VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Rimantas Belevičius, Remigijus Kutas

F O R T R A N A S

Vilnius 1998

R. Belevičius, R. Kutas. FORTRANAS. Vadovėlis. V.: 1998. 241 p.

FORTRANas visada buvo ir yra dominuojanti programavimo kalba moksliniams ir inžineriniams uždaviniams programuoti. Tai - viena pirmųjų algoritminių kalbų, savo istoriją skaičiuojanti nuo 1957 m. Visi skaičiuotojai, inžinieriai programavo, programuoja ir dar ilgai programuos FORTRANu. Nepaisant to, lietuvių kalba apie FORTRANą literatūros beveik nėra. Šią spragą ir bando užpildyti šis vadovėlis.

Knygelės pirmame skyriuje glaustai paaiškinama veiksmų seka, kurią turime atlikti, norėdami parašyti tvarkingą programą. Čia kartu trumpai apibrėžiami kai kurie terminai. Antras skyrius paaiškina algoritmo sąvoką. Kiti du skyriai skirti FORTRANo elementams ir operatoriams. Penktame skyriuje gana plačiai aprašoma įvesties/išvesties sistema. Šeštame skyriuje pateikiami nesudėtingų programų pavyzdžiai, iliustruojantys ankstesnių skyrių medžiagą. Septintas skyrius skirtas paprogramių naudojimui aptarti. Aštuntas skyrius sudomins skaitytojus, norinčius plačiau susipažinti su FORTRAN 90 standarto naujovėmis. Devintame skyriuje pateikiami FORTRANo grafikos elementai, o paskutiniame, dešimtame, supažindinama su FORTRANo programavimo aplinkomis. Knygelės prieduose galima rasti standartinių funkcijų sąrašą, kitos naudingos informacijos.

Šis leidinys skirtas inžinerinių specialybių studentams bei magistrantams, studijuojantiems informatikos bei programavimo dalykus. Jis bus naudingas ir doktorantams bei dėstytojams, programuojantiems FORTRANu.

Vadovėlis rekomenduotas Fundamentinių mokslų, Mechanikos ir Statybos fakultetų studijų komitetų.

Leidinį recenzavo prof. habil. dr. R.Čiegis ir
prof. habil. dr. G. Kulvietis

Redagavo ir maketavo autoriai.

ISBN

© R. Belevičius, R. Kutas, 1998

P R A T A R M Ė

Skaitytojau! Rankose laikote knygelę, skirtą nepelnytai lietuviškoje literatūroje pamirštai programavimo kalbai FORTRAN. Mūsų vidurinė mokykla pramoko programuoti moderniomis Pascalio, kai kuriuos mokinius ir C kalbomis, todėl studentai stebisi: kodėl, atseit, jie mokysis tokios pasenusios kalbos.

Skaitytojau! Visi skaičiuotojai, inžinieriai programavo, programuoja ir dar ilgai programuos tik FORTRANu. Pagaliau ir pats kalbos pavadinimas (FORmula TRANslation - formulių vertimas) rodo, kad kalba tiesiogiai skirta moksliniams techniniams skaičiavimams. Jokia kita kalba programuojant šios srities uždavinius neprilygsta FORTRANui nei patogumu, nei greičiu. Todėl mes paprasčiausiai neturime pasirinkimo, kokia kalba mokyti programuoti būsimuosius inžinierius.

Lietuvių kalba FORTRANo mokomosios literatūros, išskyrus mūsų universitete 1988 m. išleistą FORTRANui 77 skirtą knygelę, nėra. Bandysim šią spragą užpildyti šia knygele. Norėjome knygelę padaryti suprantamą ir pradedančiajam programuotojui, dar neparašiusiam nė vienos programos, ir kartu aprašyti paskutinįjį standartizuotą FORTRANo dialektą - FORTRANą 90. Tačiau darbo metu paaiškėjo, kad šių dviejų tikslų vienoje knygelėje suderinti nepavyks. Vis dėlto FORTRANas 90 yra pernelyg sudėtinga pradedančiajam programuotojui, per daug įvairių galimybių turinti kalba. Ko gero, pradedančiajam lengviau būtų pradėti nuo paprastesnių FORTRANo 77 konstrukcijų ir palaipsniui perprasti visas FORTRANo 90 galimybes. Tam pagaliau padeda ir FORTRANo kalbos ideologija: visados vėlesnis kalbos dialektas, tapdamas sudėtingesniu, apima visas ankstesnių dialektų galimybes ir konstrukcijas (išimtis čia numatoma tik būsimame kalbos dialekte, kuriame jau nebebus aiškiai pasenusių konstrukcijų). Tuo tarpu vėlesnio dialekto papildomos konstrukcijos ankstesniam yra nesuprantamos. Todėl šios knygelės objektas galėtų būti apibrėžtas taip: FORTRANo 77 kalbos standartas plius FORTRANo 90 standarto naujos esminės galimybės inžineriniams uždaviniams spręsti. Dėl šių priežasčių ir knygelės pavadinimas yra “FORTRANas”. Autoriai pasieks savo tikslą, jei atidus skaitytojas, perskaitęs šią knygelę, ims ir skaitys bei supras FORTRANui 90 skirtą literatūrą ir kalbos techninę dokumentaciją.

Knygelę stengėmės parašyti taip, kad ji būtų suprantama *pradedančiajam* programuotojui. Todėl į knygelę įtraukėme ir kelis skyrius, kurie tiesiogiai nesusiję su FORTRANu. Pirmame skyriuje labai glaustai aiškiname visą veiksmų seką, kuri būtina, norint parašyti tvarkingą programą. Čia kartu trumpai apibrėžiame kai kuriuos terminus (baitas, ląstelė, failas, transliatorius, ...), kurie plačiai vartojami knygelėje ir kurie turėtų būti žinomi iš kompiuterijos kurso. Antras skyrius irgi labai glaustai paaiškina algoritmo sąvoką. Po ilgų dvejonių autoriai nusprendė algoritmus užrašyti

grafiniu pavidalu. Bent kiek labiau patyrusiam programuotojui šie algoritmai, aišku, sukels šypseną, tačiau pradedančiajam programuotojui programos logiką jais paaiškinti yra tikrai paprasta. Po to, kai išmoksim loginius bei ciklo operatorius, šių algoritmų iškart atsisakysim. Dėl tų pačių priežasčių į knygelę įtraukėme ir paskutinįjį skyrių, kuriame trumpai aiškinama, kaip parašytą programą įvesti į kompiuterio atmintį, ją paleisti darbui ir gauti rezultatus.

Nors knygelės apimtis ir nėra didelė, tačiau tikslinga būtų medžiagą perprasti skaitant knygelę du kartus. Pirmiausia derėtų išmokti tik paprasčiausias kalbos konstrukcijas ir kai kurių operatorių paprasčiausias formas, kurios leistų parašyti paprastutes programas ir kartu išsiaiškinti programavimo logiką. Antrajam skaitymui būtų galima palikti visą medžiagą, kurią supratus jau galima visai neblogai programuoti. Antrajam skaitymui rekomenduojamus skyrius ar skyrių dalis žymėsim žvaigždute.

Reikia pripažinti, kad knygelėje aprašytos ne visos inžinieriams neesminės kalbos galimybės, ne visi operatoriams keliami reikalavimai ir pan. Tačiau jei bandytume tą padaryti, tai knygutės apimtis išaugtų bent jau iki tų 500 puslapių, kuriuos turi kiekvieno FORTRANo 90 kompiliatoriaus aprašymas. Antra vertus, naudojant aprašytas kalbos galimybes bei nesileidžiant į programuotojų mėgstamus triukus, reikalaujančius gilių žinių, galima rašyti modernias programas.

Patariame skaityti itin atidžiai. FORTRANo kalba nuo mūsų kalbos skiriasi logika: joje nėra jokių dviprasmybių ar nutylėjimų. Tai lemia pati kompiuterio prigimtis ir veikimo principai. Gal būt kuri nors, atrodytų, smulkmenėlė sugriaus visą žinių “kortų namelį”. Todėl besimokant programavimo kalbos negalima praleisti jokios nesuprastos temos, imti ir skaityti knygelę nuo vidurio ir t.t. Parašyta programa arba veiks, arba neveiks. Todėl ir programavimo žinioms įvertinti užtenka iš principo dviejų pažymių: moki programuoti arba nemoki.

Autoriai dėkoja profesoriams R. Čiegiui ir G. Kulviečiui, atidžiai perskaičiusiems knygos rankraštį, už vertingas diskusijas ir pastabas bei pataisytas klaidas.

Visa atsakomybė už knygoje likusius netikslumus tenka autoriams: už 1 - 4-ą bei 6 - 8-ą skyrius - R. Belevičiui; už 5, 9, 10-ą bei priedus - R. Kutui.

Sėkmės besimokant!

T U R I N Y S

Į V A D A S

FORTRANas visada buvo ir yra dominuojanti programavimo kalba moksliniams ir inžineriniams uždaviniams programuoti. Tai viena pirmųjų algoritminių kalbų, savo istoriją skaičiuojanti nuo 1957 m. Įvade ir norėtume trumpai peržvelgti FORTRANo kalbos istoriją.

Pačioje skaičiavimo mašinų eros pradžioje programavimas buvo itin sunkus, varginantis užsiėmimas. Programuotojas turėjo puikiai pažinti skaičiavimo mašinos skaičiavimo įrenginį - procesorių: jo komandų sistemą, registrus, magistrales ir daugelį kitų dalykų. Pati programa buvo rašoma *mašinos instrukcijomis* - žmogui sunkiai skaitomu dvejetainiu kodu. Laikui bėgant imta naudoti simbolines šių instrukcijų santrumpas, o šias į mašinos instrukcijas pertvarkydavo specialios programos - *asembleriai*. Tokios *asemblerio kodais* vadinamos programos irgi itin efektyviai išnaudoja mašinos procesoriaus galimybes, tačiau joms kurti ir derinti vis dėlto reikėjo pernelyg daug laiko.

Išvardintos priežastys skatino kūrimą *aukšto lygio programavimo kalbų* - paprastų ir patogių žmogui bei kartu nedaug efektyvumu tenusileidžiančių asemblerio kodams. FORTRANas buvo viena pirmųjų tokių kalbų. Ją sukūrė amerikiečių firmos IBM tyrinėtojų grupė, kuriai vadovavo John Backus. Kalba išsiskyrė tuo, kad matematinės formulės joje atrodo labai panašiai į formules matematiniame tekste - todėl ją lengva išmokti; o kruopščiai parengti jos kompiliatoriai - vertėjai į mašinos instrukcijų kalbą - garantavo itin aukštą kalbos efektyvumą. FORTRANas programavime padarė tiesiog revoliuciją, nes nuo tada kompiuteriai tapo prieinami bet kuriam inžinieriui, pasirengusiam paaukoti truputį laiko FORTRANo įvaldymui.

Augant kalbos populiarumui, radosi vis nauji FORTRANo dialektai - kalbos versijos skirtingiems procesoriams ir operacinėms sistemoms, o kartu kilo ir problemos perkelti programas iš vieno tipo kompiuterio į kitokį kompiuterį. Todėl 1966 m., po keturių darbo metų, Amerikos nacionalinis standartų institutas (ANSI) sukūrė pirmąjį kalbos standartą - FORTRANą 66. Faktiškai šis standartas apėmė bendras visiems kalbos dialektams konstrukcijas, o į kiekvieną kalbos versiją galima žiūrėti kaip į standarto išplėtimą. Programuotojams, norintiems sukurti universalias, mobilias programas, teko naudoti tik į standartą įtrauktas galimybes - jos privalomos visiems kalbos kompiliatoriams.

FORTRANas sparčiai tobulėjo ir pasirodžius 66-ojo standartui. Daugelyje kalbos versijų atsirado naujos konstrukcijos, tinkamos plintančiam *struktūrinio programavimo* stiliui, naujos duomenų įvesties/išvesties galimybės. Visa tai atvedė prie naujo kalbos standarto, įteisinto 1978 m. - FORTRANo 77. Šis standartas programavimo pasaulyje įsivyravo apie 1980 metus.

Nors FORTRANo dominavimas sprendžiant inžinerinius-mokslinius uždavinius nekėlė jokių abejonių, vis dėto kilo mintys ir šią kalbą papildyti naujomis idėjomis, kurios buvo įgyvendintos vėliau sukurtose modernesnėse programavimo kalbose. Taip į FORTRANą atėjo nauji duomenų tipai, naujos nepavojingos atminties asocijavimo formos, laisva programos teksto forma, veiksmų su masyvais automatizavimas ir pan. Šios naujos kalbos galimybės jau leidžia rašyti programas, priderintas prie uždavinio objekto - *objektiškai orientuotas programas*. Kad nebūtų prarastas milžiniškas turtas - ankstesniais dialektais sukurtos programos - ir naujesnis 90-asis kalbos dialektas, kaip įprasta FORTRANui, apima ir "supranta" 66-ąjį ir 77-ąjį dialektus. Standartas įteisintas 1991 m. Skirtingai nuo ankstesnių kalbos standartų, kurie tik įteisindavo praktikoje įsigalėjusias kalbos dialektų konstrukcijas, 90-asis standartas suteikė kalbai daug naujų bruožų, būdingų kitoms programavimo kalboms. Prie 90-ojo standarto sukūrimo daugiausia prisidėjo Europos mokslininkai. Netrukus atsirado šio standarto papildymų ir jo praplėstas variantas vadinamas 95-uoju FORTRANu. Manoma, kad kita modifikacija bus FORTRANAS 2000 ir šis standartas bus pakankama programavimo priemonė ilgesnį laiką.

Pirmą kartą paskutiniame FORTRANo standarte paskelbtas pasenusių kalbos konstrukcijų sąrašas (į sąrašą įtraukti operatoriai aritmetinis IF, ASSIGN, priskiriamasis GO TO, žymėtasis RETURN, PAUSE, EQUIVALENCE, taip pat galimybės naudoti realiojo ir dvigubojo tikslumo tipų DO kintamuosius, užbaigti ciklą kitu nei CONTINUE ar END DO operatoriumi, užbaigti kelis ciklus vienu operatoriumi, priskirti žymes FORMATui). Dalies šių operatorių, jau senokai nematytų jokiose komercinėse programose, knygelėje neaprašysime. Tikriausiai jau kitame FORTRANo standarte minėtų pasenusių operatorių ir kalbos konstrukcijų nebeliks. Na, o kad kitas FORTRANo standartas bus sukurtas, jokių abejonių nekyla. Aišku ir kuo būsimasis standartas praturtins FORTRANo kalbą - tai bus lygiagrečiojo duomenų apdorojimo galimybės. Tokių galimybių atsiradimas jau tampa naudingu ir mums, nes spręsti sudėtingus, didelių kompiuterinių resursų reikalaujančius uždavinius juos išlygiagretinant jau galima ir Lietuvoje. 1998 metais Vilniaus Gedimino technikos universitetas įsigijo pirmąjį Lietuvoje daugiaprocesorinį IBM RS/6000 SP superkompiuterį ir RS/6000 darbo stočių klasę, kur įdiegta lygiagretaus programų vykdymo aplinka. Ji leidžia programuoti duomenų lygiagretų apdorojimą High Performance FORTRANu bei kitomis (MPI, PVM) uždavinių išlygiagretinimo priemonėmis

Programavimo FORTRANu sistemos yra naudingos moksliniams bei inžineriniams skaičiavimams dar ir tuo, kad jos turi dideles matematinių-inžinerinių uždavinių sprendimo paprogramių bibliotekas (IMSL - Digital Visual FORTRANe, ESSL,

PEESL - AIX FORTRANe) žymiai palengvinančias ir pagreitinančias standartinių uždavinių, ypač veiksmų su matricomis, sprendimą.

Knygelės pirmame skyriuje glaustai aiškinama veiksmų seka, kuri būtina norint parašyti tvarkingą programą. Čia kartu trumpai apibrėžiami kai kurie terminai. Antras skyrius paaiškina algoritmo sąvoką. Kiti du skyriai skirti FORTRANo elementams ir operatoriams. Penktame skyriuje gana plačiai aprašoma įvesties/išvesties sistema. Šeštame skyriuje pateikiami nesudėtingų programų pavyzdžiai, iliustruojantys ankstesnių skyrių medžiagą. Septintas skyrius skirtas paprogramių naudojimui aptarti. Aštuntas skyrius sudomins skaitytojus, norinčius plačiau susipažinti su FORTRAN 90 standarto naujovėmis. Devintame skyriuje pateikiami FORTRANO grafikos elementai, o paskutiniame, dešimtame, supažindinama su FORTRANo programavimo aplinkomis. Knygelės prieduose galima rasti standartinių funkcijų sąrašą, kitos naudingos informacijos.

Reikia pažymėti, kad knygelėje autoriai stengėsi laikytis tik į FORTRAN 90 standartą įtrauktų operatorių bei kalbos konstrukcijų. Turime įspėti skaitytojus, kurie naudos paplitusį Microsoft FORTRAN PowerStation kompiliatorių (atitinkama programavimo terpė trumpai aprašyta paskutiniame knygos skyriuje), kad kai kurie standarto operatoriai (INTERFACE, TYPE) šiam kompiliatoriui yra nesuprantami, o kiti kompiliatoriui žinomi operatoriai (INTERFACE TO, MAP, STRUCTURE, UNION, ...) nėra įtraukti į kalbos standartą - todėl knygoje apie juos nekalbama. Dalis panašių kompiliatoriaus ir kalbos standarto neatitikimų yra aptarta antrame priede.

Kaip minėta pratarmėje, knygelės autoriai nepretenduoja išsamiai aprašyti FORTRANą 90. Skaitytojai, pageidaujantys griežto programuotojo vadovo, nukreipiami į 1, 4, 5 ir 9 rekomenduojamos literatūros šaltinius. Kalbos standartą bei gausią informaciją apie egzistuojančius FORTRANo kompiliatorius, apie siūlomus FORTRANu 90/95 parašytus programų paketus, apie FORTRANo kalbą plėtojančius pasaulio mokslo centrus, informacijos apie paskutinį dar nestandartizuotą FORTRANą 2000 bei kitas žinias rasite Interneto puslapiuose:

http://www.fortran.com/
http://www.digital.com/fortran
http://www.compaq.com/fortran
http://www.nasoftware.co.uk/fortran/

Informaciją apie lygiagrečiuosius skaičiavimus ir jiems skirtą programinę įrangą (High Performance FORTRAN, MPI, PVM) ieškokite puslapiuose:

http://www.vcpc.univie.ac.at/activities - Europos lygiagrečiųjų skaičiavimų centras Vienoje,

http://www.pdc.kth.se - Švedijos Karališkojo technologijos universiteto Lygiagrečiųjų skaičiavimų centras,

http://www.gmd.de/SCAI/lab/adaptor/adaptor_home.html - Vokietijos nacionalinis informacinių technologijų tyrimo centras,

http://www.vtu.lt/sc - VGTU skaičiavimo centras. Šiame puslapyje skyrelyje Informacija vartotojui / Fortranas rasite ir daugiau informacijos apie Digital Visual Fortran aplinką, “run-time” procedūrų sąrašą Windows NT/9x operacinėms sistemoms, šios knygos variantą PDF formate, joje pateiktų programų pavyzdžių pradinius tekstus.

1. PROGRAMOS KŪRIMO ETAPAI. PROGRAMOS APDOROJIMO KOMPIUTERYJE ETAPAI

1-ame ir 2-ame skyriuose pateiktos žinios, griežtai kalbant, yra ne apie algoritminę kalbą FORTRAN. Čia labai trumpai panagrinėsim dalykus, bendrus visoms programavimo kalboms - programos apdorojimo ir vykdymo kompiuteryje etapus bei algoritmavimą. Nežinant šių dalykų, mokytis programuoti būtų keblu.

Terminija. Iš pradžių norėtume apibrėžti tuos informatikos terminus, su kuriais iškart susidursim. Nors šie terminai skaitytojui turėtų būti žinomi, dėl viso pikto juos priminsim ir paaiškinsim prasmę, kuria jie bus vartojami tekste.

- *Baitas* (angliškai byte) - smulkiausias adresuojamas kompiuterio atminties vienetas, prieinamas programuotojui iš FORTRANo 77 programos.
- *Ląstelė* (cell) - kompiuterio atminties dalis, kurioje saugomas vienas duomuo (= konstanta, kintamasis, ...). Ląstelė yra vienokio ilgio, pavyzdžiui, sveikam duomeniui, ir kitokio - realiam, turinčiam trupmeninę dalį. Ląstelės ilgis matuojamas baitais.
- *Failas* (file) - kompiuterio atmintyje saugoma logiškai susieta duomenų seka, kuriai parinktas vardas. Kol kas mums užteks tokio supaprastinto failo apibrėžimo. 5-ame skyriuje pateikiamas tikslus apibrėžimas.
- *Operatyvioji atmintis* (main memory, primary memory, RAM) - greitoji kompiuterio atminties rūšis. Šioje atmintyje prieš apdorojimą saugomi visi duomenys, pati programa, gauti rezultatai prieš išvedimą. Techniškai tai - integrinė grandinė, integrinė schema. Tai nepastovi atmintis: išjungus kompiuterį, visa, kas buvo joje, dingsta.
- *Išorinė atmintis* (secondary memory, hard disk memory) - pastovi kompiuterio atmintis. Techniškai tai - kietasis diskas. Išjungus kompiuterį, visa, kas joje buvo įrašyta, išlieka.
- *Transliatorius, kompiliatorius* (translator, compiler) - programa-vertėjas, verčianti programą iš algoritminės kalbos į mašinos kodų kalbą.

Programos kūrimo etapai. Norėdamas užprogramuoti kokį nors uždavinį, tik patyręs programuotojas gali imti popierių ir iškart pradėti rašyti programą. Mažiau patyrusiam toks programavimo metodas duos dažniausiai apverktinus rezultatus.

Programos kūrimo etapai turėtų būti:

- Uždavinio apibūdinimas: reikia išsiaiškinti visą skaičiavimų eigą, numatyti pradinių duomenų bei rezultatų struktūrą, numatyti pradinių duomenų įvedimo į kompiuterio atmintį bei rezultatų išvedimo iš atminties formas.

- Programos loginės schemos - algoritmo - sukūrimas. Algoritmą galima užrašyti žodžiais, nubraižyti blokinės schemos pavidalu. Mes naudosim pastarąją algoritmo formą.
- Kodavimas - rašymas programos algoritmine programavimo kalba pagal sudarytą algoritmą.
- Programos vykdymas. Savo ruožtu šis etapas apima kelis smulkesnius veiksmus (žr. žemiau).
- Testavimas - klaidų programoje ieškojimas. Susidursim su trejopo pobūdžio klaidomis: sintaksės, programos vykdymo metu kylančiomis ir loginėmis. Programos operatoriai, kaip matysim, turi būti rašomi laikantis griežtų sintaksės taisyklių. Jei šie reikalavimai pažeisti - programa nevykdoma, deklaruojamos sintaksinės klaidos. Ištaisę visas sintaksės klaidas, gauname sintaksiškai korektišką, tačiau dar nežinia, ar logiškai teisingą programą: programa gali skaičiuoti kažkokius nebūtinai teisingus rezultatus. Loginės klaidos ieškomos taip: programa vykdoma su pradiniais duomenimis, kuriems teisingas rezultatas yra žinomas iš anksto, arba nesudėtingiems pradiniams duomenims “rankomis” suskaičiuojami rezultatai; po to su tais pat duomenimis vykdoma programa. Jei abu rezultatai sutampa, galima įtarti, kad programa teisinga; jei ne - tenka ieškoti loginių klaidų programoje, o gal būt ir algoritme. Tokie tikrinimai turėtų būti atlikti bent su keliais skirtingų pradinių duomenų variantais.

Dažniausiai dėl prasto programos algoritmo randasi ir vykdymo klaidos (angliškai run-time-errors): dalyba iš nulio, perpildymas (t.y. gaunamas toks skaičius, kurio įtalpinti į atmintį neįmanoma) ir pan. Šios klaidos nutraukia programos vykdymą.

- Programos dokumentavimas. Sudėtingesnė programa turi turėti instrukciją vartotojams bei techninius dokumentus. Dokumentuose vartotojams derėtų aprašyti, kaip reikia įvesti pradinius duomenis, kaip kreipti skaičiavimų eigą, pateikti skaičiavimo programa pavyzdžius. Techniniuose dokumentuose aprašomi programos tikslai, skaičiavimo metodai, pateikiami algoritmai, visų programos išvedamų meniu ir pranešimų pavyzdžiai, kartais ir programos tekstai, pagaliau nurodoma, kaip įrašyti programą į kompiuterio atmintį ir ją vykdyti.

Pradedančiajam programuotojui, matyt, atrodys, kad sudėtingiausias yra kodavimo etapas. Vis dėlto sudėtingesni yra pirmieji du. Nuo pasirinkto skaičiavimo metodo, algoritmo - trumpai tariant, programos architektūros - priklausys viso tolesnio darbo sėkmė. O daugiausia programuotojo laiko užima, kaip įprasta, testavimas. Nors sintaksės klaidas išrinkti paprasta, o labiau patyrę programuotojai jų paprasčiausiai nebedaro, tačiau loginės klaidos yra sunkiai aptinkamos. Dar daugiau, kiekviena tikrai sudėtinga programa neišvengiamai turi nepastebėtų loginių klaidų, pasireiškiančių tik tam tikriems pradinių duomenų variantams. Mes, aišku, kol kas tokių sudėtingų programų nerašysime, todėl su tokio pobūdžio problemomis nesusidursime.

Programos apdorojimo etapai. Tarkim, turim parašytą programą. Įrašėm ją į kompiuterio atmintį ir atidavėm kompiuteriui ją vykdyti. Programa kompiuteryje bus apdorojama trimis etapais:

- Programos vertimas (angliškai translation, compilation) iš algoritminės kalbos - mūsų atveju FORTRANo - į vienintelę kompiuteriui suprantamą mašinos kodų kalbą, kurią sudaro tik dvejetainiai simboliai 0 ir 1. Šiuos veiksmus atlieka labai sudėtinga programa-vertėjas - kompiliatorius, arba kitaip transliatorius. Būtent šio etapo metu transliatorius tikrina programos sintaksę ir deklaruoja visas aptiktas sintaksės klaidas. Radęs klaidų, transliatorius nutraukia programos apdorojimą. Jei kompiliavimo klaidų nėra - bus vykdomas antrasis apdorojimo etapas -
- programos saistymas (linking). Šio etapo metu prie pateiktos programos prijungiamos visos kitos reikalingos programos. Pavyzdžiui, vėliau sužinosim, kad FORTRANo programoje galima kreiptis į trigonometrinę tangento funkciją TAN(x). Tangentas skaičiuojamas atskiroje programoje, kurios programuotojui rašyti nereikia; ši tangento skaičiavimo programa yra kalbos sudėtinė dalis. Surinkimo metu šią programą lieka tik prijungti prie programuotojo parašytos programos. Panašiai ir su kitomis programomis, į kurias kreipiasi programuotojo programa. Jei saistymo metu sėkmingai randamos ir prijungiamos visos reikalingos pagalbinės programos, bus vykdomas paskutinysis programos apdorojimo etapas -
- programos vykdymas (execution). Specialiai nerašome: “skaičiavimas”, nes programa gali ne tik skaičiuoti, bet ir, pavyzdžiui, kurti tekstus ir pan. Dabar vykdomi visi programoje užprogramuoti veiksmai: pradinių duomenų įvedimas, skaičiavimai, rezultatų išvedimas.

Dar kartą pabrėžiam, kad ir sėkmingai įveikę visų trijų apdorojimo etapų barjerus, negalim būti tikri, jog programos gauti rezultatai yra teisingi. Turim patikrinti, ar programoje nesislepia loginių klaidų.

2. ALGORITMAS

Programuojant visados reikia aiškiai suvokti, kad kompiuteris gali tik:

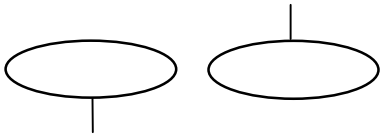
- atlikti aritmetines operacijas;
- atlikti logines santykio operacijas (palyginti, ar vienas duomuo didesnis už kitą, didesnis arba lygus, mažesnis, mažesnis arba lygus, lygus; nelygus);
- atlikti informacijos mainus tarp žmogaus / kompiuterio ar kompiuterio / žmogaus.

Bet kokį - paprastutį ar labai sudėtingą - uždavinį galėsim užprogramuoti tik tada, kai sugebėsim visą jį išskaidyti iki tokių aritmetinių, loginių ir informacijos mainų operacijų lygmens. Tokių operacijų seką vadinsime *algoritmu*. Algoritmo dalis apibrėšim rėmeliais, o atskirų tokių blokų vykdymo seką parodysim rodyklėmis.

Taigi, algoritmas - uždavinio sprendimo planas. Šis planas, užrašytas žodžiais arba pavaizduotas blokine schema, yra daug paprastesnis ir vaizdesnis už pačią programą. Todėl pradedantiesiems programuotojams naudinga prieš rašant programą sudaryti tokį uždavinio sprendimo planą. Be to, uždavinio sprendimo algoritmas nepriklauso nuo to, kokia programavimo kalba bus rašoma programa. Vėliau, įgijus pakankamai programavimo patirties, blokinių schemų galima nebebraižyti, tačiau ir tada uždavinio sprendimo būdas privalo būti visiškai aiškus ir turi būti “saugomas” programuotojo galvoje. Programos rašymas pagal turimą algoritmą yra jau palyginti nesudėtingas dalykas.

Sutarsime blokus, iš kurių sudarysime algoritmų blokines schemas, žymėti taip (2.1 lentelė).

2.1 lentelė. Blokinės schemos blokai ir jų paskirtis

Blokų vaizdavimas	***Blokų paskirtis***
	Programos pradžiai ir pabaigai žymėti
	Aritmetiniams veiksmams užrašyti

2.1 lentelės tęsinys

Blokų vaizdavimas	*Blokų paskirtis*
  taip   =0	Informacijos mainams žymėti: duomenims įvesti į operatyviąją atmintį klaviatūra ar iš išorinės atminties; duomenims išvesti į ekraną, išorinę atmintį. Loginiam reiškiniui, turinčiam dvi reikšmes (“tiesa” - taip, “melas” - ne) užrašyti Loginiam reiškiniui, kuriame aritmetinis reiškinys lyginamas su nuliu, užrašyti

Iš vieno bloko gali išeiti dvi ar trys rodyklės (loginiams blokams) arba tik viena rodyklė (visiems kitiems blokams). Į vieną bloką gali sueiti kiek norima rodyklių. Tai ir visos algoritmų užrašymo taisyklės . Reikia tik atkreipti dėmesį į tai, kad ženklas “=“ blokinėse schemose ir programose turi kitokią prasmę nei matematikoje. Čia ženklas ne parodo lygybę, o *priskiria*: duomeniui, esančiam kairėje ženklo pusėje, priskiriama reiškinio ar duomens dešinėje pusėje reikšmė. Prieskyros metu ankstesnė kairiosios pusės duomens reikšmė prapuola.

Beveik visada algoritmus, o vėliau pagal juos programas stengsimės rašyti taip, kad visi pradiniai duomenys programose figūruotų kintamųjų pavidalu, t.y. programoje matysim tik jų vardus, o konkreti reikšmė šiems kintamiesiems bus suteikiama pradinių duomenų įvesties metu. Tokiu būdu gaunamos bendresnio pobūdžio programos, tinkamos visiems tos pat klasės uždaviniams spręsti. Jei programoje pradiniai

duomenys figūruotų konstantų pavidalu, tai norėdami išspręsti tokia programa kitą uždavinį, besiskiriantį nuo pirmojo tik duomenimis, turėtume keisti programos tekstą.

Užrašysime kelių nesudėtingų uždavinių sprendimo algoritmus.

1 pavyzdys

Sudaryti algoritmą funkcijos reikšmei skaičiuoti pasirinktinai pagal išraiškas:

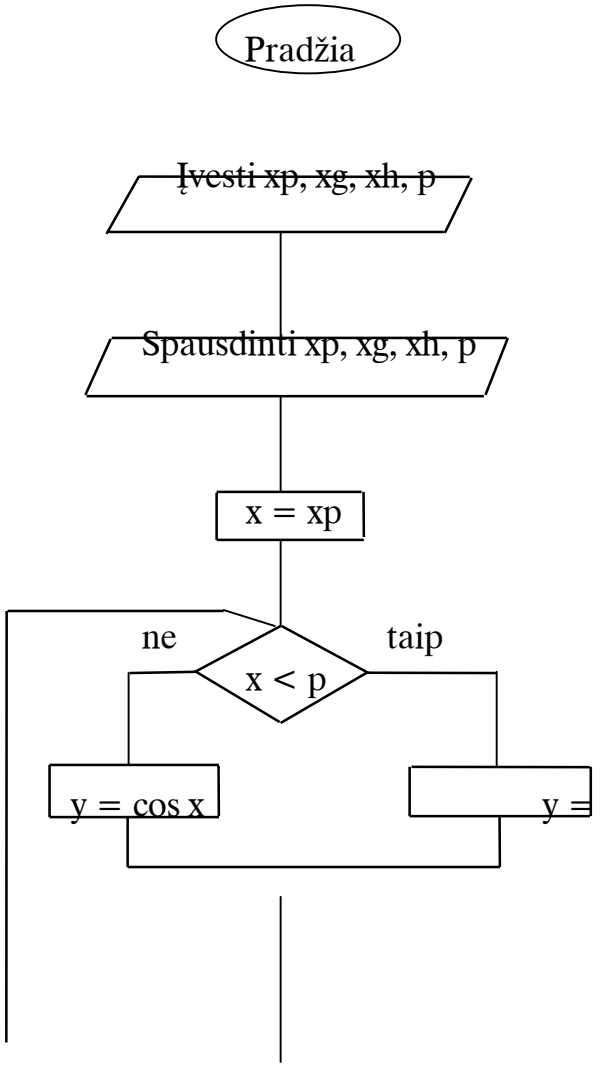
$$y = \begin{cases} \sin x, & x < 1 \ , \\ \cos x, & x \geq 1 \ . \end{cases}$$

Argumentas x kinta nuo pradinės reikšmės 0 radianų iki galinės reikšmės 1.5 radiano pastoviu žingsniu 0.1 radiano (čia ir visur kitur vietoje dešimtainio kablelio rašysim tašką - taip įprasta informatikoje).

Sprendimas

Šio uždavinio pradiniai duomenys- pradinė argumento reikšmė, žingsnis, kuriuo keičiasi argumentas; argumento reikšmė, dalijanti argumento kitimo sritį į du intervalus ir galinė argumento reikšmė. Pažymėsim šiuos duomenis vardais xp, xh, p, xg. Argumento eilinė reikšmė tegu bus x, o atitinkama funkcijos reikšmė - y. Tuo būdu kompiuterio operatyviojoj atminty toks algoritmas reikalauja vienu metu talpinti 6 kintamuosius - užimti 6 ląsteles atminties duomenims saugoti.

Blokinė schema su paaiškinimais pateikiama žemiau.



Skaičiavimų pradžia.

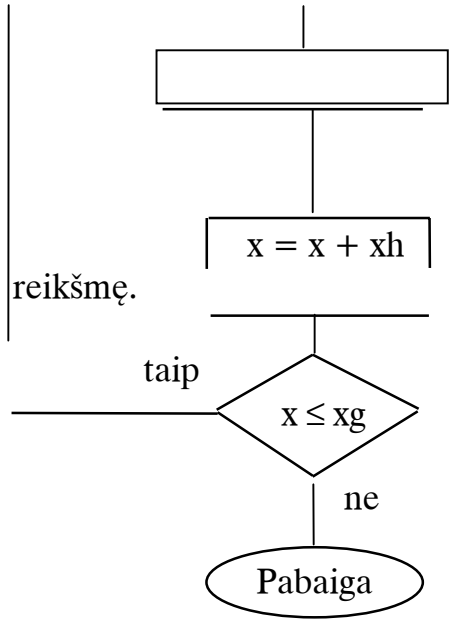
Šių kintamųjų reikšmės turi būti nuskaitytos, pavyzdžiui, iš klaviatūros į kompiuterio atmintį.

Kontrolinis įvestų reikšmių spausdinimas: kad būtų išvengta dažnų klaidų įvedimo metu.

x priskirti reikšmę xp. Pradėsime skaičiavimą nuo argumento pradinės reikšmės.

Tikrinimas, į kurį intervalą pakliūna x. Jei loginis reiškinys įgyja prasmę "tiesa" - t.y. x<p, tai funkciją skaičiuojame pagal pirmąją išraišką. "Ne" reiškia, kad x nėra mažiau už p, taigi bus didesnis arba lygus p.

Todėl šiuo atveju funkcija skaičiuojama pagal antrąją išraišką.



Argumento ir gautą funkcijos reikšmes išvesti iš operatyviosios atminties į žmogui prieinamą pavidalą (pavyzdžiui, į kompiuterio ekraną).

x priskirti žingsniu didesnę reikšmę.

Patikrinti, ar x viršija galinę reikšmę. Jei ne - skaičiuoti toliau, jei taip -

uždavinio pabaiga.

2 pavyzdys

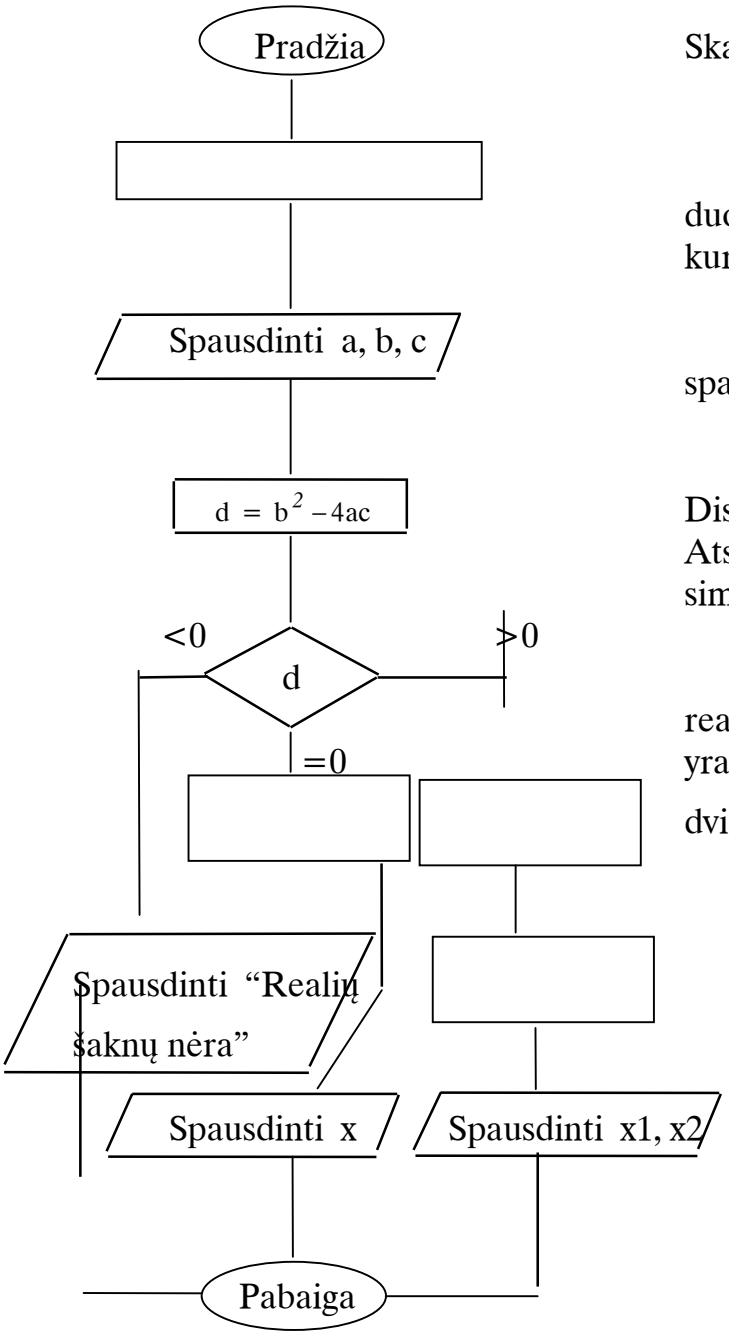
Sudaryti kvadratinės lygties $ax^2 + bx + c = 0$ sprendimo blokinę schemą .

