaizdo adapteris yra naudojamas kompiuteryje (CGA, EGA, VGA, kt.). Priklausomai nuo adapterio tipo, į mūsų sukompiliuotą programą bus įtraukta atitinkama tvarkyklė. Pvz., jei jūsų kompiuteryje yra VGA tipo tvarkyklė, tai galimi tokie grfinio režimo nustatymo sakiniai:

```
InitGraph ( VGA, … );   arba
InitGraph ( 9, … );
```

Pastebėkime, kad kintamasis VGA "atkeliauja" iš GRAPH bibliotekos (ten jis aprašytas kaip globalusis UNIT modulio *integer* tipo kintamasis), kuriam suteikta reikšmė 9.

Jeigu nežinome, koks monitoriaus adapteris yra kompiuteryje (dažniausiai taip ir būna; juk programą vieną kartą galime vykdyti vienu kompiuteriu, kitą kartą - kitu), pasinaudokime DETECT funkcija, kuri neturi parametrų. Ji automatiškai apklausia adapterį ir nustato jo rūšį. Taigi, programoje rašykime taip:

```
tvrk := DETECT;
InitGraph ( tvrk, … );
```

Čia kintamasis *tvrk* turi būti *integer* tipo.

InitGraph (…) procedūros antrasis parametras turi būti *integer* tipo kintamasis, kuriuo nurodomas grafiko išvedimo režimas: aukšta skiriamoji geba (parametro reikšmė 0) ar žema skiriamoji geba (1).

InitGraph (…) procedūros trečiuoju parametru (jis yra simbolių eilutė) nurodoma, kuriame diskų kataloge yra saugomos monitoriaus adapterių tvarkyklės. Taigi, grafinį režimą inicijuoti rekomenduojama taip:

```
USES  Graph;
VAR  tvrk, skgeb : integer;
…
tvrk := detect;  skgeb := 0;
InitGraph ( tvrk, skgeb, 'c:\tp\bgi' );   …
```

Baigus darbą su grafikais ir norint kompiuterį grąžinti į tekstinį darbo režimą, vartojama procedūra

CloseGraph;

Įvykdžius ją išlaisvinama operatyvioji atmintis, kuri grafiniame režime buvo paskirta tvarkyklėms, simbolių šriftams, kt. laikyti, sunaikinami nupiešti grafikai. Kad pereinant iš grafinio režimo į tekstinį, po to iš tekstinio vėl į grafinį, nebūtų sunaikinami nupiešti grafikai vartojamos GRAPH bibliotekos procedūros

RestoreCrtMode; ir **SetGraphMode (išvedimo-režimas);**

7.4. Vaizdo atminties puslapių junginėjimas

Monitoriaus vaizdo atmintis (ji yra videoadapteryje) daloma į puslapius. EGA, VGA, kt. videoadapteriai turi daugiau kaip vieną puslapį, kurie numeruojami pradedant nuo 0. Tam tikru laiko momentu į ekraną išvedamas tik vieno kurio nors puslapio turinys. Tai matomasis puslapis (pagal nutylėjimą 0 puslapis). Kol rodomas vaizdas, kurio pikselių informacija yra viename puslapyje, naują vaizdą galime formuoti kitame puslapyje. Taip galimi du lygiagretūs procesai. Darbui su vaizdo atminties puslapiais yra tokios procedūros:

SetActivePage (puslapio-numeris);
SetVisualPage (puslapio-numeris);

Jose puslapio-numeris turi būti *word* tipo sveikasis skaičius.

Pavyzdys:

```
. . .   SetActivePage (0);        {pagal nutylėjimą yra 0 puslapis}
        Circle ( … );             {rodo ir formuoja vaizdą 0 puslapyje}
. . .   SetActivePage (1);
        Rectangle ( … );          {rodys 0 puslapį, formuos - 1}
. . .   SetVisualPage (1);        {pradeda rodyti 1 puslapį}
```

7.5. Ekrano koordinačių sistema

Atskaitos taškas yra ekrano viršutinis kairysis taškas. Jo kordinatės yra (0,0). Apatinio dešiniojo taško koordinatės, pvz., EGA monitoriaus 640x480 pikselių dydžio ekrane dirbant EGAHi režimu (*High* skiriamoji geba), yra (639,479).

Einamoji rodyklė (CP - Current Pointer, tai darbinis Paskalio programinės įrangos kintamasis) nurodo koordinates, kur konkrečiu momentu yra nematomasis kursorius. Dirbant tekstiniu režimu kursorius yra matomas.

Prisiminkime, kad dirbant tekstiniu režimu kursorių galime perkelti GoToXY (x,y); procedūra (ji laikoma CRT modulių bibliotekoje). Nuo tos vietos toliau WRITE sakiniu bus išvedami duomenys. Pastebėkime, kad tekstiniame režime koordinatės nurodo simbolių, o ne pikselių pozicijas.

Grafiniame režime (po InitGraph(…); procedūros įvykdymo) nematomąjį kursorių galime kilnoti procedūromis

MoveTo (x, y); {*x* ir *y* reiškia absoliučias kursoriaus koordinates}
MoveRe (dx, dy); {dx ir dy nurodo poslinkį buvusios kursoriaus padėties atžvilgiu}

Nuo tos vietos ekrane, kur stovi nematomasis kursorius, bus piešiamas grafikas.

7.6. Ekranas ir langai jame

Dirbant tekstiniu režimu, lango ekrane, kur norime išvedinėti duomenis, vietą ir dydį galime nurodyti procedūra

Window (x1, y1, x2, y2);

Čia *x1*, *y1* yra lango viršutinio kairiojo taško koordinatės, *x2*, *y2* - apatinojo dešiniojo taško koordinatės visame ekrane. Tai *integer* tipo skaičiai.

Grafiniame režime langas, kuriame norime "piešti", nustatomas procedūra

SetViewPort (x1, y1, x2, y2, riba);

Šioje procedūroje parametras *riba* yra loginio (*boolean)* tipo dydis, kuriuo nurodoma, ar apriboti "piešimą", kuomet pasiekiama lango riba (šio parametro reikšmė *true*), ar leisti "piešti" ir už nustatyto lango ribų (*false*).

Susikūrus langą, jo viršutinio kairiojo taško koordinatės yra (0,0), ir būtent jis, o ne viso ekrano viršutinis kairysis taškas, tampa atskaitos tašku toliau išvedant rezultatus.

Susikurtas langas nuvalomas procedūra *ClearViewPort;*

Grįžti prie viso ekrano koordinačių galima dviem būdais: nuvalant visą ekraną *ClearDevice;* procedūra arba nustatant maksimalaus dydžio langą

SetViewPort (0, 0, MaxX, MaxY, true);

Norint sukurtą langą "nudažyti" norima spalva ar pamargintai, po procedūros *SetViewPort (...);* vartokite tokias dvi procedūras:

SetFillStyle (šablonas, spalva); {šablonas -- 0-11, spalva -- 0-15}

Bar (x1, y1, x2, y2);

Bar (...); procedūroje turės būti tokios pat koordinatės kaip ir *SetViewPort (...);* procedūroje. Pavyzdžiui:

```
…   SetViewPort (200, 70, 500, 200, true);
    SetFillStyle (10, 3);
    Bar (200, 70, 500, 200);
```

7.7. Taško ir linijos brėžimas

Taškui išvesti yra procedūra

PutPixel (x, y, spalva); {spalva -- 0-15}

Linijai nubrėžti yra keletas procedūrų:

Line (x1, y1, x2, y2); {brėš liniją tarp taškų (x1,y1) ir (x2,y2)}

Prieš brėžiant liniją, galima nustatyti jos spalvą:

SetColor (spalva);

Line (x1, y1, x2, y2);

Linijoms brėžti dar yra procedūros

LineTo (x, y); {nuo einamojo taško į nurodytą}

LineRel (dx, dy); {nuo einamojo tšk. iki tšk. pridėjus dekrementą}

Linijų pavidalas gali būti įvairus: plona, stora, punktyrinė, štrichinė, kt. Tai nustatoma procedūra

SetLineStyle (stilius, šablonas, storis);

kur *stilius* -- SolidLn arba 0, DottedLn - 1(iš taškų), CenterLn - 2 (iš taškų ir brūkšnelių), DashedLn - 3 (štrichuota), UserbitLn - 4 (gali nustatyti pats programuotojas); *šablonas* -- 0 (minėtų standartinių stilių linijos) arba 4 (programuotojo pasirinktas stilius); *storis* -- 1 (normalus storis) arba 3 (riebi).