Sprendimas

Šio uždavinio pradiniai duomenys - nežinomųjų a, b ir c koeficientai. Programoje žymėsime juos šiais vardais. Rašydami uždavinio sprendimo planą, turime turėti galvoje, kad esant kai kurioms šių duomenų kombinacijoms lygtis neturės realiųjų šaknų: būtent, kai diskriminantas (programos tekste žymėsim jį vardu d) yra neigiamas. Todėl turime iš pradžių suskaičiuoti diskriminanto reikšmę, ir, esant neigiamai jo reikšmei, išvesti į kompiuterio ekraną atitinkamą pranešimą bei po to nutraukti programos darbą. Jei į tokią galimybę rašydami programą neatsižvelgsime, tai, esant $d<0$, programoje rasis vykdymo klaida - programa baigs savo darbą nesėkmingai, klaidos pranešimu. Be abejo, tokia programa būtų nekorektiška.

Kai d neneigiamas, lygtis turi vieną arba dvi šaknis. Jas žymėsime vardais x bei x1 ir x2.

Blokinė schema pateikiama kitame puslapyje.



Skaičiavimų pradžia.

Duomenų įvedimas. Šio uždavinio duomenys - trys koeficientai, kuriuos žymėsime vardais a, b, c.

Kontrolinis duomenų spausdinimas.

Diskriminanto skaičiavimas. Atsižvelgdami į jo reikšmę, turėsim skaičiuoti 0, 1 ar 2 šaknis.

Jei diskriminantas neigiamas - realiųjų šaknų nėra; jei lygus nuliui - yra viena šaknis; jei teigiamas - dvi šaknys. Šaknų skaičiavimas.

Rezultatų spausdinimas.

Pabaiga.

3. FORTRANO ELEMENTAI

3.1. SUTARTINIAI ŽYMĖJIMAI

Programa, parašyta FORTRANo kalba, susideda iš *operatorių*, talpinamų vienoje arba keliose eilutėse. Operatorių, savo ruožtu, sudaro standartinis žodis (angliškas) ir dažnai dar ir informacija, nurodoma programuotojo nuožiūra. Ši informacija pateikiama, aišku, pagal tam tikras taisykles, tačiau konkretus jos turinys priklauso tik nuo programuotojo. Aiškindami operatorius, standartinius FORTRANo žodžius rašysime didžiosiomis raidėmis, o visą kitą informaciją - mažosiomis raidėmis kursyvu. Jei ši informacija nėra būtina, tai ją apimsime laužtiniais skliaustais.

Kai kuriuose operatoriuose tam tikra informacijos grupė gali būti kartojama keletą sykių. Tokį pakartojimą žymėsim daugtaškiu.

Programų tekstuose daugtaškiu taip pat žymėsim programos eilutės (-čių) praleidimą.

Operatoriai gali būti numeruoti. Operatorių numerius vadinsim *žymėmis* ir žymėsim *ž1, ž2, ...*

Daugeliui operatorių reikia nurodyti parametrus, kurių reikšmės turi būti parenkamos iš kelių alternatyvų tarpo. Šias alternatyvas rašysime figūriniuose skliaustuose, o atskirsim jas vieną nuo kitos vertikaliais brūkšniais. Tuos parametrus, kuriuos nurodyti nebūtina, kaip ir visą nebūtiną informaciją, apimsim laužtiniais skliaustais.

Šalia daugelio operatorių nurodomi vienodo prioriteto elementai, kurie paprastai skiriami vienas nuo kito kableliais. Tai trumpai vadinsime *sąrašu* ir žymėsim kursyvu atspausdintu žodžiu *sąrašas*.

Rašydami programų pavyzdžius, standartinius žodžius taip pat rašysim didžiosiomis raidėmis, o visa kita - mažosiomis.

Kartais tos pačios FORTRANo 77 ir FORTRANo 90 konstrukcijos smarkiai skiriasi. Tokiu atveju pateiksim abiejų dialektų konstrukcijas ir nurodysim, kurios jų priklauso vienam ir kurios - kitam dialektui. Primename, kad visada vėlesni FORTRANo dialektai apima visas ankstesnes konstrukcijas, todėl 77-ojo dialekto konstrukcijos ir operatoriai yra tinkami bet kuriam transliatoriui. Ten, kur skirtumai tarp abiejų dialektų itin dideli, apsiribosime 77-ojo standarto galimybėmis, o 90-ojo konstrukcijų aprašymą nukelsime į 8-ąjį skyrių.

Pagaliau, kaip minėta įžangoje, autoriai stengėsi parašyti tokią knygelę, kurią galėtų skaityti ir visiškai nieko apie programavimą nežinantis, ir pramokęs programuoti studentas. Pirmą kartą skaitant knygelę derėtų įsidėmėti tik tą informaciją, be kurios neįmanoma parašyti paprasčiausios FORTRANo programos (praktiškai tai būtų tik 77-

ojo standarto galimybės). Antram skaitymui rekomenduojamus skyrius pažymėsime žvaigždute. Dėl tos pat priežasties - kad knygelė neatrodytų pernelyg sudėtinga - kai kurias sudėtingesnes FORTRANo 90 temas nukėlėme į knygelės pabaigą, nors logiškai jos labiau derėtų knygutės pradžioje. Taip, pavyzdžiui, atributai atsidūrė 8-ame skyriuje, nors turėtų būti aprašyti FORTRANo elementų skyriuje.

Taigi nuo šiol, aiškindami FORTRANo konstrukcijas bei rašydami programų pavyzdžius, laikysimės tokių sutartinių žymėjimų.

Aiškinant operatorius ir konstrukcijas:

ž1 OPEN (*valdančiosios informacijos sąrašas*)	OPEN - standartinis žodis, kuris taip turi atrodyti visada; *sąrašas* - tam tikra informacija, kurią programuotojas pateikia savo nuožiūra pagal tam tikras taisykles; *ž1* - žymė.
DATA *kintamųjų sąrašas /konstantų sąrašas/ ...*	Reikėtų suprasti, kad kintamųjų bei konstantų sąrašai tokia tvarka gali būti kartojami; tuo atveju tarp pakartojimų dedami kableliai.
[FORM = {FORMATTED|UNFORMATTED|BINARY}]	FORM - nebūtinas parametras, kuris taip rašomas visada ir kuriam reikia suteikti vieną iš trijų alternatyvių reikšmių.

Programų pavyzdžiuose:

PROGRAM p1 . . . 100 WRITE (*,*,ERR=10) a*b**2 END	Čia PROGRAM, WRITE, END bei ERR yra standartiniai FORTRANo žodžiai, todėl rašomi didžiosiomis raidėmis. p1, a*b**2 - programuotojo sugalvoti dalykai, todėl rašomi mažosiomis raidėmis.

Tekste:

*** 7.8. OPERATORIUS SAVE**

Šiame skyriuje pateiktos žinios nebūtinos pradedančiajam programuotojui, todėl pirmą kartą skaitant knygelę šį skyrių galima praleisti.

3.2. SIMBOLIAI

FORTRANo programoje galima naudoti vadinamojo ASCII (American Standard Code for Informational Interchange) kodo simbolius. Taigi FORTRANo programoje gali būti tokie simboliai:

- 26 didžiosios lotyniško alfabeto raidės (nuo A iki Z) ir 26 mažosios (nuo a iki z). Kompiliatorius interpretuoja mažąsias raides lygiai taip, kaip ir didžiąsias. Raidžių tipas svarbus tik tekstinėse konstantose.
- 10 skaitmenų (nuo 0 iki 9).
- Simboliai = + - * / ( ) , . $ ‘ : ir visi kiti specialūs ASCII simboliai.
- Intervalo (tuščio tarpo) simbolis. Šis simbolis ypatingas tuo, kad neturi grafinio vaizdo. FORTRANe 77 intervalas neturi reikšmės operatorių ar vardų interpretavimui ir yra svarbus tik tekstinėse konstantose bei prieš operatorius (žr. 3.3 skyrių). Pavyzdžiui, operatoriai GO TO 10 , GOTO10 ar G OTO 1 0 transliatoriui yra visiškai vienodos prasmės. *Pastaba.* Tekste, kur intervalo simbolis yra svarbus arba kur svarbus yra intervalų kiekis, intervalą žymėsime matomu sutartiniu ženklu ⎵ .
- FORTRANe 90 intervalas neturi reikšmės tarp *leksemų*, bet intervalai neleistini leksemos viduje. Leksema - tai simbolių grupė, turinti elementarią reikšmę (pavyzdžiui, žymė, vardas, standartinis FORTRANo žodis).

Reikia pažymėti, kad programos komentaruose ir tekstinėse konstantose galima naudoti ne tik FORTRANo simbolius, bet ir visus kitus kompiuterio palaikomus grafinius simbolius, pavyzdžiui, lietuviškas ar rusiškas raides.

3.3. PROGRAMOS UŽRAŠYMO TAISYKLĖS

FORTRANas 77 reikalauja tam tikro programos formato. Užrašant programą būtina laikytis tokių taisyklių:

- Kiekvienoje programos eilutėje gali būti pateikiamas tekstas, kurio ilgis daugiausia 72 simboliai.
- Žymės rašomos pozicijose nuo 1 iki 5 imtinai.
- Operatoriai rašomi pozicijose nuo 7 iki 72 imtinai.
- 6-oji pozicija skirta taip vadinamam perkėlimo simboliui. Jei operatorius ilgesnis nei 66 simboliai, jį būtina perkelti į kitą eilutę. Tokiu atveju operatorius nutraukiamas bet kurioje vietoje, kitos eilutės 6-oje pozicijoje rašomas perkėlimo simbolis (juo gali būti bet kuris FORTRANo simbolis, tik ne nulis ir ne intervalas), ir ši eilutė tampa tąsos eilute. Kiekvienam operatoriui jų leidžiama turėti iki 19.
- Jei 1-oje pozicijoje įrašyti simboliai C, mažoji c arba *, tokia eilutė laikoma komentaro eilute ir nėra verčiama į mašinos kodus. Tokios eilutės skirtos ne kompiuteriui, bet pačiam programuotojui reikalingam aiškinamajam tekstui rašyti. Šauktuko ženklas ! , parašytas ne 6-oje pozicijoje, šioje eilutėje pradeda komentarą nuo kitos pozicijos; komentaras tęsiasi iki eilutės pabaigos. Šauktukas 6-oje pozicijoje yra tąsos simbolis.
- Dolerio ženklas $ 1-oje pozicijoje pradeda metakomandos eilutę. Metakomanda valdo kompiliatoriaus funkcijas.

* FORTRANas 90, priimdamas tokį fiksuotą programos formatą, kartu pripažįsta kur kas liberalesnes programos pateikimo taisykles:

- Eilutės ilgis - 132 simboliai.
- Operatoriai pradedami nuo bet kurios pozicijos.
- Komentarai rašomi pagal tas pat taisykles, kaip FORTRANe 77. Leistinos ir tuščiosios eilutės, sudarytos vien tik iš intervalų.
- Eilutės tąsos požymis - & (ampersendas) rašomas pratęsiamos eilutės gale. Paskutinis pratęsiamos eilutės simbolis - simbolis prieš ampersendą, o pirmas pratęsimo simbolis - pirmasis neintervalas tąsos eilutėje. Tąsos eilutę galima pradėti taip pat ampersendu: tada pirmas pratęsimo simbolis - pirmasis simbolis po ampersendo. Daugiausia galimos 39 tąsos eilutės (neskaičiuojant komentarų). Komentarų pratęsti negalima. Neleistina eilutė, kurioje yra tik vienas simbolis - ampersendas.
- Vienoje eilutėje galima talpinti kelis operatorius. Operatoriai vienas nuo kito skiriami kabliataškiu ; .
- Žymė - sveikoji aritmetinė konstanta, turinti iki 5 simbolių, kurių bent vienas nelygus nuliui. Pradiniai žymės nuliai nereikšmingi: 10 ir 010 transliatoriui reiškia tą pat.

3.4. PROGRAMOS ATLIKIMO TVARKA. OPERATORIŲ IŠDĖSTYMO PROGRAMOJE TVARKA

Programos operatoriai vykdomi paeiliui iš viršaus į apačią. Šią tvarką gali pakeisti tik vadinamieji valdymo perdavimo operatoriai, nurodantys, kuris operatorius turi būti vykdomas.

Visi operatoriai skirstomi į vykdomuosius ir nevykdomuosius. Pirmieji keičia kompiuterio atminties turinį arba programos operatorių atlikimo tvarką. Antrieji tik praneša transliatoriui tam tikrą informaciją apie vieną ar kitą programos objektą.

Vykdomųjų operatorių tvarką programoje lemia programuotinų veiksmų seka, tuo tarpu nevykdomieji operatoriai tarpusavyje išdėstomi pagal tam tikras taisykles. Visuomet nevykdomieji operatoriai programoje surašomi aukščiau pirmojo vykdomojo operatoriaus (išimtis sudaro tik nevydomieji operatoriai ENTRY, DATA, FORMAT, kurie gali būti bet kurioj programos vietoj, tačiau ne aukščiau programos įvardijimo ir ne žemiau programos užbaigimo operatorių).

Programoje operatoriai išdėstomi 3.1 lentelėje pateikta tvarka.

3.1 lentelė. Operatorių tarpusavio tvarka programoje

<table>
<tr><td colspan="3">PROGRAM, SUBROUTINE, FUNCTION, MODULE, BLOCK DATA</td></tr>
<tr><td colspan="3">USE</td></tr>
<tr><td>COMMON,
DIMENSION,
EQUIVALENCE,
EXTERNAL,
IMPLICIT,
INTERFACE,
INTRINSIC,
NAMELIST,
SAVE,
tipo apibrėžimo operatoriai</td><td>PARAMETER</td><td rowspan="3">ENTRY,
FORMAT</td></tr>
<tr><td>operatoriai - funkcijos</td><td rowspan="2">DATA</td></tr>
<tr><td>vykdomieji operatoriai</td></tr>
<tr><td colspan="3">CONTAINS</td></tr>
<tr><td colspan="3">vidinių subprogramų operatoriai</td></tr>
<tr><td colspan="3">END</td></tr>
</table>

Šioje lentelėje dešiniau esančiame stulpelyje surašyti operatoriai gali būti išdėstyti bet kurioje vietoje tarp kairesniame stulpelyje surašytų operatorių. Pavyzdžiui, operatorius DATA gali būti bet kurioje vietoje tarp vykdomųjų operatorių arba operatorių-funkcijų apibrėžimo operatorių, tačiau būtinai aukščiau operatoriaus END ir žemiau, tarkim, DIMENSION.

Lentelėje neparodyta komentarų eilučių vieta programoje. Komentarai gali būti rašomi bet kurioje programos vietoje, tačiau būtinai aukščiau operatoriaus END.

3.5. FORTRANO VARDAI

Vardai naudojami kintamiesiems, programoms ir kitiems objektams įvardyti. Vardo sudarymo taisyklės tokios:

- Vardo pirmasis simbolis turi būti raidė (didžioji ar mažoji - nesvarbu), o tarp kitų simbolių gali būti ir skaičiai. FORTRANas 90 leidžia pirmuoju vardo simboliu naudoti dolerio ženklą $ (laikoma, kad tai yra "raidė", o jos padėtis abėcėlėje - po raidės Z), o vardo ne pirmuoju simboliu leistinas ir apatinis brūkšnelis _ .
- Vardo ilgis - ne daugiau 6 (FORTRANe 77) arba 31 (FORTRANe 90) simbolių.
- FORTRANas 77 intervalus varde ignoruoja. Pavyzdžiui, vardai V1 ir V___1 transliatoriui yra identiški. 90-as standartas neleidžia intervalų nei vardo viduje, nei operatoriaus standartinio žodžio viduje. Tai yra, matyt, vienintelis atvejis, kai 90-ojo FORTRANo transliatorius negali "praleisti" pagal tokias 77-ojo taisykles parašytos programos.

3.6. DUOMENYS. DUOMENŲ TIPAI

Visa tai, ką programa įveda į kompiuterio atmintį, apdoroja skaičiavimo metu ir išveda iš kompiuterio atminties į programuotojui suprantamą pavidalą po skaičiavimų - yra duomenys.

Duomuo, kuris programos vykdymo metu nekinta, yra konstanta. Konstanta išreiškiama simboliais, iš kurių ji susideda, arba vardu, kuriam tik vieną kartą operatoriumi PARAMETER priskirta reikšmė. Duomuo, kuris programos vykdymo metu kinta, yra kintamasis. Kintamasis visada išreiškiamas vardu. Vardas šiuo atveju yra simbolinis adresas tos atminties ląstelės, kurioje saugoma kintamojo reikšmė.

FORTRANas skirsto duomenis į kelis tipus. Pagal tipą duomeniui saugoti kompiuterio atmintyje skiriama vienokio ar kitokio ilgio ląstelė. Skiriasi ir saugomo duomens tikslumas, leistinos duomens dydžio ribos ir t.t. Yra tokie duomenų tipai:

- Sveikasis (INTEGER, INTEGER*1, INTEGER*2, INTEGER*4).

- Realusis (REAL, REAL*4).
- Realusis dvigubojo tikslumo (DOUBLE PRECISION, REAL*8, REAL*16).
- Kompleksinis (COMPLEX, COMPLEX*8).
- Kompleksinis dvigubojo tikslumo (DOUBLE COMPLEX, COMPLEX*16).
- Loginis (LOGICAL, LOGICAL*1, LOGICAL*2, LOGICAL*4).
- Tekstinis (CHARACTER, CHARACTER *n; $1 \le n \le 32767$).
- Išvestinis (žr. 8-ąjį skyrių).

Čia šalia duomens tipo po žvaigždutės nurodytas skaičius reiškia ląstelės, kurioje saugomas duomuo, ilgį baitais. Kaip matyti, tekstinio duomens ilgį pasirenka pats programuotojas. Jei ląstelės ilgis šalia tipo nenurodytas, bus automatiškai suteikiamas toks ilgis baitais:

INTEGER	- 4 (FORTRANe77) ir 2 (FORTRANe 90)
REAL	- 4
DOUBLE PRECISION	- 8
COMPLEX	- 8
DOUBLE COMPLEX	- 16
LOGICAL	- 4 (FORTRANe77) ir 2 (FORTRANe 90)
CHARACTER	- 1

Knygelėje dažnai naudosim sutrumpintus tipų žymėjimus - vardo pirmąsias raides, o kartais dar ir ilgį baitais: I, R, DP, C, DC, L, CH; I*2, R*4, DC*16 ir t.t.

Sveikieji, realieji ir kompleksiniai duomenys dar vadinami aritmetinio tipo duomenimis, o visi kiti duomenys - nearitmetiniais duomenimis.

Reikiamą tipą konkrečiam duomeniui suteikia programuotojas nevykdomaisiais duomenų tipo apibrėžimo operatoriais.

Visi šie duomenų tipai, išskyrus paskutinįjį, yra *vidiniai* FORTRANo duomenų tipai. Jų charakteristikos yra žinomos iš anksto. Be šių vidinių duomenų tipų, FORTRANe 90 galimi ir *išvestiniai* duomenų tipai. Išvestiniai duomenys yra tam tikra paties programuotojo nustatoma vidinių duomenų kombinacija. Šie duomenys aprašyti 8-ajame skyriuje. Ten pat pateikiama informacija ir apie FORTRANo 90 palaikomas *duomenų tipų rūšis*.

3.7. KONSTANTOS

Šiame skyriuje išvardysime visus kalbos keliamus apribojimus konstantoms, jų tikslumą bei pateiksim konstantų pavyzdžius. Kol kas nagrinėsime tik tokias konstantas,

kurias ”pažįsta” 77-asis standartas, o papildomas 90-ojo standarto galimybes aprašysim 8-ajame skyriuje.

Sveikosios konstantos

Konstantą sudaro tik ženklai + arba - ir skaitmenys. Pliuso ženklo galima nerašyti. Konstantos kitimo ribos priklauso nuo ląstelės ilgio ir 16 bitų kompiuteriui yra:

INTEGER*1	-128 ≤ ≤ 127
INTEGER*2	-32 768 ≤ ≤ 32 767
INTEGER*4	-2 147 483 648 ≤ ≤ 2 147 483 647

Pavyzdžiai: +123 , -123 , 123 .

Realiosios konstantos

Realioji konstanta užima 4 baitų ilgio atminties ląstelę ir dažnai yra tik norimo realiojo skaičiaus aproksimacija, nes jos tikslumas - tik 6-7 reikšminiai skaitmenys. Galima užrašyti konstantoje ir daugiau reikšminių skaitmenų, tačiau atmintyje bus saugomi tik pirmieji 6 arba 7 skaitmenys. Konstantų kitimo sritis yra neigiami skaičiai nuo -3.4028235E+38 iki -1.1754944E-38 , skaičius 0 ir teigiami skaičiai nuo 1.1754944E-38 iki 3.4028235E+38. Kaip matyti iš šių konstantų pavidalo, jos gali būti užrašytos ne tik mums įprasta, bet ir *eksponentine* forma.

Konstantos sintaksė yra:

[*ženklas*][*sveikoji dalis*][*.trupmeninė dalis*][E*laipsnio rodiklis*]

Konstantoje gali būti praleista sveikoji ar trupmeninė dalis, tačiau tik ne vienu metu: bent viena dalis privalo likti. Raidė E bei po jos einantis *laipsnio rodiklis* (jei reikia - su ženklu) reiškia, kad skaičių reikia padauginti iš 10 laipsniu *laipsnio rodiklis*.

Pavyzdžiai: visos konstantos +1.23 , 1.23 , 1.2300 , +1.23E0 , .01230E2 , .01230E+2 , 123.E-2 , 123E-2 , 1230E-3 reiškia vieną ir tą patį skaičių.

Realiosios dvigubojo tikslumo konstantos

Jos užima 8 baitų atminties ląstelę, o jų tikslumas - tarp 15 ir 16 reikšminių skaitmenų. Kitimo sritis yra neigiami skaičiai nuo -1.797693134862316D+308 iki -2.225073858507201D-308, skaičius 0 ir analogiški neigiamiems teigiami skaičiai. Konstantos sintaksė yra lygiai tokia, kaip ir realiosios konstantos, tik vietoje raidės E rašoma raidė D konstantoje yra privaloma.

Pavyzdžiai: dvigubojo tikslumo skaičius 0.123 FORTRANe gali būti rašomas viena iš alternatyvių formų 0.123D0 , +.123D0 , 123.D-3 , 123D-3 , 0.000123D+3 .

Kompleksinės konstantos ir ***kompleksinės dvigubojo tikslumo konstantos***

Tai pora dviejų realiųjų arba pora dviejų realiųjų dvigubo tikslumo skaičių, todėl užimamos ląstelės ilgis yra 8 arba 16 baitų. Konstantų sintaksė yra:

[*ženklas*] (*realioji skaičiaus dalis* , *menamoji skaičiaus dalis*)

Pavyzdžiai: (12, -3.4) , -(-12, 3.4) reiškia tą pat kompleksinį skaičių 12 - 3.4i viengubojo tikslumo režime, o (12D0, -3.4D0) - tą pat kompleksinį skaičių dvigubuoju tikslumu.

Loginės konstantos

Koks ilgis bebūtų skirtas ląstelei - 1, 2 ar 4 baitai - konstanta gali įgyti tik dvi logines reikšmes: “tiesa” - .TRUE. arba “melas” - .FALSE.

Tekstinės konstantos

Jos susideda iš bet kokių simbolių. Vienam simboliui skiriamas 1 baitas atminties. Kaip buvo minėta anksčiau, tekstinio duomens ląstelė gali būti 1 ÷ 32767 baitų ilgio. Reikiamas ląstelės ilgis privalo būti nurodytas operatoriumi CHARACTER (žr. tipo apibrėžimo operatorius). Konstantą sudaro simbolis(-iai), apimtas iš abiejų pusių apostrofais ‘ arba kabutėmis “ . Jei norima konstantoje turėti šiuos simbolius, juos reikia dubliuoti (ši pora bus skaičiuojama kaip vienas simbolis). Konstantos simboliai yra ir intervalai.

Reikėtų būti atidiems keliant tekstinę konstantą į kitą eilutę: jei tekstinę konstantą perkeliamojoje eilutėje nutrauksime ne ties 72-aja pozicija, o tąsos eilutę pradėsime ne 7-aja pozicija iškart po tąsos simbolio, tai į tekstinę konstantą kartu įterpsime intervalus. Tokiu atveju paprasčiau tekstinę konstantą padalyti į kelias trumpesnes, telpančias vienoje eilutėje, konstantas.

Pavyzdžiai:

Konstanta	*Reikšmė*
‘Tekstas’	Tekstas
“Intervalai yra įskaičiuojami”	Intervalai yra įskaičiuojami
“ “ “Dvigubos” ” kabutės”	“Dvigubos” kabutės
“““““	“
“ ‘ ” “	‘

3.8. KINTAMIEJI

Kintamieji programos tekste atspindimi tik vardu - simboliniu adresu ląstelės, kurioje saugoma kintamojo reikšmė. Kintamojo tipą nustato programuotojas nevykdomaisiais tipo apibrėžimo operatoriais. Kiekvienam programos tekste paminėtam kintamajam kompiliatorius automatiškai skiria po vieną atminties ląstelę, kurios ilgis priklauso nuo kintamojo tipo.

Galima kintamųjų tipų programoje ir nenurodyti - tada jiems pagal FORTRANe plačiai taikomą nutylėjimo principą bus suteikti tam tikri tipai automatiškai: jei kintamojo vardo pirmasis simbolis yra I, J, K, L, M arba N (arba i, j, k, l, m, n), tai toks vardas, o kartu ir duomuo, laikomas sveikuoju (I*2); visi kiti vardai (duomenys) laikomi realiaisiais (R*4). Kitokių tipų nutylėjimo principas nenustato, todėl bent kiek sudėtingesnėms programoms jo aiškiai nepakanka.

3.9. MASYVAI

Paprasti kintamieji, apie kuriuos kalbėjom anksčiau, kiekvienas turi unikalius vardus ir yra saugomi vienoje atminties ląstelėje. Tačiau jei duomenis galėtume saugoti tik tokia paprasčiausia forma, daugelio uždavinių (pavyzdžiui, iš tiesinės algebros srities) sprendimas taptų labai komplikuotas arba išvis neįmanomas. Tokiems uždaviniams būtina masyvo sąvoka.

Kelių duomenų seka, turinti tik vieną vardą, yra masyvas. Masyvai yra vienmačiai, dvimačiai, ..., *n*-mačiai. Vienmačio masyvo atitikmuo matematikoje - vektorius, dvimačio - matrica, trimačio - skaičių kubas. *n* maksimali reikšmė priklauso nuo programai prieinamos kompiuterio atminties apimties (FORTRANe 77 - 7).

Į konkretų masyvo elementą kreipiamasi panašiai kaip matematikoje, pavyzdžiui, į matricos elementą: nurodant masyvo vardą ir skliaustuose indeksų reikšmes. Indeksai vienas nuo kito skiriami kableliais. Indekso reikšmė pateikiama sveikaja konstanta ar sveikojo tipo aritmetiniu reiškiniu (žr. skyrių žemiau).

Pavyzdžiai: X(10) , Y(10,5) , Z(1,J+5)

Programos viršuje tarp nevykdomųjų operatorių būtina nurodyti, kokį kiekį atminties ląstelių reikia skirti masyvo saugojimui. Šias funkcijas atkieka operatorius DIMENSION arba tipo apibrėžimo operatoriai, arba operatorius COMMON (žr. atitinkamus skyrius). Šiuose operatoriuose prie masyvo vardo nurodomos maksimalios kiekvieno indekso reikšmės, paprastai - sveikųjų konstantų pavidalu.

Kompiuterio atmintis turi tik vieną dimensiją, todėl kelių matavimų bebūtų masyvas, jis atmintyje saugomas kaip vienmatė duomenų seka: *n+1* elementas seka po

n elemento. Jei masyvas keliamatis, tai kompiuterio atmintyje masyvo elementai išdėstomi taip, kad pirmiausia kinta kairysis masyvo indeksas, o galiausiai - dešinysis.

Pavyzdys: dvimačio masyvo M, turinčio 3 eilutes ir 4 stulpelius, elementai kompiuterio atminty būtų išdėstyti tokia tvarka:

M(1,1) M(2,1) M(3,1) M(1,2) M(2,2) M(3,2) M(1,3) M(2,3) M(3,3) M(1,4) M(2,4) M(3,4)

- tai yra stulpeliais.

3.10. REIŠKINIAI

Reiškinys - tai formulė kokiai nors reikšmei skaičiuoti. Reiškinį sudaro *operandai* ir *veiksmų operatoriai* (nepainiokim jų su FORTRANo operatoriais!). Operatorius nurodo, kokią operaciją atlikti su operandais. Pavyzdžiui, reiškinyje *a*/*b* *a* ir *b* yra operandai - kintamieji, o / - aritmetinis dalybos operatorius. Yra 4 tipų reiškiniai:

- aritmetiniai (jų rezultatas - sveikasis, realusis arba kompleksinis duomuo)
- santykiniai (loginis)
- loginiai (loginis)
- tekstiniai (tekstinis)

Bet koks reiškinys nėra savarankiškas FORTRANo objektas; jis turi būti kokio nors FORTRANo operatoriaus dalis. Pavyzdžiui, prieskyros operatoriuje *c* = *a*/*b* aritmetinio reiškinio *a*/*b* reikšmė priskiriama kintamajam *c*. Tik panašiu būdu galima FORTRANą "priversti" skaičiuoti vienokį ar kitokį reiškinį.

Aritmetiniai reiškiniai

Aritmetiniuose reiškiniuose operandai - aritmetinės konstantos, kintamieji, masyvų elementai, masyvai, funkcijos, išvestiniai duomenys - sujungiami aritmetinių veiksmų operatoriais. FORTRANas žino 5 aritmetinius veiksmus: kėlimas laipsniu laikomas atskiru veiksmu. Be šių operatorių, aritmetiniuose reiškiniuose galima naudoti skliaustelius. Apskliausti veiksmai atliekami pirmiausia. Aritmetinių veiksmų operatorių atlikimo tvarka ir jų ženklai FORTRANe yra:

1. Kėlimas laipsniu ** (kėlimas laipsniu FORTRANe yra atskiras veiksmas)
2. Daugyba * ir dalyba /
3. Sudėtis + ir atimtis -

Taigi pirmiausia atliekami veiksmai skliaustuose, po to - kėlimo laipsniu veiksmai, po to - vienodo vyresnumo daugybos ir dalybos veiksmai ir pagaliau - vienodo vyresnumo sudėties ir atimties veiksmai. Jei yra keli vienodo vyresnumo veiksmai vienas po kito, tai jie atliekami iš kairės į dešinę.

Aritmetinio reiškinio operandai gali būti vienodo arba skirtingo tipo. Jei operandai vieno tipo, tai ir reiškinio rezultatas gaunamas to pat tipo. Jei operandai skirtingo tipo, tai FORTRANas pirmiausia "padaro" abu operandus vieno tipo ir tik po to atlieka veiksmus bei gauna šio tipo rezultatą. Vienodinant operandų tipus, visuomet mažiau bendro tipo operandas pervedamas į bendresnįjį tipą. Tipų bendrumo seka yra: I*1, I*2, I*4, R, DP, C, DC. Taigi skaičiuojant reiškinį $dc + i1$, kur dc yra DC tipo operandas, $i1$ - I*1 tipo operandas, pirmiausia operandas $i1$ būtų pervestas į dvigubojo tikslumo kompleksinį tipą, po to - atliktas sudėties veiksmas ir rezultatas būtų gautas taip pat DC tipo. Aritmetiniai operandai pervedami iš vieno tipo į kitą pagal lentelėje 3.2 nurodytas taisykles.

3.2 lentelė. Aritmetinių operandų pervedimo iš vieno tipo į kitą taisyklės

Operando tipas	***Pervedimo į artimiausią bendresnį tipą taisyklė***	***Pervedimo į artimiausią žemesnį tipą taisyklė***
DC (C*16)	-	Realioji ir menamoji dalys iš R*8 pervedamos į R*4 (žr. žemiau)
C (C*8)	Realioji ir menamoji dalys iš R*4 pervedamos į R*8 (žr. žemiau)	Atmetama menamoji dalis. Realioji dalis pervedama iš R*4 į R*8 (žr. žemiau)
DP (R*8)	Duomuo pervedamas iš R*8 į R*4 (žr. žemiau) ir pridedama nulinė R*4 menamoji dalis	Duomuo apvalinamas iki 6-7 reikšminių skaitmenų
R (R*4)	Duomuo papildomas nuliais iki 15-16 reikšminių skaitmenų	Atmetama trupmeninė dalis

I*4	Pridedama nulinė trupmeninė dalis	Duomuo pervedamas į I*2 formatą*

3.2 lentelės tęsinys

I*2	Duomuo pervedamas į I*4 formatą	Duomuo pervedamas į I*1 formatą*
I*1	Duomuo pervedamas į I*2 formatą	-

* Jei pervedamojo duomens dydis išeina už šiam ląstelės ilgiui nustatytų ribų, keičiamas duomens ženklas ir naudojama mažiausiai reikšminga duomens dalis.

Sakėme, kad operandais gali būti *funkcijos*. Jas nagrinėsime 7-ajame skyriuje kartu su kitomis paprogramėmis. Ten bus paaiškinta, kaip parašyti programą bet kokiai programuotojui reikiamai funkcijai. Daugelio funkcijų programų rašyti nereikia, nes dažniau naudojamos funkcijos yra užprogramuotos ir tos programos sudaro FORTRANo kalbos dalį. Tokių *vidinių, standartinių funkcijų* (INTRINSIC FUNCTION) yra daugiau nei šimtas. Kadangi tokios funkcijos reikalingos ir nesudėtingose programose, dalį jų paminėsime šiame skyriuje (viso sąrašo teks ieškoti kalbos aprašymo dokumentuose).

3.3 lentelė. Matematinis funkcijos žymėjimas, funkcijos žymėjimas FORTRANe bei jos tipas

Funkcija	***I***	***R***	***DP***	***C***	***DC***
$\sqrt{x}$	-	SQRT(x)	DSQRT(x)	CSQRT(x)	CDSQRT(x)
$\lvert x \rvert$	IABS(x)	ABS(x)	DABS(x)	CABS(x)	CDABS(x)
sin x	-	SIN(x)	DSIN(x)	CSIN(x)	CDSIN(x)
cos x	-	COS(x)	DCOS(x)	CCOS(x)	CDCOS(x)
tg x	-	TAN(x)	DTAN(x)	-	-
Arcsin x	-	ASIN(x)	DASIN(x)	-	-
Arccos x	-	ACOS(x)	DACOS(x)	-	-
Arctg x	-	ATAN(x)	DATAN(x)	-	-
e^x	-	EXP(x)	DEXP(x)	CEXP(x)	CDEXP(x)
ln x	-	LOG(x)	DLOG(x)	CLOG(x)	CDLOG(x)

lg x	-	LOG10(x)	DLOG10(x)	-	-

Pastabos:

1. Šių funkcijų argumentas gali būti aritmetinis reiškinys (arba jo dalis - kita funkcija, masyvo elementas, kintamasis, konstanta).
2. Kokio tipo yra funkcija, tokio tipo turi būti ir jos argumentas.
3. Trigonometrinių funkcijų argumentai skaičiuojami radianais.

Be šių matematinių funkcijų, FORTRANe yra standartinės funkcijos duomenims pervesti iš vieno tipo į kitą, apvalinti, skirtumui tarp dviejų duomenų gauti, duomens ženklui gauti, didžiausiam ir mažiausiam iš kelių duomenų rasti; kai kurių veiksmų su kompleksiniais ir tekstiniais duomenimis funkcijos.

Bendriniai standartinių funkcijų vardai (generic names). Praktiškai į visas 3.3 lentelėje išvardytas funkcijas galima kreiptis ir vadinamaisiais bendriniais vardais. Pavyzdžiui, modulio funkcijos bendrinis vardas yra ABS(x). Tokios funkcijos rezultato tipas priklauso nuo teikiamo argumento tipo: jei argumentas x bus sveikojo tipo, tai ir funkcijos rezultatas bus gautas sveikasis, ir t.t. Viso funkcijų bendriniais vardais sąrašo reikėtų ieškoti kompiliatoriaus dokumentuose.

Plačiau apie bendrinius funkcijų vardus skaitykite 8.2 skyriuje.

Rašant aritmetinius reiškinius tenka laikytis kelių taisyklių:

- Negalima rašyti paeiliui dviejų veiksmų ženklų (kėlimo laipsniu žymėjimas - ** - laikomas vienu ženklu).
- Negalima praleisti daugybos ženklo.
- Dalybos iš nulio aritmetinė operacija uždrausta.
- Nulinio operando kėlimas neigiamu laipsnio rodikliu uždraustas.

Pavyzdžiai: pateiksim kelis tokių aritmetinių reiškinių pavyzdžius, kurių programavimas galėtų kelti keblumų.

Reiškinys ***Reiškinys FORTRANe*** ***Pastabos***

$\frac{a\ b}{c}$	(a * b) / c arba a * b / c arba a / c * b	Daugybos ir dalybos veiksmai vienodo prioriteto.
$\frac{a+b^2\ c}{c-d}$	(a + b**2 * c) / (c - d)	Keliama laipsniu anksčiau nei dauginama, o sudėtis ir atimtis apskliaustos, kad būtų atliktos anksčiau už dalybą.
sin 2x	SIN(2.*x)	
$\cos^2 x^3$	COS(x**3) **2	
$\sqrt[3]{x}$	x**(1./3.) arba tiesiog - x**0.3333333	Skliaustai nurodo pirmiausia atlikti dalybą ir tik po to - kėlimą laipsniu. Bent viena iš konstantų “1” ar “3” turi būti realiojo tipo, nes tik taip gausim realųjį rezultatą. Jei abi konstantos būtų sveikojo tipo, tai ir dalybos rezultatas būtų sveikasis, t.y. trupmeninė dalis būtų atmesta ir liktų reikšmė “0” .
$\ln(x+\sqrt{1-x^2})$	LOG(x + SQRT(1-x*x))	

Santykiniai reiškiniai

Tai - patys paprasčiausi loginiai reiškiniai, gaunami sujungus du aritmetinius reiškinius loginiu santykio operatoriumi. Jų rezultatas yra loginis duomuo (LOGICAL) ir turi tik dvi reikšmes: “tiesa” - .TRUE. ir “melas” - .FALSE. . Visi santykio operatoriai yra mažesnio vyresnumo nei aritmetiniai operatoriai. Santykio operatoriai yra:

<	.LT.	(FORTRANe 90 yra ir sinonimas - ženklas <)
≤	.LE.	(<=)
>	.GT.	(>)
≥	.GE.	(>=)
=	.EQ.	(==)
≠	.NE.	(/=)

Su C ir DC tipų operandais leistini tik santykio operatoriai .EQ. ir .NE. .