Pavyzdžiui:

```
…   SetLineStyle (3, 0, 1);
    Line (50, 20, 100, 150);                {brėžia liniją}
    Rectangle (100, 50, 300, 200);          {piešia stačiakampį}
```

Čia ir linija, ir stačiakampis bus brėžiami štrichuota (3) normalaus storio (1) linija.

7.8. Įvairių figūrų piešimas

Prieš piešiant figūras, jų linijų spalvą, pavidalą, figūros uždažymą ar kt. galima nustatyti tokiomis procedūromis:

SetColor (spalva); {spalva -- 0-15}
SetBkColor (spalva); {fono spalvai nustatyti}
SetLineStyle (stilius, šablonas, storis); {žiūr. skyrelį anksčiau}
SetFillStyle (šablonas, spalva); {uždažys figūras nurodytu būdu}

Programoje nesant nurodymų, naudojamos pagal nutylėjimą nustatytos parametrų reikšmės.

Figūrų piešimo procedūrų aprašus galima rasti Paskalio kalbos vadovėliuose arba Turbo Pascal aplinkos pagalbose (*help*). Čia paminėsime tik dalį procedūrų.

Stačiakampių piešimas:

Rectangle (x1, y1, x2, y2);
Bar (x1, y1, x2, y2); {uždažo nupieštą stačiakampį}

Daugiakampių piešimas:

DrawPoly (kiek-kampų, masyvas-su-viršūnių-koordinatėmis);
FillPoly (kiek-kampų, masyvas-su-viršūnių-koordinatėmis); {uždažo nupieštą daugiakampį}

Apskritimų ir lankų piešimas:

Circle (x, y, spindulys); {x, y – apskritimo centro koordinatės}
Arc (x, y, prad-kampas, galut-kampas, spindulys); {lankams piešti}
PieSlice (x, y, prad-kampas, galut-kampas, spindulys);
{uždažytiems skritulių sektoriams piešti}
Ellipse (x, y, prad-kampas, galut-kampas, x-ašis, y-ašis);
{elipsėms arba jų lankams piešti;
kai *prad-kampas* = 0, *galut-kampas* = 360, piešia visą elipsę}
FillEllipse (x, y, x-ašis, y-ašis); {uždažo visą elipse}
Sector (x, y, prad-kampas, galut-kampas, x-ašis, y-ašis);
{uždažytiems elipsių sektoriams piešti}

Norint uždažyti bet kokios formos sritį, kuri yra apribota kažkokios spalvos kontūru (pvz., nupieštą CIRCLE(…), RECTANGLE(…), LINE(…), kt. procedūromis), naudojama procedūra

FloodFill (x, y, kontūro-spalva);

Joje (x,y) yra bet kurio taško kontūru apibrėžtoje srityje koordinatės.

7.9. Tekstų išvedimas grafiniu režimu

Nupieštuose grafikuose dažnai reikia tekstinių paaiškinimų. Dirbant grafiniu režimu, tekstams išvesti yra tokios procedūros:

OutText(simbolių-eilutė); {tekstą išveda nuo einamojo nematomo kursoriaus taško}
OutTextXY (x, y, simbolių-eilutė); {tekstą išveda nuo (x,y) vietos}

Aritmetinių kintamųjų reikšmėms pervesti į simbolių eilutės pavidalą yra procedūra

Str (aritm-kint:m:n, simb-eilutė);

Pavyzdys:

```
VAR   a : real;
      S : string;   . . .
a := 25.37;
Str (a:8:3, s);   {gausime eilutę '   25.380' }
OutTextXY (100, 25, 'Rezultatas:' + s);
```

Šriftui nustatyti yra procedūra

SetTextStyle (šriftas, kryptis, simbolių-dydis);
{*šriftas* -- 0-4, DefaultFont - 0, TriplexFont -1, SmallFont - 2,
SanserifFont - 3, GothicFont - 4;

kryptis -- 0 arba 1, HorizDir - 0, VertDir - 1;
simbolių-dydis -- 1-10, kiek kartų padidinti simbolius}

Išvedamų simbolių dydį dar galima nustatyti procedūra

SetUserCharSize (xdg, xdl, ydg, ydl);

Joje santykis xdg/xdl duoda simbolių plotį, ydg/ydl - simbolių aukštį.

8. DINAMINIAI KINTAMIEJI. RODYKLĖS

8.1. Dinaminio kintamojo samprata

Visi lig šiol nagrinėti kintamieji visą programos vykdymo laiką išlaiko jiems nustatytas charakteristikas. Jau kompiliavimo metu jiems suplanuojama vieta atmintyje, kurios dydis baitais ir reikšmės tuose baituose saugojimo pavidalas priklauso nuo duomenų tipo (longint, single, kt.).

Tačiau dažnai programos vykdymo eigoje ne visi kintamieji yra reikalingi iš karto. Taupant kompiuterio atmintį, nėra būtino reikalo dar kompiliavimo metu jiems visiems suplanuoti vietą atmintyje ir vėliau, vykdant programą, laikyti ją užimtą visą laiką. Paskalyje yra galimybė kintamiesiems paskirti vietą atmintyje ne programos kompiliavimo metu, o vėliau, t.y. programos vykdymo metu. Prireikus kintamojo, jam paskiriama vieta atmintyje, o kai kintamasis nebereikalingas, turėta atmintis išlaisvinama ir gali būti panaudota kitiems reikalams. Tokie kintamieji vadinami dinaminiais (anksčiau nagrinėti kintamieji buvo statiniai).

Dinaminiai kintamieji visada yra susiję su rodyklėmis (angl. *pointer*). Rodyklė – tai tokio tipo kintamasis, kuriame įsimenamas adresas atminties, paskirtos duomenų kintamajam. Kadangi duomenų kintamieji yra įvairių tipų (integer, real, string, array[] of …, record, kt.), tai ir rodyklės yra tokių pačių tipų (turime juk žinoti, kokio tipo duomenis talpinsime atmintyje nuo kažkokio adreso). Paskalio programose rodyklių apraše naudojamas simbolis ‘^’, po kurio rašomas duomenų tipą nusakantis žodis (integer, real, kt.). Pavyzdžiui:

```
TYPE  gatve = string[30];
      Anketa = record
                    pavarde : string[15];
                    pareigos : string[20];
                    alga : array[1..12] of real;
               end;
      masyvas = array[1..10000] of real;
VAR   a : ^integer;
      b : ^real;
      c : ^gatve;
      d : ^anketa;
      f : ^masyvas;
```

Čia kintamieji *a, b, c, d, f* yra rodyklės, galinčios įsiminti adresus. Kiekvienu šiuo adresu gali būti įsimenamos tik atitinkamo tipo reikšmės (kintamieji).

Negali būti rodyklių procedūroms/funkcijoms ir failams.

8.2. Dinaminių kintamųjų vartojimas

Dinaminis kintamasis programoje “gimsta” tuomet, kai jam paskiriama vieta atmintyje. Tam reikia atitinkamai rodyklei suteikti atminties adresą. Pavyzdžiui:

```
VAR   a : ^integer;
      b : integer;
begin - - -
      NEW(a);        {rodyklė  a  įgijo laisvos vietos atmintyje adresą}
      a^ := 123;     {toliau jau galima naudoti dinaminį kintamąjį a^ }
      b := 3 * a^ ;
      WRITELN ( a^ , b );   - - -
      DISPOSE(a);  {rodyklė a neteko reikšmės, toliau a^ naudoti nebegalima}
       - - -
end.
```

Matome, kad dinaminis kintamasis žymimas tokiu pačiu vardu kaip ir rodyklė, tačiau jo vardo gale dar pridedamas ‘^’ ženklas.

Kitas pavyzdys:

```
VAR   x, y : ^real;
- - -
NEW(x);        {paskiriama atmintis dinaminiam kintamajam x^ }
NEW(y);        {paskiriama atmintis dinaminiam kintamajam y^ }
x^ := 8.3;
y^ := -3;
x^ := x^ + y^;          { x^ reikšmė pasidarė lygi 5.3 }
x := y;          { rodyklė x įgijo rodyklės y reikšmę, t.y. abu dinaminiai
                   kintamieji x^ ir y^ atmintyje dabar laikomi toje pačioje vietoje}
y^ := 0;         {dabar ir y^ , ir x^ reikšmė tapo lygi 0}
WRITELN( x^ );       {bus išvestas 0}
DISPOSE(x);  DISPOSE(y);
```

Procedūra NEW(x) paskiria dinaminiam kintamajam *x^* vietą atmintyje, o jos adresą įsimena rodyklėje *x*. Tik po NEW(x) įvykdymo galima pradėti naudoti dinaminį kintamąjį *x^* .