Loginiai reiškiniai

Jų operandai gali būti tik loginio tipo konstantos, kintamieji bei santykiniai reiškiniai. Yra tokios loginės operacijos (išdėstytos pagal vyresnumą):

1. Neigimas .NOT.
2. Loginė daugyba .AND.
3. Loginė sudėtis .OR.
4. Lygiareikšmiškumas .EQV.
   ir nelygiareikšmiškumas .NEQV.

Visų šių loginių operacijų vyresnumas mažesnis už aritmetinių operacijų bei loginių santykių vyresnumą.

Visos loginės operacijos, išskyrus loginį neigimą, turi po du operandus. Loginis neigimas paneigia operando reikšmę: jei *a* reikšmė buvo .TRUE., tai .NOT. *a* reikšmė taps .FALSE. . Kitų loginių operacijų prasmė bus aiški iš 3.4 lentelės, kurioje pateiksim kiekvienos operacijos rezultatą pagal operandų reikšmes.

3.4 lentelė. Loginių operacijų reikšmės

Operandų a ir b reikšmės	***a .AND. b***	***a .OR. b***	***a .EQV. b***	***a .NEQV. b***
abi .TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.
viena .TRUE. , kita .FALSE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
abi .FALSE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.

Pavyzdys: tarkim, skaičių ašyje išskirti keli intervalai:

sritis ① - x < x1;
sritis ② - x2 < x < x3;
sritis ③ - x > x4 .

Parašysime loginius reiškinius, kurie įgyja reikšmę “tiesa”, kai argumentas *x* yra:

1) srityje ① - x .LT. x1
2) srityje ② - x .GT. x2 .AND. x .LT.x3
3) srityse ① arba ③ - x .LT. x1 .OR. x .GT. x4
4) srityse ①, ② arba ③-
   x .LT. x1 .OR. x .GT. x2 .AND. x .LT. x3 .OR. x .GT. x4

Skliaustai šiuose loginiuose reiškiniuose, turint omeny loginių santykių ir loginių operacijų vyresnumą, nebūtini. Tačiau jei kyla kokių nors abejonių, skliaustus galim įrašyti - jie nieko nepagadins. Pavyzdžiui, reiškinys (x .GT. x2) .AND. (x .LT.x3) identiškas jau užrašytam antruoju punktu reiškiniui.

Tekstiniai reiškiniai

Jų operandais gali būti tik tekstinio tipo konstantos, kintamieji, funkcijos bei *tekstinės subeilutės* (character substrings). Yra vienintelis tekstinių reiškinių operatorius - konkatenacija (FORTRANe žymimas //). Šis operatorius sujungia du tekstinius dydžius į vieną.

Pavyzdys. Tegu dvi tekstinės konstantos yra ‘AB␣c’ ir ‘DE␣␣’. Reiškinio ‘AB␣c’//’DE␣␣’ rezultatas bus viena tekstinė konstanta ‘AB␣cDE␣␣’ .

***** Pakeliui šiame skyrelyje paaiškinsime ir *tekstinės subeilutės* sąvoką. Tai - nuosekli tekstinio tipo kintamojo dalis. Jos sintaksė yra:

tekstinis kintamasis([*ip*] : [*ig*]),

kur *ip* yra sveiko tipo reiškinys (arba kintamasis, konstanta), nurodantis pirmąjį į subeilutę “paimamą” *kintamojo* simbolį, o *ig* - analogiškai galinį. *subeilutės* ilgis tuo būdu yra *ig - ip + 1* .Jei *ip* nenurodytas, pagal nutylėjimą imama jo reikšmė 1, o jei nenurodytas *ig* , tai *subeilutė* baigiama paskutiniu kintamojo simboliu. Jei *tekstinio kintamojo* ilgis yra *ilgis*, tai turi būti išlaikytos sąlygos $1 <= ip <= ig <= ilgis$.

Pavyzdys. Tegu 14 baitų ilgio tekstinio kintamojo vardas reikšmė yra “Jonas␣Jonaitis”. Tada žemiau parašytų subeilučių reikšmės bus:

vardas(:5) “Jonas”

vardas(7:)	“Jonaitis”
vardas(6:8)	“ Jo”
vardas(:)	“Jonas Jonaitis”
vardas(0:)	neteisinga subeilutė
vardas(:15)	neteisinga subeilutė
vardas(2:1)	neteisinga subeilutė

4. OPERATORIAI

Šiame skyriuje aprašysime didžiąją FORTRANo operatorių dalį. Išdėstysime juos ne alfabetine tvarka, ne skirstydami juos į vykdomuosius ar nevykdomuosius ir pan., o tokia tvarka, kokia yra patogiausia besimokant programavimo. Dėl šios priežasties dalis operatorių perkelta į skyrių, aprašantį informacijos mainų sistemą (READ, WRITE, ...), dalis - į skyrių, skirtą paprogramiams (ENTRY, INTRINSIC, EXTERNAL, ...) ir t.t. Visi operatoriai sąlyginai sugrupuoti į kelias grupes.

Operatoriai aprašomi tokiu būdu: nurodoma, ar operatorius yra vykdomasis ar nevykdomasis; pateikiamas operatoriaus bendrasis pavidalas; paaiškinami operatoriaus atliekami veiksmai; operatoriaus veikimas iliustruojamas pavyzdžiais.

4.1. PROGRAMOS PRADŽIOS IR PABAIGOS OPERATORIAI. TUŠČIASIS OPERATORIUS

Nevykdomasis programos pradžios operatorius, kurio bendrasis pavidalas yra

PROGRAM programos vardas

įvardija programą. Šis vardas neturi sutapti su jokiu kitu programos objekto vardu. Šis operatorius programoje nebūtinas; jei jo nėra, programą įvardija operacinė sistema. Jei programoje operatorius yra, jis turi būti pirmasis operatorius.

Programos pabaigos operatorius

END

nutraukia programos darbą ir grąžina valdymą operacinei sistemai. Privalo būti paskutinis programos operatorius. * Paprogramyje parašytas operatorius nutraukia paprogramės darbą; valdymas grąžinamas kviečiančiajai programai.

---

* Vykdomasis operatorius

STOP [*tekstinė arba sveikoji konstanta*]

nutraukia programos darbą ir perduoda valdymą operatoriui END. Jei šalia operatoriaus buvo nurodytas pranešimas, jis parodomas kompiuterio ekrane suveikus operatoriui. Jei šalia operatoriaus nenurodytas joks pranešimas, ekrane parodomas standartinis pranešimas apie programos darbo nutraukimą:

STOP - Program terminated.

Vykdomasis operatorius

žymė CONTINUE

neatlieka jokių veiksmų. Dažniausiai naudojamas *žymei* "nešti".

4.2. DUOMENŲ TIPO APIBRĖŽIMO OPERATORIAI

Tai - nevykdomieji operatoriai. Paprasčiausiose FORTRANo programose jie nebūtini. Priminsim, kad šiuo atveju duomenys suskirstomi į INTEGER ir REAL tipus pagal nutylėjimo principą: jei duomens vardas prasideda raide I, J, K, L, M, N ar i, j, k, l, m, n, tai duomuo laikomas sveikuoju duomeniu; antraip - realiuoju.

Beveik visų tipo apibrėžimo operatorių sintaksė yra:

*tipa*s[**ilgis1*] *duomens vardas*[**ilgis2*] [(*dimensijos*)] [/*reikšmės*/] ...

Šie operatoriai duomeniui, apibrėžiamam *duomens vardu*, priskiria *tipą*, šalia kurio dar galima nurodyti duomens ląstelės *ilgį1* baitais. Jei *ilgis1* nenurodytas, automatiškai suteikiamas standartinis šio tipo ląstelės ilgis (žr. 3.6 skyrių). Kaip matyti, operatoriai, be *tipo*[**ilgis1*] deklaravimo masyvo vardui, dar gali ir aprašyti tą masyvą: skliausteliuose tada nurodomos maksimalios indeksų(-o) reikšmės (žr. 4.3 skyrių). Pagaliau operatoriai gali kartu ir suteikti norimas *reikšmes* deklaruojamam duomeniui. *Reikšmės*, jei jų keletas, viena nuo kitos skiriamos kableliais. Jei norima užduoti *n* vienodų reikšmių, patogu naudoti tam numatytą trumpą konstrukciją *n*reikšmė*.

Tokių elementų sąrašas gali būti kartojamas. Tada, jei *ilgis2* nenurodytas, visi sąrašo vardai gauna *ilgį1*. Šalia bet kurio sąrašo vardo nurodant *ilgį2*, šiam priskiriamas būtent šis *ilgis2*, o ne *ilgis1*.

Tipas[**ilgis1*] gali būti (abėcėlės tvarka):

CHARACTER, CHARACTER**n*
COMPLEX, COMPLEX*8, COMPLEX*16
DOUBLE COMPLEX
DOUBLE PRECISION
INTEGER, INTEGER*1, INTEGER*2, INTEGER*4
LOGICAL, LOGICAL*1, LOGICAL*2, LOGICAL*4
REAL, REAL*4, REAL*8

Atidus skaitytojas, matyt, pastebėjo, kad dalis čia išvardintų tipų yra sinonimai: C ir C*8, C*16 ir DC, DP ir R*8, I ir I*2, L ir L*2, R ir R*4.

Ilgis2 privalo būti leistinas deklaruotam duomens tipui. Pavyzdžiui, jei duomeniui deklaruotas tipas REAL, tai *ilgis2* gali būti tik 4 arba 8.

Pavyzdžiai:

INTEGER i, x, z	Kintamieji i, x, z programoje bus laikomi sveikojo tipo kintamaisiais; jų ląstelės ilgis - 2 baitai.
INTEGER i, x*4, z	i, x, z bus sveikieji kintamieji; x ląstelės ilgis - 4 baitai, kitų dviejų kintamųjų - 2 baitai.
INTEGER i/10/, x*4(5,10)/50*0/, z	Kintamieji i ir z bus I*2 tipo; be to, pirmajam dar priskiriama reikšmė 10. x bus I*4 tipo dvimatis masyvas, turintis 5 eilutes ir 10 stulpelių; visi masyvo elementai įgis reikšmes 0.

Pats bendriausias tipo apibrėžimo operatorius yra IMPLICIT, suteikiantis tipą ne konkrečiam vardui, kaip aukščiau paaiškinti operatoriai, o vardo pirmajai *raidei*. *Raidės* gali būti nurodomos sąrašais arba diapazonais pagal lotyniškąją abėcėlę. Diapazonas apibrėžiamas pirmąja ir paskutiniąja diapazono raidėmis, o tarp jų įrašomas brūkšnelis. Bendrasis operatoriaus pavidalas yra:

IMPLICIT *tipas*[**ilgis*] (*raidės*) ...

Čia *tipas*[**ilgis*] turi lygiai tą pat prasmę kaip aukščiau paaiškintuose operatoriuose.

Operatorius IMPLICIT yra, aišku, nevykdomasis operatorius ir turi būti aukščiau bet kurio vykdomojo operatoriaus. Tipo apibrėžimo operatoriai konkrečiam vardui gali suteikti kitokį tipą nei numato operatorius IMPLICIT.

Vienoje programoje gali būti ir keli operatoriai IMPLICIT, tačiau konkreti *raidė* gali būti apibrėžta tik viename iš jų.

Pavyzdžiai:

IMPLICIT REAL*8(a-h, o-z), INTEGER*4(i,j,k), LOGICAL*1(l)

Visi duomenys, kurių vardai prasideda raidėmis a, b, c, ... , h ir o, p, r, ... , z bus R*8 tipo; visi duomenys, kurių vardai prasideda raidėmis i, j, k - I*4; o raide l -L*1 tipo. Iš visos abėcėlės liko neapibrėžtos tik raidės m, n, kurios pagal nutylėjimo principą suteiks vardui I*2 tipą.

IMPLICIT INTEGER*4(a,b)
REAL*8 a, a1(10)

Pirmasis operatorius suteikia visiems raidėmis a, b prasidedantiems vardams I*4 tipą, tačiau antrasis operatorius, "paneigdamas" IMPLICIT įtaką, konkretiems vardams a ir a1 suteikia R*8 tipą.

FORTRANe 90 įvesta ir nauja operatoriaus IMPLICIT forma

IMPLICIT NONE

Programoje, esant tokiam operatoriui, kitų IMPLICITų nebegali būti. Toks operatorius reiškia, kad programoje visi programuotojo sugalvoti duomenų vardai privalo būti apibrėžti šio skyriaus pradžioje minėtais tipo apibrėžimo operatoriais; principas pagal nutylėjimą išjungiamas. Programoje suradęs neapibrėžtą vardą, transliatorius deklaruos klaidą.

4.3. MASYVŲ APRAŠYMO OPERATORIUS

Tai - nevykdomasis operatorius DIMENSION, atkeliavęs iš ankstesnių FORTRANo dialektų. Kaip ką tik sužinojom, masyvai gali būti aprašomi tiesiog tipo apibrėžimo operatoriuose. Operatoriaus bendrasis pavidalas yra:

DIMENSION *masyvo vardas* (*dimensijos*) ...

Operatorius paskiria masyvui saugoti nurodytą ląstelių kiekį. Maksimalus masyvo matavimų kiekis priklauso nuo prieinamos kompiuterio atminties (FORTRANe 77 - 7).

Dimensijos nurodomos arba maksimalia atitinkamo indekso reikšme (pavyzdžiui, x(5,10) - dvimačiam masyvui skiriamos 50 ląstelių atminties), arba kiekvieno matavimo apatiniąja ir viršutiniąja ribomis, skiriamomis dvitaškiu. Pastarasis masyvo aprašymas būtų patogus, tarkim, tokiu atveju. Masyvas expr skirtas eksperimentų nuo 20-ojo iki 100-ojo duomenims saugoti. Tada patogu aprašyti masyvą tokioms dimensijoms: expr(20:100). Dabar, pavyzdžiui, 77-ojo bandymo duomenis gausim kreipdamiesi į 77-ąjį masyvo elementą. Jei masyvas būtų aprašytas įprastu būdu, tai atrinkdami norimo bandymo rezultatus turėtume perskaičiuoti indeksą.

Dimensijų ribos gali būti neigiamos, nulinės arba teigiamos aritmetinės konstantos (jei jos ne sveikosios - automatiškai paverčiamos sveikojo tipo konstantomis). Viršutinė riba turi būti ne mažesnė nei apatinė.

Pavyzdys:

DIMENSION x(10), y(2,-2:2) — Masyvui x skiriamos 10 ląstelių, dvimačiam masyvui y - 10 ląstelių (2 eilutės ir 5 stulpeliai, kurių indeksai -2, -1, 0, 1, 2).

4.4. PRADINIŲ REIKŠMIŲ SUTEIKIMO OPERATORIAI

Pradines reikšmes simbolinėms konstantoms, t.y. vardu išreiškiamoms konstantoms, suteikia nevykdomasis operatorius

PARAMETER (*konstantos vardas = konstantos reikšmė ...*) .

Simbolinės konstantos, tokiu būdu įgavusios kokią nors reikšmę, programoje keisti nebegalima. Bet koks bandymas pakeisti reikšmę bus laikomas sintaksės klaida.

Pageidautina, kad simbolinė konstanta ir ją atitinkanti reikšmė būtų vienodo tipo. Jei tipai skiriasi, reikšmės tipas automatiškai keičiamas į konstantos tipą.

Pavyzdys:

PARAMETER (pi = 3.14159, radian = 57.2958)

Jei programoje tenka dažnai kreiptis į π ar į laipsnių kiekį radiane, patogu konstantų reikšmes suteikti šiuo būdu: dabar jų reikšmių

net ir sąmoningai negalėtume pakeisti.

Kintamiesiems pradines reikšmes galima suteikti operatorium

DATA *kintamųjų sąrašas /kintamųjų reikšmių sąrašas/ ...*

Kaip matyti, operatorius yra ankstesnių FORTRANo dialektų reliktas, nes 90-asis dialektas leidžia šiuos veiksmus atlikti tipo apibrėžimo operatoriais. Vienas esminių DATA skirtumų tas, kad kintamųjų sąraše gali būti *neišreikštasis ciklas* (žr. *ciklus*). Šis operatorius, lygiai kaip ir tipo apibrėžimo operatoriai, leidžia pasikartojančias kintamųjų reikšmių sąrašo reikšmes trumpai rašyti forma *n*reikšmė*; jeigu reikia, automatiškai pakeičia reikšmės tipą - kad sutaptų su atitinkamo kintamojo tipu.

Operatoriaus vieta programoje - tarp vykdomųjų operatorių (žr. 3.1 lentelę).

***** Operatorių reikia atsargiai naudoti paprogramiuose, iškviečiamuose darbui kelis kartus: operatorius DATA faktiškai vykdomas kompiliavimo metu tik *vieną* kartą, todėl antrame ir paskesniuose paprogramių iškvietimuose jis bus ignoruojamas. Operatoriumi negalima suteikti reikšmių kintamiesiems, įtrauktiems į COMMONų sąrašus (žr. 7-ą skyrių).

Pavyzdžiai:

DATA pi, radian /3.14159, 57.2958/	Kintamiesiems (ne konstantoms !) pi ir radian kompiliavimo metu suteikiamos atitinkamos reikšmės.
REAL x(100) DATA x /100*0/	Visiems vienmačio masyvo x elementams suteikiama reikšmė 0. . Prieš reikšmės suteikimą ši iš sveikojo tipo automatiš- kai pervedama į realųjį.

4.5. PRIESKYROS OPERATORIUS

Tai pats paprasčiausias iš visų vykdomųjų operatorių - jo standartas yra tik lygybės ženklas:

kintamasis = reiškinys

Operatoriaus vykdymas: kairėje pusėje esančiam *kintamajam* priskiriama suskaičiuota dešinėje pusėje esančio *reiškinio* reikšmė. *Kintamasis* gali būti kintamasis,

masyvas, masyvo elementas arba išvestinis duomuo. *Reiškinys* gali būti reiškinys arba jo dalis - kintamasis ar tiesiog konstanta.

Jeigu *reiškinys* yra aritmetinis, tai ir *kintamasis* turi būti aritmetinis. Jei jų tipai skiriasi, tai aritmetinio *reiškinio* reikšmės tipas pervedamas į *kintamojo* tipą, o tik po to atliekamas priskyrimas.

Jeigu *reiškinys* yra loginis, tai ir *kintamasis* turi būti loginis. Loginio *kintamojo* ir loginio *reiškinio* reikšmės ilgiai baitais gali būti skirtingi; tai neturi įtakos prieskyros rezultatui.

Jeigu *reiškinys* yra tekstinis, tai ir *kintamasis* turi būti tekstinis. Tekstinio *kintamojo* ir tekstinio *reiškinio* reikšmės ilgiai baitais gali būti skirtingi: jei *kintamasis* ilgesnis už *reiškinį* - reiškinio reikšmė papildoma iš dešinės pusės intervalais; jei *kintamasis* trumpesnis už *reiškinį* - reiškinio reikšmės atliekami simboliai iš dešinės pusės ignoruojami.

Pavyzdžiai: panagrinėkim programos fragmentą

```
INTEGER  i1, i2
REAL  r
COMPLEX  c
LOGICAL  k
CHARACTER  t1*10, t2*3

i1 = 1
i1 = i1 + 2          Sveikojo kintamojo  i1  reikšmė bus  3

i2 = 2.999           Sveikojo kintamojo  i2  reikšmė bus 2 :
                     realioji reikšmė  2.999  pervedama į sveikąjį
                     tipą atmetant trupmeninę dalį, o tik po to
                     atliekamas priskyrimas

r = 2 / 3            Realiojo kintamojo  r  reikšmė bus  0. :
                     iš pradžių skaičiuojama sveikojo reiškinio 2/3
                     (nes abu jo operandai sveikojo tipo) reikšmė
                     - bus  0 ; reikšmė  0  pervedama į realųjį tipą
                     -  0., ir tik tada atliekamas priskyrimas


t1 = 'Tekstas'       Tekstinio kintamojo  t1  reikšmė bus
                     'Tekstas___' - iš dešinės pusės 3 intervalai;

t2 = 'Tekstas'       o  t2  -  'Tek' - atliekami dešinės pusės
```

	simboliai ignoruojami
c = (1.0, 2.0)	Kompleksinio kintamojo c reikšmė bus 1. + 2.i
i1 = 10 i2 = 11 k = i1 .GT. i2	Loginio kintamojo k reikšmė bus .FALSE.
t1 = r i1 = ‘Tekstas’ k = r / 2.	Visi trys priskyrimai klaidingi, kadangi *kintamojo* ir *reiškinio* prasmės tipai skirtingi

4.6. VALDYMO PERDAVIMO OPERATORIAI

Tai vykdomieji operatoriai, kuriais galima nutraukti nuoseklią operatorių vykdymo tvarką ir atiduoti valdymą, t.y. pradėti vykdyti reikiamą operatorių. Paprasčiausias jų - ***besąlyginio valdymo perdavimo operatorius***

GO TO *žymė*

Po šio operatoriaus bus vykdomas operatorius su žyme *žymė*. *Žymė* privalo būti parašyta prie vykdomojo operatoriaus.

Pavyzdys:

GO TO 10	Kitas bus vykdomas operatorius su žyme 10.

*** *Skaičiuojamasis valdymo perdavimo operatorius***:

GO TO (*žymių sąrašas*) *sveikasis aritmetinis reiškinys*

Žymių sąraše viena ir ta pati žymė gali būti įrašyta kelis kartus. *Reiškinys*, aišku, gali būti supaprastėjęs iki kintamojo ar konstantos. Operatorius vykdomas taip: skaičiuojama *reiškinio* sveikoji reikšmė r , o po to valdymas atiduodamas r-ajai žymei iš *žymių sąrašo*. Jei gauta r viršija žymių kiekį sąraše ar r < 1, operatorius nevykdo jokių veiksmų; paprasčiausiai valdymą gauna žemiau parašytas operatorius.

Pavyzdys:

```
kitas = 1
GO TO (10, 20) kitas          Valdymą gaus operatorius su žyme  10 .
```

**** Priskiriamasis valdymo perdavimo operatorius***

GO TO *kintamasis* [(*žymių sąrašas*)]

Čia *kintamasis* yra sveikasis kintamasis, kuriam programoje anksčiau priskirta vykdomojo operatoriaus žymė (žr. operatorių ASSIGN). Galima šalia parašyti *žymių sąrašą*, kuriame išvardijamos visos leistinos kintamajam įgyti žymės. Šis operatorius įtrauktas į pasenusių FORTRANo operatorių sąrašą. Matyt, kitame kalbos standarte jo nebeliks.

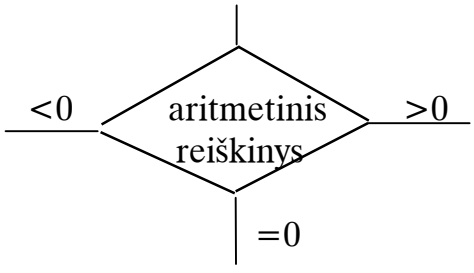
Pavyzdžiai:

```
ASSIGN  10  TO  n
GO TO n                       Valdymą gaus operatorius su žyme  10 .

ASSIGN  100  TO  m
   . . .
GO TO  m (10, 20, 30)         Klaidingas programos fragmentas:  m gali
                              įgyti tik reikšmes  10, 20 arba 30. Bus
                              deklaruota vykdymo klaida.
```

4.7. SĄLYGOS OPERATORIAI

Tai vykdomieji operatoriai. FORTRANe yra trys sąlygos operatoriai.

Aritmetinis sąlygos operatorius skirtas tokiam blokinės schemos fragmentui programuoti:



Taigi operatorius gali perduoti valdymą vienai iš trijų atskirų krypčių atsižvelgiant į reiškinio reikšmę. Bendrasis operatoriaus pavidalas yra:

IF (*aritmetinis reiškinys*) *ž1, ž2, ž3*

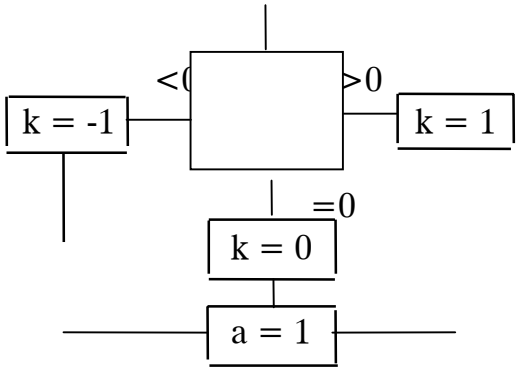
Aritmetinio reiškinio tipas gali būti: sveikasis, realusis, dvigubojo tikslumo. Žymės *ž1, ž2, ž3* rašomos tik prie vykdomųjų operatorių; žymių sąraše žymės gali kartotis.

Vykdant operatorių pirmiausia skaičiuojama aritmetinio reiškinio reikšmė ir pagal tai valdymas perduodamas: jei reikšmė neigiama - operatoriui su žyme *ž1* , jei nulinė - su žyme *ž2* , jei teigiama - su žyme *ž3* .

Operatorius įtrauktas į pasenusių FORTRANo konstrukcijų sąrašą.

Pavyzdžiai:

1. FORTRANe užrašysime tokį blokinės schemos fragmentą:




```
    IF ( m )  10, 20, 30
10  k = -1

    GO TO 40

20  k = 0

    GO TO 40

30  k = 1
```

Žymes parenkame savo nuožiūra, nebūtinai didėjimo tvarka.

Jei m neigiamas, valdymą gaus šis operatorius.

Po operatoriaus “k = -1” turi būti vykdomas operatorius “a = 1.”, prieš kurį parašysime žymę “40”.

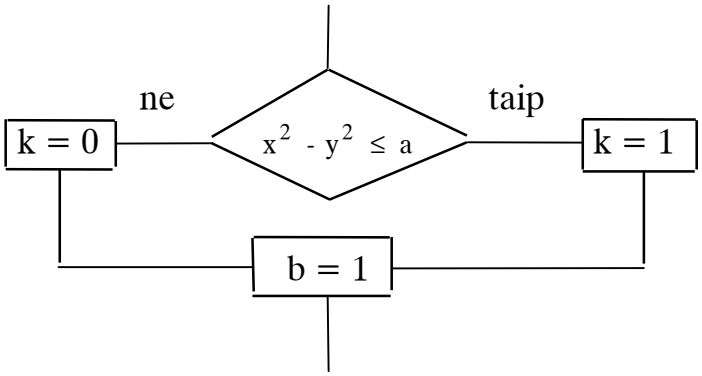
Jei m nulinis, valdymą gaus šis operatorius.

Po jo taip pat turi būti vykdomas operatorius “a = 1.”.

Jei m teigiamas, valdymą gaus šis

operatorius, o po to bus vykdomas
40 a = 1. operatorius “a = 1.”.

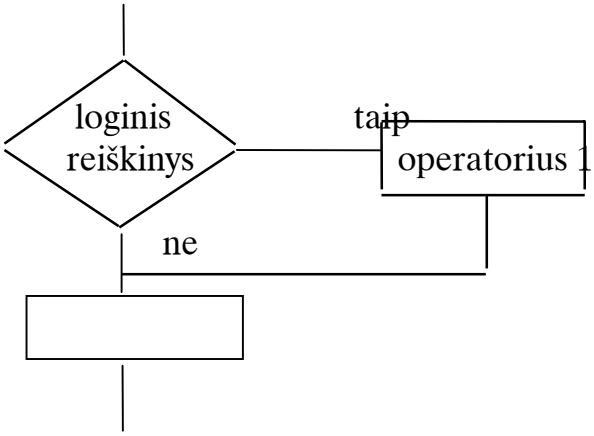
2. Užprogramuosime kitą blokinės schemos fragmentą:



IF (x*x - y*y - a) 10, 10, 20 Vienintelis būdas aritmetiniu IF užprogramuoti panašų fragmentą - parašyti aritmetinį reiškinį, kuriame iš vieno lyginamojo dydžio $x^2 - y^2$ atimamas kitas - a. Taigi sutarkim schemoje kryptį “taip” žymėti žyme 10, o “ne” - 20. Panagrinėkim, kokį turim parašyti žymių sąrašą: kairioji žymė gaus valdymą tada, kai reiškinys $x^2 - y^2$-a neigiamas, o neigiamas jis tik tada, kai $x^2 - y^2 < a$; šiuo atveju, kaip matome iš schemos, turime valdymą atiduoti pagal kryptį “taip” , t.y. žymei 10 . Vidurinė žymė gaus valdymą nulinės reiškinio reikšmės atveju, o tokią turėsim, kai $x^2 - y^2 = a$; šiuo atveju, kaip matom iš schemos, turime valdymą atiduoti pagal tą pat kryptį, t.y. žymei 10. Dešinioji žymė gaus valdymą reiškinio teigiamos reikšmės atveju, o tokią turėsim, kai $x^2 - y^2 > a$; dabar, kaip matom iš schemos, valdymą turim atiduoti pagal kryptį “ne”, t.y. žymę 20. Taigi žymių sąrašas turi būti 10, 10, 20.

10 k = 1
GO TO 30 Operatorių “b=1” žymėsim žyme 30
20 k = 0
30 b = 1.

Loginis sąlygos operatorius skiriamas programuoti tokiam blokinės schemos fragmentui:



Bendrasis operatoriaus pavidalas yra

IF (*loginis reiškinys*) *operatorius1*
operatorius2

Tiesą sakant, loginio IF bendrasis pavidalas yra tik pirmoji mūsų užrašyta eilutė; antrąją - *operatorius2* - pridėjome tik tam, kad būtų paprasčiau paaiškinti vykdymo logiką. *Operatorius1* gali būti bet koks vykdomasis operatorius, išskyrus CASE, DO, ELSE, ELSE IF, END, END IF, END SELECT CASE, SELECT CASE, kitas loginis IF.

Vykdant operatorių, pirmiausia skaičiuojama *loginio reiškinio* reikšmė; po to, jei reikšmė .TRUE. - vykdomas *operatorius1*, o jei reikšmė .FALSE. - *operatorius2*.

Pavyzdžiai: pasitelkę loginį IF užrašykim paskutinįjį nagrinėtą pavyzdį. Pirmiausa didaktikos dėlei pateiksim *neteisingą* variantą. Atrodytų, kad patogiausia būtų užrašyti tokį programos fragmentą:

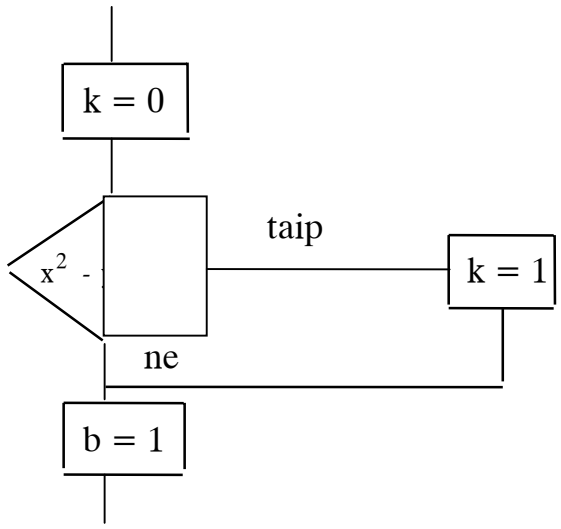
```
IF ( x*x - y*y .LE. a )  k = 1
k = 0
b = 1.
```

Jei loginio reiškinio reikšmė “tiesa”, bus vykdomas operatorius “k = 1” , o po to - operatorius “k = 0” ir t.t. Taigi šiuo loginio reiškinio reikšmės atveju blokinė schema realizuojama netiksliai.

Teisingas variantas būtų toks:

```
      IF ( x*x - y*y .LE. a ) GO TO 10      Jei loginio reiškinio reikšmė "tiesa",
                                            vykdyti operatorių su žyme 10; jei sąlyga
      GO TO 20                              netenkinama, "GO TO 10" bus igno-
                                            ruojamas, o vykdomas šis operatorius.
10  k = 1
      GO TO 30
20  k = 0
30  b = 1.
```

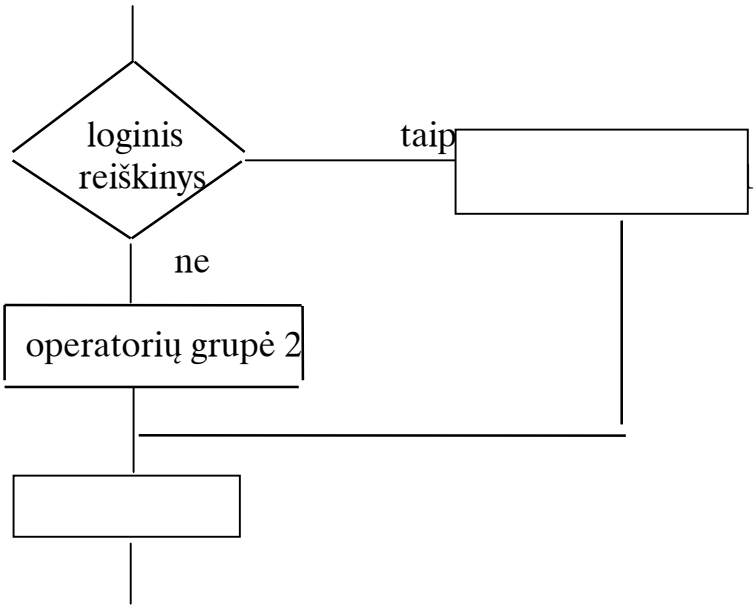
Paprasčiausia panašius veiksmus galima atlikti prieš tai pakoregavus blokinę schemą (jos rezultatas bus toks pat):




```
      k = 0
      IF ( x*x - y*y .LE. a )  k = 1     Jei sąlyga patenkinama, pakeisime  k  reikšmę
                                         į "1".
      b = 1.
```

Blokinis sąlygos operatorius

Pirmiausia paaiškinsime paprastesnį operatoriaus pavidalą (be blokinio ELSE). Operatorius skirtas tokiam blokinės schemos fragmentui:



Operatoriaus supaprastintas pavidalas yra:

IF (*loginis reiškinys*) THEN
operatorių grupė 1
[ELSE
operatorių grupė 2]
END IF
operatorius 3

Operatorių grupes turi sudaryti bent vienas vykdomasis operatorius. Į šias operatorių grupes tiesiog perduoti valdymą iš už IF srities ribų negalima; į jas leistina įeiti tik per operatorių IF.

Vykdant operatorių pirmiausia skaičiuojama *loginio reiškinio* reikšmė. Jei ji "tiesa" - vykdoma *operatorių grupė 1*, o po to - *operatorius 3*. Jei reikšmė yra "melas" - vykdoma *operatorių grupė 2*, o po to - *operatorius 3*. Kaip matyti, galimas ir operatoriaus pavidalas be ELSE ir *operatorių grupės 2*. Šiuo atveju, esant reiškinio reikšmei "melas", iškart bus vykdomas *operatorius 3*.

Pavyzdys: šiuo operatoriumi užprogramuosime paskutinįjį nagrinėtą pavyzdį.

```
IF ( x*x - y*y .LE. a )  THEN      Jei loginio reiškinio reikšmė "tiesa" -
  k = 1                            vykdomas šis operatorius; po to "b = 1.";
ELSE                               antraip -
  k = 0                            šis operatorius; po to "b = 1.";
END  IF                            IF srities pabaiga.
```

b = 1.

***** Blokiniame IF galimas ir kitas, vidinis blokinis sąlygos operatorius, ir blokinis ELSE operatorius. Pastarojo pavidalas yra

ELSE IF (*loginis reiškinys*) THEN

Jei reiškinio reikšmė “tiesa”, bus vykdomi ELSE IF bloko operatoriai, o po jų - kitas to pat lygmens END IF; jei “melas” - tai kitas ELSE IF, ELSE ar to paties lygmens END IF operatorius. Kaip ir blokiniame IF, negalima iš išorės atiduoti valdymą bet kuriam vidinės srities operatoriui.

Šias ganėtinai sudėtingas logines struktūras lengviausia išsiaiškinti, matyt, nagrinėjant konkrečius pavyzdžius. Paaiškinimus užrašysime komentaruose, o programos fragmento logiką paryškinsim patraukdami žemesnio lygmens operatorius į dešinę pusę.

Pavyzdžiai:

```
  IF ( j .GT. 1000 )  THEN
!                       čia parašyti operatoriai vykdomi, jei   j>1000
       ELSE IF ( j .GT. 100 )  THEN
!                       čia parašyti operatoriai vykdomi, jei   j>100  ir   j≤1000
          ELSE IF ( j .GT. 10 )  THEN
!                       čia parašyti operatoriai vykdomi, jei   j>10  ir  j≤100
             ELSE
!                       čia parašyti operatoriai vykdomi, jei   j≤10
    END  IF
```

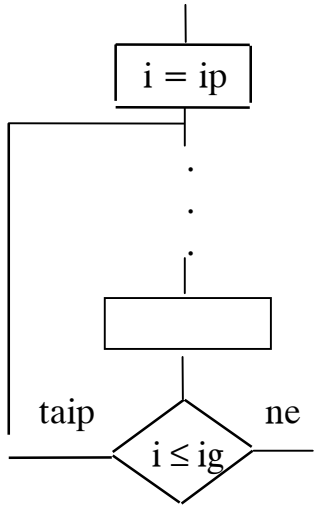
```
  IF ( i .LT. 100 )  THEN
!                       čia parašyti operatoriai vykdomi, jei   i<100
       IF ( j .LT. 10 )  THEN
!                       čia parašyti operatoriai vykdomi, jei   i<100  ir  j<10
       END  IF
!                       čia parašyti operatoriai vykdomi, jei   i<100
  ELSE
!                       čia parašyti operatoriai vykdomi, jei   i≥100
       IF ( j .LT. 10 )  THEN
!                       čia parašyti operatoriai vykdomi, jei i≥100  ir  j<10
       END  IF
```

```
!                          čia parašyti operatoriai vykdomi, jei i≥100
  END  IF
```

Šiame pavyzdyje į vieno operatoriaus IF sritį įdėtas kitas operatorius IF; į šio operatoriaus vidinę sritį gali būti įdėti dar kiti IFai. Tokio įdėtinumo laipsnis neturi viršyti 50.

4.8. CIKLAI

Blokinės schemos fragmentas



realizuoja *ciklą*, t.y., operatoriai, esantys tarp blokų $i = ip$ bei $i = i + ih$, bus kartojami keletą kartų atsižvelgiant į *ciklo kintamojo* i pradinę reikšmę ip, galinę reikšmę ig ir prieaugį ih. Pirmasis blokinės schemos blokas yra ciklo pradžia, o kiti du blokai užbaigia ciklą.

Pastaba: tai supaprastinta ciklo schema. Iš tikrųjų, kai $ip>ig$, ciklas nevykdomas nė karto. Faktiška ciklo realizacija aprašyta 54 p.

Tokią labai dažnai programose pasitaikančią struktūrą patogiausia programuoti specialiu *ciklo operatoriumi* DO:

```
        DO žymė [,] [ i = ip, ig [, ih] ]
              . . .
žymė    CONTINUE
```

Kaip matome, operatorius DO rašomas vietoje pradinio ciklo bloko, o vietoje ciklo pabaigos - tuščiasis vykdomasis operatorius CONTINUE. Į ciklo sritį įrašomi

operatoriai, kuriais užprogramuojami ciklo viduje esantys blokai (jie pažymėti daugtaškiu). *Žymės* paskirtis - nurodyti paskutinįjį ciklo operatorių.