Procedūra DISPOSE(x) išlaisvina atmintį, kurios adresą nurodo rodyklė x.

Panaudojus kelis kartus NEW( ) procedūrą tam pačiam kintamajam, kiekvieną kartą paskiriama vis nauja atmintis. Tokiu būdu galime turėti kelias to paties kintamojo versijas. Įvykdžius DISPOSE( ), išlaisvinsime tos versijos atmintį, kurią nurodys rodyklė.

Dinaminiams kintamiesiems atmintis dar gali būti paskiriama ir išlaisvinama GETMEM( ) ir FREEMEM( ) procedūromis. Pastarosios turi šiek tiek platesnes galimybes.

Priskiriant vienos rodyklės reikšmę kitai, jų tipai turi būti vienodi. Rodyklėms galima priskirti konstantos NIL reikšmę. Tai speciali Paskalio konstanta, priskyrus kurią rodyklė nebetenka reikšmės (nenurodo jokio atminties adreso). Pvz., priskyrus x := NIL; toliau nebegalima naudoti dinaminio kintamojo *x^* . Kol rodyklei nepavartosime NEW(x); procedūros, jos reikšmė bus neapibrėžta (nebūtinai lygi NIL).

Su rodyklėmis yra leistinos tik dvi palyginimo operacijos: = (lygu) ir <> (nelygu). Pvz.,

```
IF  x = NIL  THEN  NEW(x);
```

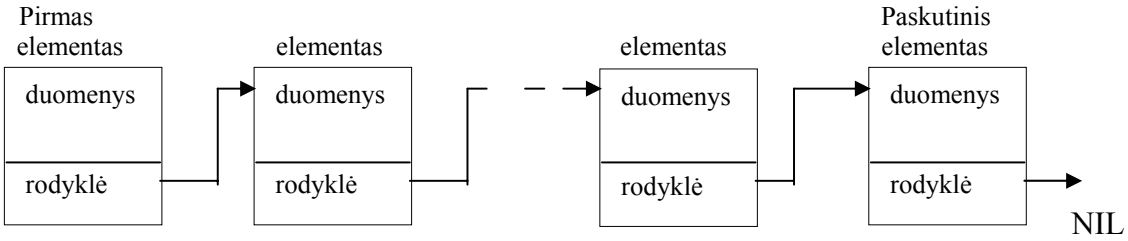
Dinaminiams kintamiesiems vieta skiriama vadinamojoje Heap atmintyje (angl. *heap* – krūva, daugybė). Tai atminties dydis, neviršijantis DOS'ui leistinų ribų. Informaciją apie Heap atmintį galime gauti standartinėmis funkcijomis *Memavail* ir *Maxavail* (jos laikomos standartinėje SYSTEM bibliotekoje). Šios funkcijos neturi argumentų. *Memavail* rezultatas yra kiek iš viso dar laisvų baitų yra Heap atmintyje, o *Maxavail* – koks didžiausias vientisas atminties dydis yra laisvas (darbo metu ši atmintis gali būti fragmentuota). Pavyzdžiui, pačioje programos pradžioje sakiniu WRITELN(Memavail); bus atspausdintas visos Heap atminties dydis baitais (jis priklauso nuo aplinkos, kuri yra kompiuteryje vykdant programą, t.y. apie 200-500 tūkst. baitų). Atkreipkime dėmesį į tai, kad įprastu atveju Paskalio programos kintamiesiems iš viso yra skirama tik 64K= 65535 baitų. Jei šitiek atminties per maža, naudokime dinaminius kintamuosius. Vienas dinaminis kintamasis (pvz., masyvas ar įrašas) atmintyje gali užimti ne daugiau kaip 65521 baitą.

8.3. Sąrašai

Sąrašas – tai atitinkamai sudėlioti duomenys. Paprasčiausias pavyzdys yra masyvas, kuriame nuosekliai vienas po kito (vientisoje atminties dalyje) yra talpinami jo elementai. Masyvo sąvybė yra ta, kad aprašant jį būtina nurodyti elementų kiekį (indeksų kitimo ribos nurodomos konstantomis). Tuo tarpu sąrašai neturi fiksuoto ilgio, jų elementai gali būti išmėtyti atmintyje (laikomi ne vientisoje srityje). Todėl jiems organizuoti reikia lankstesnio būdo. Lankstumą duoda dinaminiai kintamieji, įgalinantys kurti įvairias dinamines duomenų struktūras: tiesinius sąrašus, žiedinius sąrašus, eiles, stekus, medžius, piramides.

Tiesinis sąrašas. Paprasčiausio – vienos krypties tiesinio sąrašo - elemente turi būti du laukai: duomenys (elemento informacija) ir rodyklė (kur atmintyje laikomas kitas iš eilės sąrašo elementas). Sąrašo ilgis gali būti keičiamas bet kurioje sąrašo vietoje įterpiant naują elementą arba pašalinant bet kurį esantį.

Nupieškime vienos krypties tiesinį sąrašą grafiškai.



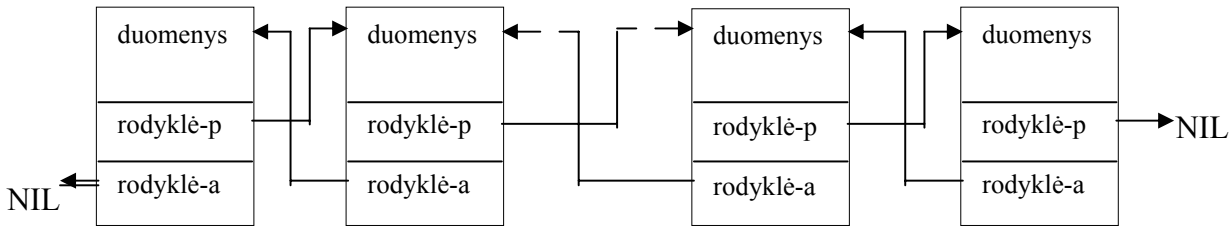
8.1 pav. Vienos krypties tiesinio sąrašo schema.

Žinant kur yra pirmasis sąrašo elementas (užfiksavus atitinkamos rodyklės reikšmę), galima susirasti bet kurį sąrašo elementą. Paskutiniame elemente esančiai rodyklei priskiriama NIL reikšmė, kaip požymis, kad sąrašas baigėsi.

Pavyzdžiui, sudarykime Paskalio programą gyventojų telefono sąrašui sudaryti kompiuterio atmintyje (žiūr. 8.2 pav.).

```
- - -   TYPE  Trod = ^Telement;
              Telement = record
                            pavarde : string [20];      { pavarde , adresas  ir numeris
                            adresas : string [30];        yra elemento duomenys}
                            numeris : string [8];
                              rodykle : Trod;           {rodykle – nuorodai į kitą
                          end;                                          elementą}
        VAR   pirmas, x, y : Trod;
              n : integer;
        begin - - -              {įvedame duomenis ir sudarome sąrašą iš 25 elementų}
                    n := 0;
                    pirmas := NIL;
                    REPEAT
                        NEW(x);
                        READLN( x^.pavarde);
                        READLN(x^.adresas);
                        READLN(x^.numeris);
                        x^.rodykle := NIL;
                        IF pirmas = NIL     THEN  pirmas := x
                                            ELSE   y^.rodykle := x;
                       y := x;
                        n := n + 1;
                    UNTIL  n = 25 ;
                                  { toliau išvedame į monitorių sąrašo elementus }
                    x := pirmas;
                    REPEAT
                        WRITELN( x^.pavarde:20,  x^.adresas:30,  x^.numeris:10 );
                        x := x^.rodykle;
                    UNTIL  x = NIL;    - - -
        end.
```

Tiesinio sąrašo elemente pavartojus dvi rodykles, kurių viena būtų skirta nurodyti kitam iš eilės einančiam sąrašo elementui, o antra - prieš einančiam sąrašo elementui, turėtume dviejų krypčių tiesinį sąrašą. Pavaizduokime jį grafiškai.



8.3 pav. Dviejų krypčių tiesinio sąrašo schema

Trumpai apibūdinkime kitokias dinamines duomenų struktūras:

žiedinis sąrašas yra panašus į tiesinį sąrašą, tačiau jame yra papildomas ryšys (nuoroda) tarp paskutinio ir pirmo elementų;

eilė – vienos krypties tiesinio sąrašo variantas, kuriam leistini tik du veiksmai: naujo elemento pridėjimas tik sąrašo gale ir elemento pašalinimas tik nuo sąrašo pradžios;

stekas – vienos krypties tiesinio sąrašo variantas, kuriame naujas elementas gali būti pridėtas tik gale, o pašalintas taip pat tik nuo galo. Šio sąrašo galinis elementas vadinamas steko viršūne;

medis - tai pasirinktos konfigūracijos hierarchinė duomenų struktūra. Medžio elementai vadinami viršūnėmis (mazgais);

piramidė (surūšiuotas medis) – tai medis, kurio viršūnių reikšmės visada didėja arba mažėja pereinant į kitą lygį.

Grįžkime prie vienos krypties tiesinio sąrašo ir programos pavyzdžiu pailiustruokime elemento pašalinimą iš sudaryto sąrašo. Tam pasinaudokime 8.2 pav. sudarytu gyventojų telefonų sąrašu. Iš to sąrašo pašalinkime elementą su nurodyta gyventojo pavarde. Pavardę įveskime klaviatūra.

```
- - -
LABEL galas;
TYPE  Trod = ^Telement;
        Telement = record
                        pavarde : string [20];      { pavarde , adresas  ir numeris  }
                        adresas : string [30];      {  yra elemento duomenys          }
                        numeris : string [8];
                          rodykle : Trod;           {rodykle – nuorodai į kitą elementą }
                      end;
VAR   pirmas, x, y : Trod;
        n : integer;
        salinti : string [20];
begin  - - -              {įvedame duomenis ir sudarome sąrašą iš 25 elementų}
        n := 0;
        pirmas := NIL;
        REPEAT
           NEW(x);
           READLN( x^.pavarde);  READLN(x^.adresas);  READLN(x^.numeris);
           x^.rodykle := NIL;
           IF  pirmas = NIL   THEN  pirmas := x
                                ELSE   y^.rodykle := x;
           y := x;
           n := n + 1;
        UNTIL  n = 25 ;
        - - -                    { toliau iš sąrašo šaliname nurodytus elementus  }
        x := pirmas;     y := NIL;
        IF  x = NIL  THEN  begin
```

```
                                WRITELN ( ‘Sąraše nėra nė vieno elemento’);
                                GOTO galas;
                            end;
        READLN ( salinti );          { įvedame gyventojo pavardę, kurio duomenis }
                                     { norime panaikinti is sąrašo                }
        REPEAT
           IF x^.pavarde <> salinti THEN  begin
                                             y := x;
                                             x := x^.rodykle;
                                         end;
        UNTIL   (x = NIL)  or  (x^.pavarde = salinti);
        IF  x^.pavarde = salinti
              THEN    IF  x^.rodykle = NIL
                          THEN y^.rodykle := NIL
                          ELSE  IF x = pirmas THEN pirmas := NIL
                                              ELSE y^.rodykle := x^.rodykle
              ELSE     WRITELN ( ‘Tokios pavardes sarase nera’ );
        - - -
        x := pirmas;             { išvedame į monitorių sąrašą be jau pašalintų elementų}
        REPEAT
           WRITELN( x^.pavarde:20,  x^.adresas:30,  x^.numeris:10 );
           x := x^.rodykle;
        UNTIL  x = NIL;    - - -
galas : end.
```

Analogiškai atliekamas ir naujo elemento įterpimas į norimą sąrašo vietą. Programą siūlome pabandyti sudaryti patiems skaitytojams.

9. OBJEKTINIS PROGRAMAVIMAS

9.1. Programavimo metodologijos. Objektinio programavimo sąvokos

Objektinis programavimas (OOP - Objektiškai Orientuotas Programavimas) - tai viena iš programavimo metodologijų. Kita programavimo metodologija yra struktūrinis programavimas. Poreikis laikytis vienokios ar kitokios programavimo metodologijos atsirado rašant sudėtingas programas ar programų sistemas. Tai neišvengiama, norint kad rašomos programos būtų:

- lengvai, patogiai skaitomos. Tam reikia vengti neaiškių, nevaizdžių Paskalio (ar kitos programavimo kalbos) konstrukcijų. Programas reikia rašyti taip, kad jose būtų kuo mažiau "šokinėjimų" iš vienos vietos į kitą (vengti GOTO sakinių), kad programos tekstas būtų skaitomas nuosekliai, kt.;
- efektyvios. Programų efektyvumas pasiekiamas skaidant jas į modulius taip, kad lengviau būtų ieškoti klaidų ir taisyti jas, kad pertvarkant vieną modulį nereikėtų keisti kitų;
- patikimos. Patikimomis programomis laikomos tokios, kurias galima nuosekliai tikrinti (testuoti) ir derinti. Tai pasiekiama jau aukščiau minėtais būdais - skaidant sudėtingas programas į modulius, rašant lengvai skaitomus programų tekstus (jos lengviau analizuojamos);
- greitai ir pigiai kuriamos. Tam reikia protingai organizuoti darbą, tinkamai jį padalyti tarp programuotojų kolektyvo narių. Čia vėlgi neapsieinama be sudėtingų programų skaidymo į modulius, kas plačiai taikoma struktūriniame programavime.

Vienas iš struktūrinio programavimo principų yra "skaldyk ir valdyk" (dekompozicija). Tam sudėtingas uždavinys (problema) skaidomas į smulkesnius nepriklausomus uždavinius, kuriuos būtų lengva suprasti ir spręsti. Tai ir yra modulinio programavimo principas (procedūros, funkcijos, UNIT bibliotekiniai moduliai, DLL, kt.). Programų skaidymas į modulius ir hierarchinis modulių grupavimas turi įtakos į sudėtingų programų kokybę. Programos sudėtingumo rodikliai: 1) į kelis modulius kreipiasi vienas kažkuris modulis ir 2) keli moduliai kreipiasi į vieną kažkurį modulį. Laikoma, kad optimali šių rodiklių reikšmė yra 7 (jokiu būdu ne daugiau 10).

Struktūrinio programavimo pagrindiniai reikalavimai yra šie:

1. Sudėtinga programa turi būti skaidoma į nepriklausomas dalis - modulius.
2. Modulis - tai nepriklausomas programos blokas, kurio programa fiziškai ir logiškai atskirta nuo kitų blokų.
3. Modulis turi būti skirtas tik vienai loginei uždavinio funkcijai.
4. Modulyje neturėtų būti daugiau kaip 100 programavimo kalbos sakinių.
5. Modulyje turi būti tik po vieną įėjimo ir išėjimo tašką.
6. Ryšiai tarp modulių nustatomi pagal hierarchinę struktūrą.
7. Kiekvieno modulio programos tekstas turi prasidėti komentarais, paaiškinančiais modulio paskirtį, parametrų, kurių reikšmės perduodamos į modulį ir iš jo, paskirtį, kokie kiti moduliai kreipiasi į jį ir į kokius kitus modulius jis kreipiasi.
8. Vengti bereikalingų žymių ir GOTO sakinių. Jei labai jų reikia, GOTO naudoti tik "peršokimams" į modulio pradžią ar pabaigą.
9. Modulių ir kintamųjų vardai turi būti prasmingi.
10. Giminingos grupės vardai (pvz., Antano ūgis, svoris, bato numeris, kt.) turi prasidėti vienodu priešdėliu (prefiksu, pvz., *antug, antsv, antbn*).
11. Programose naudoti tik standartines valdymo konstrukcijas (pvz., ciklams organizuoti - FOR, REPEAT, WHILE; išsišakojimams - IF, CASE).

12. Vienoje eilutėje reikėtų rašyti tik po vieną sakinį. Jei vienam sakiniui užrašyti reikia daug eilučių, antra ir kitos eilutės pastumiamos pirmosios eilutės pradžios atžvilgiu.
13. Vartoti IF sakinių "gylį" ne didesnį kaip 3.
14. Vengti programavimo "triukų" (mandrybių, pasipuikavimų).

Objektinis programavimas, kaip ir ką tik aptartas struktūrinis programavimas, palengvina sudėtingų programų kūrimą, supaprastina darbus, sumažina jų kainą. Objektinis programavimas sudėtingų programų atvejais yra patogesnis (pranašesnis) už struktūrinio programavimo metodologiją.

Kaip jau minėjome, vienas iš programavimo darbų organizavimo principų yra dekompozicija ("skaldyk ir valdyk"). Skiriami du dekompozicijos būdai:

- algoritminis. Jis pagrindinai naudojamas struktūriniame programavime, kur programa skaidoma į nepriklausomus modulius ir nustatomi ryšiai (sąveika) tarp modulių. Modulius gali programuoti atskiri darbuotojai;

- objektinis. Skaidant uždavinį į dalis, visų pirma stengiamasi įvertinti ryšių tarp uždavinio (tuo pačiu ir programos) objektų pobūdį, kas nuo ko priklauso, kas kam pavaldus.