Ciklo operatoriaus užrašymo taisyklės:

- *i* gali būti I, R arba DP tipų kintamasis.
- *ip, ig* gali būti I, R arba DP tipų aritmetiniai reiškiniai.
- *ih* gali būti I, R arba DP tipų aritmetinis reiškinys, tačiau jo reikšmė *0* neleistina. Jei *ih* praleistas, pagal nutylėjimo principą jo reikšmė bus *1*.
- Ciklą užbaigia nebūtinai CONTINUE; *žymę* galima prirašyti ir prie kito paskutiniojo vykdomojo ciklo operatoriaus, tačiau tas paskutinysis operatorius negali būti GO TO, aritmetinis ir blokinis IF, CASE, CYCLE, DO, ELSE, ELSE IF, END, END IF, END SELECT CASE, EXIT, RETURN, SELECT CASE, STOP - šiuo atveju teks ciklą baigti operatorium CONTINUE.
- Ciklo kintamojo *i* reikšmės ciklo viduje jokiais operatoriais keisti negalima.
- Negalima perduoti valdymo iš už ciklo srities ribų į ciklo srities vidų; įėjimas į ciklo sritį galimas tik per operatorių DO. Perduoti valdymą iš ciklo srities į išorę galima.
- Jei ciklo srityje įrašyti operatoriai blokinis IF arba SELECT CASE, tai ir jų sritis užbaigiantys operatoriai END IF bei END SELECT CASE turi būti patalpinti DO srityje.
- Jei ciklo srityje yra kitas ciklo operatorius, tai šio, vidinio, ciklo operatoriaus sritis turi būti išorinio ciklo operatoriaus srities viduje. Tokie kartotiniai ciklai gali būti baigiami vienu operatoriumi.

Ciklo iteracijų kiekį riboja tik ciklo kintamojo maksimali leistina reikšmė. Pavyzdžiui, jei ciklo kintamasis yra I*2 tipo, tai maksimalus iteracijų kiekis - 32767.

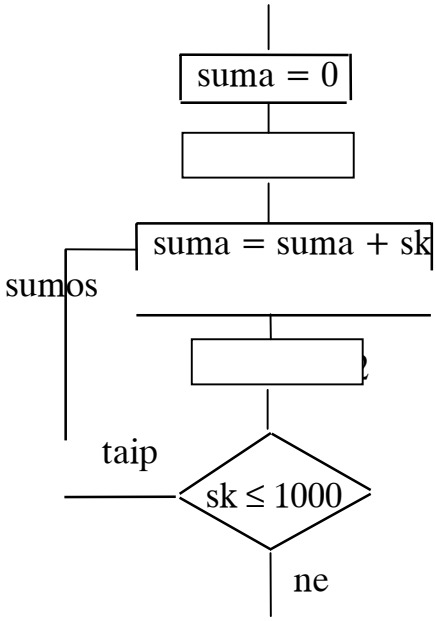
Ciklo operatoriaus vykdymas

Supaprastintai galime laikyti, kad operatorius visiškai realizuoja šio skyriaus pradžioje pateiktą blokinės schemos fragmentą.

Atskiru atveju, kai ciklo kintamasis ir jo kitimo ribos nenurodytos, realizuojamas *begalinis ciklas*. Tokį ciklą užbaigti galima tik iš ciklo srities perduodant valdymą į ciklo išorę ar operatorium EXIT (žr. žemiau).

Pavyzdžiai:

1. Žemiau pateiktas blokinės schemos fragmentas skaičiuoja visų lyginių skaičių pradedant 2 ir baigiant 1000, sumą. Sumą žymime vardu “suma”, o eilinį lyginį skaičių - vardu “sk”.



Sumavimo uždaviniuose sumą visada iš pradžių prilyginame nuliui.
Ciklo pradžia: apibrėžiamas pirmasis lyginis skaičius.
Skaičius pridedamas prie ankstesnės reikšmės.
Pereiname prie kito lyginio skaičiaus.

Ar dar neviršijome galinės skaičiaus reikšmės? Jei viršijome - skaičiavimą baigti.

FORTRANe šis fragmentas atrodys taip:

```
      INTEGER  suma, sk
      suma = 0
      DO 10 sk = 2, 1000, 2
      suma = suma + sk
   10 CONTINUE
```

arba

```
      INTEGER  suma, sk
      suma = 0
      DO 10 sk = 2, 1000, 2
   10 suma = suma + sk
```

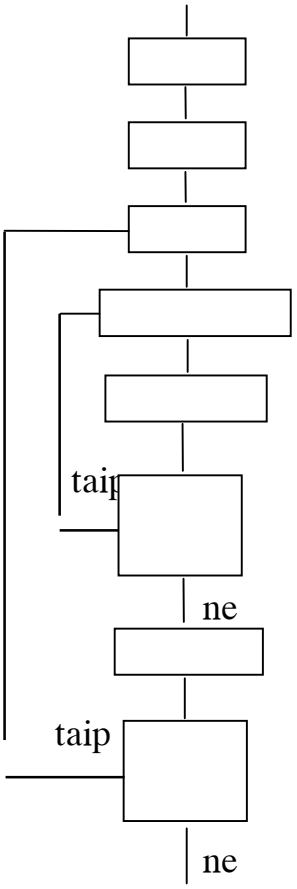
šiuo atveju CONTINUE nebūtinas; ciklą užbaigiame žymę prirašę prie paskutinio vykdomojo ciklo operatoriaus

2. Pateiksime vieną pavyzdį su kartotiniais ciklais, t.y. kai vieno ciklo viduje yra kitas ciklas. Užprogramuosime blokinės schemos fragmentą dvigubos sumos

skaičiavimui (be abejo, tai nėra racionalus algoritmas, tačiau jis labai patogus dvigubo ciklo iliustracijai):

$$s = \sum_{i=1}^{10}\sum_{j=1}^{5} i \times j \ , \quad \text{t.y. } S = 1*1+1*2+1*3+...+1*5 \ + \ 2*1+2*2+...+2*5 \ + \ \ldots$$

$$+ \ 10*1+10*2+...+10*5 \ .$$



suma, kaip įprasta, prilyginama nuliui

išorinio ciklo pradžia

vidinio ciklo pradžia

tai - vienintelis blokas ciklų viduje

šie du blokai žymi vidinio ciklo pabaigą

šie du blokai žymi išorinio ciklo pabaigą

Visus tris žemiau parašytus programos fragmentus transliatorius “supras” visiškai vienodai: kartotinius ciklus galima užbaigti vienu operatorium; pagaliau šiuo atveju operatorius CONTINUE nebūtinas.

```
      s = 0                  s = 0                  s = 0
      DO 10 i = 1,10         DO 10 i = 1,10         DO 10 i = 1,10
      DO 20 j = 1,5          DO 10 j = 1,5          DO 10 j = 1,5
      s = s +i*j             s = s +i*j          10 s = s +i*j
   20 CONTINUE            10 CONTINUE
   10 CONTINUE
```

***** Faktiškai ciklo operatorius vykdomas taip:

1. Suskaičiuojamos reiškinių *ip, ig, ih* reikšmės. Jei reikia, šios reikšmės pervedamos į tokį tipą, kokio yra ciklo kintamasis *i*.
2. Tikrinama, ar reikia vykdyti operatorius ciklo viduje: skaičiuojamas iteracijų skaitiklis pagal išraišką MAX(INT((*ig*-*ip*+*ih*)/*ih*), *0*). Operatoriai nebus vykdomi, jei gauta skaitiklio reikšmė *0*; valdymas tada perduodamas pirmajam operatoriui žemiau ciklo srities pabaigos. Ciklas taip pat nebus vykdomas, jei *ip*>*ig* ir *ih*>*0* arba *ip*<*ig* ir *ih*<*0*.
3. Jei iteracijų skaitikio reikšmė didesnė už nulį, operatoriai ciklo viduje vykdomi.
4. Įvykdžius paskutinįjį ciklo operatorių, *i* reikšmė didinama dydžiu *ih*.
5. Iteracijų skaitiklio reikšmė sumažinama vienetu.
6. Tikrinama iteracijų skaitiklio reikšmė: jei ji didesnė už nulį, vėl vykdomi operatoriai ciklo viduje, ir t. t.

Taigi ciklui pasibaigus, ciklo kintamojo reikšmė yra dydžiu *ih* didesnė, nei jo galinė reikšmė, su kuria paskutinį kartą įvykdytas ciklas.

Čia aprašytas DO operatorius atėjo iš 77-ojo FORTRANo dialekto. 90-ajame standarte kai kurios operatoriaus galimybės skelbiamos pasenusiomis: galimybė naudoti R ir DP tipo ciklo kintamuosius, galimybė baigti ciklą kitu nei CONTINUE operatoriumi bei galimybė kartotinius ciklus baigti vienu operatorium.

Kita ciklo operatoriaus forma

Vėlesniuose dialektuose naudojama ir DO forma be žymėtosios ciklo pabaigos; šiuo atveju ciklą baigia specialus ciklo pabaigos operatorius END DO:

```
DO  [ i = ip, ig [,ih] ]
        . . .
END DO
```

Operatoriaus vykdymas visiškai toks, kaip aprašyta aukščiau. Vienintelis skirtumas - kad kartotiniuose cikluose kelių ciklų jau negalime baigti vienu ir tuo pačiu operatorium.

Pavyzdys:

Perrašysime paskutinįjį pavyzdį naudodami šią operatoriaus formą. Jei ten turėjome tris alternatyvius programos fragmentus, tai dabar galimas tik toks programos fragmentas:

```
s = 0
DO  i = 1, 10
  DO  j = 1, 5
    s = s + i*j
  END DO
END DO
```

**** Loginis ciklo operatorius DO WHILE***

Operatorius

DO [*žymė*[,]] WHILE (*loginis reiškinys*)
. . .
ciklo pabaiga

vykdo ciklo viduje esančius operatorius tol, kol loginio reiškinio reikšmė yra .TRUE. . Ciklą galima užbaigti arba žymėtuoju operatorium - tada žymė po DO yra būtina, arba operatorium END DO - tada žymės po DO neturi būti. Abiem atvejais ciklo pabaigai galioja taisyklės, minėtos kalbant apie abi operatoriaus DO formas. Be to, galioja ir visos kitos ten užrašytos operatorių sričių taisyklės.

Operatoriaus vykdymas:

1. Skaičiuojama loginio reiškinio reikšmė.
2. Jei reikšmė yra .FALSE., jokie ciklo viduje esantys operatoriai nevykdomi; valdymas atiduodamas pirmam operatoriui žemiau ciklo srities pabaigos.
3. Jei reikšmė yra .TRUE., vykdomi ciklo viduje esantys operatoriai.
4. Po ciklo pabaigos operatoriaus valdymas gražinamas į ciklo pradžios operatorių, vėl kartojamas 1-as punktas ir t.t.

Pavyzdys:

Naudodami šį operatorių perrašykime pirmąjį su DO užrašytą programos fragmentą:

```
suma = 0.
sk = 2.
DO  WHILE  (sk .LE. 1000.)
   suma = suma + sk
   sk = sk + 2.
END DO
```

**** Operatoriai CYCLE ir EXIT***

Ciklo viduje parašytas operatorius

CYCLE

perduoda valdymą tiesiai į ciklo pabaigą, “peršokdamas” visus žemiau esančius ciklo operatorius.

Ciklo viduje parašytas operatorius

EXIT

nutraukia ciklo vykdymą ir perduoda valdymą pirmam vykdomajam operatoriui žemiau ciklo pabaigos operatoriaus.

Šie operatoriai naudotini kartu su IF operatoriais. Pavyzdžiui, kai ciklo viduje išpildžius kokią nors sąlygą reikia nutraukti ciklo vykdymą - patogus operatorius EXIT. Panašiai, jei ciklo viduje išpildžius kokią nors sąlygą reika vykdyti “sutrumpintą” ciklą praleidžiant dalį ciklo vidaus galinių operatorių - naudotinas CYCLE.

**** Neišreikštasis ciklas***

Kai kuriems FORTRANo operatoriams (WRITE, DATA) vietoj kintamųjų sąrašų galima įrašyti vadinamuosius neišreikštuosius ciklus:

(*sąrašas*, $i = ip,ig$ [,*ih*])

Čia *sąrašas* gali būti: tuščiasis sąrašas, t.y. nieko; kintamasis, masyvo elementas, išvestinio duomens elementas, masyvas, bet koks reiškinys, pagaliau kitas neišreikštasis ciklas. Šio ciklo veikimas analogiškas DO operatoriaus veikimui.

Mes neišreikštąjį ciklą dažniausiai naudosime masyvų išvesties operatoriuose, kur jis leidžia išvesti norimą masyvo dalį, tik tam tikrus masyvo elementus ir pan. Todėl vietoje *sąrašo* nurodysime masyvo vardą ir skliausteliuose I tipo kintamąjį - masyvo elemento indeksą (dvimačiam masyvui, aišku, bus du kintamieji). Neišreikštuoju ciklu reikiama tvarka keisdami šį kintamąjį ir apibrėšime norimą masyvo dalį.

Pavyzdžiai: programos fragmentas

```
      REAL  x(100), y(100), z(5,8)
            . . .
```

```
      WRITE (5,100) x                                    !  1 atvejis
      WRITE (5,100) (x(i), i = 1,10)                     !  2 atvejis
      WRITE (5,100) (x(i), i = 1,100,2)                        !  3 atvejis
      WRITE (5,100) (x(i), y(i), i = 1,10)               !  4 atvejis
      WRITE (5,100) (x(i), i = 1,10), (y(i), i = 1,10)         !  5 atvejis
      WRITE (5,200) z                                    !  6 atvejis
      WRITE (5,300) ((z(i,j), i = 1, 3), j = 1,5)              !  7 atvejis
      WRITE (5,400) ((z(i,j), j = 1, 8), i = 1,5)              !  8 atvejis
      WRITE (5,500) (i,( z(i,j), j = 1,8), i = 1,5)      !  9 atvejis
100    FORMAT( 10F8.2 )
200    FORMAT( 5F8.2 )
300    FORMAT( 3F8.2 )
400    FORMAT( 8F8.2 )
500    FORMAT( I3, 8F8.2)
      END
```

į displėjaus ekraną išves:

1) visus 100 masyvo x elementų - nes įvesties/išvesties sąraše nurodytas masyvo vardas, o šiuo atveju operuojama tiek masyvo elementų, kiek jų nurodyta aprašymo operatoriuje;

2) pirmuosius 10 masyvo x elementų;

3) tik nelyginius masyvo x elementus;

4) masyvų x ir y elementus tokia tvarka: x(1), y(1), x(2), y(2), ... , x(10), y(10);

5) masyvų x ir y elementus tokia tvarka: x(1),x(2), ... , x(10), y(1), y(2), ... , y(10),

6) visus z masyvo 40 elementų, išdėstytų stulpeliais: z(1,1), z(2,1), ... ,z(5,1), z(1,2), z(2,2), ... , z(5,2), ... , z(5,8) - nes spausdinami visi masyvo elementai ir tokia tvarka, kokia išdėstyti kompiuterio atmintyje;

7) 15-a masyvo z elementų tokia tvarka: z(1,1), z(2,1), z(3,1), z(1,2), z(2,2), z(3,2), ... , z(3,5). Elementai išdėstomi vėlgi stulpeliais, nes toks dvigubas neišreikštasis ciklas vykdomas taip: suteikiama reikšmė indeksui j, o po to visas savo reikšmes “prabėga” indeksas i, po to kitą reikšmę gauna indeksas j ir t.t. Galima sakyti, kad kartotiniai ciklai vykdomi “iš dešinės į kairę”.

8) visus 40 masyvo z elementų eilutėmis: z(1,1), z(1,2), ... , z(1,8), z(2,1), z(2,2), ... , z(2,8), ... , z(5,8);

9) visus 40 masyvo z elementų eilutėmis ir dar eilutės indeksus: 1, z(1,1), z(1,2), ... , z(1,8), 2, z(2,1), z(2,2), ... , z(2,8), ... , 5, z(5,1), z(5,2), ... , z(5,8).

Pavyzdžių su dvimačiu masyvu FORMATuose nurodytas toks spausdinimo specifikacijų kiekis, kad vienoje displėjaus ekrano eilutėje būtų spausdinama viena masyvo eilutė ar stulpelis; primename, kad vienoje ekrano eilutėje telpa 80 simbolių.

4.9. PAPRASČIAUSI DUOMENŲ ĮVESTIES/IŠVESTIES OPERATORIAI

Trumpai aptarsime paprasčiausias duomenų įvesties ir išvesties operatorių formas, kurių pakanka nesudėtingoms programoms sudaryti. Universalios šių operatorių formos bei dauguma operatorių galimybių aprašytos 5-ame skyriuje.

Duomenims įvesti klaviatūra naudojamas operatorius:

READ(*, *) [*įvesties duomenų sąrašas*]

Duomenims išvesti į displėjaus ekraną naudojami operatoriai:

PRINT *, [*išvesties duomenų sąrašas*]
WRITE(*,*) [*išvesties duomenų sąrašas*]

Duomenų sąrašuose pateikiami atskiriant kableliu norimų įvesti ar išvesti kintamųjų vardai ar konstantos. Vykdydamas operatorių READ(*,*) kompiuteris “lauks”, kol klaviatūra bus surinktos *duomenų sąraše* išvardintų kintamųjų reikšmės ir nuspaustas įvesties klavišas Enter. Jei šį klavišą nuspausime anksčiau, nei bus surinktos visos reikšmės, kompiuteris “lauks”, kol baigsime įvesti tiek reikšmių, kiek yra išvardinta objektų *įvesties duomenų sąraše*. Jei surinksime daugiau reikšmių, tai paskutinės nereikalingos bus ignoruojamos. Duomenų įvesties eiliškumas turi atitikti išvardintus sąraše duomenis. Įvedamos konstantos atskiriamos kableliu arba tuščiu tarpu. Žvaigždutės operatorių parametrų sąrašuose nurodo, kad duomenys perduodami standartiniu, kompiliatoriaus nustatytu formatu. Platesnė informacija apie įvesties/išvesties operatorių sintaksę pateikiama 5.10-5.13 skyriuose.

Pavyzdys:

Parašysime programą, kuri apskaičiuotų kvadratinio polinomo $a x^2 + b x + c$ reikšmę duotoms konstantų a, b, c ir kintamojo x reikšmėms:

```
WRITE(*,*) ‘ Įveskite konstantų a b c ir kintamojo x reikšmes’
READ(*,*) a, b, c, x
```

```
WRITE(*,*) ‘ a b c x ‘,a, b, c, x
y = a*x**2+b*x+c
WRITE(*,*) ‘ y =‘, y
END
```

Kaip matote, prieš skaitant duomenis iš klaviatūros, rekomenduojame į ekraną išvesti atitinkamą pranešimą, kad programos vartotojas žinotų, jog jam šiuo metu reikia įvesti nurodytų duomenų reikšmes. Blogas toks programavimo stilius, kai į ekraną išvedami tik “pliki” skaičiai, nes tada pasakyti ką jie reiškia gali tik mintinai žinantis programos tekstą žmogus. Todėl pageidautina visuomet išvesti ir bent minimalią paaiškinamąją informaciją. Pateiktame pavyzdyje tai - trumpas tekstas “y =“, paaiškinantis, kad išvestas skaičius yra y reikšmė. Labai patartina taip pat išvesti ir įvestus pradinius duomenis patikrinimui, ar jie buvo įvesti teisingai: įvesties metu dažnai daromos klaidos.

Jei pradinius duomenis norime skaityti iš failo, o rezultatus taip pat užrašyti į failą, turime pirmiausiai “atidaryti” šiuos failus operatoriumi OPEN, o operatoriuose READ ir WRITE vietoj pirmos žvaigždutės rašyti operatoriuje OPEN nurodyto loginio įrenginio numerį (žr. 5.5 skyrių). Kaip šiuo atveju atrodytų programa? Tarkim, kad pradinių duomenų failo vardas bus “pr_duom.dat”, o rezultatų “sk_duom.rez”. Juos susiesime atitinkamai su loginiais įrenginiais 1 ir 2:

```
OPEN(UNIT=1, FILE = ‘pr_duom.dat)
OPEN(UNIT=2, FILE = ‘sk_duom.rez)
READ(1, *) a, b, c, x
WRITE(2, *) ‘ a=‘, a, ’ b=‘, b, ’ c=‘, c, ’ x=‘, x
y = a*x**2+b*x+c
WRITE(2, *) ‘ y =‘, y
CLOSE(1)
CLOSE(2)
END
```

Pradinių duomenų failą pr_duom.dat turime sukurti patys kokiu nors DOS ar FORTRANo aplinkos tekstų redaktoriumi ir įrašyti į disko aktyvųjį katalogą. Konstantos jame rašomos skiriant jas bent vienu tarpu arba kableliu. Rezultatų failą sk_duom.rez sukurs pati programa.

Operatoriai CLOSE “uždaro” nurodytus failus, kai šie yra nebereikalingi. Tokie operatoriai nėra būtini, nes programai korektiškai baigiant darbą, visi atidaryti failai automatiškai uždaromi. Vis dėlto rekomenduojame operatorius naudoti, nes programai nesėkmingai baigiant darbą, failai liks tinkamai neuždaryti ir gali tapti nebeprieinami, t.y. jų nebegalėsime nei nuskaityti, nei nieko į juos įrašyti.

* 4.10. SELEKTORIUS

Selektoriaus konstrukcija:

SELECT CASE (*reiškinys*)

CASE (*reikšmių sąrašas 1*)
[*operatorių grupė 1*]

[CASE (*reikšmių sąrašas 2*)
[*operatorių grupė 2*]]
. . .
[CASE DEFAULT
[*operatorių grupė n*]]

END SELECT

leidžia perduoti valdymą į atskirai *operatorių grupei i* , kurią turi sudaryti bent vienas vykdomasis operatorius atsižvelgiant į tai, ar *reiškinio* reikšmė sutampa su *reikšmių sąrašo i* bent vienu elementu. Jei *reiškinio* reikšmė nesutampa nė su vienu visų *reikšmių sąrašų* elementu, valdymas atiduodamas *operatorių grupei n*, parašytai po CASE DEFAULT operatoriumi; o jei šio operatoriaus nėra - tai pirmam operatoriui po END SELECT. Įvykdžius bet kurią iš operatorių grupių, valdymas taip pat atiduodamas pirmam operatoriui po END SELECT.

Šioje konstrukcijoje *reiškinys* gali būti tik I, L arba CH tipo.

Tik tokio tipo galimi ir *reikšmių sąrašų* elementai; jų tipas turi atitikti *reiškinio* tipą. *Reikšmių sąraše* galima pateikti atskiras reikšmes, atskirtas viena nuo kitos kableliais, ir/arba reikšmių diapazonus, kuriuose pradinė reikšmė nuo galinės skiriamos dvitaškiu. Tačiau diapazono negalima naudoti L tipo reikšmėms. Jei reikšmės tekstinio tipo, tai jų išdėstymo tvarka tokia, kokia priimta ASCII koduose. Pavyzdžiui, diapazonas 5:10 apims visas I tipo konstantas nuo 5 iki 10 imtinai; ‘i’:’n’ - visas CH*1 tipo konstantas i, j, k, l, m, n. Diapazono žemesnioji riba gali būti praleista - tada bus laikoma, kad sąraše yra visos reikšmės, mažesnios arba lygios viršutinei ribai. Panašiai bus ir praleidus viršutinę diapazono ribą. Pavyzdžiui, diapazonas 5: reiškia, kad reikšmių sąrašo elementai bus visos I tipo konstantos, didesnės arba lygios 5. Viena kuri nors reikšmė gali būti tik viename kuriame nors *reikšmių sąraše*.

Operatorių grupės gali ir nebūti. Tokia tuščia grupė, matyt, prasminga tada, kai norima parodyti, jog kažkuriam *reikšmių sąrašui* nereikia atlikti jokių veiksmų.

Konstrukcijoje galimas tik vienas CASE DEFAULT operatorius.

Panašiai kaip ciklo operatoriai, selektoriaus operatoriai gali būti kartotiniai. Šiuo atveju kiekvienas selektorius privalo baigtis atskiru pabaigos operatoriumi. Išorinio selektoriaus sritis turi apimti vidinio selektoriaus sritį; sritys kirstis negali. Analogiškos taisyklės operatorių sritims galioja ir tuo atveju, kai selektorius įdėtas į ciklo, sąlygos, ELSE ar ELSE IF operatorių, arba atvirkščiai.

Negalima perduoti valdymo iš už selektoriaus srities ribų į selektoriaus vidų; įėjimas į vidų galimas tik per SELECT CASE operatorių. Išėjimas iš srities vidaus į operatoriaus išorę galimas. Negalima perduoti valdymo iš vienos *operatorių grupės* į kitą.

Pavyzdys 1: Ženklo funkciją sign galima užprogramuoti taip:

```
INTEGER  number, sign
  . . .
SELECT  CASE (number)
  CASE ( :-1 )
        sign = -1
  CASE ( 0 )
        sign = 0
CASE ( 1: )
        sign = 1
END  SELECT
    . . .
```

Pavyzdys 2: Programos fragmentas iškviečia skirtingus paprogramius, kai kintamojo ch reikšmė lygi 0; 1, 2, 3, ... , 9; A arba a; D arba d; H arba h. Jei ch reikšmė kitokia, į ekraną išvedamas pranešimas.

```
CHARACTER  ch*1
  . . .
SELECT  CASE ( ch )
  CASE ( '0' )
        CALL  OpenFiles
  CASE ( '1':'9' )
        CALL  RetrieveFiles
  CASE ( 'A', 'a' )
        CALL  AddEntry
  CASE ( 'D', 'd' )
        CALL  DeleteEntry
```

```
  CASE ( 'H', 'h' )
       CALL  Help
  CASE  DEFAULT
       WRITE ( *,* ) 'Command not recognized'
END  SELECT
  . . .
```

* 4.11. ŽYMĖS PRIESKYROS OPERATORIUS

Žymę sveikam *kintamajam* galima priskirti operatoriumi

ASSIGN *žymė* TO *kintamasis*

Taip parinkta žymė privalo būti tame pat programiniame vienete kaip ir operatorius ASSIGN. *Kintamasis* gali būti naudojamas priskiriamajame GO TO arba įvesties/išvesties operatoriuose kaip FORMATo žymė.

Operatorius įtrauktas į pasenusių FORTRANo operatorių sąrašą.

Pavyzdys:

ASSIGN 100 TO i priskiria kintamajam i žymės 100 reikšmę. Jei vėliau bus sumanyta kintamąjį i naudoti kaip aritmetinį kintamąjį, tai reikia turėti omenyje, kad žymės reikšmė ir kintamojo reikšmė yra skirtingi dalykai. Tarkim, norėdami turėti kintamojo i reikšmę 100, turim ją suteikti priskyrimo, DATA ar READ operatoriais.

*5. DUOMENŲ ĮVESTIES IR IŠVESTIES SISTEMA

Bet kokios programos, taip pat ir FORTRANo programos tikslas yra atlikti tam tikrus veiksmus su turimais ar programos vykdymo metu įvestais pradiniais duomenimis ir gautus rezultatus pateikti programos naudotojui. Paprasčiausias duomenų pradinių reikšmių nustatymo būdas yra jų prieskyra betarpiškai pačioje programoje, panaudojant prieskyros operatorius. Tačiau tokia programa bus neuniversali: skaičiuojant uždavinį su kitomis pradinių duomenų reikšmėms tektų taisyti programos tekstą, ją iš naujo kompiliuoti ir komponuoti. Daug efektyvesnis toks pradinių duomenų įvedimas, kai nereikia koreguoti programą ir yra patogu programos vartotojui. Programos vartotojui taip pat labai svarbu gauti ir programos darbo rezultatus jam patogia ir suprantama forma. Visa tai leidžia padaryti pakankamai išplėtota ir sudėtinga FORTRANo duomenų įvesties ir išvesties (Į/I) sistema. Dėl šio skyriaus sudėtingumo siūlome jo medžiagą išmokti etapais. Pirmame etape reikėtų susipažinti su 5.1 skyrelyje pateiktomis pagrindinėmis sąvokomis ir 4.9 skyrelyje aprašytais paprasčiausiais duomenų įvesties ir išvesties organizavimo būdais. To pakaktų paprastoms programoms sudaryti. Sudėtingesniam duomenų išvesties organizavimui reikės susipažinti su formatinių įrašų parengimu, aprašomu 5.9 skyrelyje. Tiems, kurie imsis didesnių programų sudarymo, teks rimtai susipažinti su visa skyriaus medžiaga

5.1. PAGRINDINĖS SĄVOKOS

FORTRANo įvesties/išvesties sistemoje duomenys yra saugomi ***failuose (bylose, rinkmenose)*** ir gali būti perduodami iš vieno failo į kitą.

Failas - tai įvardyta vienodos struktūros ir kreipties logiškai susietų įrašų išorinėje atmintyje seka. Kreipties būdui tenkinti failas fiziškai gali būti suskirstytas į įrašus, blokus arba kitus apimčių vienetus. Skiriami du pagrindiniai failų tipai: išorinis ir vidinis failas.

Kreipties sąvoka naudojama kreipimosi į failą būdui nusakyti. Galime į failą kreiptis nuosekliai įrašas po įrašo arba tiesiogiai į konkretų įrašą. Tai priklauso nuo failo tipo, aptariamo kitame skyrelyje.

Išorinis failas - tai išorinės atminties įrenginyje užrašytas failas. Kompiuterio įvesties ir išvesties įrenginiai: klaviatūra, displėjus, spausdintuvas traktuojami taip pat kaip išoriniai failai.

Vidinis failas - tai atminties dalis, kurioje saugomas tekstinio tipo kintamasis, masyvo elementas, struktūros elementas, tekstinis ar netekstinis masyvas, tekstinė paeilutė.

Įrašas - tai simbolių ar skaitinių duomenų reikšmių seka arba, kitaip tariant, viena duomenų įvesties ar išvesties porcija. Įrašo ilgis matuojamas baitais. Pavyzdžiui 5 simbolių tekstinių duomenų įrašo ilgis bus 5 baitai (1 baitas x 5 = 5 baitai), dviejų dvigubo tikslumo skaičių įrašo ilgis bus 16 baitų (8 baitai x 2 = 16 baitų). Įrašai gali būti formatiniai, neformatiniai, dvejetainiai ir failo pabaigos įrašai.

Įvestis - tai duomenų perdavimas iš failo į vidinę kompiuterio atmintį.

Išvestis - tai duomenų perdavimas iš vidinės atminties į failą.

Eilutė - duomenų darinys, kurį sudaro simbolių seka.

Buferis - rezervuota atminties sritis, skirta tarpiniam duomenų laikymui.

Norint duomenis perduoti įvesties/išvesties operacijomis, reikia atlikti šiuos veiksmus:

- susieti failą su loginiu įrenginiu, per kurį bus vykdoma Į/I operacija. Šis veiksmas dar vadinamas failo atidarymu;
- parengti perduodamų duomenų sąrašą;
- nustatyti duomenų perdavimo formatą.

Viena svarbiausių Į/I sistemos sąvokų yra failas. Šiame skyrelyje tik labai trumpai išskyrėme du failų tipus bei paminėjome jo pagrindinį struktūrinį vienetą - įrašą. Kitame skyrelyje plačiau apžvelgsime failų tipus, jų klasifikavimą

5.2. FAILAI

Kaip jau minėjome, failus sudaro įrašai. Duomenų formatas ir kreipties į failą metodas nulemia failo įrašų tipą. Kiekvienas failo tipas turi savų privalumų, o kokio tipo failą pasirinkti, sprendžia pats programuotojas pagal konkrečios programos poreikius: ar reikia didesnio skaitymo/rašymo greičio, ar svarbiau duomenų kompaktiškumas, ar kokia kita savybė.

Pagal kreipties į failą būdą visi failai skirstomi į ***nuosekliosiosios kreipties*** ir ***tiesioginės kreipties*** failus. Kreipties į failą metodas paskiriamas operatoriuje OPEN parametru ACCESS.

Nuosekliosiosios kreipties failuose (ACCESS='SEQUENTIAL') įrašai skaitomi ir užrašomi vienas po kito nuosekliai. Tokiame faile negalime, pavyzdžiui, perskaityti penkto įrašo, neperskaitę prieš tai keturių pirmų. Vidiniams failams, taip pat traktuojamiems kaip išoriniai failai fiziniams įrenginiams, tokiems kaip klaviatūra, ekranas ir spausdintuvas, galimas tik nuosekliosiosios kreipties metodas.

Tiesioginės kreipties failuose (ACCESS='DIRECT') galima įrašus skaityti ir rašyti laisva tvarka. Pavyzdžiui, užrašyti trečią įrašą, po to pirmą ir t.t. Tiesioginės kreipties failai turi būti tik diskuose. Visi įrašai faile yra numeruojami. Pirmas įrašas turi numerį 1.

Viename duomenų faile visi įrašai gali būti tik vienodo tipo. Pagal duomenų pateikimo būdą galimi ***formatiniai***, ***neformatiniai*** ir ***dvejetainiai*** įrašai. Įrašų tipas nurodomas operatoriuje OPEN parametru FORM. Pagal įrašų tipą failai skiriami į formatinius, neformatinius ir dvejetainius failus.

Formatinį įrašą (FORM='FORMATTED') sudaro ASCII kodais išreikštų simbolių seka. Duomenų išvesties į formatinį failą metu jie pervedami iš vidinio pateikimo būdo į išorinį (simbolinį) pateikimo būdą, o įvesties metu, priešingai, iš simbolinio į vidinį. Su šiais duomenų pateikimo būdais skaitytojas turėjo būti susipažinęs pradiniame informatikos kurse, tad dabar tik trumpai priminsime. Kuo skiriasi vidinis ir išorinis duomenų pateikimo būdai? Vidiniu pateikimu vadinama ta duomenų forma, kuria jie yra saugomi kompiuterio atmintyje. Ši forma priklauso nuo duomenų tipo. Sveikieji skaičiai saugomi tiesiogiai pervesti į dvejetainę skaičiavimo sistemą. Maksimali tokio skaičiaus reikšmė priklauso nuo jam paskirtos ląstelės ilgio. Pavyzdžiui, INTEGER*2 tipo skaičius 86 bus saugomas dviejų baitų ląstelėje kaip 0000000001010110. Realieji skaičiai užrašomi normalizuota kanonine forma ir taip pat pervedami į dvejetainę skaičiavimo sistemą išskiriant tam tikrą bitų kiekį mantisės ir skaičiaus eilės dvejetainiams kodams. Jiems paprastai skiriama 4 arba 8 baitai. Tekstiniams duomenims saugoti naudojami vieno baito (8 bitų) ilgio kodai (ASCII kodai). Tai reiškia, kad kiekvienam teksto simboliui saugoti skirtoje atminties ląstelėje užrašomas jį atitinkantis kodas. Pavyzdžiui, raidė A bus vaizduojama kaip 01000001, simbolis ∑ kaip 11100100 ir t.t. Šių kodų lenteles galima rasti daugelyje informatikai ar programavimui skirtų leidinių. Išorinis duomenų pateikimo būdas - tai duomenų, kaip simbolių sekos, užrašymas ASCII kodais, kurie duomenų išvesties įrenginiuose (displėjaus ekrane, popieriaus lape) paverčiami grafiniais atitinkamų simbolių vaizdais. Pavyzdžiui, tas pats skaičius 86, vaizduojamas išoriškai, taip pat bus dviejų baitų ilgio, nes jį sudaro du simboliai: simbolis 8 (jo ASCII kodas 00111000) ir simbolis 6 (jo ASCII kodas 00110110) ir jis bus išreikštas kodu 0011100000110110.

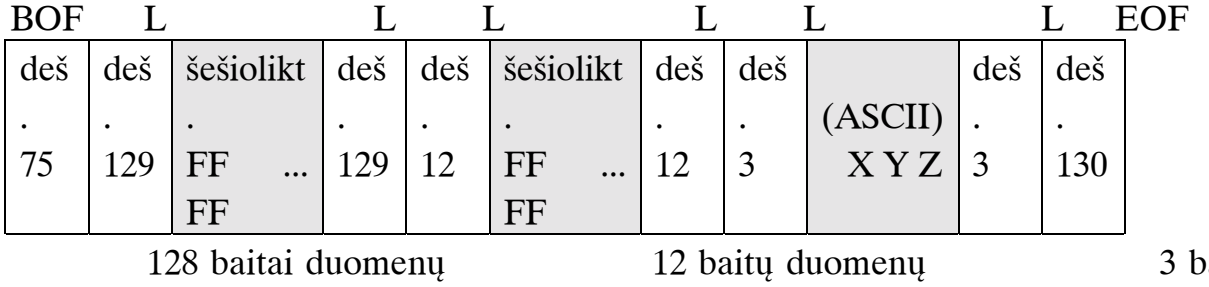
Formatinio failo įrašus išvesties įrenginyje (rodomus ekrane ar atspausdintus spausdintuvu) mes galime perskaityti, nes matome žmogui suprantamų simbolių grafinį vaizdą. Formatinius įrašus galima sutvarkyti pageidaujama forma panaudojant formatų sąrašą operatoriuje FORMAT. Formatiniai įrašai paprastai naudojami pradinių duomenų ir skaičiavimo rezultatų failuose. Formatinio įrašo ilgį nusako simbolių (baitų) skaičius. Formatinio nuosekliosiosios kreipties failo įrašai gali būti skirtingo ilgio.

Formatiniame tiesioginės kreipties faile visi įrašai yra vienodo ilgio, kuris nurodomas parametru RECL operatoriuje OPEN. Jeigu duomenys tiesioginiame faile užima ne visą įrašą, tai jis papildomas tuščio tarpo simboliais. Kiekvienas išorinio formatinio failo įrašas baigiamas duomenų pabaigos požymiu: karietėlės grįžties simboliu

(šešioliktainis OD) ir perėjimo į naują eilutę simboliu (šešioliktainis OA). Plačiau apie formatinį duomenų perdavimą bus kalbama 5.9-5.12 skyreliuose.

Neformatiniuose įrašuose (FORM='UNFORMATTED') duomenys užrašomi tokia forma, kokia jie saugomi kompiuterio atminties įrenginiuose, t.y. vidinio pateikimo būdu. Neformatinių įrašų ilgis išreiškiamas baitais. Neformatiniai failai naudojami tarpiniams skaičiavimo rezultatams, kurie nereikalingi programos vartotojui ir kuriuos naudoja tik atskiros programos dalys, patalpinti išorinėje atmintyje. Neformatinių įrašų skaitymo/rašymo greitis yra didesnis nei formatinių, nes nereikia pervesti duomenų iš vieno jų pateikimo būdo į kitą. Neformatinio failo tikrasis turinys žmogui yra neprieinamas, galime matyti tik jo baitų šešioliktainius kodus, o visos kompiuterio failų vizualizacijos priemonės kiekvieną jo baitą priims kaip simbolio ASCII kodą ir pateiks ekrane šių simbolių virtinę. Teisingai bus parodyti tik tekstiniai duomenys, nes jų vidinis ir išorinis pateikimo būdai sutampa.