Praktikoje dažniausiai naudojami abu dekompozicijos būdai kartu. Objektinio programavimo metodai naudojami pradžioje, sudarant sprendžiamo uždavinio objektų hierarchiją (kas nuo ko priklauso, kaip susiję). Tuo atspindima uždavinio esmė. Po to pereinama prie algoritminės dekompozicijos, skaidant uždavinį į modulius.

Mokantis objektinio programavimo, būtina gerai suprasti “objekto” sąvoką. Objektui apibūdinti naudojami ne tik jam charakteringi duomenys (pvz., stačiakampis apibūdinams ilgiu, pločiu, uždažymo spalva, kt.), bet ir veiksmai, leistini su tuo objektu (pvz., pasukti, pastumti, padidinti, sumažinti, kt.). Objekto duomenų apjungimas su tam objektui leistinais veiksmais (procedūromis/funkcijomis) yra vadinamas **inkapsuliacija**. Objektai, kaip ir kitokie kintamieji, Paskalio programose aprašomi VAR skyriuje. Kaip visa tai daroma, matysime truputį vėliau.

Spendžiant sudėtingus uždavinius, be jų dekompozicijos (skaidymo į dalis) svarbus yra gautų dalių hierarchinis išdėstymas. Tiek struktūrinio, tiek objektinio programavimo metodai leidžia formuoti ryšių tarp uždavinio dalių hierarchinį medį. Struktūrinio programavimo metode hierarchijos medis formuojamas skaidant sudėtingą uždavinį į nepriklausomas dalis, tuo tarpu objektinio programavimo metodas įgalina į tai žiūrėti kitaip. Šiuo atveju hierarchija formuojama taip, kad žemesnio hierarchijos lygio elementai (vaikai, palikuonys) paveldi vyresnio lygio elementų (tėvų) sąvybes. **Paveldėjimas** – tai tokie santykiai (ryšiai) tarp objektų, kai vienas objektas (palikuonis) perima kito objekto (tėvo) struktūrą ir veiksmus.

Dar viena svarbi objektinio programavimo metodo sąvoka – **polimorfizmas**. Tai objektų gebėjimas tokį pat veiksmą atlikti įvairiai (pvz., bėga dramblys, bėga pelė, bėga vėžlys). Paprasto (ne objektinio) programavimo atveju kiekvieno objekto (pvz., dramblio, pelės, vėžlio) konkrečiam veiksmui (pvz., bėgti, rėkti, gintis) atlikti reikėtų rašyti atskirą modulį (funkciją ar procedūrą). Objektinio programavimo atveju programuotojui tereikia nurodyti, kuriam objektui ir kokį veiksmą reikia atlikti. Kaip konkretus objektas tą veiksmą darys, priklausys nuo to objekto apraše esančių metodų (funkcijų, procedūrų).

9.2. Objekto-tipo ir objekto-kintamojo aprašai

Objektai yra programavimo kalbos elementai (kaip ir *integer, real*, *array*, *record, kt.* duomenų tipai ar šių tipų kintamieji). Paskalio programose jie aprašomi TYPE ir VAR skyriuose. TYPE skyriuje aprašomi objektai-tipai, o VAR skyriuje – objektai-kintamieji. Objektai savo aprašo forma yra labai panašūs į įrašus (*record*). Įrašų apraše naudojamas žodelis *record*, o objektams naudojamas žodelis *object*. Pavyzdžiui,

įrašo tipo aprašas	objekto tipo aprašas

```
TYPE  Tir =  record                    TYPE   Tob =  object
                a : real;                              x : integer;
                b : integer;                           y : integer;
               - - -                                   procedure p1 ( … );
              end;                                     function f1 ( … ): real;
                                                      - - -
                                                     end;
```

Kaip matome iš šio pavyzdžio, objekto tipo apraše be jį kiekybiškai charakterizuojančių elementų (čia *x,y*) dar gali būti procedūrų/funkcijų antraštės. Šiomis procedūromis/funkcijomis nurodomi veiksmai, kurie gali būti atliekami su tokios sudėties objektu. Objektuose apibrėžtos procedūros/funkcijos yra vadinamos **metodais**.

Toliau, aiškindami objektus, naudosime šachmatų žaidimo programavimo pavyzdį. Šachmatų figūros poziciją lentoje nurodyti galima tokiu objektu-tipu (1 pav.):

```
TYPE
Tst = 'a' .. 'h';      Teil = 1  .. 8;
Tpoz = object
          stulp : Tst;
          eil : Teil;
       end;
VAR
x, y : Tpoz;
```

1 pav. Objekto aprašo pavyzdys (šachmatų figūros pozicija lentoje).

Čia su objektais-kintamaisiais (egzemplioriais) *x, y* yra leistini tokie patys veiksmai, kaip ir įrašų (*record*) atveju. Pavyzdžiui,

```
x.stulp := 'd';   x.eil := 4;
```

Pastebėkime, kad *stulp* ir *eil* yra mūsų apsirašytų objektų duomenų laukai (duomenys).

Tiesiogiai kreiptis į objektų-kintamųjų duomenų laukus nerekomenduojama. Tam tikslui objektų aprašuose nurodomi veiksmai, leistini su objektu. Šie veiksmai objekto apraše atspindimi kaip procedūrų/funkcijų antraštės. Prisiminkime, kad objektų veiksmai vadinami **metodais**.

Išplėskime 1 pav. pateiktą objekto aprašą nurodydami galimus šachmatų figūroms metodus (veiksmus): 1) figūros pradinės pozicijos lentoje nustatymą; 2) figūros pozicijos lentoje stulpelio gavimą; 3) figūros pozicijos lentoje eilutės gavimą.

```
TYPE
      Tst = 'a' .. 'h';     Teil = 1  .. 8;
      Tpoz = object
                stulp : Tst;
                eil : Teil;
                procedure  pradpoz (s:Tst; e:Teil;);
                function  gautst : Tst;
                function  gauteil : Teil;
             end;
VAR  x, y : Tpoz;
```

2 pav. Objekto su metodais pavyzdys.

Pastebėkime, kad čia panaudotos funkcijos *gautst* ir *gauteil* neturi parametrų (jie šiuo atveju nebūtini). Kreipusis į jas, atitinkamai gaunamas *Tst* arba *Teil* tipo rezultatas. Tuo tarpu procedūra *pradpoz* turi du parametrus, kurie reikalingi nustatant šachmatų figūros poziciją lentoje.

Atkreipkime dėmesį (žiūr. 2 pav.), kad objekto-tipo apraše būtina pirmiau išvardinti duomenis *stulp, eil* , o tik po jų nurodyti metodus (procedūras/funkcijas) *pradpoz, gautst, gauteil*.

Objekte-tipe nurodytų metodų (procedūrų/funkcijų) tekstai rašomi įprastoje Paskalio programų vietoje - po VAR skyriaus prieš pagrindinį programos sakinių skyrių. Išplėskime 2 pav. pateiktą pavyzdį, papildydami jį reikiamų metodų (procedūrų/funkcijų) tekstais (žiūr. 3 pav.).

```
TYPE
Tst = 'a' .. 'h';          Teil = 1  .. 8;
Tpoz = object
          stulp : Tst;
          eil : Teil;
          procedure  pradpoz (s:Tst; e:Teil;);
          function  gautst : Tst;
          function  gauteil : Teil;
        end;
VAR   x, y : Tpoz;

Procedure   Tpoz.pradpoz (s:Tst; e:Teil);
   begin   stulp := s;   eil := e;   end;

Function  Tpoz.gautst : Tst;
   begin   gautst := stulp;   end;

Function  Tpoz.gauteil : Teil;
   begin   gauteil := eil;   end;

begin
        - - -   {pagrindinės programos dalies sakiniai}
end.
```

3 pav. Objekto aprašo pavyzdys su metodų (procedūrų/funkcijų) tekstais.

Atkreipkime dėmesį, kad prieš metodų (procedūrų/funkcijų) vardus nurodomas dar ir objekto tipas (mūsų atveju *Tpoz*), t.y. šie metodai gali būti naudojami tik nurodyto tipo objektams-kintamiesiems *x, y.*

Pagrindinėje programos sakinių dalyje (žiūr. 3 pav.) galime elgtis dvejopai: 1) su objektais dirbti taip, kaip derėtų; 2) su objektais elgtis kaip su įrašais (*record*), kas nerekomenduojama. Detalizuokime šiuos abu leistinus variantus:

1) darbas su objektais, kaip derėtų.

Geras programavimo stilius reikalauja kreiptis į objekto duomenų laukus (mūsų nagrinėjamame pavyzdyje *stulp, eil*) naudojant metodus, nurodytus objekto apraše (*pradpoz, gautst, gauteil*).

```
TYPE
        Tst = ‘a’ .. ‘h’;     Teil = 1  .. 8;
        Tpoz = object
                    stulp : Tst;
                    eil : Teil;
                    procedure  pradpoz (s:Tst; e:Teil;);
                    function  gautst : Tst;
                    function  gauteil : Teil;
                end;
VAR   x, y : Tpoz;
        xs, ys : Tst;
        xe, ye : Teil;

Procedure   Tpoz.pradpoz (s:Tst; e:Teil);
        begin   stulp := s;   eil := e;   end;

Function  Tpoz.gautst : Tst;
        begin   gautst := stulp;   end;

Function  Tpoz.gauteil : Teil;
        begin   gauteil := eil;   end;

begin  - - -
        x.pradpoz (‘c’, 1);       {nustatom figūros x poziciją}
        - - -
xs := x.gautst;            { gauname figūros x pozicijos }
        xe := x.gauteil;                    { stulpelio ir eilutės reikšmes }
        - - -
        y.pradpoz (‘b’, 1);       {nustatom figūros y poziciją}
        - - -
end.
```

4 pav. Darbas su objektais.

Pastebėkime, kad 4 pav. pavyzdys papildytas VAR skyriuje aprašytais kintamaisiais *xs, ys* (figūrų pozicijų stulpeliams įsiminti) bei *xe, ye* (figūrų pozicijų eilutėms įsiminti).

4 pav. pagrindinę programos sakinių dalį galima supaprastinti, panaudojus WITH sakinį:

```
begin  - - -
        WITH  x  DO
           begin
                pradpoz (‘c’, 1);                {nustatom figūros x poziciją}
                - - -
                xs := gautst;                    {gauname figūros x pozicijos
                xe := gauteil;                   stulpelio ir eilutės reikšmes}
           end;
        - - -
end.
```

2) darbas su objektais kaip su įrašais (*record*), kas nerekomenduojama.

Pagrindinė programos sakinių dalis 4 pav. galėtų būti ir tokia:

```
begin  - - -
       x.stulp := 'c';          {nustatom figūros x poziciją}
       x.eil :=  '1';
       - - -
       xs := x.stulp;           { gauname figūros x pozicijos  }
       xe := x.eil;             { stulpelio ir eilutės reikšmes  }
       - - -
end.
```

Panaudojus WITH sakinį, tai galima būtų užrašyti taip:

```
begin  - - -
       WITH  x  DO
          Begin
               stulp := 'c';     {nustatom figūros x poziciją}
               eil :=  '1';  - - -
               xs := stulp;      {gauname figūros x pozicijos stulpelio ir }
               xe := eil;        {eilutės reikšmes                         }
            end;   - - -
end.
```

Pastebėkime svarbią objektų sąvybę. Jų metodai veikia taip, lyg po kreipimosi jų programos tekste būtų WITH sakinys. Pavyzdžiui (žiūr. 4 pav.), po kreipinio

```
x.pradpoz ('c', 1);
```

procedūra *Tpoz.pradpoz* traktuojama taip, lyg jos programos tekstas būtų toks:

```
Procedure  Tpoz.pradpoz (s:Tst; e:Teil);
    begin
          WITH  x  DO
             begin   stulp := s;   eil := e;   end;
    end;
```

Objekto metodų antraštėse visada yra "nematomas" (nutylimas) formalus parametras vardu *self*. Kreipusis į metodą, *self* parametrui perduodamas kreipinyje nurodytas objekto-kintamojo vardas (mūsų atveju *x*). Tą nutylimą metodų dalį galima įsivaizduoti taip (tolesniame pavyzdyje ji paryškinta):

```
Procedure  Tpoz.pradpoz ( self : pointer; s : Tst; e : Teil );
  begin
       WITH  self  DO
          Begin   stulp := s;   eil := e;   end;
  end;
```

9.3. Objektai ir UNIT moduliai

Pastebėkime, kad UNIT modulių *interface* dalis ir objektų aprašai turi tą panašumą, kad juose nurodomos tik metodų (procedūrų/funkcijų) antraštės, o patys procedūrų/funkcijų programų tekstai rašomi atskirai kitoje vietoje (UNIT atveju *implementation* dalyje). Kad tais pačiais objektais galėtų naudotis skirtingos programos, reikia naudoti UNIT modulius, aprašant objektus UNIT modulio *interface* dalyje, o procedūras/funkcijas (metodus) *implementation* dalyje. Pailiustruokime tai remdamiesi mūsų nagrinėto šachmatų žaidimo programavimo pavyzdžiu.

```
UNIT  sachmat;
INTERFACE
   TYPE
        Tst = ‘a’ .. ‘h’;      Teil = 1 .. 8;
        Tpoz = object
                        stulp : Tst;
                        eil : Teil;
                        procedure  pradpoz (s:Tst; e:Teil;);
                        function  gautst : Tst;
                        function  gauteil : Teil;
                  end;
IMPLEMENTATION
        Procedure  Tpoz.pradpoz (s:Tst; e:Teil);
                  begin    stulp := s;   eil := e;    end;
        Function Tpoz.gautst : Tst;
                  begin    gautst := stulp;    end;
        Function Tpoz.gauteil : Teil;
                  begin   gauteil := eil;      end;
begin
end.
```

9.4. *Private* ir *public* nuorodos objektuose

Private ir *public* nuorodomis apibrėžiama objekto elementų galiojimo sritis.

Objekto apraše sutiktas *private* atlieka panašų vaidmenį kaip ir *implementation* UNIT moduliuose. *Private* apriboja kai kurių objekto elementų veikimo sritį. Taip objektuose galima susikurti uždarus duomenų ir metodų laukus. Mūsų turėtame UNIT sachmat; pavyzdyje (žiūr. 5 pav.) išplėskime objektą, panaudodami *private* ir *public* nuorodas.

```
UNIT  sachmat;
INTERFACE
        TYPE    Tst = ‘a’ .. ‘h’;     Teil = 1  .. 8;
                Tpoz = object
                        function  gautst : Tst;
                        function  gauteil : Teil;
                        private
                        stulp : Tst;
                        eil : Teil;
                        procedure  pradpoz (s:Tst; e:Teil;);
                  end;
IMPLEMENTATION
    Procedure  Tpoz.pradpoz (s:Tst; e:Teil);
                  begin    stulp := s;   eil := e;    end;
    Function  Tpoz.gautst : Tst;
                  begin    gautst := stulp;    end;
    Function  Tpoz.gauteil : Teil;
                  begin   gauteil := eil;      end;
begin   - - -     end.
```

6 pav. *Private* nuoroda objekte UNIT modulyje.

Objekto elementai po *private* nuorodos galioja tik UNIT modulio *implementation* dalyje.

Tarkime, kad turime programą, kuri naudoja 6 pav. parodytą UNIT sachmat; modulį:

```
PROGRAM  kuku;
USES  sachmat;
VAR   x, y : Tpoz;
        stulp : Tst;   - - -
begin   - - -
   x.stulp := 'b';              { šie du sakiniai neleistini dėl objekto apraše   }
   x.pradpoz ( 'g', 6 );        { UNIT sachmat; modulyje panaudoto private }
   - - -
   stulp := x.gautst;           { šis sakinys leistinas }
end.
```

Atkreipkime dėmesį į tai, kad objekto apraše tuoj po žodelio *object* einančių elementų galiojimo sritis nėra ribojama. Elementai, kurių galiojimo sritį norime apriboti *private* nuoroda, aprašomi objekto gale. Tačiau, jeigu apribotos galiojimo srities elementus norime turėti objekto aprašo pradžioje, tai po jų reikės rašyti nuorodą *public* ir objekto elementus, kurių galiojimo sritis neribojama (žiūr. 7 pav.).

```
UNIT  sachmat;
INTERFACE
     TYPE
          Tst = 'a' .. 'h';     Teil = 1  .. 8;
          Tpoz = object
                        private
                        stulp : Tst;
                        eil : Teil;
                        procedure  pradpoz (s:Tst; e:Teil;);
                        public
                        function  gautst : Tst;
                        function  gauteil : Teil;
                   end;
IMPLEMENTATION  - - -
```

7 pav. *Private* ir *public* nuorodų naudojimo pavyzdys

7 pav. parodytas objekto aprašas leidžia į apribotos galiojimo srities elementus kreiptis tik panaudojant objekto metodus. To kaip tik reikalauja geras objektinio programavimo stilius.