Neformatiniame nuosekliosiosios kreipties (ACCESS='SEQUENTIAL') faile įrašai gali būti skirtingo ilgio, tačiau jie skaitomi ar rašomi nuosekliai vienas po kito. Neformatinį nuosekliosiosios kreipties failą sudaro 130 baitų ar mažiau (paskutinis) fiziniai blokai. Kiekvienas fizinis blokas susideda iš siunčiamų duomenų (iki 128 baitų) ir dviejų kompiliatoriaus įterpiamų bloko pradžioje ir gale vadinamųjų "ilgio baitų", kuriuose užrašomas duomenimis užimtų bloko baitų skaičius. Loginis įrašas gali turėti vieną ir daugiau fizinių blokų, taigi būti bet kokio dydžio. Jei įrašas susideda iš kelių blokų, tai pirmųjų pilnų blokų ilgio baitams priskiriama reikšmė 129, rodanti, kad už šio bloko eina dar kitas to paties įrašo blokas, o paskutinio bloko ilgio baito reikšmė yra ≤ 128. Nuosekliojo neformatinio failo pirmas ir paskutinis baitai rezervuoti, pirmojo reikšmė 75, paskutiniojo 130. FORTRANas naudoja šiuos baitus klaidoms tikrinti ir failo pabaigai pažymėti. Nuosekliojo neformatinio failo iš dviejų 140 ir 3 baitų ilgio įrašų, kurių pirmą sudaro 35 skaičiai -1, o antrą simboliai X, Y ir Z, struktūra atrodytų taip:



5.1 pav. Neformatinio failo struktūra. Čia BOF - failo pradžios baitas, L - fizinio bloko "ilgio" baitas, EOF - failo pabaigos baitas, deš.- dešimtainis

Šių įrašų struktūra gali šiek tiek keistis pagal konkrečią FORTRANo realizaciją.

Neformatinis tiesioginės kreipties (ACCESS='DIRECT') failas yra taip pat neformatinių įrašų seka. Šie įrašai yra vienodo ilgio, kuris nustatomas operatoriuje OPEN parametru RECL. Juo galime rašyti ir skaityti įrašus laisva tvarka. Jei faktiškas įrašo ilgis yra mažesnis už nustatytą, jis papildomas ASCII NUL simboliais. Neformatiniame tiesioginės kreipties faile įrašai nedalijami į blokus ir tarp jų nėra skiriamųjų simbolių. Apie neformatinį duomenų perdavimą plačiau kalbama 5.13 skyrelyje.

Dvejetainiai įrašai (FORM='BINARY') yra tam tikra neformatinių įrašų atmaina. Juose, kaip ir neformatiniuose, duomenys užrašomi vidinio pateikimo būdu. Šie įrašai nuo neformatinių šiek tiek skiriasi tik savo struktūra. Jie neskirstomi į blokus, juose nėra jokių papildomų tarnybinių baitų, todėl yra kompaktiškesni ir gali būti naudojami dideliems duomenų kiekiams saugoti bei skaityti kitomis kalbomis sukurtus failus. Dvejetainiai įrašai analogiškai formatiniams ar neformatiniams įrašams gali būti ir nuosekliosiosios, ir tiesioginės kreipties.

5.3. LOGINIO ĮRENGINIO IDENTIFIKATORIUS

Kiekviena Į/I operacija su vidiniu ar išoriniu failu atliekama per loginį įrenginį, t.y. įvesties/išvesties operatoriuose į failą kreipiamasi nurodant loginio įrenginio identifikatorių.

Loginio įrenginio identifikatorius vidiniams failams - tai to failo vardas.

Loginio įrenginio identifikatorius (arba trumpiau - tiesiog įrenginys, kanalas) išoriniams failams nustatomas operatoriuje OPEN parametru UNIT. Tai gali būti sveikojo tipo aritmetinis reiškinys, kintamasis arba konstanta, kurių reikšmė yra intervale nuo -32767 iki 32767, taip pat simbolis žvaigždutė (*). Ryšys tarp įrenginio ir išorinio failo turi būti nustatytas prieš kreipimąsi į failą įvesties/išvesties operatoriais. Šio ryšio nustatymas yra vadinamas failo atidarymu arba susiejimu. Galimi trys failo susiejimo su įrenginiu būdai: tiesioginis, netiesioginis ir išankstinis.

Tiesioginis failo susiejimas su įrenginiu atliekamas operatoriumi OPEN. Failas liks susietas (atidarytas) iki tol, kol to nenutrauks operatorius CLOSE arba nesibaigs programos vykdymas. Pavyzdžiui, programos fragmente:

```
OPEN(UNIT=1, FILE='duom1.dat')
READ(1,*) a,b,c
```

išorinis failas vardu duom1.dat susiejamas su įrenginiu 1 ir iš jo skaitomos kintamųjų a,b ir c reikšmės.

Netiesioginis failo susiejimas su įrenginiu atliekamas pirmu kreipimusi į operatorių READ arba WRITE, jeigu programoje nenaudojamas operatorius OPEN. Failas automatiškai atjungiamas pasibaigus programai.

Išankstinis susiejimas reiškia tai, kad išorinis failas susiejamas su įrenginiu pagal nutylėjimą programos vykdymo pradžioje. Iš anksto atidaromi tik keturi įrenginiai:

* (žvaigždutė)	Klaviatūra ir ekranas
0	Klaviatūra ir ekranas
5	Klaviatūra
6	Ekranas

Įrenginys * (žvaigždutė) įvesties ir išvesties operatoriuose visada nurodo klaviatūrą, kada skaitoma, ir ekraną, kada rašoma. Šis įrenginys vadinamas standartiniu, jis negali būti susietas su kitais išoriniais failais operatoriumi OPEN, taip pat būti naudojamas operatoriuose CLOSE, REWIND, BACKSPACE ir ENDFILE.

Su įrenginiais 0, 5 ir 6 operatoriumi OPEN galima susieti bet kokį kitą išorinį failą. Šis susiejimas galios, kol nebus panaudotas operatorius CLOSE. Uždarius failą operatoriumi CLOSE, šie įrenginiai vėl automatiškai bus iš anksto susieti su atitinkamu fiziniu įrenginiu.

5.4. FAILO VARDAS

Kiekvienas failas turi savo vardą.

Vidinio failo vardas - tai tekstinio kintamojo, tekstinės paeilutės, tekstinio masyvo elemento, tekstinio struktūros elemento, tekstinio ar netekstinio masyvo vardas.

Išorinio failo vardas - bet koks vardas, tenkinantis operacinės sistemos failų vardų reikalavimus. DOS operacinėje sistemoje kai kurie fiziniai įrenginiai turi savo vardus, kurių negalima naudoti kitiems failams įvardyti. Įvesties/išvesties operacijos su šiais įrenginiais gali būti vykdomos naudojant jų sisteminius vardus: CON - klaviatūra arba ekranas, PRN - spausdintuvas, NUL - fiktyvus įrenginys, COM1 - pirmas nuoseklus prievadas ir t.t.

Išoriniam failui suteikti vardą galima šiais būdais:

1. Parametru FILE operatoriuje OPEN.
2. Programos iškvietimui vykdyti komandoje arba atsakyme į atitinkamą užklausą jos vykdymo metu. Tai atliekama tuo atveju, jei programoje naudojamas loginis įrenginys, nesusietas su failu operatoriumi OPEN arba parametro FILE reikšmė operatoriuje OPEN yra tuščio tarpo simboliai.
3. Pagal nutylėjimą. Šis būdas taikomas failams, kurie susieti su loginiu įrenginiu operatoriumi OPEN, bet jame praleistas parametras FILE. Toks failas laikomas

atsitiktiniu (STATUS=‘SCRATCH’). Jam operacinė sistema suteikia vardą Zzxyyyyy, kur x - vienas iš simbolių 0,a,b,... atitinkamai pirmam, antram, trečiam ir t.t. atsiitiktiniam failui, o yyyyy - penkiaženklis operacinės sistemos pateiktas skaičius. Pavyzdžiui, trečias atsitiktinis failas gali turėti vardą ZZb36548.

5.5. ĮVESTIES/IŠVESTIES SĄRAŠAS

Įvesties/išvesties sąrašas (Į/I sąrašas) nurodo programos objektus, kurių reikšmės dalyvauja perduodant duomenis. Sąrašo elementais gali būti šie programos objektai:

- Kintamojo vardas, masyvo elemento, struktūros elemento ar tekstinės paeilutės vardas. Jie pažymi, kurie kintamieji, masyvo ar struktūros elementai bei kurios tekstinės paeilutės bus įvedamos ar išvedamos. Pavyzdžiui,

```
WRITE(*,*) n, bbt(n), stud%pav(n), txt(1:n)
```

- Masyvo vardas. Jei sąraše nurodytas tik masyvo vardas, tai perduodant duomenis dalyvauja visi masyvo elementai tokia tvarka, kokia jie išdėstyti atmintyje. Pavyzdžiui, programoje

```
INTEGER danr(2, 3)/3*5, 3*2/
WRITE(*, *) danr
```

bus išvesta į displėjaus ekraną: 5 5 5 2 2 2.

- Reiškiniai. Aritmetiniai, loginiai ir tekstiniai reiškiniai galimi tik operatoriuose PRINT ir WRITE.
- Neišreikštasis ciklas. Šis ciklas leidžia įvesti ar išvesti masyvo elementus su nurodytais jų indeksais Ciklo rėžių reikšmės gali būti perskaitytos tuo pačiu operatoriumi. Pavyzdžiui:

```
OPEN(10, FILE=‘test.dat’)
READ(10,*) ( mydata(i), i = 5 , 10 )
READ(10, *) k, (regt(i), i = 1, k )
DO i = 1 , m
        WRITE(*,*) (ssk(i,j), j = 1 , n)
END DO
```

Pirmuoju operatoriumi READ perskaitomi iš failo test.dat 6 skaičiai į masyvą mydata, pradedant penktuoju masyvo elementu. Antruoju READ iš to paties failo perskaitoma cikolo rėžio k reikšmė ir po to k pirmų masyvo regt elementų. Operatoriumi WRITE į ekraną eilutėmis išvedamas dvimatis masyvas ssk.

- Tuščiasis sąrašas. Šiuo atveju operatoriumi WRITE galime užrašyti į failą nulinio ilgio įrašą ar formato deskriptorių (žr. 5.9 skyrelį), o operatoriumi READ(*, *) padaryti pauzę , kuri bus nutraukta klavišu Enter.

5.6. ĮVESTIES/IŠVESTIES OPERATORIAI

Duomenis įvesti ir išvesti, failams ir įrenginiams valdyti, prijungiamo prie loginio įrenginio failo savybėms nustatyti, informacijai apie failo būseną gauti naudojami šie operatoriai:

Operatorius	Funkcija
BACKSPACE	Nustato failo skaitymo/rašymo poziciją ankstesnio įrašo pradžioje
CLOSE	Išjungia loginį įrenginį (uždaro failą)
ENDFILE	užrašo failo pabaigos įrašą
FORMAT	Nustato duomenų įvesties/išvesties formatą
INQUIRE	pateikia informaciją apie įrenginio arba įvardinto failo savybes;
LOCKING	Valdo kreipinius į tiesioginės kreipties failus
OPEN	Susieja failą su loginiu įrenginiu (atidaro failą), nustato jo savybes
PRINT	Išveda duomenis į displėjaus ekraną
READ	Įveda (skaito) duomenis
REWIND	Nustato skaitymo/rašymo poziciją failo pradžioje
WRITE	Išveda (užrašo) duomenis

Į/I operatoriuose failo savybės ir duomenų perdavimo būdas nustatomi pasirinktiniais (nebūtinais) parametrais, kurių tam tikras reikšmes FORTRANo transliatorius priima pagal nutylėjimą, jei šie parametrai yra praleidžiami. Įvesties/išvesties operatoriuose naudojami šie parametrai:

Parametras	Į/I operatoriai	Paskirtis.
ACCESS	INQUIRE, OPEN	Nustato kreipties į failą būdą: nuosekląjį ar tiesioginį
BLOCKSIZE	INQUIRE, OPEN	Nustato vidinio atminties buferio, skirto Į/I operacijoms, dydį
END	Visi, išskyrus PRINT	Leidžia valdyti programą, radus įrašą “failo pabaiga”
ERR	Visi, išskyrus PRINT	Leidžia valdyti programą įvykus klaidai Nurodo operatoriaus, kuris bus vykdomas

Parametras	Į/I operatoriai	Paskirtis.
		įvykus klaidai, žymę
FILE	INQUIRE, OPEN	Paskiria arba nurodo failo vardą
[FMT=]	PRINT, READ, WRITE	Nurodo formatinių įrašų perdavimo būdą
FORM	INQUIRE, OPEN	Nustato failo įrašų tipą: formatinį, neformatinį arba dvejetainį
IOSTAT	Visi, išskyrus PRINT	Leidžia valdyti programą įvykus klaidai Apibrėžia kintamojo reikšmę, kuri parodo, ar įvyko klaida
MODE	INQUIRE, OPEN	Paskiria leidžiamas įvesties/išvesties operacijas kitiems procesams, kai dirbama kompiuterių tinkle
[NML=]	PRINT, READ, WRITE	Nurodo norimų įvesti ar išvesti kintamųjų grupę, aprašytą operatoriumi NAMELIST
REC	LOCKING, READ, WRITE	Nustato įrašo numerį, kuris bus skaitomas iš failo, rašomas į failą ar užrakinamas.
SHARE	INQUIRE, OPEN	Paskiria leidžiamas atlikti tuo pačiu metu įvesties/išvesties operacijas, kai tą patį failą naudoja kelios programos, dirbančios tinkle
UNIT	Visi, išskyrus PRINT	Paskiria ar nurodo loginio įrenginio identifikatorių

Jei parametro reikšmė pateikiama kaip reiškinys, tai joje neturi būti kreipinio į funkcijas, turinčias įvesties/išvesties operatorius bei kreipinio į failo pabaigos nustatymo vidinę funkciją EOF.

Paprastai programose, organizuojant duomenų įvestį ir išvestį, laikomasi tokios operatorių išdėstymo tvarkos. Pirmiausiai operatoriumi OPEN atidaromi failai, nustatomos jų savybės, toliau duomenys perduodami (skaitomi/rašomi) operatoriais READ, WRITE ir PRINT. Dirbant su nuosekliosiosios kreipties failais gali tekti panaudoti failų valdymo operatorius BACKSPACE, REWIND ir ENDFILE, kurie nustato rašymo/skaitymo poziciją reikalingoje failo vietoje. Baigus dirbti su failu galima jį atjungti nuo loginio įrenginio (uždaryti) operatoriumi CLOSE. Programos vykdymo metu informaciją apie failą galima gauti operatoriumi INQUIRE. Failo apsaugai dirbant su įjungtu į tinklą kompiuteriu naudojamas operatorius LOCKING bei parametrai MODE ir SHARE.

5.7. ĮVESTIES/IŠVESTIES OPERATORIŲ SINTAKSĖ

Dabar išsamiau aptarsime Į/I operatorių sintaksę. Laikysimės abėcėlinės operatorių išdėstymo tvarkos.

BACKSPACE

Paskirtis. Nustato failo skaitymo/rašymo poziciją ankstesnio įrašo pradžioje;
Sintaksė

BACKSPACE {*unitspec* | ([UNIT=]*unitspec* [,ERR=*errlabel*] [,IOSTAT=*iocheck*])}

Jeigu UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras. Kitų parametrų tvarka yra laisva.

unitspec - sveikojo tipo aritmetinė išraiška, apibrėžianti loginio įrenginio numerį. Jei failas nurodytu įrenginiu nėra atidarytas, įvyksta programos vykdymo klaida.

errlabel - vykdomojo operatoriaus žymė. Jei *errlabel* yra nurodyta, tai įvykus klaidai programos vykdymas perduodamas operatoriui su žyme *errlabel*.

iocheck - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kuriam suteikiamas nulis, jei operatoriaus vykdymo metu neįvyko klaida, ir programos vykdymo klaidų pranešimo numeris, jei operatoriaus vykdymo metu aptikta klaida.

Pastabos

Operatorius BACKSPACE visada grąžina failo skaitymo/rašymo poziciją į ankstesnio įrašo pradžią, išskyrus šiuos specialius atvejus:

Nėra ankstesnių įrašų	Failo skaitymo/rašymo poziciją nepakinta
Ankstesnis įrašas yra failo pabaigos įrašas	Failo skaitymo/rašymo pozicija nustatoma prieš failo pabaigos įrašą
Failo skaitymo/rašymo pozicija yra įrašo viduje	Failo skaitymo/rašymo pozicija nustatoma šio įrašo pradžioje

Pavyzdžiai:

```
BACKSPACE 7
BACKSPACE ( 7 )
BACKSPACE ( iout )
BACKSPACE ( UNIT= nt10, ERR=1000, IOSTAT=ios )
```

CLOSE

Paskirtis.. Išjungia loginį įrenginį (uždaro failą);
Sintaksė

CLOSE ([UNIT=]*unitspec* [,ERR=*errlabel*] [,IOSTAT=*iocheck*] [STATUS=*stat*])

Jeigu UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras. Kitų parametrų tvarka yra laisva.

unitspec - sveikojo tipo aritmetinė išraiška, apibrėžianti loginio įrenginio numerį. Jei failas nurodytu įrenginiu nėra atidarytas, programos vykdymo klaida nefiksuojama.

errlabel - vykdomojo operatoriaus žymė. Jei *errlabel* yra nurodyta, tai įvykus klaidai programos vykdymas perduodamas operatoriui su žyme *errlabel.*

iocheck - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kuriam suteikiamas nulis, jei operatoriaus vykdymo metu neįvyko klaida, ir programos vykdymo klaidų pranešimo numeris, jei operatoriaus vykdymo metu aptikta klaida

stat - tekstinė išraiška, kurios reikšmė yra 'KEEP' - palikti arba 'DELETE' - ištrinti failą iš išorinės atminties įrenginio. Jei failas buvo atidarytas, be nurodyto vardo, toks failas yra atsitiktinis (scratch file). Atsitiktiniai failai yra laikini failai ir jie visuomet ištrinami programai normaliai pasibaigus. Nuoroda tokiems failams STATUS='KEEP' sukels programos vykdymo klaidą. Visiems kitiems failams pagal nutylėjimą *stat* reikšmė yra 'KEEP'.

Pastabos

Atidarytus failus nėra būtina uždaryti tiesiogiai operatoriumi CLOSE. Normaliai užbaigiama programa uždarys visus failus tam tikru statusu.

Įrenginio 0 uždarymas automatiškai perjungs įrenginį 0 į klaviatūrą ar ekraną.

Įrenginių 5 arba 6 uždarymas automatiškai perjungs juos atitinkamai į klaviatūrą ar ekraną. Įrenginio *(žvaigždutė) uždarymas sukels programos kompiliavimo klaidą.

Pavyzdžiai:

```
CLOSE (10)
CLOSE ( 7, STATUS = 'DELETE')
```

ENDFILE

Paskirtis.. Užrašo failo pabaigos įrašą nurodytame įrenginyje.

Sintaksė

ENDFILE {*unitspec* | ([UNIT=]*unitspec* [,ERR=*errlabel*] [,IOSTAT=*iocheck*])}

Pastabos

Užrašius failo pabaigos įrašą, skaitymo/rašymo pozicija nustatoma už failo pabaigos įrašo. Toliau duomenis perduoti į failą ir iš failo bus uždrausta, kol nebus įvykdytas operatorius BACKSPACE arba REWIND.

Jei ENDFILE naudojamas tiesioginės kreipties faile, tai už naujo failo pabaigos įrašo visi kiti įrašai yra ištrinami.

FORMAT

Paskirtis. Nustato duomenų įvesties/išvesties formatą.

Sintaksė

ž FORMAT (*flist*)

flist - formatų sąrašas, kurį sudaro formato deskriptorių seka, apibrėžianti duomenų įvesties/išvesties formatą ir tikslumą. Deskriptorius - tai konkretaus formato aprašas.

ž - formato žymė.

Plačiau formatinę duomenų įvestį/išvestį aptarsime 5.8 ir 5.9 skyreliuose.

Pastabos

FORMAT operatorius visada turi žymę.

Dėl klaidingo formato elementų sąrašo atsiranda kompiliavimo arba programos vykdymo klaida.

INQUIRE

Paskirtis. Pateikia informaciją apie Į/I įrenginio ir failo savybes.

Sintaksė

INQUIRE ({ [UNIT=]*unitspec* | [FILE=*file*] } [,ACCESS=*access*]
[,BINARY= *binary*]* [,BLANK=*blank*] [,BLOCKSIZE=*blocksize*]* [,DIRECT=*direct*]
[,ERR=*errlabel*] [,EXIST=*exist*] [,FORM=*form*] [,FORMATTED=*formatted*]
[,IOSTAT=*iostat*] [,NAME=*name*] [,NAMED=*named*] [,NEXTREC=*nextrec*]
[,NUMBER=*num*] [,OPENED=*opened*] [,RECL=*recl*] [,SEQUENTIAL=*seq*]
[,UNFORMATTED=*unformatted*])

Jei UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras, o kitų parametrų tvarka yra laisva.

unitspec - sveikojo tipo aritmetinė išraiška arba žvaigždutė (*). Jeigu UNIT= yra nurodytas, tai negalima įtraukti parametro NUMBER=, nes tai sukels vykdymo klaidą. Aišku, kad gali būti nurodytas tik *unitspec* arba *file*, bet ne abu kartu. Jei nurodomas *unitspec*, tokia operacija vadinama “paieška pagal įrenginį”.

file - tekstinė išraiška, nurodanti failo, apie kurį norime gauti informaciją (pasiteirauti), vardą. Kai nurodome failo vardą, ši operacija vadinama “paieška pagal vardą”.

access - tekstinis kintamasis, tekstinio masyvo ar struktūros elementas. Įgauna reikšmę ‘SEQUENTIAL’, jei failas atidarytas duomenims papildyti arba yra nuosekliosiosios kreipties, ir reikšmę ‘DIRECT’, jei failas yra tiesioginės kreipties. Jeigu vykdysime “paiešką pagal įrenginį, kuris nebus susietas su jokiu failu, tai *access* reikšmė bus neapibrėžta.

binary - tekstinis kintamasis, tekstinio masyvo ar struktūros elementas. Įgauna reikšmę ‘YES’, jei failas atidarytas kaip dvejetainis, t.y. su parametru FORM= ‘BINARY’. Kitais atvejais įgauna reikšmę ‘NO’ arba ‘UNKNOWN’.

blank - tekstinis kintamasis, tekstinio masyvo arba struktūros elementas. Įgauna reikšmę “NULL’, jei faile ignoruojami tušti tarpai skaitinių duomenų įvesties laukuose, ir reikšmę ‘ZERO’, jei tušti tarpai interpretuojami kaip nuliai. Šie režimai nustatomi operatoriumi OPEN arba formatų deskriptoriais BN ir BZ (žr. 5.9 skyrelį).

blocksize - sveikojo tipo kintamasis, masyvo ar struktūros elementas. Įgauna Į/I buferio apimties reikšmę.

direct - sveikojo tipo kintamasis, masyvo ar struktūros elementas. Įgauna reikšmę ‘YES’, jei failas yra tiesioginės kreipties, ir reikšmę ‘NO’ arba ‘UNKNOWN’ kitais atvejais.

errlabel - vykdomojo operatoriaus žymė. Jei *errlabel* yra nurodyta, tai įvykus klaidai programos vykdymas perduodamas operatoriui su žyme *errlabel*.

exist - loginio tipo kintamasis, masyvo ar struktūros elementas. Įgauna reikšmę ‘TRUE’, jei nurodytas įrenginys ar failas egzistuoja, ir reikšmę .FALSE. kitais atvejais.

form - tekstinis kintamasis arba masyvo elementas. Įgauna reikšmę ‘FORMATTED’, jei failui ar įrenginiui nustatyta formatinė Į/I, ir reikšmę ‘UNFORMATTED’, jei nustatyta neformatinė Į/I.

formatted - tekstinis kintamasis, masyvo ar struktūros elementas. Įgauna reikšmę ‘YES’, jei failui ar įrenginiui nustatyta formatinė Į/I, ir reikšmę ‘NO’ arba ‘UNKNOWN’ kitais atvejais.

iostat - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kuriam suteikiamas nulis, jei operatoriaus vykdymo metu neįvyko klaida, ir programos vykdymo klaidų pranešimo numeris, jei operatoriaus vykdymo metu klaida aptikta.

name - tekstinis kintamasis, masyvo arba struktūros elementas. Paieškoje pagal įrenginį įgauna susieto su šiuo įrenginiu failo vardo reikšmę. Jei su nurodytu įrenginiu nesusietas joks failas, *name* reikšmė yra neapibrėžta.

named - loginis kintamasis, masyvo arba struktūros elementas. Įgauna reikšmę .FALSE., jei nurodytas parametru FILE ar UNIT failas nėra atidarytas arba jis yra

atsitiktinis (scratch file), ir reikšmę .TRUE. visais kitais atvejais. Pavyzdžiui, programoje

```
LOGICAL nmd
OPEN (10, FILE = 'AZD1.DAT')
INQUIRE (UNIT = 10, NAMED = nmd)
```

loginio kintamojo reikšmė bus .TRUE.

nextrec - sveikojo tipo kintamasis, masyvo arba struktūros elementas. Įgauna kito įrašo numerio reikšmę tiesioginės kreipties faile. (Pirmas įrašas turi eilės numerį 1.)

num - sveikojo tipo kintamasis, masyvo arba struktūros elementas. Paieškoje pagal vardą įgauna nurodyto failo loginio įrenginio numerį. Jei nurodytas failas nėra atidarytas, reikšmė neapibrėžta. Pavyzdžiui:

```
OPEN (UNIT = 4, FILE = 'PROG2.REZ')
INQUIRE (FILE = 'PRG2.REZ', NUMBER = num1)
INQUIRE (FILE = 'AAA.DAT', NUMBER = num2)
```

Kintamojo num1 reikšmė bus 4, nes nurodytas failas jo atidarymo metu susietas su loginiu įrenginiu 4, o kintamojo num2 reikšmė bus neapibrėžta, nes failas vardu AAA.DAT nėra atidarytas.

opened - loginis kintamasis, masyvo arba struktūros elementas. Paieškoje pagal loginį įrenginį įgauna reikšmę .TRUE. jei bet koks failas susietas su nurodytu įrenginiu, ir .FALSE. kitais atvejais. Paieškoje pagal failo vardą įgauna reikšmę .TRUE., jei nurodytas failas yra susietas su kokiu nors loginiu įrenginiu, ir reikšmę .FALSE. kitais atvejais.

Pavyzdžiui:

```
LOGICAL opnd1, opnd2
OPEN (UNIT = 15, FILE = 'PROG2.REZ')
      INQUIRE (UNIT = 15, FILE = 'PROG2.REZ',
OPENED=opnd1)
INQUIRE (UNIT= 'AAA.DAT', OPENED = opnd2)
```

Kintamojo opnd1 reikšmė bus .TRUE., o kintamojo opnd2 reikšmė bus .FALSE.

recl - sveikojo tipo kintamasis, masyvo arba struktūros elementas. Įgauna įrašo ilgio (baitais) reikšmę tiesioginės kreipties failuose.

seq - tekstinis kintamasis, masyvo ar struktūros elementas. Įgauna reikšmę 'YES', jei nurodytas failas yra atidarytas kaip nuosekliosios kreipties failas, ir reikšmę 'NO' arba 'UNKNOWN' kitais atvejais.

unformatted - tekstinis kintamasis, masyvo ar struktūros elementas. Įgauna reikšmę ‘YES’, jei nurodytas failas yra atidarytas kaip neformatinis, ir reikšmę ‘NO’ arba ‘UNKNOWN’ kitais atvejais.

Pastabos

Operatorius INQUIRE suteikia tas įvairių failo parametrų reikšmes, su kuriomis failas buvo atidarytas. INQUIRE operatorius negali nustatyti neatidaryto failo charakteristikų bei atskirti, kurie parametrai buvo nustatyti programuotojo, o kurie priimti kompiliatoriaus pagal nutylėjimą.

Operatorių INQUIRE galima rašyti bet kurioje programos teksto vietoje tarp programos pradžios ir pabaigos operatorių. Jis grąžins parametrų reikšmes, esančias programos vykdymo momentu.

Pavyzdžiai:

```
C       Šioje programoje prieš atliekant Į/I operacijas, tikrinama ar egzistuoja
C       failas aktyviajame kataloge nurodytu vardu. Jei tokio failo nėra,
C       prašoma vėl nurodyti kitą failo vardą.
          CHARACTER*12 fname
          LOGICAL exists
          exist = .FALSE.
                DO WHILE (.NOT. exist)
                WRITE (*, ‘(2X, A\ )’) ‘Įveskite failo vardą: ’
                READ (*, ‘(A)’) fname
C       Teiraujamasi apie failo egzistavimą:
                INQUIRE (FILE = fname, EXIST = exists)
                IF (.NOT. exists)  THEN
            WRITE (*, ‘(12A1)’)’ >> Nerastas failas vardu’, fname
          END IF
          END DO
          OPEN (UNIT = 1, FILE = fname)
          END
```

OPEN

Paskirtis. - Susieja (atidaro) loginio įrenginio numerį su išoriniu failu ar Į/I įrenginiu. Apibrėžia įvairias atidaromo failo savybes.

Sintaksė

OPEN ([UNIT =]*unitspec*, [ACCESS = *access*] [, BLANK = *blank*]
[, BLOCKSIZE = *blocksize*] [, ERR = *errlabel*] [, STATUS = *status*]
[, FILE = *file*] [, FORM = *form*] [, IOSTAT = *iocheck*] [, RECL = *recl*]

Jei UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras. Kitų parametrų eilės tvarka yra laisva.

unitspec - sveikojo tipo išraiška, apibrėžianti loginio įrenginio numerį.

access - tekstinė išraiška, kurios reikšmė gali būti ‘APPEND’, ‘DIRECT’ arba ‘SEQUENTAL’ (priskiriama pagal nutylėjimą).

blocksize - sveikojo tipo išraiška, apibrėžianti vidinio Į/I buferio dydį baitais.

errlabel - vykdomojo operatoriaus žymė tame pačiame programos modulyje. Jei *errlabel* yra nurodytas, įvykus Į/I operacijos klaidai, programos vykdymas perduodamas operatoriui su žyme *errlabel.*

status - tekstinė išraiška, kurios galimos reikšmės yra ‘OLD’, ‘NEW’, ‘UNKNOWN’ ir ‘SCRATCH’.

‘OLD’ - failas turi jau egzistuoti. Jei jis jau įrašytas į diską, šis atidaromas, priešingu atveju pateikiamas pranešimas apie klaidą. Kai atidaromas egzistuojantis failas ir vykdoma rašymo operacija, tai ankstesnieji failo įrašai dingsta. Jei norime tęsti įrašus faile, išsaugodami senus, reikia jį atidaryti su parametru ACCESS = ‘APPEND’, kuris automatiškai failo rašymo/skaitymo poziciją nustatys prieš failo pabaigos įrašą.

‘NEW’ - failo neturi būti aktyviajame kataloge, jis yra sukuriamas. Jei failas jau egzistuoja, įvyksta programos vykdymo klaida.

‘SCRATCH’ - ši parametro STATUS reikšmė suteikia laikino failo statusą. Jis ištrinamas, jei failas uždaromas ar programa baigiama vykdyti.

‘UNKNOWN’ - kai failas atidaromas su šiuo parametru, tai jo statusas laikomas ‘OLD’, jei failas jau egzistuoja, ir ‘NEW’ - priešingu atveju.

file - tekstinė išraiška, išreiškianti atidaromo failo vardą. Jei *file* yra praleistas, atidaromas laikinas failas. Jei failo vardas išreiškiamas tuščiu tarpo simboliu, t.y. FILE = ‘ ‘, tai jo vardą reikia nurodyti programos iškvietimo eilutėje kaip argumentą arba atsakyti į atitinkamą operacinės sistemos pranešimą.

form - tekstinė išraiška, kurios reikšmė gali būti ‘FORMATTED’, ‘UNFORMATTED’ arba ‘BINARY’. Tiesioginės kreipties failams pagal nutylėjimą imama reikšmė ‘UNFORMATTED’, nuosekliosios kreipties failams - ‘FORMATTED’.

iocheck - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kuris įgauna reikšmę nulis, jei failo atidarymas įvyksta be klaidų, neigiamą sveiką reikšmę, jei susiduriama su pabaigos įrašu, arba klaidos pranešimo numerio reikšmę, jei įvyksta klaida.

recl - sveikojo tipo išraiška, nustatanti tiesioginės kreipties failuose įrašo ilgį baitais.

Pastabos

Jei atidaromas failas su loginiu įrenginiu, su kuriuo jau atidarytas kitas failas, tai šis failas pirmiausia uždaromas ir tada atidaromas naujas failas. Pranešimas apie klaidą nepateikiamas. Tuo pačiu metu negalima failo susieti su keliais loginiais įrenginiais.

Jei programoje panaudojami READ ar WRITE operatoriai su neatidarytais failais, tai į ekraną išvedamas pranešimas su prašymu nurodyti failo, su kuriuo bus atlikta Į/I operacija, vardą.

Pavyzdžiai:

```
C           Atidarome kanalu 1 nuosekliosios kreipties jau egzistuojantį formatinį failą
C           nurodytu vardu,
      CHARACTER fname * 12
      WRITE (*, *) ‘  Įveskite failo vardą: ‘
      READ (*, ’(A)’) fname
      OPEN (UNIT = 1, FILE = fname, STATUS = ‘OLD’)
C           Atidarome kanalu 2 tiesioginės kreipties neformatinį failą, kurio įrašų ilgis 80
C           baitų, vardu tarp_rez.
      OPEN (UNIT=2, FILE = ‘tarp_rez’, ACCESS = ‘DIRECT’,
     + FORM = ‘UNFORMATTED’, RECL = 80)
```

PRINT

Paskirtis. - išveda duomenis į ekraną.

Sintaksė

PRINT {*,|*formatspec*|*namelist*} [, *iolist*]

formatspec - formato specifikatorius.

namelist - išvedamų duomenų vardų įrašo specifikatorius (žr.5.11 sk.)

iolist - Į/I duomenų sąrašas

Pastabos

Jei naudojamas duomenų vardų sąrašo specifikatorius *namelist*, Į/I sąrašo neturi būti.

Operatorius PRINT rašo tik į įrenginį *.

Į/I duomenų sąraše negali būti struktūrinio tipo duomenų, gali būti tik struktūrų elementai.

Pavyzdžiai:

```
PRINT *, ‘ X=‘,xalfa, ‘  Y=‘,ybeta
PRINT ‘(/15X,A/)’  ‘ Rezultatai’
```

READ

Paskirtis. - skaito duomenis iš failo, susieto su įrenginiu *unitspec* ir jų reikšmes suteikia atitinkamiems Į/I sąrašo elementams.

Sintaksė

READ {*formatspec* | ([UNIT=] *unitspec*
[,{[FMT=] *formatspec* | [NML=] namelist}
[,END=*endlabel*] [,ERR=*errlabel*]
[,IOSTAT=*iochec*] [,REC=*recl*])} *iolist*

Jei žodžiai UNIT ir FMT yra praleidžiami, tai parametrai *unitspec* ir *formatspec* turi būti atitinkamai pirmieji.

formatspec - formato specifikatorius (žr. 5.8, 5.9 sk.).

unitspec - loginio įrenginio specifikatorius (žr. 5.8, 5.9 sk.).

namelist - vardų sąrašo grupės identifikatorius. Šiuo atveju neturi būti Į/I sąrašo. Įvesties/išvesties operacijos vardų sąrašu galimos tik su nuosekliosios kreipties failais (žr. 5.11 sk.).

endlabel - vykdomojo operatoriaus, kuriam perduodamas programos vykdymas, jei aptinkama failo pabaiga nepasibaigus Į/I sąrašui, žymė.

errlabel - vykdomojo operatoriaus žymė tame pačiame programos modulyje. Jei *errlabel* yra nurodytas, įvykus Į/I operacijos klaidai, programos vykdymas perduodamas operatoriui su žyme *errlabel.*

iochec - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kurio reikšmė bus lygi nuliui, jei operatoriaus vykdymo metu neįvyko klaida, arba lygi programos vykdymo klaidų pranešimo numeriui, jei operatoriaus vykdymo metu klaida aptikta.

recl - teigiama sveikojo tipo išraiška, nurodanti įrašo eilės numerį, kuris turi būti perskaitytas tiesioginės kreipties faile.

Pastabos

Operatorius READ *formatspec, iolist sąrašas* skirtas duomenims skaityti iš standartinio įrenginio (klaviatūros).

Jei failas nesusietas su nurodytu loginiu įrenginiu *unitspec* operatoriumi OPEN, tai operatorius READ pats atlieka prijungimą, kuris tolygus tokio operatoriaus įvykdymui:

OPEN(*unitspec*, FILE=‘ ‘, STATUS=‘OLD’, ACCESS=‘SEQUENTAL’,
FORM=*form*)

Tekstinio tipo parametro *form* reikšmė imama ‘FORMATTED’ formatiniam operatoriui READ ir ‘UNFORMATTED’ neformatiniam.

Pavyzdžiai:

```
C           Skaitome iš klaviatūros A, B ir C reikšmes
            READ *,  A, B, C
C           Skaitome laisvu formatu iš failo, atidaryto kanalu 1 kintamųjų n, m
C           reikšmes ir masyvo   a_mas   n  reikšmių
      READ(1, *) n, m
            READ(1, * , IOSTAT=klaid)  ( a_mas(i), i=1,n)
      IF(klaid  .NE.  0) THEN
                  WRITE (*, * ) ‘  Masyvo   a_mas   skaitymo klaida’, klaid
                  STOP
            END IF
```

REWIND

Paskirtis. - nustato duomenų skaitymo/rašymo poziciją faile prieš pirmą įrašą.

Sintaksė

REWIND { *unitspec* | ([UNIT=] *unitspec* [,ERR=*errlabel*] [,IOSTAT=*iochec*])}

Jei UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras. Kitų parametrų eilės tvarka yra laisva.

unitspec - išorinio failo loginio įrenginio specifikatorius.

errlabel - vykdomojo operatoriaus tame pačiame programos modulyje žymė. Jei *errlabel* yra nurodytas, įvykus Į/I operacijos klaidai, programos vykdymas bus perduotas operatoriui su žyme *errlabel.*

iochec - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kurio reikšmė lygi nuliui, jei operatoriaus vykdymo metu neįvyko klaida, arba lygi programos vykdymo klaidų pranešimo numeriui, jei operatoriaus vykdymo metu įvyko klaida.

WRITE

Paskirtis. - Užrašo nurodytus Į/I sąraše duomenis į failą, susietą su loginiu įrenginiu *unitspec.*

Sintaksė

WRITE ([UNIT=] *unitspec* [,[* |{ [FMT=] *formatspec*] | [NML=] *namelist*}]
[,ERR=*errlabel*] [,IOSTAT=*iochec*] [,REC = rec]) *iolist*

Jei UNIT= yra praleistas, tai *unitspec* turi būti pirmas parametras. Kitų parametrų eilės tvarka yra laisva.

unitspec - išorinio failo loginio įrenginio specifikatorius.

errlabel - vykdomojo operatoriaus žymė tame pačiame programos modulyje. Jei *errlabel* yra nurodytas, įvykus Į/I operacijos klaidai programos vykdymas perduodamas operatoriui su žyme *errlabel.*

iochec - sveikojo tipo kintamasis, masyvo ar struktūros elementas, kurio reikšmė lygi nuliui, jei operatoriaus vykdymo metu neįvyko klaida, arba lygi programos vykdymo klaidų pranešimo numeriui, jei operatoriaus vykdymo metu klaida aptikta.

5.8. FORMATINIŲ ĮRAŠŲ PERDAVIMO BŪDAI

Perduoti duomenis formatiniais įrašais galima vienu iš dviejų būdų: kai įvestis/išvestis yra valdoma formatų sąrašo (formatinė įvestis/išvestis) arba yra valdoma įvesties/išvesties sąrašo (sąrašu valdoma įvestis/išvestis). Duomenų sutvarkymo būdą nurodo parametras [FMT=] *formatspec,* Jis išreiškia formatinės įvesties/išvesties operatoriuose PRINT, READ ir WRITE formato specifikatorių.

Formato specifikatoriumi gali būti:

- Operatoriaus FORMAT žymė. Jei nurodyta operatoriaus FORMAT žymė, duomenys bus perduoti formatu, aprašytu šio operatoriaus formatų sąraše *flist*. Toks formato nurodymas priskiriamas prie pasenusių FORTRANo konstrukcijų.

WRITE(*unitspec*, *ž*) [*iolist*]
ž FORMAT(*flist*)

- Sveikojo tipo kintamojo vardas. Šiuo atveju operatoriumi ASSIGN reikia susieti sveikąjį kintamąjį su operatoriaus FORMAT žyme ir šiuo kintamuoju nurodyti įvesties/išvesties operacijoje naudojamo operatoriaus FORMAT žymę, t.y.

ASSIGN *ž* TO *sveikojo tipo kintamasis*
ž FORMAT (*flist*)
WRITE(*unitspec, sveikojo tipo kintamasis*) *iolist*

Pavyzdžiui:

```
      ASSIGN 2000 TO myfmt
2000  FORMAT ( /5X,3F7.2/ )
      WRITE( 4, myfmt) a, b , c
```

- Tekstinio tipo reiškinys. Tai gali būti tekstinio tipo konstanta, paeilutė, kintamasis ar masyvo vardas, t.y.

WRITE(*unitspec,* ‘(*flist*)’) *iolist*

arba

form1 = ‘(*flist*)’
WRITE(*unitspec*, form1) *iolist*

Pavyzdžiui:

```
CHARACTER  form1*10
form1 = ’(10X,A,I4)’
WRITE( *, ‘( /5X,3F7.2/ )’) a, b , c
WRITE( *, form1 ) ‘N =‘,n
```

Čia formatų sąrašą sudarantys formatų deskriptoriai 10X, A, I4 ir kiti aprašyti 5.9 skyrelyje.

- Simbolis * (žvaigždutė). Žvaigždute išreikštas formato specifikatorius parodo, kad vykdoma sąrašo valdoma Į/I operacija (žr. 5.11 skyrių). Šiuo atveju kompiliatorius kiekvienam Į/I sąrašo elementui nustato standartinį formatą pagal jo tipą. Tokios Į/I operacijos dar vadinamos Į/I operacijomis laisvu formatu.Pavyzdžiui.:

WRITE(*unitspec*,*) *iolist*
READ(*unitspec*,*) *iolist*

5.9. FORMATINĖ ĮVESTIS/IŠVESTIS

Formatinė įvestis/išvestis suteikia galimybę programuotojui norima forma įvesti pradinius duomenis bei išvesti tekstinius programos darbo rezultatus. Duomenų įvesties/išvesties formatą nustato paimtas į skliaustus formatų sąrašas, kurį sudaro atskirti vienas nuo kito kableliu formato deskriptoriai (aprašai). Trumpumo dėlei juos toliau vadinsime tiesiog formatais. Formatų sąrašas pateikiamas arba tiesiogiai operatoriuose READ ir WRITE kaip tekstinė išraiška arba operatoriuje FORMAT. Yra *kartotiniai ir nekartotiniai* formatų deskriptoriai (formatai).

Kartotiniai *formatai* nurodo perduodamų iš failo Į/I sąrašo elementams duomenų pateikimo būdą ir atvirkščiai. Prieš kiekvieną *kartotinį formatą* gali būti parašytas ne didesnis kaip 255 jo pakartojimų skaičius. Skirtingų tipų duomenims perduoti naudojami šie *kartotiniai formata*i:

Formatas	**Duomenų tipas**
I*w*[.*m*]	Sveikasis
F*w*.*d*	Realusis įprastine forma
E*w*.*d*[E*e*]	Realusis eksponentine forma
G*w*.*d*[E*e*]	Realusis įprastine ar eksponentine forma
D*w*.*d*	Realusis dvigubojo tikslumo
A*w*	Tekstinis
Z*w*	Bet koks tipas šešioliktainėje skaičiavimo sistemoje
L*w*	Loginis

Čia I, F, E, G, D, A, L ir Z nurodo skirtingų tipų duomenų pateikimo būdą ir yra vadinami formatų kodais;

w - duomens išorinėje pateikimo formoje įvesties/išvesties lauko ilgis (pozicijų skaičius);

d - skaičiaus trupmeninės dalies skaitmenų kiekis įvesties/išvesties lauke;

m - skaitmenų kiekis sveikųjų skaičių išvesties lauke;

e - skaičiaus eksponentės skaitmenų kiekis išvesties lauke;

w, d, m ir *e* - sveikojo tipo konstantos be ženklo.

Nekartotiniais formatais valdomas pats duomenų perdavimas bei perduodami duomenys betarpiškai iš *formato* įrašui.

Formatas	**Paskirtis.**
eilutė	Tekstinės eilutės perdavimas į išvesties įrenginį
*n*H	Esančių už H *n* simbolių perdavimas į išvesties įrenginį
T*n*	Pozicijos žymeklio nustatymas *n*-oje įrašo pozicijoje
TL*n*	Pozicijos žymeklio perkėlimas per *n* įrašo pozicijų į kairę
TR*n*	Pozicijos žymeklio perkėlimas per *n* įrašo pozicijų į dešinę
*n*X	Pozicijos žymeklio perkėlimas per *n* įrašo pozicijų į dešinę
SP	Ženklo “+” rašymo prieš teigiamus skaičius režimo nustatymas
SS	Ženklo “+” rašymo prieš teigiamus skaičius režimo anuliavimas
S	Ženklo “+” rašymo prieš teigiamus skaičius standartinis režimas
/	Perėjimas į kitą įrašą
\	To paties įrašo tęsimas
:	Išvedimo nutraukimas, pasibaigus Į/I sąrašui
*k*P	Mastelio daugiklio *k* nustatymas F, E, D ir G formatams
BN	Režimo, ignoruojančio tuščius tarpus, nustatymas
BZ	Režimo, pakeičiančio tuščius tarpus nuliais, nustatymas

Formatų sąraše *kartotiniai ir nekartotiniai formatai* gali eiti vienas po kito pakaitomis. Keletą *formatų* galima sujungti į grupę skliaustais. Prieš kiekvieną grupę galima parašyti pakartojimų skaičių. Pavyzdžiui, formatų sąrašą (/I4,3F8.2/I4,3F8.2/) galime užrašyti (/2(I4,3F8.2/)). Tokių apskliaudimų gali būti ne daugiau kaip 16.

Vykdant formatinę duomenų įvestį/išvestį reikia turėti omenyje šiuos Į/I įpatumus:

1. Išvedant duomenis į ekraną ar spausdinimo įrenginį pirmas įrašo simbolis interpretuojamas kaip valdantysis simbolis. Jis nespausdinamas ir nerodomas ekrane. Visi tokie simboliai, išskyrus " 0, 1, ir + ", interpretuojami kaip intervalas. Valdantieji simboliai nurodo praleidžiamų eilučių kiekį:

Simbolis	***Veiksmas***
Intervalas	Pasistūmėjimas per vieną eilutę
0	Pasistūmėjimas per dvi eilutes
1	Pasistūmėjimas iki pirmos kito lapo eilutės (išvestyje į ekraną ignoruojamas)
+	Judėjimas blokuojamas. Galima atlikti dvigubą spausdinimą

Pavyzdys:

```
      WRITE(*,10)
10    FORMAT('Vardas ?')
```

Įvykdžius operatorių WRITE ekrane atsiras užrašas:

ardas ?

Raidė V interpretuojama kaip valdymo simbolis "intervalas". Šiuo atveju reikia parašyti FORMAT(' Vardas ?'), t.y. prieš V palikti bent vieną intervalą.

Jei duomenys išvedami į failą, pirmas simbolis jame rašomas. Jei vykdoma Į/I sąrašo valdoma įvestis/išvestis, tai pirmas įrašo simbolis nelaikomas valdymo ir priimamas kaip duomenų simbolis.

2. Jeigu įvesties ar išvesties operatoriuje yra Į/I sąrašas, tai formatų sąraše turi būti bent vienas kartotinis formatas.

3. Duomenų įvestyje tuščio tarpo simboliai prieš konstantas ignoruojami, kitose vietose jų interpretacija priklauso nuo operatoriaus OPEN parametro BLANK bei formatų BN ir BZ, veikiančių duotuoju metu. Laukas tik iš tuščio tarpo simbolių yra nulinis laukas.

4. Duomenų išvestyje jų reikšmės prisiglaudžia prie dešiniojo lauko krašto. Jei išvedami duomenys turi mažiau simbolių nei lauko ilgis, tai likusi lauko dalis užpildoma tuščio tarpo simboliais.

5. Jei duomenų išvestyje išvedamų simbolių kiekis viršija lauko ilgį arba eksponentė viršija nurodytą formatuose E*w.d*E*e* bei G*w.d*E*e* eksponentės lauko ilgį *e*, tai visas

laukas užpildomas simboliais * (žvaigždutė). Jei Realusis skaičius po dešimtainio taško turi daugiau skaitmenų nei nurodyta formate ar leidžia lauko ilgis, jis suapvalinamas.

6. Duomenų įvestyje formatu su kodais I, F, E, G, D, L ir Z laukų skyrikliu galima naudoti kablelį. Eilinių duomenų skaitymas pradedamas simboliu, einančiu po kablelio. Pavyzdžiui, jeigu operatoriumi

READ (*,'(BZ,3I5)') i,j,k

įvedama tokia eilutė

5, 3␣, 8␣␣ ,

tai kintamieji i, j ir k įgaus reikšmes 5, 30 ir 800. Čia simboliu "␣" pažymėti intervalai. Kablelio, kaip laukų skyriklio naudojimo dėka vartotojui nereikia rašyti priekinių intervalų bei nulių laukuose. Naudoti kablelį nerekomenduotina, jei formatų sąraše naudojami formatai T, TL, TR ir *n*X.

7. Kompleksinio tipo duomenims naudojami du formatai su kodais F, E, G ir D: pirmas - skaičiaus realiajai daliai, antras - menamajai. Be to, abu formatai nebūtinai turi būti vienodi ir gali būti atskirti nekartotiniu formatu.

Formatinės įvesties/išvesties operatoriuose formatų sąrašas kartu su įvesties/išvesties sąrašu apibrėžia perduodamų įrašų kiekį, dydį bei kiekvieno įrašo struktūrą. Duomenų pavidalą nulemia formatai. Formatų išsidėstymas apibrėžia įrašo struktūrą. Į/I sąraše pateikiami skaitomų ar rašomų duomenų vardai.

Jei įvesties/išvesties sąraše yra bent vienas elementas, tai formatų sąraše turi būti bent vienas kartotinis formatas. Jei įvesties/išvesties sąrašo nėra, formatų sąrašas gali būti tuščias ( ) arba turėti tik nekartotinius formatus. Jei formatų sąrašas tuščias, tai įvestyje praleidžiamas vienas įrašas, o išvestyje užrašomas tuščias įrašas.

Formatas I*w*[.*m*]

Formatas I*w*[.*m*] skirtas sveikojo tipo duomenims perduoti. Čia *w* nurodo *Į/I lauko* ilgį, t.y. pozicijų kiekį, skirtą skaičiaus skaitmenims ir ženklui užrašyti. Prieš teigiamus skaičius ženklas + nespausdinamas. Nebūtinas parametras *m* duomenų išvestyje nurodo, kiek pozicijų (skaičiuojant iš dešinės) turi būti užpildyta skaitmenimis. Jei skaičius turi mažiau *m* skaitmenų, tai jų priekyje rašomi nuliai. Įvesties metu užrašomas skaičius turi būti glaudžiamas prie dešiniojo lauko krašto. Laisvos pozicijos dešinėje traktuojamos kaip nuliai. Pavyzdžiui, jei kintamojo j reikšmė lygi 34, galima įvesti taip:

Įvesties laukas	Formatas	j reikšmė
␣34	I3	34
34	I2	34
␣␣␣34	I5	34

⎵⎵34⎵	I5	340
⎵⎵34	I3	3

Duomenų išvestyje skaitmenys spausdinami taip pat dešiniosiose nurodyto lauko pozicijose. Jei laukas per mažas, tai jame vietoj skaičiaus išvedamos žvaigždutės. Pavyzdžiui, jei kintamieji k, j, m ir n turi reikšmes 215, 3, 25498 ir 776, tai užrašyti operatoriumi

```
WRITE(*,'(4X,I3,2I4,I7.5)')  k, j, m, n
```

jie atrodys taip (čia ir vėliau ⎵ - tuščio tarpo simbolis):

⎵⎵⎵215⎵⎵⎵3****⎵⎵00776

Pirmus 3 tuščius tarpus palieka formatas 4X (pirma pozicija lieka valdymo simboliams, žr. 85 p.); k reikšmė 215 užima visas tris formatu I3 jai skirtas pozicijas; j reikšmė 3 užima tik vieną poziciją I4 lauko dešinėje, n reikšmė turi 5 skaitmenis; ji I4 lauke netelpa, todėl spausdinamos tik 4 žvaigždutės; m reikšmei skirtos I7.5 formatu septynios pozicijos, iš kurių penkios turi būti užpildytos skaitmenimis, todėl prieš 776 atsiranda dar du nuliai.

Formatai F*w.d*, E*w.d*[E*e*], D*w.d*

Šie formatai skirti realiesiems (R*4) ir dvigubojo tikslumo realiesiems (R*8) skaičiams perduoti. Skaičiui iš viso skiriama *w* pozicijų, įskaitant jo sveikąją, trupmeninę dalis, ženklą ir dešimtainį tašką, iš jų *d* pozicijų trupmeninei daliai ir *e* pozicijų skaičiaus eksponentei. Formatas F skirtas įprastiniam skaičiaus vaizdavimui, formatai E ir D eksponentine forma.

Įvesties lauke duomuo pateikiamas kaip realioji konstanta su eksponente arba be jos. Jei duomuo turi dešimtainį tašką, parametras *d* ignoruojamas, jei jis praleistas, tai *d* pirmų simbolių iš dešinės priskiriami trupmeninei daliai. Raidė E arba D gali būti praleista, jei po jos eksponentėje eina "+" arba "-" ženklas. Įvestyje formatų F*w.d*, E*w.d*[E*e*] ir D*w.d* veikimas identiškas (parametras *e* neveikia).

Pavyzdys

```
      READ(*,100) a, b, c, g
  100 FORMAT(F6.2, E10.3, F7.0, D12.4)
```

Įvesties laukas:

⎵⎵2154⎵-2475E-04⎵25.89⎵⎵⎵⎵⎵⎵1.235+3

Įvesties metu kintamasis a pagal formatą F6.2 iš pirmų šešių pozicijų įgis reikšmę 21.54. Dydžio b reikšmė bus $-2.475\cdot10^{-4}$, c įgaus reikšmę kaip ir parašyta įvesties

lauke - 25.89, čia dešimtainis taškas nėra praleistas, todėl *d* lygus 0 yra ignoruojamas, o g dydis bus lygus 1235.

Kaip matome iš šio pavyzdžio, tikrąsias įvedamų duomenų reikšmes, užrašytas įvesties lauke, galime nustatyti tik žinodami jų įvesties formatą. Jei programose naudojame formatinę duomenų įvestį, aiškumo dėlei patariame naudoti konstantas su dešimtainiu tašku ir vienodo ilgio įvesties laukus. Tai padės patiems lengviau perskaityti pradinių duomenų failus.

Išvestyje formatu F*w.d* duomuo išvedamas kaip realioji konstanta be eksponentės (įprasta forma). Išvesties lauke turi tilpti ženklas (būtinas neigiamiems skaičiams), dešimtainis kablelis, skaičiaus sveikojoji dalis bei *d* suapvalintos trupmeninės dalies skaitmenų. Jei duomuo netelpa *w* pozicijų lauke, spausdinama *w* žvaigždučių. Pavyzdžiui, kintamieji a = 42.3, b = -2.35936 ir c = 569.45 operatoriumi

```
WRITE(2, '(1X,F4.1, F9.4, F5.2)' ) a,  b,  c
```

bus išvesti taip:

```
␣42.3␣␣-2.3594*****
```

Šiuo atveju formatą F5.2 reikėtų pailginti bent viena pozicija, t.y. užrašyti F6.2, o kad "nesuliptų" su prieš tai einančiu skaičiumi - F7.2 ir daugiau. F formatą galima naudoti tais atvejais, kai žinomas išvedamų skaičių reikšmių intervalas ir galima nurodyti lauką, kuriame tilps visi išvedamo skaičiaus simboliai, t.y. vietoj skaičiaus negausime tik žvaigždutes.

Jei išvedame labai didelius ar mažus skaičius, patogiau juo užrašyti eksponentine forma (su dešimtaine eile). Tam naudojamas formatas E*w.d*. Čia *w* rodo pozicijų kiekį, kuris reikalingas skaičiui eksponentine forma užrašyti, įskaitant ženklą, dešimtainį tašką, mantisę, raidę E, eilės ženklą ir pačią eilę; *d* nurodo, kiek iš jų skirta mantisei. Eilei nurodyti standartiškai skiriamos dvi pozicijos. Jei eilei reikalingos trys pozicijos, praleidžiama E raidė. Eilės pozicijų kiekį galima atskirai nurodyti parametru *e*. Šiuo formatu išvedamos realiųjų skaičių reikšmės yra normalizuojamos, t.y. dešimtainis daugiklis parenkamas toks, kad sveikojoji dalis būtų lygi nuliui, o pirmas mantisės skaitmuo nelygus nuliui. Normalizuoto skaičiaus mantisė absoliučiuoju didumu visada didesnė arba lygi 0.1 ir mažesnė už 1. Pavyzdžiui, skaičiai 127.5 ir -0.00045987 formatu E11.5 bus išvesti: ␣.12750E+03 ir ␣-.45987E-03. Išvedant duomenis šiuo formatu reikia išlaikyti sąlygą $w \geq d+6$, nes be *d* pozicijų trupmeninei daliai reikia skirti dvi pozicijas skaičiaus ir eilės ženklui bei pačiai eilei ir po vieną poziciją dešimtainiui taškui bei raidei E (sveikojoji dalis lygi 0 yra praleidžiama).

Formatas D*w.d* analogiškas formatui E*w.d*. Jis skirtas dvigubojo tikslumo kintamiesiems ir skiriasi nuo E formato tuo, kad, užrašant konstantas, vietoj E rašoma raidė D.

Formatas G*w.d*[E*e*]

Formatas G*w.d* skirtas taip pat realiesiems duomenims perduoti. **Įvesties** operacijose jis analogiškas formatui F*w.d,* kur duomenys užrašomi kaip realiosios konstantos su eksponente arba be jos.

Išvestyje formatas G*w.d* interpretuojamas kaip F*w.d* arba E*w.d*, atsižvelgiant į išvedamo duomens reikšmę. Jei duomens reikšmė "telpa" į formatą F*w.d*, jis išvedamas tuo formatu, priešingu atveju - E*w.d* formatu. Žemiau pateikiama detali šio formato interpretacija, atsižvelgiant į duomens *y* reikšmę:

Duomens reikšmė	***Formatas***	***Interpretacija***
$y < 0{,}1$	G*w.d*	E*w.d*
$0{,}1 \le y < 1$	G*w.d*	F(*w*-4).*d*,4X
$1 \le y < 10$	G*w.d*	F(*w*-4).(*d*-1),4X
$10^{(d-2)} \le y < 10^{(d-1)}$	G*w.d*	F(*w*-4).1,4X
$10^{(d-1)} \le y < 10^{(d)}$	G*w.d*	F(*w*-4).0,4X
$10^{(d)} \le y$	G*w.d*	E*w.d*

Formatą G*w.d* patartina naudoti programos derinimo metu, kada nėra žinomas išvedamų duomenų reikšmių intervalas.

Formatas A[*w*]

Formatas A[*w*] skirtas bet kokio tipo duomenims perduoti simboliniu pavidalu. Perduodamų simbolių skaičius priklauso nuo lauko pločio *w* ir atitinkamo Į/I sąrašo elemento ilgio *len*. Jei parametras *w* yra praleistas, tai perduodamų simbolių kiekis (lauko plotis) laikomas lygus atitinkamo Į/I sąrašo elemento ilgiui *len*.

Įvestis. Jei $w \ge len$, tai skaitoma *len* simbolių iš dešinės, jei $w < len$, tai perskaitoma ir perduodama elementui *w* simbolių, o kiti *len*-*w* elemento baitai užpildomi tuščio tarpo simboliais.

Išvestis. Jei $w > len$, tai išvesties lauke bus *w*-*len* tuščių tarpų ir po jų atitinkamo Į/I sąrašo elemento *len* simbolių. Jei $w \le len$ tai išvesties lauke bus *w* kairiųjų atitinkamo Į/I sąrašo elemento simbolių.

1 pavyzdys

```
CHARACTER *10 txt
READ(*,'(A15)') txt
```

Jeigu klaviatūra bus surinkti šie 13 simbolių: ABCDEFGHIJKLM, tai įvedant tekstinė konstanta bus papildyta dar dviem tarpo simboliais ABCDEFGHIJKLM␣␣ ir kintamajam txt bus perduota 10 simbolių iš dešinės, t.y. - FGHIJKLM␣␣.

2 pavyzdys

```
      CHARACTER *4 a/'4567'/ , b/'ABBA'/
      CHARACTER *9 c/'Elementas'/
      OPEN(UNIT=3, FILE='aaaa.txt', FORM='FORMATTED')
      ...
      WRITE(3,100) a, b, c
100 FORMAT('0',A6,A5,A12)
```

Vykdant išvesties operatorių WRITE, į failą aaaa.txt bus išvestos kintamųjų a, b, ir c reikšmės formatu, nurodytu operatoriumi FORMAT su žyme 100:

␣␣4567␣ABBA␣␣␣Elementas

Jei formate A lauko ilgis *w* yra praleidžiamas, tai jis lygus faktiškam išvedamo tekstinio kintamojo ilgiui *len* .

Formatas Z[*w*]

Formatas Z[*w*] skirtas bet kokio tipo duomenims perduoti šešioliktainiu pavidalu, t.y. išvesties įrenginyje duomenys bus išreikšti šešioliktainiais simboliais (0-9 ir A-F). Kiekvienas toks simbolis vidinėje pateiktyje atitiks 4 bitus, nes vienas šešioliktainis simbolis išreiškiamas 4-iais dvejetainiais simboliais. Tai reiškia, kad kiekvieno baito turinys formatu Z bus pateiktas dviem šešioliktainiais simboliais. Pavyzdžiui, simbolis "m", kurio ASCII kodas yra 01101101 bus pateiktas kaip 6D, nes dvejetainius simbolius 0110 atitinka šešioliktainis 6 ir simbolius 1101 atitinka šešioliktainis D. INTEGER*2 tipo duomuo 27380, vidinėje pateiktyje kompiuterio atmintyje bus saugomas dvejetainiu skaičiumi 0110101011110100, o formatu Z jis bus pateiktas šešioliktainiais simboliais 6AF4.

Parametras *w* nurodo perduodamų šešioliktainių simbolių kiekį. Jei jis nenurodytas, pagal nutylėjimą jis laikomas lygus 2**len*, kur *len* - įvesties/išvesties elemento ilgis baitais. Pavyzdžiui, INTEGER*4 tipo duomenys bus išreikšti aštuoniais šešioliktainiais simboliais. Išvestyje, jei $w > 2*len$, laisvos pozicijos kairėje užpildomos nuliais, jei $w < 2*len$, kairieji simboliai neišvedami. Įvesties metu šešioliktainiai simboliai po du talpinami viename atminties baite. Duomens reikšmė atmintyje priklauso nuo atitinkamo įvesties/išvesties sąrašo elemento ilgio. Jeigu $w>2*len$, tai kairieji įvesties lauko simboliai neįvedami į atmintį, o jei $w < 2*len$, tai duomuo iš kairės papildomas šešioliktainiais nuliais. Tušti tarpai įvesties lauke interpretuojami kaip nuliai.

1 pavyzdys

```
      CHARACTER *2 alpha
```

```
INTEGER*2               num
alpha = 'YZ'
num   = 4096
…
WRITE( *, '(1X, Z, 1X, Z2, 1X, Z6)' ) alpha, alpha, alpha
WRITE( *, '(1X, Z, 1X, Z2, 1X, Z6)' ) num, num, num
…
END
```

Įvykdžius šią programą, ekrane bus išvesta:

```
 595A 5A 00595A
 1000 00 001000
```

2 Pavyzdys

Tarkime, kad įvesties lauke užrašyta šešioliktainė konstanta 595A (simbolių "YX" ASCII šešioliktainis kodas) ir atitinkamas įvesties sąrašo elementas yra CHARACTER*2 tipo. Tada šis įrašas bus perskaitytas taip:

Formatas	Reikšmė
Z	YZ
Z2	0Y
Z6	YZ

Kompleksiniams skaičiams reikia naudoti du Z formatus, pirmas iš kurių skirtas realiajai daliai ir antras menamajai kompleksinio skaičiaus daliai.

Formatas L*w*

Formatas L*w* skirtas loginiams duomenims perduoti. Duomenys **įvesties** lauke užima *w* pozicijų, kuriose rašomi arba pilni žodžiai TRUE ar FALSE, arba tik raidės T ar F, arba sutrumpinti žodžiai (pvz., TR, FAL). Abiejose šių simbolių pusėse galimi taškai ar tušti tarpai. Kiti simboliai yra ignoruojami. **Išvesties** laukas susideda iš *w*-1 tuščių tarpų ir raidės T ar F atsižvelgiant į tai, ar loginio duomens reikšmė "tiesa" ar "melas".

Formatai *eilutė* ir *n*H

Formatai *eilutė* ir *n*H naudojami simbolinių konstantų reikšmėms išvesti. Konstantos gali būti užrašytos dvejopai: naudojant kabutes ar apostrofus, kaip įprasta FORTRANe, arba kaip holeritinės konstantos, kur H raidė rodo šios konstantos

požymį, o teigiamas natūrinis skaičius *n* - po jos esančios tekstinės konstantos simbolių kiekį. Formatai *eilutė* ir *n*H nenaudojami su operatoriumi READ.

Pavyzdys

```
C       Čia trys WRITE operatoriai išveda ABA'DEF
C       Priekinis tuščias tarpas konstantoje laikomas valdymo
C       simboliu
        WRITE(*, 750)
750             FORMAT(' ABA''DEF' )                    ! formatas eilutė
        WRITE(*, 751)
751             FORMAT( 8H ABA'DEF )                    ! formatas nH
        WRITE(*, '(" ABA""DEF")' )              ! formatas eilutė
C       Kitas  WRITE taip pat išveda ABA'DEF. Sąrašo valdomoje
C       išvestyje valdymo simbolis nereikalingas
        WRITE(*, *) 'ABA''DEF'
```

Formatai T*n*, TL*n*, TR*n* ir *n*X

Formatai T*n*, TL*n*, TR*n* ir *n*X naudojami nustatyti arba pakeisti poziciją įraše.

Formatas T*n* nurodo poziciją, nuo kurios turi būti pradėti duomenų mainai.

Formatai TR*n* ir *n*X nurodo, kiek pozicijų reikia praleisti skaitant įrašą duomenų įvestyje ir kiek pozicijų užpildyti tuščio tarpo simboliais duomenų išvestyje.

Formatas TL*n* nurodo, kad duomenų mainus reikia tęsti grįžus *n* pozicijų į kairę.

Įvestyje formatai T*n* ir TL*n* suteikia galibybę tuos pačius įrašų laukus įvesti keletą kartų.

Pavyzdys

```
        WRITE(*, 100)
        WRITE(*, 200)
100     FORMAT(10X, '1', 9X, '2' )
200 FORMAT(1X, '-------------------',TL15, '+', 3(4X, '+') )
```

Displėjaus ekrane bus išvesta:

```
                1                   2
- - - +- - - - +- - - - +- - - - +
```

Formatai S, SP ir SS

Formatai S, SP ir SS valdo ženklo "+" išvedimą teigiamiems skaičiams. Standartinis režimas pagal nutylėjimą arba formatu S teigiamus skaičius išveda be pliuso ženklo. Toks režimas veiks iki bus sutiktas SP formatas. Jis nurodo, kad visi kiti teigiami skaičiai bus išvedami su "+" ženklu. Formatas SS panaikina SP veikimą. Duomenų įvestyje formatai S, SP ir SS ignoruojami.

Pavyzdys

```
      INTEGER*2 j
C     Žemiau esantys operatoriai užrašys:
C     254  +254   254  +254   254
      j = 254
      WRITE( *, 100) j, j, j, j, j
  100 FORMAT(1X, I5, SP, I5, SS, I5, SP, I5, S, I5 )
```

Formatas "/ "

Formatas " / " (*slash*) nurodo įrašo pabaigą. Po šio simbolio duomenys bus rašomi į naujo įrašo (eilutės) pradžią arba skaitomi iš sekančio įrašo pradžios. Pavyzdžiui formatas (///F10.3, A15/5X, I6, E12.4) apibrėžia, kad pirmos trys eilutės bus praleistos (pirmi trys tušti įrašai), o duomenys bus dviejuose įrašuose.

Formatas " \ "

Formatas " \ " (*backslash*) naudojamas tik išvedant duomenis į spausdintuvą arba ekraną. Paprastai atlikus formatų sąrašu valdomus duomenų mainus, suformuojamas įrašo pabaigos požymis ir kitas įvesties/išvesties operatorius duomenų mainus vykdo nuo naujo įrašo pradžios. Jei formatų sąraše paskutiniuoju nurodytas " \ ", įrašo pabaigos požymis neformuojamas ir kitas įvesties/išvesties operatorius duomenų mainus vykdo tame pačiame įraše (eilutėje). Šis mechanizmas plačiai naudojamas įvedant dialoginius duomenis, kada ekrane pateikiama nuoroda ką įvesti, o renkami klaviatūra duomenys matomi šalia toje pačioje eilutėje.

Pavyzdys

```
      WRITE(*, '(2X,A\)')   'Mazgų skaičius: '
      READ(* ,*) m
     WRITE(*, '(2X,A\)')   Elementų skaičius: '
      READ(* ,*) n
```

Ekrane matysime, pavyzdžiui, tokį vaizdą:
Mazgų skaičius: 24
Elementų skaičius: 50

Formatas " : "

Formatas " : " naudojamas nutraukti formatų sąrašo peržiūrą, pasibaigus Į/I sąrašui. Jis patogus tuo atveju, kai paruošiamas bendras formatas, o konkrečiu atveju Į/I sąraše būna mažiau elementų nei deskriptorių formatų sąraše. Šiuo atveju nebus išvedamos nereikalingos tekstinės konstantos.

Pavyzdys

```
C       Šiame pavyzdyje pirmu WRITE operatoriumi išvedamos dviejų
C       kintamųjų  a ir b reikšmės, o antruoju WRITE - visų keturių.
C
        DATA a, b, c, d / 4.2, 12.8, 0.25, -45.3 /
        WRITE(*,100) a, b
        WRITE(*,100) a, b, c, d
  100   FORMAT('  A =', F5.2 , '  B =', F5.2 , : , '  C =', F5.2,'  D =', F6.2
        END
```

Ekrane matysime:

A = 4.20 B =12.80
A = 4.20 B =12.80 C = .25 D =-45.30

Jei nebūtų parašytas formatas ":", tai pirmuoju atveju turėtume:

A = 4.20 B =12.80 C =␣␣␣␣␣D =␣␣␣␣␣

Formatai BN ir BZ

Formatai BN ir BZ naudojami tuščių tarpų interpretavimo būdui nurodyti skaitinių duomenų įvestyje.

BN - nurodo, kad tušti tarpai įvesties lauke ignoruojami. Laukas vien tik iš tuščių tarpų atitinka nulinę reikšmę.

BZ - nurodo, kad tušti tarpai įvesties lauke laikomi nuliais.

Kiekvieno formatinio operatoriaus vykdymo pradžioje pagal nutylėjimą veikia formatas BN arba operatoriuje OPEN parametru BLANK (pagal nutylėjimą tušti tarpai ignoruojami) nustatytas režimas.

Pavyzdys

```
        READ( *, '(I5)') j
```

Sakykime, kad skaitomas įrašas 708␣␣. Jeigu veikia BN formatas, kintamasis j įgaus reikšmę 708, jei BZ formatas - reikšmę 70800.

Formatas *k*P

Formatas *k*P nurodo mastelio daugiklį *k*, keičiantį realiųjų duomenų, įvedamų F, E, G, ir D formatais, reikšmes. Mastelio daugiklio veikimas galioja visiems už jo esantiems formatams, kol nebus sutiktas kitas mastelio daugiklio formatas. Kiekvieno formatinio Į/I operatoriaus veikimo pradžioje šis daugiklis lygus nuliui.

Įvedami realieji duomenys neeksponentine forma F, E, D ar G formatais dauginami iš $10^{-k.}$ Jei įvedamas duomuo užrašytas eksponentine forma, $k = 0$. Duomenų išvestyje F formatu jie dauginami iš 10^{k}. Naudojant E ir D formatus duomens reikšmė nekinta, nes mastelio daugiklis šiuo atveju pakeičia dešimtainio kablelio vietą ir eksponentės reikšmę. Išvedant duomenis G formatu, mastelio daugiklis ignoruojamas, jei duomens reikšmė yra intervale, kuriam galioja F formatas, o kitais atvejais jis veikia kaip E formatui.

1 pavyzdys

```
C       Šioje programoje mastelio daugiklis vartojamas įvestyje.
        DO i = 1, 4
              READ( *, 100) a, b, c
          WRITE(*,200) a, b, c
        END DO
100 FORMAT(F10.6,1P, F10.6,-2P,F10.6)
200 FORMAT(3F11.3)
        END
```

Tarkime, kad cikle įvedami šie duomenys:

```
 48210000   48210000   48210000
-   48.21      48.21      48.21
 48.21E0    48.21E0    48.21E0
 48.21E3    48.21E3    48.21E3
```

Programos rezultatas bus displėjaus ekrane išvesti tokie skaičiai:

```
    48.210      4.821   4821.000
    48.210      4.821   4821.000
    48.210     48.821     48.210
48210.000  48210.000  48210.000
```

2 pavyzdys

```
C       Šioje programoje mastelio daugiklis vartojamas išvestyje.
```

```
      s = 48.21
      WRITE(*, 100) s, s, s, s, s,s
100 FORMAT(1X,F9.4,E11.4E2,2P,F9.4,E11.4E2,-2P,F9.4,E11.4E2)
      END
```

Programos rezultatas bus displėjaus ekrane išvesti tokie skaičiai:

␣␣48.2100␣␣.4821E+024821.0000␣48.210E+00␣␣␣␣..4821␣␣.0048E+04

5.10. SĄRAŠO VALDOMA ĮVESTIS/IŠVESTIS

Jei operatoriuose PRINT, READ ar WRITE formato identifikatoriumi yra simbolis " * " (žvaigždutė), tai duomenų, perduodamų iš failo Į/I sąrašo elementams ir atvirkščiai, pateikimas priklauso nuo Į/I sąrašo elemento tipo. Tokie operatoriai vadinami sąrašo valdomos Į/I operatoriais. Jie dar vadinami įvesties/išvesties laisvu formatu operatoriais.

Įvestyje duomenys skaitomame įraše pateikiami atitinkamo tipo konstantomis, atskirtomis viena nuo kitos tuščiais tarpais arba kableliu. Konstantos tipas turi sutapti su atitinkamu įvesties sąrašo elementu. Išskyrus tekstines, jos negali turėti tuščio tarpo simbolių. Įvedami duomenys gali būti talpinami keliuose įrašuose (eilutėse), bet kiekviena skaitinė konstanta, išskyrus kompleksinę, turi būti užrašyta toje pačioje eilutėje. Kompleksinė konstanta (sveikųjų ar realiųjų konstantų, atskirtų kableliu, pora, paimta į skliaustus) gali būti išdėstyta dviejuose nuosekliuose įrašuose: realioji dalis pirmame, o menamoji - antrame įraše. Skiriamasis kablelis gali būti bet kuriame šių įrašų. Tekstinė konstanta (simbolių seka, paimta į kabutes ar apostrofus) taip pat gali užimti keletą įrašų. Jei tekstinės konstantos ilgis mažesnis už atitinkamo įvesties sąrašo elemento ilgį, tai įvesties metu ji papildoma iš dešinės tuščiais tarpais. Jei konstanta ilgesnė už įvesties sąrašo elementą, tai ji sutrumpinama iš dešinės.

Duomenų kiekis neturi būtinai sutapti su įvesties sąrašo elementų skaičiumi. Tiems įvesties sąrašo elementams, kurių reikšmių nereikia perskaityti iš einamojo įrašo, galima naudoti nesamo duomens pažymėjimą. Nesamas duomuo sąrašo pradžioje pažymimas vienu kableliu, viduje - dviem kableliais, įrašo gale ženklu "/". Simbolis "/" nutraukia įvesties operatoriaus veikimą, tad likusių įvesties sąrašo elementų reikšmės nepakinta. Besikartojantiems vienodiems arba nesamiems duomenims galima naudoti kartojimo daugiklį *k**, kur *k* - sveikojo tipo teigiama konstanta.

Išvesties įrašo ilgis lygus 80 simbolių. Kiekvienas išvesties įrašas pagal nutylėjimą prasideda tuščio tarpo valdymo simboliu. Duomenys įraše atskiriami vienu ar keliais tarpais. Išvesties sąrašo elementų reikšmės perduodamos fiksuotu formatu, atsižvelgiant į išvedamo elemento tipą. Loginiai duomenys rašomi formatu L1, sveikojo tipo - formatu I11. Realiųjų duomenų formatas priklauso nuo jų reikšmės. Jei $1 \le$

reikšmė ≤ 10^7 paprastojo tikslumo elementas bus išvedamas formatu 0PF15.6, dvigubojo tikslumo - formatu 0PF24.15. Jeigu *reikšmė* < 1 arba *reikšmė* > 10^7, tai naudojamas formatas1PE15.6E2 paprastajam tikslumui ir 1PE24.15E3 dvigubojo tikslumo realiesiems duomenims. Tekstinio tipo elementai išvedami formatu A. Loginių duomenų reikšmės išvedamos simboliais T ir F.

5.11. VARDŲ SĄRAŠO VALDOMA ĮVESTIS/IŠVESTIS

Vardų sąrašu valdoma Į/I yra patogus duomenų skaitymo ar rašymo į failą bei displėjų būdas. Apibrėžę vieną ar kelis kintamuosius vardų sąrašo grupėje, toliau juos skaityti ar rašyti galime Į/I operatoriais nekartodami jų vardų Į/I sąraše.

Duomenys bus išvedami stulpeliu, nurodant jų vardus, po kurių eina lygybės ženklas ir reikšmė. Išvedimas pradedamas simboliu "&" (ampersedas) ir po jo einančiu vardų sarašo grupės identifikatoriumi. Toliau kiekvienoje eilutėje (įraše) užrašomas kintamojo (masyvo) vardas, lygybės simbolis ir reikšmė (reikšmės). Išvedimas baigiamas simboliu "/". Duomenys išvedami sąrašu valdomos Į/I formatu (žr. ankstesnį skyrelį), tik tekstinės reikšmės paimamos į apostrofus. Analogiškai užrašomi duomenys faile, jei mes norime juos perskaityti su vardų sąrašu.

Vardų sąrašo grupė yra sukuriama NAMELIST operatoriumi. Jo Sintaksė

NAMELIST /*namelist*/ *variablist*

Čia *namelist* yra grupės identifikatorius, o *variablist* - kintamųjų ar masyvų vardų sąrašas.