9.5. Objektų paveldėjimas ir paveldėjimo taisyklės

Kalbant objektų paveldėjimo klausimu, naudojamos **objekto-tėvo** ir **objekto-sūnaus** sąvokos. Objektas-tėvas yra tas, kurio savybes perima kitas objektas, t.y. objektas-sūnus. Matysime, kad objekto-sūnaus apraše po žodelio *object* nurodomas objekto-tėvo vardas.

8 pav. pateiktame pavyzdyje objektas-sūnus *Tzaidejas* (čia turime objektus-tipus, o ne objektus-kintamuosius) taip pat turi visus objekto-tėvo *Tpoz* elementus: *stulp, eil,* metodus *gautst, gauteil*. Pastebėkime, kad tokio paties vardo metodai *pradpoz* (čia jie skiriasi parametrų kiekiu) yra objekto-tėvo ir objekto-sūnaus aprašuose. Todėl objekte-sūnuje galioja tik jame aprašytas metodas. Tačiau į tokio paties vardo objekto-tėvo metodą galima kreiptis nurodant patikslintą vardą *Tpoz.pradpoz (...);* Tą matome objekto-sūnaus metodo (procedūros) *Tzaidejas.pradpoz* programos tekste.

Apskritai, viename objekto-sūnaus metode gali būti kreipiniai į kitus to objekto-sūnaus metodus (pvz., metode *Tzaidejas.pasalinti* yra kreipinys į metodą *valyti*) ir į objekto-tėvo metodus.

Objektui-tėvui priklausančias sąvybes objektas-sūnus paveldi laikantis šių pagrindinių taisyklių:

1) objekto-tėvo duomenų laukus ir metodus (procedūras/funkcijas) objektai-sūnūs paveldi nepriklausomai nuo hierarchijos lygių kiekio (gylio). Objekto-sūnaus sąvybes gali paveldėti dar jaunesnis objektas-sūnus (objektas-anūkas) ir t.t.;

```
TYPE
        Tst = 'a' .. 'h';       Teil = 1 .. 8;       Tsplv = (juoda, balta);
        Tpoz = object                                          {objektas-tėvas}
                  private
                    stulp : Tst;
                    eil : Teil;
                    procedure  pradpoz (s:Tst; e:Teil;);
                  public
                    function  gautst : Tst;
                    function  gauteil : Teil;
                end;
        Tzaidejas = object (Tpoz)                              {objektas-sūnus}
                      private
                        spalva : Tsplv;
                        stovis : boolean;
                        procedure pradpoz (s:Tst; e:Teil; sp:Tsplv);
                      public
                        function gautsplv : Tsplv;
                        function figstovis : boolean;
                        procedure valyti;
                        procedure pasalinti;
                        procedure rodyti;
                    end;
        - - -                              { Tpoz  metodus žiūr. 6 pav. }
procedure  Tzaidejas.pradpoz (s:Tst; e:Teil; sp:Tsplv);
        begin   Tpoz.pradpoz ( s, e );   spalva := sp;   stovis := true;    end;
function Tzaidejas.gautsplv : Tsplv;                    {gauti figūros spalvai}
        begin     gautsplv := spalva;     end;
function Tzaidejas.figstovis : boolean;                 {ar figūra yra lentoje?}
        begin     figstovis := stovis;     end;
procedure Tzaidejas.valyti;                        {nuvalo figūrą šachmatų lentoje }
        begin   - - -     end;
procedure Tzaidejas.pasalinti;            {nuvalo figūrą lentoje kreipdamasi į }
        begin                             {procedūrą valyti ir priedo pažymi,  }
          valyti;                         {kad figūra pašalinta (numušta)        }
          stovis := false;
        end;
procedure Tzaidejas.rodyti;               {parodo figūrą einamojoje pozicijoje}
        begin    - - -     end;
```

2) objekto-tėvo duomenų laukai ir metodai prieinami objektui-sūnui taip, lyg jie būtų aprašyti objekte-sūnuje;

3) objekto-sūnaus duomenų laukų vardai (identifikatoriai) negali būti tokie patys kaip objekto-tėvo duomenų laukų vardai. Šis reikalavimas taikomas ir metodų formaliųjų parametrų vardams. Tačiau metodų vardai gali būti vienodi;

4) objektas-sūnus šalia paveldėtų objekto-tėvo sąvybių gali turėti bet kokį savo duomenų laukų ir metodų kiekį;

5) bet koks objekto-tėvo metodo programos teksto pakeitimas automatiškai turi įtakos juos naudojančių objektų-sūnų metodams.

Objektų sąvybių paveldėjimo mechanizme yra naudojama nuoroda **inherited** (angl. paveldėtas). Kadangi objekto-sūnaus metodo vardas gali būti toks pat kaip ir objekto-tėvo metodo vardas, tai, prireikus artimiausio(!!!) objekto-tėvo metodo, kreipinio priekyje rašomas *inherited*. Pavyzdžiui, 8 pav. *procedure Tzaidejas.pradpoz (s:Tst; e:Teil; splv:Tsplv);* viduje yra kreipinys į objekto-tėvo procedūrą *Tpoz.pradpoz (s, e);* Šį kreipinį galima būtų užrašyti taip: *Inherited pradpoz (s, e);*

Inherited negalima naudoti tokių objektų metoduose, kurie neturi objektų-tėvų.

Objektų-sūnų metoduose kreipimasis į iš objektų-tėvų paveldėtus metodus vyksta pagal šias taisykles:

1) sutikęs programos tekste kreipinį į metodą, kompiliatorius visų pirma ieško, ar metodas tokiu vardu nėra apibrėžtas pačiame objekte-sūnuje. Jei toks metodas objekte-sūnuje yra, tai į sukompiliuotą programą įstatomas kreipinys būtent jo adresu;

2) jei objekte-sūnuje nėra apibrėžtas metodas į kurį kreipiamasi, tuomet jo ieškoma artimiausiame objekte-tėve, o neradus čia, kopiama hierarchijos laiptais aukštyn. Niekur neradus, pranešama, jog yra klaida;

3) suradus paveldimą metodą, kompiliuojamoje programoje įstatomas kreipinys į jį. Toks metodas dirbs taip, kaip jis buvo sukompiliuotas objektui-tėvui. Jei paveldėtas objekto-tėvo metodas savo ruožtu kreipiasi į kitus metodus, tai pastarieji gali būti tik objekto-tėvo arba aukštesnės hierarchijos objektų. Objekto-tėvo metoduose negali būti kreipinių į objektų-sūnų metodus.

9.6. Statiniai bei virtualūs metodai

Dar kartą prisiminkime, kad metodas – tai procedūra/funkcija skirta veiksmams su objekto duomenų laukais. Metodai skirstomi į dvi rūšis: statinius ir virtualius.

Ligi šiol mūsų turėtuose objektų-tipų pavyzdžiuose buvo naudojami **statiniai metodai**. Sukompiliavus, pvz., 4 pav. pateiktą programą, visų objekto-tipo *Tpoz* metodų *pradpoz, gautst, gauteil* adresai statiškai (visam programos vykdymo laikui) susiejami su atitinkamais objektais-kintamaisiais *x, y*. Kreipiniais x.*pradpoz(...);* ir y.*pradpoz (...);* , nors jie taikomi skirtingiems objektams-kintamiesiems (egzemplioriams), bus iškviečiamas tas pats metodas *pradpoz* ir bus atliekami tokie patys veiksmai.

Virtualieji metodai suteikia lankstesnes galimybes, įgalina realizuoti **polimorfizmą.** Kompiliavimo metu virtualus metodas nėra susiejamas su objektais-kintamaisiais (iš anksto objektui-kintamajam nėra nustatomas adresas virtualaus metodo, kuriuo jis bus apdorojamas). Šiuo atveju objekto-tipo apraše po metodo (procedūros/funkcijos) antraštės gali būti rašomas *virtual* arba *virtual 247* (čia skaičius gali būti iš intervalo 1-65535). Pavyzdžiui,

```
TYPE  Tzaidejas = object (Tpoz)     - - -
                    procedure rodyti; virtual;     - - -
                    procedure put( aa: string );   - - -
                  end;
```

```
Trikis = object (Tzaidejas)   - - -
            procedure rodyti; virtual;   - - -
          end;
Tzirgas =  object (Tzaidejas)   - - -
             procedure rodyti; virtual;   - - -
           end;  - - -
procedure Tzaidejas.rodyti;   begin   put(‘ ‘); - - -  end;
procedure Tzaidejas.put(aa: string );  begin  - - -  end;
procedure Trikis.rodyti;   begin  put(‘rikis’); - - -  end;
procedure Tzirgas.rodyti;   begin  put(‘zirgas’); - - -  end;
```

Virtual po metodo antraštės neberašomas ten, kur yra visas metodo programos tekstas.