Išvardintų sąraše kintamųjų reikšmės užrašomos į failą ar ekraną bei skaitomos operatoriais WRITE ir READ, vietoj formato nurodant grupės identifikatorių:

WRITE(*unitspec*, [NML=]*namelist*)
READ(*unitspec*, [NML=]*namelist*)

NML nėra būtinas; jis reikalingas, jei naudojami ir kiti raktažodžiai (END=, ERR=).

Pateikiamame pavyzdyje yra deklaruojamas kintamųjų vardų sąrašas, jiems suteikiamos reikšmės ir išvedamos į ekraną vardų sąrašu. Po to dalies kintamųjų reikšmės perskaitomos iš failo namelst.dat ir visa kintamųjų grupė išvedama į ekraną.

```
INTEGER          inn1*1,inn2*2,inn4*4, array(3)
LOGICAL          log1*1,log2*2,log4*4
REAL                   real4*4,real8*8
COMPLEX      z8*8,z16*16
CHARACTER    char1*1,char10*10

OPEN(unit=4,file='namelst.dat')
```

```
NAMELIST /example/ inn1,inn2,inn4,log1,log2,log4,
                   real4,real8,z8,z16,char1,char10,array

inn1   = 11
inn2   = 12
inn4   = 14
log1   = .TRUE.
log2   = .TRUE.
log4   = .TRUE.
real4  = 24.0
real8  = 28.0D0
z8     = (38.0, 0.0)
z16= (-316.0D0, 0.0D0)
char1  = 'A'
char10         = '0123456789'
array(1) = 41
array(2) = 42
array(3) = 43

WRITE(*,example)

READ(4,example)

WRITE(*,example)
END
```

Pirmasis WRITE(*,example) operatorius į ekraną išves tokią informaciją:

```
&EXAMPLE
 INT1 =         11
 INT2 =         12
 INT4 =         14
 LOG1 =  T
 LOG2 =  T
 LOG4 =  T
 REAL4 =        24.000000
 REAL8 =        28.000000000000000
 Z8 =          (38.000000,0.000000E+00)
```

```
 Z16 =       (-316.000000000000000,0.000000000000000E+000)
 CHAR1 = 'A'
 CHAR10 = '0123456789'
 ARRAY =          41         42         43
 /
```

Tegu faile namelst.dat buvo užrašyti tokie duomenys:

```
&EXAMPLE
 INT1 = 99
 array(1) = 99
 real4=99.9
 char1='Z'
 char10(5:9)='GrUMp'
 log1=F
 /
```

Antrasis WRITE(*,example) operatorius į ekraną išves informaciją

```
&EXAMPLE
 INT1 =          99
 INT2 =          12
 INT4 =          14
 LOG1 =  F
 LOG2 =  T
 LOG4 =  T
 REAL4 =        99.900000
 REAL8 =        28.000000000000000
 Z8 =           (38.000000,0.000000E+00)
 Z16 =       (-316.000000000000000,0.000000000000000E+000)
 CHAR1 = 'Z'
 CHAR10 = '0123GrUMp9'
 ARRAY =          99         42         43
 /
```

5.12. NEFORMATINĖ ĮVESTIS/IŠVESTIS

Ankstesniuose skyreliuose aptarta formatinė duomenų įvestis/išvestis naudojama pradiniams duomenims įvesti bei rezultatams išvesti. Įvairiems tarpiniams duomenims saugoti bei perduoti kitiems programos moduliams naudingiau naudoti neformatinius arba dvejetainius failus ir organizuoti neformatinę įvestį/išvestį. Neformatiniuose

dvejetainiuose failuose duomenys rašomi dvejetaine forma taip kaip jie saugomi kompiuterio atmintyje, jie nepervedami į išorinį vaizdavimo būdą, todėl neformatinė Į/I yra greitesnė ir leidžia perduoti bet kurį iš 256 ASCII kodų lentelės simbolį.

Neformatinei Į/I atidaromas neformatinis arba dvejetainis failas, pavyzdžiui:

```
OPEN(UNIT=1, FILE='pav2.dat', FORM='UNFORMATTED')
```

Rašoma ir skaitoma operatoriais WRITE ir READ, nurodant įrenginio identifikatorių ir, jei reikia, kitus parametrus, išskyrus formatą. Žemiau pateikiame du programų pavyzdžius, kuriuose naudojami neformatiniai nuosekliosios bei tiesioginės kreipties failai.

1 pavyzdys

```
C  Šiame pavyzdyje duomenys rašomi į neformatinį nuosekliosios kreipties failą,
C  tada vėl perskaitomi ir parodomi ekrane. Šio failo struktūra parodyta
C  5.2 skyrelyje.
C  Dešimtainis -1 atmintyje saugomas šešioliktainiu FF FF FF FF
C
      CHARACTER          xyz(3)
      INTEGER*4     idata(35)
      INTEGER*1          iras1(140), iras2(3)
      DATA   idata / 35 * -1 /, xyz / 'X', 'Y', 'Z' /
C
C  Atidaromas failas ir į jį užrašomas masyvas idata
C  ( 140 baitų ilgio) ir masyvas xyz ( 3 baitų ilgio)
C
      OPEN(3, FILE='nefrm1.dat', FORM='UNFORMATTED')
      WRITE (3) idata
      WRITE (3) xyz
      CLOSE (3)
C
C  Skaitome failą ir išvedame jo turinį į ekraną
C
      OPEN(3, FILE='nefrm1.dat', FORM='UNFORMATTED')
     READ (3) iras1
     READ (3) iras2
     WRITE (*, 100) iras1, iras2
     WRITE (*, 101) iras2
```

```
  100      FORMAT(2X, 'iras1 & iras2 16-je sistemoje:'/ 35(1X,4Z)/1X,
     +   3Z)
  101      FORMAT(2X, 'iras2 simboliniame formate:  ' ,3(A1,X1))
           CLOSE(3)
           END
```

2 pavyzdys

```
C          Šiame pavyzdyje duomenys rašomi į neformatinį tiesioginės
C          kreipties failą, tada vėl perskaitomi ir parodomi ekrane
C
           INTEGER*4          intd
           LOGICAL*4          logd
           CHARACTER*6     chard
C                  Atidaromas failas ir į jį užrašomi duomenys
           OPEN(3, FILE='nefrm2.dat', FORM='UNFORMATTED',
      +           ACCESS='DIRECT', RECL=10)
           WRITE (3, REC=3) .TRUE., 'abcdef'
           WRITE (3, REC=1) 2094
           CLOSE (3)
C
C          Skaitome failą ir išvedame jo turinį į ekraną
C
           OPEN(3, FILE='nefrm2.dat', FORM='UNFORMATTED',
       +           ACCESS='DIRECT', RECL=10)
         READ (3, REC=1) intd
         READ (3, REC=3) logd, chard
         WRITE (*, 100) intd
         WRITE (*, 101) logd, chard
    100 FORMAT(2X, 'Įrašas 1 : ', I5)
    101 FORMAT(2X, 'Įrašas 2 : ', L6, 4X, A6 )
           CLOSE(3)
           END
```

Tam tikra neformatinės įvesties/išvesties atmaina galime laikyti dvejetainius įrašus. Dvejetainė įvestis/išvestis apibrėžiama failo atidarymo operatoriuje OPEN parametru FORM='BINARY'. Kaip jau buvo minėta 5.2 skyrelyje, dvejetainis failas nėra struktūrizuotas, neturi įrašus skiriančių baitų ar kitų specialių simbolių. Duomenys skaitomi ar rašomi į dvejetainį failą, nekeičiant nei jų formos, nei ilgio. Jame esančių

baitų seka visiškai atitinka Į/I sąraše išvardytų kintamųjų baitų seką kompiuterio atmintyje. Žemiau pateikiamas pavyzdys iliustruoja dvejetainių failų naudojimą.

```
C       Ši programa naudoja dvejetainį nuosekliosios kreipties failą.
C       Į jį rašomas sakinys "ka tu matai, ta tu ir gauni!" bei po jo einantys 4
C       simboliai @  ( jo ASCII kodas yra dešimtainis 64 arba šešioliktainis 40)
C       Tai Microsof FORTRAN išplėtimas, kitos FORTRANo realizacijos gali jo
C       ir neturėti
        INTEGER*1       aa(4)
        INTEGER                 intd(3)
        CHARACTER               skait(32), chard*4
      DATA   aa / 4 * 64 /
      DATA   chard / ' ta ' / , intd / 'ka t', 'u ma', 'tai,'/
C       Sukuriame failą ir užrašome sakinį
C
        OPEN(UNIT=2, FILE='binar.dat', FORM='BINARY')
        WRITE(2) intd, chard
        WRITE(2) 'tu ', 'ir gauni!'
        WRITE(2) aa
        CLOSE(2)
C
C       Skaitome duomenis iš failo ir išvedame juos į ekraną
C
        OPEN(UNIT=2, FILE='binar.dat', FORM='BINARY')
        READ(2) skait
        WRITE(*,'(1X,32A1)') skait
        END
```

5.13. VIDINIS FAILAS

Vidinį failą sudaro formatiniai įrašai. Jis skirtas perduoti duomenis iš vienos atminties srities į kitą. Tai atliekama tik formatinės nuosekliosios kreipties Į/I operatoriais READ ir WRITE (sąrašu valdoma įvestis/išvestis yra negalima)

Kai vidinis failas yra simbolinis kintamasis, simbolinio masyvo elementas, paeilutė ar nesimbolinis masyvas, tai jį sudaro vienas įrašas, kurio ilgis sutampa su atitinkamo simbolinio kintamojo, simbolinio masyvo elemento, paeilutės ar nesimbolinio masyvo ilgiu. Jei failas yra simbolinis masyvas, tai kiekvienas jo elementas yra failo įrašas.

Vidiniai failai dažnai naudojami skaitinio tipo kintamiesiems pervesti į simbolinį tipą ir atvirkščiai. Pavyzdžiui, grupė išorinių failų turi vardus file001, file002 ir t.t. Suformuoti tokį failo vardą pagal jo eilės numerį galima tokiu būdu:

```
      CHARACTER fvard*7, eilnr*3
            WRITE(*,'(A\)') ' Iveskite failo eiles Nr. :'
            READ(*,*) id
C           Rašome įvestą skaičių į vidinį failą eilnr
            WRITE(eilnr, '(I3.3)') id
C           Formuojame failo vardą
            fvard = 'file'//eilnr
C           Atidarome išorinį failą
            OPEN(1, FILE=fvard)
      END
```

Kitas pakankamai dažnas vidinio failo naudojimo atvejis gali būti formatų aprašuose, kai norime tuo pačiu formatu užrašyti skirtingus duomenų kiekius, t.y. kartotinių formatų kartojimo daugiklius išreikšti ne konstantomis, o kintamaisiais. Šiuo atveju formatui aprašyti reikia naudoti simbolinius kintamuosius ir kartojimo skaičius įrašyti į juos kaip į vidinius failus. Žemiau pateiktame programos fragmente formuojamas skirtingo realiųjų ir sveikųjų skaičių išvedimo formatas, išreikštas 16 baitų ilgio kintamuoju form1, kuriame kartojimo koeficientai užima 5-6 ir 12-13 pozicijas. Prieš kiekvieną Į/I operaciją jose užrašomos išvedamų/įvedamų realiųjų ir sveikųjų skaičių reikšmės.

```
...
form1 = '(5X,xxF7.2,yyI4)'
...
n_rel = 3
n_int = 5
WRITE(form1(5:6), '(I2)') n_real
WRITE(form1(12:13), '(I2)') n_int
WRITE(*,form1) (rmas(i),i=1,n_real), imas(j),j=1,n_int)
...
```

Skaičių pervedimas iš simbolinio pavidalo į skaitinį tipą atliekamas skaitymu iš vidinio failo. Pavyzdžiui, programos fragmentas

```
...
txt = '0123456789'
READ(txt(4:5), '(A2)') m
WRITE(*,*) m
```

...

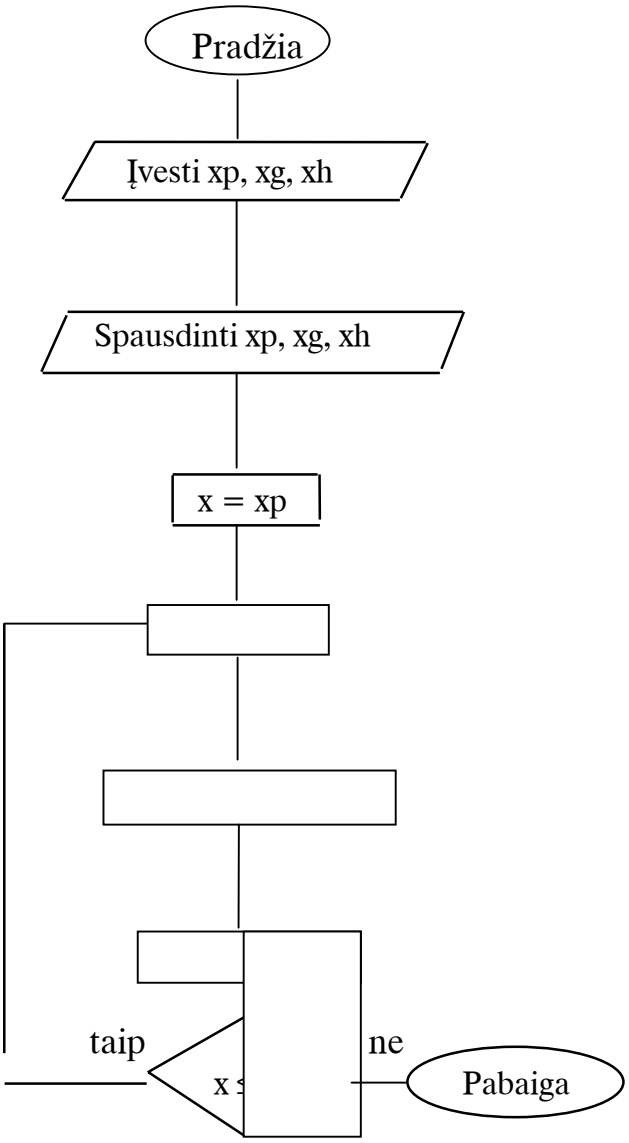
ekrane pateiks kintamojo m reikšmę, lygią 34.

6. PROGRAMŲ PAVYZDŽIAI

Neįmanoma išmokti programuoti pačiam nerašant programų. Todėl jau pirmą kartą perskaičius knygelę siūlome pabandyti užprogramuoti bent keletą nesudėtingų uždavinių. Kad pradžia būtų lengvesnė, šiame skyriuje pateiksim kelis programų pavyzdžius su išsamiais paaiškinimais.

Programų pavyzdžiai išdėstyti pagal sudėtingumą: nuo paprasčiausio sudėtingesnių link, operuojančių masyvais ir universaliais įvesties/išvesties operatoriais.

Programa 1. Turim suskaičiuoti funkcijos $y = \sin x$ reikšmes, kai argumentas x kinta nuo pradinės reikšmės xp iki galinės reikšmės xg pastoviu žingsniu xh.
Programos algoritmas:



Skaičiavimų pradžia.

Šių kintamųjų reikšmės turi būti nuskaitytos, pavyzdžiui, iš ekrano į kompiuterio atmintį.

Kontrolinis įvestų reikšmių spausdinimas: kad būtų išvengta dažnų klaidų įvedimo metu.

x priskirti reikšmę xp. Pradėsime skaičiuoti nuo argumento pradinės reikšmės.

Funkcijos skaičiavimas.

Argumento ir gautą funkcijos reikšmes išvesti iš operatyviosios atminties į, pavyzdžiui, kompiuterio ekraną.

x priskirti žingsniu didesnę reikšmę.

Patikrinti, ar x neviršija galinės reikšmės. Jei taip - tęsti skaičiavimus, jei ne - uždavinio pabaiga.

Kadangi tai pirmoji mūsų programa, naudosim paprasčiausias operatorių formas: duomenis įvesim iš ekrano, gautus rezultatus išvesim irgi į ekraną laisvuoju formatu, o kintamųjų tipus apibrėšim nutylėjimo principu.

Programos (įvardinkim ją vardu p1) tekstas su komentarais:

```
C           Funkcijos skaičiavimo programa
      PROGRAM p1
C           Duomenų įvedimas ir kontrolinis spausdinimas
      WRITE(*,*) ' Įveskite xp, xg, xh'
      READ(*,*)   xp, xg, xh
      WRITE(*,*) ' Duomenys', xp, xg, xh
C           Funkcijos skaičiavimo ir rezultatų spausdinimo
C           ciklas
      x = xp
      WRITE(*,*) ' Rezultatai'
   10 y = SIN( x )
      WRITE(*,*) ' x=', x, '   y=', y
      x = x+xh
      IF( x .LE. xg )  GO TO 10
      END
```

Surinkus programos tekstą, ji kompiliuojama, komponuojama ir paleidžiama vykdyti (žr. 10-ą skyrių). Programos p1 pirmasis WRITE operatorius ekrane parodo užrašą

```
Įveskite xp, xg, xh
```

Dabar turim ekrane surinkti pradinius duomenis taip, kaip reikalauja įvesties iš ekrano taisyklės - skirdami skaičius kableliais arba intervalais, ir nuspausti įvedimo klavišą. Tarkim, pradinė argumento reikšmė turi būti 0 radianų, galinė - 0.9 radiano, o žingsnis - 0.1 radiano. Taigi ekrane renkame:

```
0. 0.9 0.1
```

Šiuos skaičius operatorius READ nuskaito į kompiuterio operatyviąją atmintį, į ląsteles, paskirtas kintamųjų xp, xg ir xh saugojimui. Toliau suveikia kontrolinio spausdinimo operatorius WRITE bei visi funkcijos skaičiavimo ir spausdinimo ciklo operatoriai. Kiekviename ciklo žingsnyje spausdinama argumento reikšmė su paaiškinamuoju tekstu " x=" bei atitinkama funkcijos reikšmė su tekstu " y="

(konstantoje prieš simbolį y specialiai atspausdinti keli intervalai - kad rezultatų spausdiniai “nesuliptų” vienas su kitu). Skaitinių reikšmių spausdinimo forma tokia, kokią nurodo laisvasis formatas. Ciklas kartojamas 10 kartų, po to sąlygos operatorius IF nukreipia valdymą programos pabaigos operatoriui. Taigi kompiuterio ekrane matysim tokius spausdinius:

```
 Duomenys   0.000000E+00    9.000000E-01    1.000000E-01
 Rezultatai
 x=   0.000000E+00   y=   0.000000E+00
 x=   1.000000E-01   y=   9.983342E-02
 x=   2.000000E-01   y=   1.986693E-01
 x=   3.000000E-01   y=   2.955202E-01
 x=   4.000000E-01   y=   3.894183E-01
 x=   5.000000E-01   y=   4.794255E-01
 x=   6.000000E-01   y=   5.646425E-01
 x=   7.000000E-01   y=   6.442177E-01
 x=   8.000000E-01   y=   7.173561E-01
 x=   9.000000E-01   y=   7.833270E-01
```

**Pastaba.* Šioje programoje panaudota ir viena nerekomenduotina konstrukcija, būtent, loginis reiškinys x.LE.xg , lyginantis vieną su kitu du realiuosius duomenis. Realieji duomenys kompiuterio atminty dėl ląstelės baigtinio ilgio yra visad tik tikrosios duomenų reikšmės aproksimacijos. Todėl labai gali būti, kad žingsnio xh reikšmė kompiuterio atminty yra ne 0.1, o 0.1000001 ar 0.999999. Pirmuoju suapvalintos reikšmės atveju prarastume vieną ciklo žingsnį. Kokia turėtų būti korektiška programa? Atsakymas - įvesti sveikojo tipo ciklo žingsnių skaitiklį, suskaičiuoti, kiek reiks atlikti ciklo žingsnių, ir loginiame reiškinyje palyginti tarpusavyje tik sveikuosius duomenis, kurie kompiuterio atmintyje saugomi tiksliai:

```
                . . .
C           Funkcijos skaičiavimo ir rezultatų spausdinimo
C           ciklas
      ig = ANINT( xg/xh ) + 1     !  funkcija ANINT apvalina iki sveikojo duomens
      x = xp
      i = 1
      WRITE(*,*) ' Rezultatai'
   10 y = SIN( x )
      WRITE(*,*) ' x=', x, '   y=', y
      x = x+xh
      i = i+1
      IF( i .LE. ig )  GO TO 10
```

. . .

Programa 2. Užprogramuosime pirmąjį algoritmą iš 2-ojo skyriaus. Programos tekstas:

```
C           Funkcijos skaičiavimo pasirinktinai programa
        PROGRAM p2
C           Duomenų įvedimas ir kontrolinis spausdinimas
        WRITE(*,*) ' Įveskite xp, xg, xh, p'
        READ(*,*)   xp, xg, xh, p
        WRITE(*,*) ' Duomenys', xp, xg, xh, p
C           Funkcijos skaičiavimo ir rezultatų spausdinimo
C           ciklas
        x = xp
        WRITE(*,*) ' Rezultatai'
   10   IF( x.LT.p ) THEN
          y = SIN( x )
        ELSE
          y = COS( x )
        END IF
        WRITE(*,*) ' x=', x, '   y=', y
        x = x+xh
        IF( x .LE. xg )  GO TO 10
        END
```

Įvedę programai duomenis

0. 0.1 0.9 0.5

gausim tokius programos spausdinius:

```
Duomenys   0.000000E+00    9.000000E-01    1.000000E-01    5.000000E-01
Rezultatai
x=   0.000000E+00   y=   0.000000E+00
x=   1.000000E-01   y=   9.983342E-02
x=   2.000000E-01   y=   1.986693E-01
x=   3.000000E-01   y=   2.955202E-01
x=   4.000000E-01   y=   3.894183E-01
x=   5.000000E-01   y=   8.775826E-01
x=   6.000000E-01   y=   8.253356E-01
x=   7.000000E-01   y=   7.648422E-01
```

```
x=  8.000000E-01   y=  6.967067E-01
x=  9.000000E-01   y=  6.216099E-01
```

**Pastaba.* Žr. pastabas ankstesnei programai.

Programa 3. Užrašysim programą antrajam algoritmui iš 2-ojo skyriaus:

```
C          Programa kvadratinės lygties sprendimui
           PROGRAM p3
C          Duomenų įvedimas ir kontrolinis spausdinimas
           WRITE(*,*) ' Įveskite a, b, c'
           READ(*,*)   a, b, c
           WRITE(*,*) ' Duomenys: a, b, c', a, b, c
C          Diskriminanto skaičiavimas
           d = b*b - 4.*a*c
C          Ar yra realios šaknys ?
           IF( d )  10, 20, 30
C          Nėra
      10   WRITE(*,*) ' Rezultatai:  Realių šaknų nėra'
           STOP
C          Yra viena šaknis
      20   x = -b/(2.*a)
           WRITE(*,*) ' Rezultatai:  x=', x
           STOP
C          Yra dvi šaknys
      30   x1 = (-b + SQRT(d))/(2.*a)
           x2 = (-b - SQRT(d))/(2.*a)
           WRITE(*,*) ' Rezultatai:     x1=', x1, '      x2', x2
C
           END
```

Įvedę šiai programai duomenis

2.5 4. -0.6

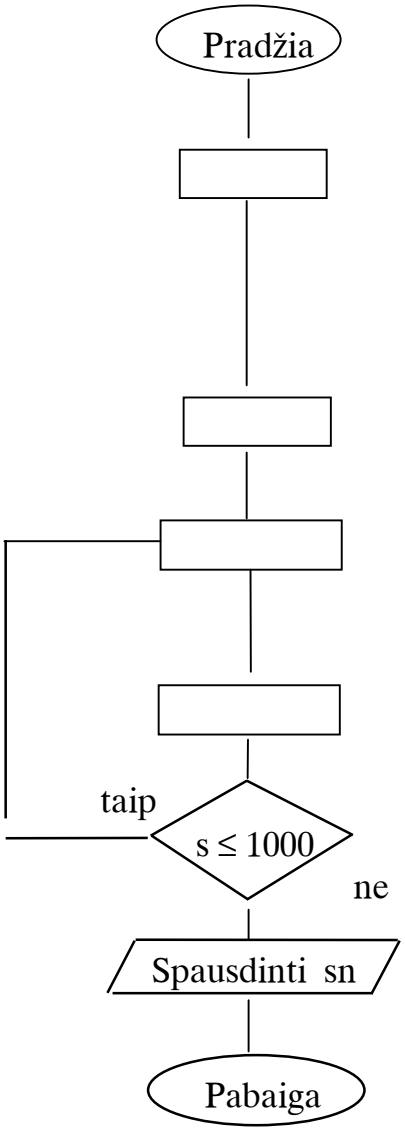
gausim tokius programos spausdinius:

```
Duomenys: a, b, c       2.500000        4.000000   -6.000000E-01
Rezultatai:    x1=   1.380831E-01      x2=    -1.738083
```

Programa 4. Užduotis: suskaičiuoti visų nelyginių skaičių, esančių tarp 0 ir 1000, sumą.

Šiuo trivialiu uždaviniu bandysim išsiaiškinti sumavimo algoritmą, todėl uždavinį spręsim pačiu paprasčiausiu būdu. Nelyginių skaičių sumą pažymėsim vardu sn, o eilinį nelyginį skaičių - s. Pradinių duomenų šiam uždaviniui neįvesim; paprasčiausiai juos pranešime programos tekste. Aišku, tai apriboja programos universalumą.

Algoritmo blokinė schema:



Esminis dalykas sumavimo uždaviniuose - sumą prieš ciklą turim prilyginti nuliui: kitaip nebus aišku, kokią jos reikšmę imti pirmą kartą vykdant 4-ąjį bloką.

Pradėsim skaičiavimą nuo pirmojo nelyginio skaičiaus - 1 .

Eilinį nelyginį skaičių pridedame prie ankstesnės sumos reikšmės.

Pereiname prie kito nelyginio skaičiaus.

Ar dar tęsti skaičiavimus ?

Pasibaigus ciklui, ląstelėje sn yra sukaupta nelyginių skaičių suma.

Programa.

Nelyginių skaičių suma bei eilinis nelyginis skaičius pagal savo prasmę yra sveikieji duomenys, o jiems parinkti vardai - sn bei s - pagal FORTRANo nutylėjimo principą

būtų laikomi realiais kintamaisiais. Todėl programoje šiuos kintamuosius turime deklaruoti sveikaisiais tipo apibrėžimo operatorium INTEGER. Ciklą kintamajam s keisti užrašysim ciklo operatorium.

```
C          Sumos skaičiavimo programa
      PROGRAM p4
      INTEGER  sn, s
C
      sn = 0
      DO s = 1, 999, 2
        sn = sn+s
      END DO
C
      WRITE(*,*)  ' Nelyginių skaičių suma', sn
C
      END
```

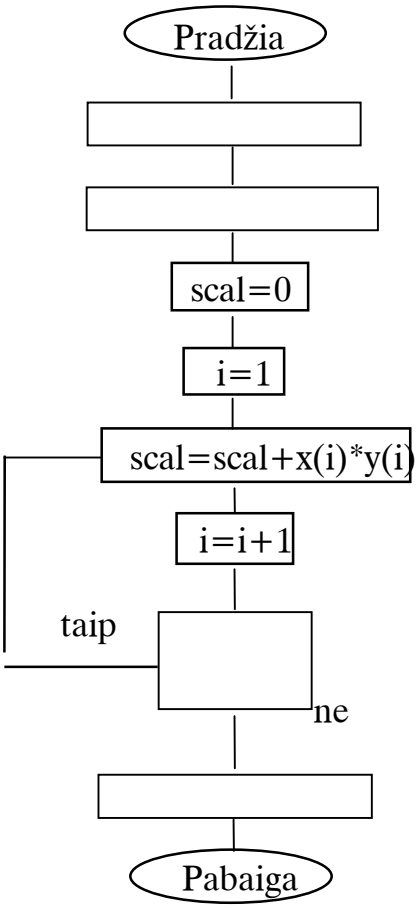
Programos darbo rezultatai:

Nelyginių skaičių suma 250000

Programa 5. Duoti du realiųjų skaičių vektoriai, turintys ne daugiau kaip po 20 elementų. Parašysime programą jų skaliarinei sandaugos rasti.

Vienmačiams masyvams (taip FORTRANe vadinami vektoriai) parenkame realius vardus x ir y. Programos pradžioje šiems masyvams saugoti turime paskirti po 20 ląstelių atminties. Elementų kiekį kiekviename masyve žymėsim sveikuoju vardu n , o skaliarinę sandaugą - realiuoju vardu scal. Priminsim, kad skaliarinė sandauga skaičiuojama taip: sudauginami atitinkami masyvų elementai ir šios sandaugos susumuojamos: $(x, y) = \sum_i x_i y_i$. Masyvo elementus išskirsime kreipdamiesi į juos indeksu i (sveikas kintamasis pagal nutylėjimo principą).

Algoritmas: žr. kitą puslapį.



Sumą prilyginame nuliui.

Perrinksime masyvų elementų poras pradėdami nuo pirmosios.

Pereiname prie kitos indekso reikšmės - kitos elementų poros.

Ar indeksas neišėjo už masyvo ribų?

Programa:

```
C       Skaliarinės masyvų sandaugos skaičiavimo
C       programa
      PROGRAM p5
      DIMENSION x(20), y(20)
C       Duomenų įvedimas ir spausdinimas
      WRITE(*,*) ' Įveskite duomenis: n / x / y'
      WRITE(*,*) 'DUOMENYS'
      READ(*,*) n
      WRITE(*,*) ' n=', n
      READ(*,*) (x(i), i=1,n)
      WRITE(*,*) ' x=', (x(i), i=1,n)
      READ(*,*) (y(i), i=1,n)
      WRITE(*,*) ' y=', (y(i), i=1,n)
C       Skaičiavimas
      scal = 0.
      DO i=1,n
```

```
          scal = scal + x(i)*y(i)
        END DO
C            Rezultatų spausdinimas
        WRITE(*,*) ' REZULTATAI'
        WRITE(*,*) ' Skaliarinė sandauga', scal
C
        END
```

Programai įvedus tokius pradinius duomenis:

```
5
-10.2  4.2  0.  -4.1  5.7
0.04  -14.4  -5.2  7.6  15.8
```

bus gauti programos spausdiniai:

```
 DUOMENYS
 n=          5
 x=    -10.200000        4.200000    0.000000E+00       -4.100000
       5.700000
 y=   4.000000E-02      -14.400000       -5.200000        7.600000
      15.800000
 REZULTATAI
 Skaliarinė sandauga      -1.987998
```

Programa 6. Rasime realiųjų skaičių sekos, turinčios ne daugiau 50-ies elementų, maksimalų narį bei jo eilės sekoje numerį.

Šiuo uždaviniu išsiaiškinsim maksimumo-minimumo (principas lygiai tas pats) ieškojimo algoritmą.

Jei ankstesni programų pavyzdžiai buvo suprantami, tai, matyt, loginiai ir ciklo operatoriai Jums nebesudaro keblumų, ir toliau galim apsieiti be algoritmų blokinių schemų. Uždavinio algoritmus nuo šiol trumpai užrašysime žodžiais. Taigi šio uždavinio sprendimo planas toks: iš pradžių darome prielaidą, kad masyvo (tegu jo vardas bus realusis x) maksimalus elementas (vardas tebūnie realusis xmax) yra pirmasis masyvo elementas, o tada jo eilės numeris (tegu - sveikasis vardas imax) masyve - 1. Dabar perrinksime visus likusius masyvo elementus ir patikrinsime, ar parinkta prielaida yra teisinga ar ne. Jei lyginant xmax su eiliniu masyvo elementu pasirodys, kad prielaida yra teisinga - nieko nekeisdami pereisime prie prielaidos lyginimo su kitu elementu; jei ne - turėsim pakeisti prielaidos reikšmę į surasto didesnio elemento reikšmę ir atitinkamai pakeisti imax.

Aišku, nebūtina kaip prielaidą imti pirmąjį masyvo elementą; prielaida galime pasirinkti ir bet kurį kitą masyvo narį.

Programa:

```
C           Programa maksimaliam sekos nariui
C           rasti
      PROGRAM p6
      DIMENSION x(50)
C           Duomenų įvedimas ir spausdinimas
      WRITE(*,*) ' Įveskite duomenis: n / x'
      READ(*,*) n
      READ(*,*) (x(i), i=1,n)
      WRITE(*,*) ' DUOMENYS'
      WRITE(*,*) ' n=', n
      WRITE(*,*) ' x=', (x(i), i=1,n)
C           Funkcinė dalis
      xmax = x(1)
      imax = 1
      DO i=2,n
        IF( xmax.LT.x(i) )  THEN
          xmax = x(i)
          imax = i
        END IF
      END DO
C           Rezultatų spausdinimas
      WRITE(*,*) ' REZULTATAI'
      WRITE(*,*) ' Maksimalus elementas', xmax, '  jo indeksas', imax
C
      END
```

Duomenims

```
5
10.  2.5  -15.0  154.  0.0
```

gaunami rezultatai

```
 DUOMENYS
n=         5
x=      10.000000       2.500000      -15.000000      154.000000
```

```
  0.000000E+00
 REZULTATAI
 Maksimalus elementas     154.000000  jo indeksas         4
```

Pastaba. Jei programai pateiktoje skaičių sekoje būtų keli vienodi didžiausi elementai, tai programos rezultatas būtų pirmasis iš tų elementų. Jeigu programos operatoriuje IF įrašytume loginio santykio ženklą .LE., tai rezultatas būtų paskutinis iš didžiausių elementų.

* Programa 7. Parašysime programą realiųjų skaičių matricos x, kurios matiškumas neviršija 10 eilučių ir 10 stulpelių, normai rasti: rasti maksimalią sumą tarp visų matricos eilučių elementų absoliutinių dydžių sumų: $\|x\|_\infty = \max_i \sum_j \left|x_{ij}\right|$.

Dvimatį masyvą pažymime vardu x, jo eilučių kiekį - n, stulpelių - m, normą - vardu norma. Algoritmas: iš pradžių suskaičiuosime visas eilučių sumas. Jų yra tiek, kiek masyve eilučių, todėl šį tarpinį rezultatą saugosime vienmačiame masyve es. Tada šiame masyve es pagal jau žinomą algoritmą rasime masyvo didžiausią elementą - tai ir bus matricos norma.

Duomenis įvesime iš failo kietajame diske. Jeigu šis failas įrašytas tame pat kataloge, kaip ir programos tekstas, tada atidarant failą operatoriumi OPEN užteks nurodyti vien tik failo vardą - duom.dat. Matricos elementai faile išdėstyti eilutėmis. Rezultatus išvesime vėlgi į failą diske vardu rezult.dat. Tiek kontrolinis spausdinimas, tiek rezultatų spausdinimas - formatinis.

```
C          Programa matricos normos skaičiavimui
      PROGRAM p7
      REAL norma
      DIMENSION x(10,10), es(10)
      OPEN(UNIT=1, FILE ='duom.dat', STATUS ='OLD')
      OPEN(UNIT=2, FILE ='rezult.dat', STATUS ='NEW')
C        Duomenų įvedimas ir kontrolinis spausdinimas
      READ(1,*) n,m
      WRITE(2,10)
      WRITE(2,11)  n, m
      READ(1,*) ((x(i,j), j=1,m), i=1,n)
      WRITE(2,12) ((x(i,j), j=1,m), i=1,n)
C        Funkcinė dalis
C        Eilučių sumos
      DO i=1,n
        s = 0.
```

```
          DO j=1,m
            s = s + ABS( x(i,j) )
          END DO
          es(i) = s
        END DO

C           Maksimalaus es elemento radimas
        norma = es(1)
        DO i=2,n
          IF( norma.LT.es(i) )  norma = es(i)
        END DO
C           Rezultatų spausdinimas
        WRITE(2,20)
        WRITE(2,21) norma
10      FORMAT(//20X,'DUOMENYS')
     11 FORMAT(10X,'Eilučių kiekis',I3 /10X, 'Stulpelių kiekis', I3)
     12 FORMAT(10X,'Matrica' / (5G10.3) )
     20 FORMAT(//20X,'REZULTATAI')
     21 FORMAT(10X,'Matricos norma', G10.3)
        END
```

Prieš atiduodant šią programą vykdyti, darbiniame kataloge turi būti įrašytas duomenų failas duom.dat. Sakykim, kad jo turinys yra:

```
3 5
1. 2. 3. 4. 5.
10. 20. 30. 40. 50.
-100. 200. -300. 400. -500.
```

Programa suformuos darbiniame kataloge tokį rezultatų failą rezult.dat:

```
                DUOMENYS
        Eilučių kiekis  3
        Stulpelių kiekis  5
        Matrica
  1.00      2.00      3.00      4.00      5.00
  10.0      20.0      30.0      40.0      50.0
 -100.      200.     -300.      400.     -500.


                REZULTATAI
```

Matricos norma 1500.

7. PAPROGRAMIAI

7.1. BENDROS ŽINIOS

Didelė programa niekada nėra apiforminama kaip vienas programinis vienetas. Tokia programa paprastai suskirstoma į atskiras mažesnes programas, viena iš kurių yra *pagrindinė* programa, o kitos - *paprogramiai*. Pagrindinė programa reikiamu metu "iškviečia" darbui paprogramius. Savo ruožtu ir paprogramiai gali iškviesti kitus, žemesnio lygio, paprogramius.

Tokia programos struktūra turi nemažai pranašumų, nes FORTRANo kompiliatorius kiekvieną programinį vienetą apdoroja atskirai. Todėl galima kiekvieną programą atskirai versti į mašinos kodų kalbą, testuoti, o tik po to surinkti visas reikiamas programas į visumą. Testuoti mažesnes programas yra kur kas lengviau nei dideles. Be to, kai kurie veiksmai, pavyzdžiui, tiesinių lygčių sistemų sprendimas, kartojasi daugelyje uždavinių. Tokią veiksmų seką patogu apiforminti kaip paprogramį. Vėliau susidūrus su tuo pačiu uždaviniu, jo iš naujo jau nebereikės užprogramuoti; tereiks tik panaudoti jau parašytą paprogramį. Paprogramių bibliotekas kaupia ne tik programuotojai. Dideles standartinių paprogramių bibliotekas kuria ir parduoda programinę įrangą kuriančios firmos. Yra paprogramių rinkinių tiesinės algebros, diferencialinių lygčių, matematinės statistikos ir pan. uždaviniams spręsti. Laisvai platinamų paprogramių rinkinių galima parsisiųsdinti, pavyzdžiui, iš http://www.fortran.com/fortran/libraries.html. Tokios paprogamių bibliotekos labai palengvina programuotojų darbą.

Pagrindinė programa ir paprogramiai yra savarankiški programiniai vienetai. Duomenimis šios programos keičiasi tik per vadinamuosius *argumentų sąrašus*, į kuriuos įtraukiami visi informacijos mainams reikalingi duomenų vardai: *formaliųjų argumentų sąrašas* nurodomas paprogramio deklaravimo operatoriuje, o jį atitinkantis *faktiškųjų argumentų sąrašas* - pagrindinės programos operatoriuje, kviečiančiame darbui paprogramį. Į argumentų sąrašus dažniausiai įtraukiami kintamųjų ir masyvų vardai. Dalis argumentų, kuriais pagrindinė programa perduoda duomenis paprogramiui, vadinama dar *įėjimo argumentais*, o argumentai, kuriais paprogramis perduoda darbo rezultatus ją kviečiančiai programai - *išėjimo argumentais*.