Virtualieji metodai neišvengiamai yra susiję su objektų-tėvų ir objektų-sūnų paveldėjimo klausimais. Visa esmė slypi tame, kad objekto-tėvo virtualus metodas gali būti pakeistas atitinkamu objekto-sūnaus virtualiu metodu. Jeigu objekte-tėve yra virtualusis metodas, tai ojektai-sūnūs turi tenkinti šiuos reikalavimus:

1) objekto-sūnaus metodas, kurio vardas sutampa su objekto-tėvo virtualaus metodo vardu, taip pat turi būti virtualus. Taip realizuojama galimybė objekto-tėvo virtualųjį metodą pakeisti objekto-sūnaus, į kurį programoje buvo kreiptasi, virtualiuoju metodu (**polimorfizmas**);

2) tokių pačių virtualiųjų metodų antraštės objekte-tėve ir objekte-sūnuje turi būti identiškos (turi būti vienodi metodų vardai, parametrų kiekis ir jų tipai turi sutapti);

3) kiekvienas objektas-tipas, kuriame yra bent vienas virtualus metodas, turi turėti konstruktorių (apie konstruktorius žiūr. toliau).

9.7. Konstruktoriai ir destruktoriai

Konstruktorius – tai tokio tipo procedūra, kuri parengia objekto virtualiems metodams reikiamas darbo sąlygas (mechanizmą). Programoje iki kreipimosi į kurio nors objekto virtualų metodą būtinai turi būti kreiptasi į to objekto konstruktorių. Nesant kreipinio į konstruktorių, gali sutrikti viso kompiuterio (sistemos) darbas. Kiekvienas objektas-kintamasis (egzempliorius) inicializuojamas atskiru kreipiniu į konstruktorių. Priskyrimo sakiniu priskiriant vieno objekto-kintamojo reikšmę kitam (jie abu turi būti tokio paties tipo), perduodamos tik objekto duomenų reikšmės, o inicializacija nepersiduoda. Rekomenduotina konstruktoriams suteikti vardą *init*.

Pavyzdys.

```
TYPE
Tst  =  ‘a’..‘h’;    Teil = 1..8;    Tsplv = (juoda, balta);
Tpoz = object
          private
            stulp : Tst;
            eil : Teil;
            procedure  pradpoz (s:Tst; e:Teil;);
          public
            function  gautst : Tst;
            function  gauteil : Teil;
        end;
Tzaidejas = object (Tpoz)
              private
                spalva : Tsplv;
                stovis : boolean;
                constructor init (s:Tst; e:Teil; sp:Tsplv);
```

Žiūr. tęsinį

```
                public
                  function gautsplv : Tsplv;
                  function figstovis : boolean;
                  procedure valyti;
                  procedure pasalinti;
                  procedure eiti (s:Tst; e:Teil); virtual;
                  function tikrinti (s:Tst; e:Teil): boolean; virtual;
                  procedure rodyti;  virtual;
                  destructor  atlikta;  virtual;
              end;
Trikis = object (Tzaidejas)
                constructor init (s:Tst; e:Teil; sp:Tsplv);
                function tikrinti (s:Tst; e:Teil): boolean; virtual;
                procedure rodyti; virtual;
                destructor atlikta;  virtual;
         end;
Tzirgas = object (Tzaidejas)
                constructor init (s:Tst; e:Teil; sp:Tsplv);
                function tikrinti (s:Tst; e:Teil): boolean; virtual;
                procedure rodyti; virtual;
                destructor atlikta;  virtual;
          end;  - - -
VAR   bz1, bz2, jz1, jz2 : Tzirgas;
      br1, br2, jr1, jr2 : Trikis;     - - -
constructor Tzaidejas.init ( s:Tst; e: Teil, splv: Tsplv );
      begin  pradpoz ( s, e );  spalva := splv;  stovis := true; end;
constructor Tzirgas.init ( s: Tst; e: Teil; splv: Tsplv );
      begin  inherited  init ( s, e, splv );  end;
constructor Trikis.init ( s: Tst; e: Teil; splv: Tsplv );
      begin  inherited  init ( s, e, splv );  end;
procedure Tzaidejas.eiti ( st: Tst; eil: Teil );
      begin  if  tikrinti (st, eil)  then  rodyti
                                  else … writeln (‘klaidingas ejimas’);
      end;
destructor  Tzaidejas.atlikta;  begin  end;
destructor  Tzirgas.atlikta;  begin  end;
destructor  Trikis.atlikta;  begin  end;
- - -            { čia turėtų būti ir ir visų kitų metodų programų tekstai}
begin  - - -    { pagrindinės programos sakinių skyrius }
      bz1.init (‘b’, 1, balta );   bz2.init ( ‘g’, 1, balta );  { konstruktorių }
      jz1.init ( ‘b’, 8, juoda );  jz2.init ( ‘g’, 8, juoda );  { vykdymas     }
      br1.init ( ‘c’, 1, balta );  - - -
      - - -
      bz1.eiti ( ‘c’, 3 );   - - -
      br1.eiti ( ‘d’, 2 );   - - -
      bz1.atlikta;  - - -              {atliekamas destruktorius bz1 figūrai }
end.
```

Toliau lentelėje parodytos objekto *bz1* (pirmo baltųjų žirgo) reikšmės ir sąsajos programos pradžioje ir po sakinio *bz1.init ('b', 1);* įvykdymo.

Lentelė

Objekto *bz1* (žiūr. 9 pav.) reikšmės ir sąsajos

Duomenys, metodai	Programos pradžioje	Įvykdžius sakinį *bz1.init ('b', 1);*	Paaiškinimai
Stulp		'b'	
Eil		1	
Spalva		Balta	
Stovis		True	
Pradpoz	Tpoz.pradpoz adresas	Tpoz.pradpoz adresas	Paveldėtas, statinis
Gautst	Tpoz.gautst adresas	Tpoz.gautst adresas	Paveldėtas, statinis
Gauteil	Tpoz.gauteil adresas	Tpoz.gauteil adresas	Paveldėtas, statinis
Gautsplv	Tzaidejas.gautsplv adresas	Tzaidejas.gautsplv adresas	Paveldėtas, statinis
Figstovis	Tzaidejas.figstovis adresas	Tzaidejas.figstovis adresas	Paveldėtas, statinis
Valyti	Tzaidejas.valyti adresas	Tzaidejas.valyti adresas	Paveldėtas, statinis
Pasalinti	Tzaidejas.pasalinti adresas	Tzaidejas.pasalinti adresas	Paveldėtas, statinis
Init	Tzirgas.init adresas	Tzirgas.init adresas	Konstruktorius
Rodyti		Tzirgas.rodyti adresas	Virtualus
Eiti		Tzaidejas.eiti adresas (!)	Virtualus
Tikrinti		Tzirgas.tikrinti adresas	Virtualus
Atlikta		Tzirgas.atlikta adresas	Virtualus

Vykdant 9 pav. programos *bz1.eiti(...);* sakinį bus kreipiamasi į objekto *Tzirgas* paveldimą metodą *Tzaidejas.eiti(...).* Atkreipkime dėmesį, kad metodo *Tzaidejas.eiti(...)* viduje yra kreipiniai į funkciją *tikrinti (...)* ir procedūrą *rodyti (...).* Dėl *virtual* nuorodų šiuo atveju bus kreipiamasi ne į *Tzaidejas.tikrinti* ir *Tzaidejas.rodyti,* bet į *Tzirgas.tikrinti* ir *Tzirgas.rodyti*.

Destruktorius – tai specialaus pavidalo metodas (procedūra), dažnai dar vadinamas "šiukšlių surinkėju" (žiūr. 9 pav.). Aprašant destruktorių vietoje žodžio *procedure* naudojamas *destructor* . Destruktoriumi inicializuojamos standartinės sisteminės funkcijos nurodytam objektui su virtualiais metodais, kurios yra vadinamos epilogu. Destruktoriaus sakinių skyrius dažniausiai būna tuščias.

Pažintis su šiame skyriuje pateiktomis objektinio programavimo sąvokomis ir principais padės skaitytojams ateityje lengviau įsisavinti sudėtingas šiuolaikines programavimo sistemas.