Formaliųjų ir faktiškųjų argumentų sąrašai privalo sutapti pagal kiekį, tipą ir prasmę, o atitinkamų faktinių ir formaliųjų argumentų vardai gali ir skirtis. Pavyzdžiui, jei paprogramio formaliųjų argumentų sąraše yra 10 argumentų, tiek pat jų turi būti ir paprogramį kviečiančio pagrindinės programos operatoriaus faktiškųjų argumentų sąraše. Jei pirmas formalus argumentas yra R*8 tipo dvimatis masyvas, tai ir pirmas faktiškasis argumentas turi būti tokio pat tipo dvimačio masyvo vardas.

Formalieji argumentai gali būti: kintamieji, masyvų vardai, kitų paprogramių vardai ir žvaigždutė * (paskutines dvi galimybes paaiškinsime vėliau). Faktiškieji argumentai gali būti: konstanta, kintamasis, reiškinys, masyvo vardas, masyvo elementas, kito paprogramio vardas, žymė. Jei faktiškasis argumentas yra reiškinys, tai prieš perduodant valdymą paprogramei jis suskaičiuojamas.

Kaip per argumentų sąrašus vyksta duomenų mainai? Iškvietus paprogramį, jam perduodami ląstelių, kuriose saugomi faktiškieji argumentai, adresai. Taigi formaliųjų argumentų sąrašas iškvietimo metu tampa sąrašu ląstelių, kuriomis duomenys perduodami paprogramiui, adresų. Supaprastintai galima sakyti, kad formalūs ir faktiškieji argumentai saugomi tose pačiose atminties ląstelėse.

Iš to išeina viena svarbi išvada. Jei kviečiant paprogramį faktiškasis argumentas yra konstanta, reiškinys ar kito paprogramio vardas, tai pačiame paprogramyje šį argumentą atitinkančiam formaliajam argumentui negalima priskirti jokios reikšmės - tokiu atveju paprogramio darbo rezultatas būtų nenuspėjamas.

Lokalūs ir globalūs duomenys. Visi duomenys, neįtraukti į argumentų sąrašus, yra *lokalūs*, t.y. apie juos žino tik ta programa, kuri jais operuoja. Kitos programos apie šiuos lokalius duomenis “nežino” nieko. Dėl šios priežasties atskirose programose galima naudoti lokalių duomenų žymėjimui tuos pat vardus, naudoti tas pačias žymes ir pan. Duomenys, įtraukti į argumentų sąrašus, yra *globalūs* toms programoms, kurios tuos argumentų sąrašus naudoja.

Paprogramių rūšys. Kol kas kalbėsime tik apie FORTRANo 77 paprogramius; FORTRANo 90 papildomai įvesti paprogramiai aprašyti 8-ame skyriuje. Taigi FORTRANe 77 skiriami tokie paprogramiai: *funkcijos* ir betarpiškai *paprogramiai* SUBROUTINE. Funkcijos rezultatas yra tik vienas duomuo, o paprogramio - keli duomenys, tame tarpe gali būti ir masyvai. Įėjimo argumentų abeji paprogramiai gali turėti kelis (kraštutiniu atveju - ir nė vieno).

7.2. PAPROGRAMIAI - FUNKCIJOS

Kaip minėta, tai - paprogramiai, galintys turėti kelis įėjimo argumentus ir tik vieną išėjimo argumentą - rezultatą. Galimas aritmetinio, loginio arba tekstinio tipo rezultatas. Gauto funkcijos rezultato reikšmė užrašoma į atminties ląstelę, turinčią paprogramio-funkcijos vardą. Todėl funkcijos vardui galioja visos vardus liečiančios taisyklės: pavyzdžiui, jei laukiamas R*8 tipo funkcijos rezultatas, tai tokio tipo turi būti ir funkcijos vardas.

Yra trijų rūšių funkcijos: *vidinės* (INTRINSIC) *funkcijos*, *išorinės* (EXTERNAL) *funkcijos* ir *operatoriai-funkcijos*.

Vidinės funkcijos yra FORTRANo kalbos dalis. Kelias vidines aritmetinio tipo funkcijas paminėjome trečiame skyriuje, aiškindami aritmetinius reiškinius. Viso daugiau nei šimto vidinių funkcijų sąrašo teks ieškoti kalbos aprašymo dokumentuose.

Visos funkcijos iškviečiamos vienodai: paminint fukcijos vardą ir nurodant faktiškųjų argumentų sąrašą:

funkcijos vardas ([*faktiškųjų argumentų sąrašas*])

Toks kreipinys galimas, pavyzdžiui, reiškinyje, išvesties operatoriuje ir pan. Laužtiniai skliaustai rodo, kad nesant formaliųjų argumentų sąrašo neturi būti ir faktiškųjų argumentų.

Pavyzdys

```
y = SIN( SQRT( x**3 ) )
```

Čia mums jau pažįstamos vidinės sinuso ir kvadratinės šaknies funkcijos iškviečiamos jų vardus ir faktiškųjų argumentų sąrašus paminint aritmetiniame reiškinyje. Faktinis argumentas sinuso funkcijai - kita vidinė funkcija, kvadratinė šaknis; o šiai - aritmetinis reiškinys x^3. Suskaičiuoto aritmetinio reiškinio reikšmė perduodama funkcijai SQRT , šios gautas rezultatas įrašomas į ląstelę vardu SQRT, o šios adresas perduodamas funkcijai SIN. Šios funkcijos rezultatas įrašomas į ląstelę vardu SIN, o priskyrimo operatorius jos turinį nukopijuoja į ląstelę vardu y.

Operatorius - funkcija

Tai pati paprasčiausia funkcija. Naudojama tada, kai visą funkcijos skaičiavimą galima užrašyti vienu operatoriumi. Funkcijos kūno bendrasis pavidalas yra:

funkcijos vardas ([*formaliųjų argumentų sąrašas*]) = *reiškinys*

Šis operatorius - nevykdomasis. Programoje jis turi būti po visų apibrėžimo operatorių. *Funkcijos vardas* ir *reiškinys* turi atitikti vienas kitą (atitikimo taisyklės paaiškintos kalbant apie priskyrimo operatorių 4-ame skyriuje). Pavyzdžiui, jei *funkcijos vardas* yra loginis, tai ir *reiškiny*s turi būti loginis. *Reiškinio* tipas visada pakeičiamas į tokį tipą, kokio yra *funkcijos vardas*. Pavyzdžiui, jei *vardas* deklaruotas kaip R*8 tipo vardas, o *reiškinys* yra I*4 tipo, tai jo reikšmė prieš priskyrimą pervedama į R*8 tipą. Faktiškieji argumentai operatoriui-funkcijai visada pervedami į tokį tipą, kokio yra atitinkami formalieji argumentai.

Pavyzdys Parašysime programos fragmentą (be duomenų įvedimo ir rezultatų spausdinimo), pagal trikampio kraštinių a, b, c ilgius skaičiuojantį trikampio kampus alfa, beta, gama. Panaudosime elementarios trigonometrijos formulę: kampas prieš kraštinę a yra: $\alpha = \arccos \frac{b^2 + c^2 - a^2}{2bc}$. Trikampio kampo skaičiavimo procedūrą užrašysime operatoriuje-funkcijoje kampas. Kadangi laukiamas realiojo tipo rezultatas, šį vardą reikia deklaruoti kaip R tipo vardą. Kiti programos vardai apibrėžiami nutylėjimo principu.

```
      REAL  kampas
C   Operatorius-funkcija kampui prieš kraštinę  x  skaičiuoti. Naudojama vidinė
C   arkkosinuso funkcija, kurios darbo rezultatą pervedame į laipsnius
      kampas ( x, y, z )  =  57.2958*ACOS(  (y*y + z*z - x*x) / (2.*y*z)  )
            . . .
C   Kampų prieš kraštines  a, b, c   alfa, beta, gama  skaičiavimas
      alfa  =  kampas ( a, b, c )
      beta  =  kampas ( b, c, a )
      gama  =  kampas ( c, a, b )
            . . .
      END
```

Išorinė funkcija

Naudojama tada, kai funkcijos skaičiavimui užprogramuoti neužtenka vieno operatoriaus. Funkcijos kūno pavidalas yra:

```
[tipas]  FUNCTION  vardas ( [formaliųjų argumentų sąrašas] )
   . . .
vardas  =  rezultatas
   . . .
END
```

Tipas (gali būti pateiktas vienas iš 3.6 skyriuje nurodytų operatorių) apibrėžia funkcijos *vardo* tipą. Jei jis praleistas, *vardo* tipas apibrėžiamas nutylėjimo principu. *Formaliųjų argumentų sąrašo* gali ir nebūti, tačiau ir tada skliaustelius reikia palikti. *Argumentų* tipai apibrėžiami pačios funkcijos viduje (ne ją kviečiančioje programoje!) esančiais tipo apibrėžimo operatoriais. Dėl šios priežasties formalusis argumentas - masyvas turi būti aprašytas funkcijoje, o jį atitinkantis faktiškasis argumentas - masyvas taip pat turi būti aprašytas kviečiančioje programoje. Plačiau apie masyvų aprašymą paprogramiuose - 7.4 skyriuje. Kadangi funkcijos darbo rezultatas yra ląstelėje vardu

vardas, tai funkcijos kūno viduje bent vieną kartą reikia šiam *vardui* priskirti rezultato reikšmę. Funkcijos paskutinysis operatorius turi būti END. Funkcijos darbą taip pat užbaigti ir valdymą perduoti tiesiai operatoriui END galima operatoriumi RETURN. Funkcijoje jų gali būti keli; kaip matyti, jų atliekami veiksmai yra tokie pat, kaip operatoriaus STOP - pagrindinėje programoje.

Funkcijos viduje galimi bet kurie operatoriai, išskyrus BLOCK DATA, FUNCTION, INTERFACE, PROGRAM, SUBROUTINE.

Rekomenduojame funkcijos argumentus naudoti tik kaip pradinius duomenis ir funkcijoje jų reikšmių nekeisti. Argumento reikšmės pakeitimas funkcijoje keičia atitinkamos ląstelės turinį ir kviečiančioje programoje, todėl argumentų sąrašais irgi galima perduoti funkcijos rezultatus, tačiau toks programavimo stilius laikomas netinkamu.

Pavyzdys Parašysime išorinę funkciją faktorialo skaičiavimui ir pagrindinę programą, kuri šia funkcija suskaičiuotų 8! ir 10! . Funkcijos rezultatas, aišku, bus sveikasis skaičius ir iki 10^6 eilės skaičius, todėl funkcijos vardui turim parinkti I*4 tipą.

```
C    Funkcija  factor  skaičiaus  n  faktorialui skaičiuoti
          INTEGER*4  FUNCTION   factor ( n )
          factor  =  1
          DO  i  =  1, n
            factor  =  factor * i
          END DO
          END
C    Pagrindinė programa skaičių 8 ir 10 faktorialų  f8  bei f10  skaičiavimui
          INTEGER*4  factor, f8, f10
          f10  =  factor ( 8 )
          f15  =  factor ( 10 )
          WRITE (*,*)  ‘ 8! = ‘, f8,  ‘  10! = ‘, f10
          END
```

Kaip matyti iš pateikto pavyzdžio, atiduodant kompiuteriui apdoroti programų tekstus, visiškai nesvarbi kviečiančiosios programos ir paprogramiai tekstų eilė. Tačiau įprasta pirmiau pateikti pagrindinės programos tekstą, o vėliau visų paprogramių pagal kvietimo eilę tekstus.

7.3. PAPROGRAMIS SUBROUTINE

Paprogramio kūno pavidalas yra

SUBROUTINE *vardas* [(*formaliųjų argumentų sąrašas*)]
. . .
END

Globalus paprogramio *vardas* neteikia jokios informacijos ir skirtas tik paprogramiui identifikuoti. Todėl duomenų mainams organizuoti *formaliųjų argumentų sąraše* turi būti tiek įėjimo, tiek ir išėjimo argumentai. Šiame sąraše gali būti kintamieji, masyvai, kitų paprogramių vardai ir alternatyvaus valdymo sugrąžinimo į kviečiančiąją programą žymė - * (žr. žemiau). *Argumentų* tipas kaip ir išorinėje funkcijoje suteikiamas paprogramio viduje rašomais operatoriais. Atskiru atveju *formaliųjų argumentų sąrašo* gali ir visai nebūti - tada duomenų mainai organizuojami kitais būdais (žr. 7.7 skyrių).

Paprogramio viduje negalimi BLOCK DATA, FUNCTION, INTERFACE, PROGRAM, SUBROUTINE operatoriai.

Paprogramio darbą bet kurioje vietoje nutraukti ir valdymą grąžinti kviečiančiajai programai galima operatoriumi RETURN. Viename paprogramyje gali būti ir keli operatoriai RETURN. Primename, kad paprogramyje parašytas operatorius STOP nutrauktų visos programos darbą.

Paprogramis darbui iškviečiama specialiu operatoriumi

CALL *vardas* [(*faktiškųjų argumentų sąrašas*)]

Paprogramiui baigus darbą, valdymas grąžinamas į kviečiančiąją programą vykdomajam operatoriui, esančiam iškart po operatoriaus CALL.

Faktiškųjų ir formaliųjų argumentų atitikimo taisyklės pateiktos 7.1 skyriuje.

Pavyzdys Perrašysime ankstesnio skyriaus pavyzdį su paprogramiu-SUBROUTINE.

```
C    Pagrindinė programa skaičių 8 ir 10 faktorialų  f8  bei f10  skaičiuoti
        INTEGER*4  f8, f10
        CALL  factor ( 8, f8 )
        CALL  factor ( 10, f10 )
        WRITE (*,*)  '8! = ', f8,  '10! = ', f10
        END
```

```
C   Paprogramis factor skaičiaus n  faktorialui nf skaičiuoti
C    n - įeities argumentas, nf - išeities argumentas
      SUBROUTINE  factor ( n, nf )
      INTEGER*4  nf
      nf  =  1
      DO  i  =  1, n
        nf  =  nf * i
      END DO
      END
```

**** Alternatyvus valdymo grąžinimas kviečiančiajai programai***

leidžia sugrąžinti valdymą į norimą kviečiančiosios programos operatorių. Tam būtina speciali operatoriaus RETURN žymėtoji forma, galima tik paprogramiuose SUBROUTINE: RETURN *žymė*.

Alternatyvus valdymo grąžinimas realizuojamas taip:

1. Paprogramio formaliųjų argumentų sąraše įrašoma tiek žvaidždučių, kiek yra alternatyvų.

2. Kvietimo operatoriuje CALL atintinkami faktiškieji argumentai bus reikiamų kviečiančiosios programos operatorių žymės su žvaigždutėmis: **žymė* . Pavyzdžiui,

```
SUBROUTINE  subr ( a, b, c, *, *,* )          ir atitinkamai
CALL  subr ( a1, b1, c1, *100, *200, *300 ) .
```

3. Paprogramyje būtinas bent vienas žymėtasis operatorius RETURN. Pateiktame pavyzdyje šiems operatoriams reikėtų nurodyti žymes: 1 - tam operatoriui, kuris turės perduoti valdymą kviečiančiosios programos operatoriui su žyme 100, 2 - tam, kuris perduos valdymą operatoriui su žyme 200, ir 3 - su žyme 300. Jei paprogramis baigia valdymą paprastu operatorium RETURN ar pasiekia RETURN, kurio žymė neturi atitikmens formaliųjų argumentų sąraše (mūsų atveju, pavyzdžiui, RETURN 4), valdymas grąžinamas įprastu būdu.

Pavyzdys

```
C  Kviečiančiosios programos fragmentas
C  Jei  number  bus lygus 100 - valdymas grįš į 10-ąjį operatorių; jei 200 - į 20-ąjį;
C  jei 300 - į 30-ąjį; jei nelygus šiems skaičiams - į 1-ąjį.
                . . .
      CALL  fiction ( number, *10, *20, *30 )
```

```
1       WRITE ( *,* )  'normal return'
        GO TO  100
10      WRITE ( *,* )  'return to 10'
        GO TO  100
20      WRITE ( *,* )  'return to 20'
        GO TO  100
30      WRITE ( *,* ) 'return to 30'
100     CONTINUE
              . . .
        SUBROUTINE  fiction ( n, *, *, * )
        IF ( n .EQ. 100 ) RETURN 1
        IF ( n .EQ. 200 ) RETURN 2
        IF ( n .EQ. 300 ) RETURN 3
        RETURN
        END
```

7.4. MASYVŲ IR TEKSTINIŲ DUOMENŲ APRAŠYMAS PAPROGRAMIUOSE

Šią temą išskyrėme į atskirą skyrių tik dėl to, kad tokiuose aprašymuose padaroma nemaža klaidų.

Masyvai. Paprogramiuose, skirtingai nei pagrindinėje programoje, masyvus, kurių vardai įtraukti į argumentų sąrašą, galima aprašyti ne tik konstantomis, bet ir kintamaisiais. Tai įmanoma dėl to, kad paprogramio iškvietimo metu vietoj formalaus argumento - masyvo vardo - perduodamas faktinio argumento - taip pat masyvo - pirmojo elemento adresas. Paprogramis gali panaudoti visą atminties sritį, skirtą masyvui pagrindinėje programoje arba jos dalį, tačiau jokiu būdu ne didesnę sritį.

Pavyzdžiui, jei pagrindinėje programoje vienmatis masyvas aprašytas

```
DIMENSION  x(100) ,
```

tai į paprogramio, turinčio apdoroti šį masyvą, argumentų sąrašą paprastai įtraukiamas ir masyvo formalus vardas (pavyzdžiui, y), ir elementų kiekis masyve (pavyzdžiui, n), o pats masyvas paprogramyje aprašomas taip:

```
DIMENSION  y(n) .
```

Kintamajam n prieš atiduodant valdymą paprogramiui gali būti priskirtos reikšmės nuo 1 iki 100 imtinai.

Tačiau labai dažnai programuotojai vienmačius masyvus paprogramiuose daug nesukdami galvos aprašo vienetiniam ilgiui, pavyzdžiui,

DIMENSION y(1) .

Formaliai žiūrint, paprogramyje, kreipiantis į masyvo elementą su didesniu už 1-ą indeksu, daroma klaida. Tačiau FORTRANo kompiliatorius nekontroliuoja, ar indeksas viršija aprašymo operatoriuje nurodytą ribą, ar ne; ir klaida nedeklaruojama. Iš tikrųjų klaidos ir nebus, jei paprogramyje kreipdamiesi į masyvo elementą neišeisim už faktiškajam masyvui skirtos atminties srities ribų.

Truputį sudėtingiau su dvimačių masyvų aprašymu. Kaip žinia, dvimatis masyvas atmintyje saugomas kaip vienmatė masyvo elementų seka; elementų išdėstymo tvarka - stulpeliais. Programoje kreipiantis į (i,j)-ąjį masyvo elementą, iš šios sekos jis bus atrinktas pagal indeksą k = (j - 1) * n + i , kur n yra masyvo eilučių kiekis, nurodytas aprašant masyvą. Taigi jei paprogramyje aprašydami masyvą nurodysim skirtingą nei kviečiančiojoje programoje eilučių kiekį, tai "sumaišysim" masyvo elementų atrinkimo tvarką. Tuo tarpu antrojo matavimo reikšmė gali ir skirtis, svarbu tik, kad ji neviršytų kviečiančiojoje programoje nurodyto stulpelių kiekio. Pavyzdžiui, tegu pagrindinėje programoje masyvas aprašytas

REAL x(10,10) .

Paprogramyje šį atitinkantis formalus masyvas y gali būti aprašytas

REAL y(10,10) arba
REAL y(10,1) arba bet kuriam kitam stulpelių skaičiui $2 \le j \le 10$, arba
REAL y(n,1) arba - rekomenduotinas variantas -
REAL y(n,m) ,

kur n ir m reikšmės, perduodamos paprogramiui, turi būti: n = 10, o $1 \le m \le 10$.

Analogiškai, aprašant paprogramyje trimatį masyvą, teks tiksliai išlaikyti pagrindinėje programoje nurodytus du pirmuosius masyvo matavimus, o trečiasis matavimas neturi viršyti pagrindinėje programoje nurodytos reikšmės ir t.t.

Pavyzdys Parašysim paprogramį dviejų realiųjų skaičių matricų sandaugai:

r = a * b, a matiškumas yra n eilučių ir m stulpelių, b - m ir l . Rezultato matrica r tada bus iš n eilučių ir l stulpelių, o jos elementai turi būti skaičiuojami taip:

$$r_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad .$$

Naudodami parašytą paprogramį, sudauginsime matricas

$$a = \begin{bmatrix} 10.4 & -2.3 & 5.6 \\ 1.2 & 0. & -15.2 \end{bmatrix} \quad \text{ir} \quad b = \begin{bmatrix} 1.1 & 4.4 \\ 0. & 17.1 \\ 5.7 & 0.4 \end{bmatrix} .$$

```
C  Pagrindinė programa
      REAL  a(2,3), b(3,2), ab(2,2)
      DATA  a /10.4,1.2,-2.3,0.,5.6,-15.2/,  b/1.1,0.,5.7,4.4,17.1,0.4/
      CALL  md (a, b, ab, 2, 3, 2)
      WRITE (*,*)  'matricų sandauga', ((ab(i,j), j=1,2), i=1,2)
      END

C  Paprogramis  matricų daugybai md
C  a - pirma dauginamoji matrica, n*m
C  b - antra dauginamoji matrica, m*k
C  r - rezultato matrica, n*k
      SUBROUTINE  md (a, b, r, n, m, k)
      REAL  a(n,1), b(m,1), r(n,1), s
      DO  i=1,n
       DO  j=1,k
       s=0.                ! programa greitesnė, jei cikluose įvesim  s, o ne r(i,j):
        DO  kk=1,m         ! kreipinys į masyvo elementą užima daugiau laiko
          s=s+a(i,kk)*b(kk,j)
        END DO
       r(i,j)=s
       END DO
      END DO
      END
```

Tekstiniai kintamieji. Apibrėžiant paprogramyje tekstinio duomens, įtraukto į formaliųjų argumentų sąrašą, ilgį, vėlgi turime neišeiti iš kviečiančioje programoje atitinkamam faktiškajam argumentui skirtos atminties srities. Skirtumas nuo masyvų aprašymo tik tas, kad ilgis gali būti nurodytas ir žvaigždute * . Šiuo atveju formalusis argumentas "paveldi" atitinkamo faktiškojo argumento ilgį. Žvaigždute galima suteikti ir tekstinio tipo funkcijos vardo ilgį (išimtys: vidinė, modulinė ir rekursinė funkcijos bei funkcijos, grąžinančios rezultatą-masyvą ir rezultatą-nuorodą - žr. 8.2 skyrių).

Pavyzdys

```
C  Kviečiančiosios programos fragmentas
        CHARACTER   t1*5, t2*10
              . . .
        CALL  fiction( t1, t2 )
              . . .
C  Paprogramio fragmentas
        SUBROUTINE  fiction( a, b )
        CHARACTER*(*)  a, b   !  a  ilgis bus 5 baitai, o  b  - 10 baitų
              . . .
```

7.5. PAPROGRAMIŲ VARDAI ARGUMENTŲ SĄRAŠUOSE

Į argumentų sąrašus įtraukiamus kaip argumentus kitų paprogramių vardus būtina aprašyti nevykdomuosiuose operatoriuose

INTRINSIC *vardų sąrašas*

arba

EXTERNAL *vardų sąrašas* .

INTRINSIC operatoriui nurodomi visų FORTRANo vidinių funkcijų vardai, o EXTERNAL - visų paties programuotojo sukurtų paprogramių (tiek FUNCTION, tiek SUBROUTINE) vardai.

Pavyzdys Tarkim, parašėme išorinę funkciją apibrėžtiniam integralui $\int_a^b f(x)\,dx$ apskaičiuoti Simpsono metodu. Ją pradedame funkcijos deklaravimo operatorium su argumentais, kurie iš eilės reiškia integravimo rėžius ir pačią integruojamąją funkciją:

```
        FUNCTION  simps ( a, b, fx )
              . . .
        END
```

Norėdami šią funkciją panaudoti integralo $y1 = \int_0^1 \sin(x)\,dx$ skaičiavimui, kviečiančiojoje programoje turėtume vidinę sinuso funkciją aprašyti operatoriuje INTRINSIC, o tada jau galėtume iškviesti funkciją simps:

```
INTRINSIC SIN
y1 = simps ( 0., 1., SIN )
      . . .
```

Analogiškai, skaičiuodami $y1 = \int_0^1 \sin(x)\,dx$, turėtume parašyti savo išorinę funkciją pointegraliniam reiškiniui vardu, pavyzdžiui, sin2:

```
FUNCTION sin2 ( x )
sin2 = SIN( x/2. )**2
END
```

o kviečiančiojoje programoje šią funkciją aprašytume operatoriuje EXTERNAL:

```
EXTERNAL sin2
y2 = simps ( 0., 1., sin2 )
      . . .
```

* 7.6. KELI ĮĖJIMAI Į TĄ PATĮ PAPROGRAMĮ

Paprastai į paprogramį įeinama nuo pradžios. Operatorium ENTRY galima nurodyti kitus įėjimo į paprogramį taškus. Operatoriaus bendrasis pavidalas panašus į paprogramio deklaravimo operatoriaus pavidalą:

ENTRY *įėjimo vardas* [([*formaliųjų argumentų sąrašas*])]

Įėjimo vardas - įėjimo taško vardas, kitoks nei pačio paprogramio vardas. Jei tas *įėjimo vardas* yra išorinėje funkcijoje, tai jam keliami lygiai tokie pat reikalavimai, kaip ir funkcijos vardui; ir analogiškai - jei paprogramyje SUBROUTINE. Operatorius ENTRY neleistinas blokinio sąlygos operatoriaus IF ir ciklo operatoriaus DO sričių viduje. Paprogramyje ENTRY kiekis neribojamas.

Paprogramio ir jame esančio operatoriaus ENTRY *argumentų sąrašai* gali skirtis. Jei ENTRY *argumentų sąraše* yra naujas argumentas, kurio nebuvo paprogramio argumentų sąraše, tai šiuo argumentu operuoti vykdomuosiuose operatoriuose aukščiau ENTRY negalima. Pavyzdžiui, įtraukus į ENTRY argumentų sąrašą naujo masyvo vardą, jį aprašyti galima ir reikia po paprogramio deklaravimo operatoriaus

esančiuose nevykdomosiuose operatoriuose - t.y. aukščiau ENTRY. Tačiau kreiptis į šį masyvą galima tik žemiau ENTRY operatoriaus.

Norint, kad paprogramis pradėtų darbą nuo pirmojo žemiau ENTRY esančio operatoriaus, reikia paprogramiai iškvietimo operatoriuje nurodyti būtent šį ENTRY *įėjimo vardą*.

Pavyzdys Parašysime paprogramį dviejų dvimačių masyvų sumai ir skirtumui skaičiuoti, o po to su juo gausim masyvų x(5,5) ir y(5,5) sumą xpy(5,5) bei skirtumą xmy(5,5).

```
      SUBROUTINE  add ( a, b, r, n, m )
C   a, b  -  pradiniai masyvai, r  -  masyvas-rezultatas,  visi n×m
C   Masyvų suma
      REAL  a(n,m), b(n,m), r(n,m)
      DO  10 i = 1, n
        DO  10 j = 1, m
          r(i,j) = a(i,j) + b(i,j)
10    CONTINUE
      RETURN
C   Masyvų skirtumas
      ENTRY  sub ( a, b, r, n, m )
      DO  20 i = 1, n
        DO  20 j = 1, m
          r(i,j) = a(i,j) - b(i,j)
20    CONTINUE
      END

C   Kviečiančiosios programos fragmentas masyvų  x(5,5)  ir  y(5,5)  sumai  xpy(5,5)
C   bei skirtumui  xmy(5,5)  rasti
          . . .
      CALL  add ( x, y, xpy, 5, 5 )
      CALL  sub ( x, y, xmy, 5, 5 )
          . . .
```

* 7.7. KITAS DUOMENŲ PERDAVIMO TARP PROGRAMŲ BŪDAS

Informacijos mainai tarp pagrindinės programos ir paprogramių arba tarp paprogramių galimi ne tik per argumentų sąrašus, bet ir per bendrąsias atminties sritis

COMMON. Galima informaciją perduoti ir iš dalies per argumentų sąrašus, iš dalies per COMMONus.

Laikoma, kad šis operatorius yra pasenusi FORTRANo konstrukcija, ir, matyt, kitame FORTRANo standarte jų jau nebebus.

Nevykdomasis operatorius COMMON mašinos operatyvinėje atmintyje apibrėžia atminties sritį, bendrą visoms programoms, turinčioms šį operatorių. Yra nežymėtoji operatoriaus forma - COMMON be vardo, ir žymėtoji - su vardu. Programoje gali būti tik vienas nežymėtasis COMMON ir keli žymėtieji. Operatoriaus bendrasis pavidalas yra

COMMON [*/vardas/*] *sąrašas*

Sąraše gali būti kintamieji ir masyvų vardai (be matavimų arba su matavimais-konstantomis - šiuo atveju masyvams kartu išskiriama atmintis). Galima į COMMON *sąrašą* įtraukti tik masyvų vardus, o juos aprašyti kitais mums jau žinomais operatoriais.

Sąrašai visose programose, turinčiose atitinkamą COMMONą, yra sulyginami pagal kiekvieną *sąrašo* elementą. Šie elementai saugomi tose pačiose atminties ląstelėse - o tai ir leidžia programoms viena kitai perduoti duomenis per šią bendrą atminties sritį. Atitinkami *sąrašo* elementai skirtingose programose gali turėti skirtingus vardus, tačiau turėtų būti vienodo tipo ir ilgio. Yra ir šios taisyklės išimčių, tačiau čia jų nenagrinėsime (žr. [1]).

Pavyzdys tarkim, kad vienoje programoje yra operatoriai

```
REAL  a, b
INTEGER*2  k
COMMON  /c1/  a, k, b(3)
```

o kitoje -

```
REAL  x, y1, y2, y3
INTEGER*2  m
COMMON  /c1/  x, m, y1, y2, y3
```

Šiose programose bus toks duomenų ekvivalentiškumas:

$a \equiv x,\ k \equiv m,\ b(1) \equiv y1,\ b(2) \equiv y2,\ b(3) \equiv y3$.

Jei trečiojoje programoje norėtume turėti kintamąjį z, lygų b(1) ir y1, o kitų kintamųjų reikšmės trečiai programai būtų nereikalingos, tai srities /c1/ sąraše z

įrašytume trečiuoju, o pirmaisiais - du bet kokius fiktyvius vardus, kurių antrasis būtinai būtų dviejų baitų ilgio:

```
REAL  f1
INTEGER*2  f2
COMMON  /c1/  f1, f2, z
```

Apribojimai COMMON sritims. Į COMMON sąrašus negalima įtraukti vardų duomenų iš argumentų sąrašų arba operatoriaus EQUIVALENCE sąrašų. Negalima duomenų vardams iš žymėtojo COMMON sąrašo parinkti pradinių reikšmių operatoriumi DATA - tai reikia atlikti specialia paprograme BLOCK DATA.

Paprogramisje BLOCK DATA galimi tik operatoriai žymėtieji COMMON, DATA, PARAMETER, EQUIVALENCE, DIMENSION ir tipo apibrėžimo operatoriai. Jei programoje reikalingas tik vienas paprogramis, jis gali būti bevardis; jei keli - paprogramio vardas būtinas.

Pavyzdys Programoje turime

```
INTEGER  k1, k2
REAL x(20)
COMMON  /c2/  k1, k2, x
```

ir norime suteikti šiems kintamiesiems nulines reikšmes operatoriumi DATA. Prieš programą tada turėsim parašyti tokį paprogramį:

```
BLOCK  DATA
INTEGER  k1, k2
REAL  x(20)
COMMON  /c2/  k1, k2, x
DATA  k1, k2, x  / 2*0, 20*0. /
END
```

* 7.8. OPERATORIUS SAVE

SAVE [*vardų sąrašas*]

vardų sąraše išvardintiems kintamiesiems, masyvams arba žymėtųjų COMMON sričių visiems elementams leidžia išsaugoti paprogramisje įgytas reikšmes. *Vardų sąraše* negali

būti paprogramių argumentų. COMMON sričių vardai pateikiami kartu su brūkšneliais.

Pavyzdys

```
      LOGICAL  called /.FALSE./
      CALL  sub (called)
      called = .TRUE.
      CALL  sub (called)
      END
C
      SUBROUTINE  sub (called)
      LOGICAL  called
      INTEGER  k
      SAVE  k
      IF (called .EQV. .FALSE.)  THEN
        k = 1
      ELSE
        k = k + 1
      END IF
      WRITE (*,*)  k     ! Pirmo paprogramio kvietimo metu spausdins
                                 !      reikšmę 1
                         ! Antro paprogramio kvietimo metu spausdins
                                 !      reikšmę 2
      END
```

* 8. NAUJOS FORTRANO 90 GALIMYBĖS

Skyriuje aptariami dalykai priklauso tik FORTRANui 90. Primename, kad daugelį nesudėtingų 90-ojo standarto dalykų (programos teksto formatą, operatorių tvarką programoje, vardų taisykles, vykdomuosius operatorius) jau esame išnagrinėję ankstesniuose skyriuose. Tai buvo įmanoma, nes tie operatoriai (tarkim, SELECT CASE ar DO WHILE) neįneša nieko revoliucingo lyginant su FORTRANu 77; be to, daugelis FORTRANo 77 kompiliatorių šiuos operatorius ir pripažino kaip FORTRANo 77 išplėtas. Šiame gi skyriuje pateiksime tas konstrukcijas, kurios iš esmės papildo 77-ąjį standartą. Apie jas anksčiau kalbėti būtų buvę gal ir logiškiau, tačiau tai būtų keblu.

Dauguma FORTRANo 90 nevykdomųjų operatorių yra tiek sudėtingi, kad jų bendrajam pavidalui pateikti sukurta netgi speciali sintaksė. Mes ja čia neužsiimsim, o operatorius pradėsim aiškinti nuo atskirų jų galimybių, dažnai nuo pavyzdžių pereisim prie bendrųjų formų. Beje, visos FORTRANui 90 skirtos knygos parašytos kaip tik tokiu stiliumi.

Aprašysime tik tas kalbos konstrukcijas, kurios yra įtrauktos į FORTRANo 90 standartą, todėl lieka nepaminėti kai kurie paplitusių asmeninių kompiuterių kompiliatorių operatoriai. Dėl šios priežasties skyriuje aprašomas išvestinių duomenų deklaravimo operatorius TYPE, o ne Microsoft FORTRANo analogiškas operatorius STRUCTURE ir pan. Kai kurie svarbesni standarto ir Microsoft FORTRANo neatitikimai paminėti knygelės priede.

Daugelį veiksmų (pavyzdžiui, apibrėžiant kintamojo tipą ir suteikiant pradinę reikšmę) 90-ajame standarte galima atlikti keliais alternatyviais būdais. Stengsimės programų pavyzdžiais iliustruoti kiek įmanoma daugiau tokių alternatyvų.

Kadangi Jūs jau susipažinę su 77-uoju FORTRANu, visas žinias šiame skyriuje pateiksime glausčiau.

Keli sudėtingesni šio skyriaus programų pavyzdžiai arba jų idėjos paimti iš J.Kerrigano ir Br. Hahno knygų.

8.1. DUOMENYS

Yra *vidiniai* duomenų tipai (intrinsic types) ir *išvestiniai* (derived types).

Vidinių duomenų tipai: trys aritmetiniai - sveikieji, realieji ir kompleksiniai (INTEGER, REAL, COMPLEX) ir du nearitmetiniai - loginiai ir tekstiniai (LOGICAL, CHARACTER). Kiekvieno tipo duomenys dar skirstomi į atskiras *rūšis*. Aritmetiniai ir loginiai duomenys turi tam tikras apibrėžtas duomenų rūšis. Duomens rūšis parenkama nurodant duomens *tipo rūšies parametrą* (kind type parameter).

Rūšies parametras apibrėžia:

- sveikiesiems duomenims - duomens ilgį baitais, o kartu ir duomens kitimo diapazoną;
- realiesiems ir kompleksiniams - duomens ilgį baitais, o kartu ir duomens kitimo diapazoną bei duomens tikslumą;
- loginiams - duomens ilgį baitais;
- tekstiniams duomenims yra ilgio ir rūšies parametrai. Pirmas apibrėžia duomens ilgį baitais (ir simboliais), o antras - simbolių komplekto numerį. Šio parametro reikšmė "1" atitinka ASCII simbolius, o kitoms parametro reikšmėms gali būti priskirtos kitokios simbolių sekos, pavyzdžiui, kinų hieroglifų grafiniai vaizdai.

Daugumai procesorių leistinos tokios rūšies parametrų reikšmės: sveikiesiems ir loginiams duomenims - 1, 2, 4, 8; realiesiems ir kompleksiniams - 4, 8, 16 (t.y. rūšies parametro reikšmė sutaps su duomens ilgiu baitais). Jei duomens rūšies parametras nenurodytas, duomuo laikomas standartinės rūšies (šie standartiniai duomenų ilgiai pateikti 3-iame skyriuje).

Konstantos. Norint naudoti nestandartinę aritmetinės ir loginės konstantos rūšį, šalia konstantos po apatinio brūkšnelio rašomas rūšies parametras sveikosios konstantos arba sveikojo kintamojo pavidalu:

konstanta_rūšies parametras

Tekstinei konstantai rūšies parametras rašomas pirma:

rūšies parametras_tekstinė konstanta

Pavyzdžiai: 1234567890_8 - 8-os rūšies sveikoji konstanta; jos ilgis yra 8 baitai
1_"TEKSTAS" - 1-os rūšies tekstinė konstanta, kurios reikšmė yra TEKSTAS

Nežinant, kokia sveikojo duomens rūšis reikalinga norint operuoti sveikaisiais duomenimis iš intervalo $(-10^n, 10^n)$, galima kreiptis į vidinę FORTRANo 90 funkciją SELECTED_INT_KIND(*n*). Šios funkcijos rezultatas bus reikiama rūšies parametro reikšmė. Pavyzdžiui, jei programoje reikalingas sveikųjų duomenų diapazonas (-999999, 999999), tai atitinkamą rūšies parametro reikšmę galime sužinoti kreipiniu SELECTED_INT_KIND(6) . Jei norima duomens rūšis neįmanoma, tai funkcija grąžina reikšmę -1. Kitos dvi vidinės funkcijos atlieka atvirkščius veiksmus: KIND(*duomuo*) ir RANGE(*duomuo*) grąžina atitinkamai *duomens* rūšies parametro reikšmę ir *duomens* kitimo diapazoną dešimties laipsnio rodikliu. Pavyzdžiui, kreipinio į funkciją KIND(100) rezultatas būtų standartinės sveikosios konstantos rūšies

parametro reikšmė - 2, o kreipinio RANGE(100) - standartinės rūšies sveikosios konstantos kitimo diapazono ilgis *n* - 4 (nes kitimo diapazonas yra -32768 ÷ 32767).

Nežinant, kokia realiojo duomens rūšis reikalinga norint turėti tam tikras realiojo duomens charakteristikas, galima kreiptis į vidinę funkciją SELECTED_REAL_KIND(*p, n*), kur *p* yra norimas duomens tikslumas reikšminių skaitmenų kiekiu, o *n* prasmė tokia pat, kaip analogiškoje funkcijoje sveikiesiems duomenims. Ši funkcija, jei norimų charakteristikų duomuo neįmanomas, grąžina neigiamas reikšmes: -1 - kai negalima pasiekti norimo tikslumo; -2 - kai negalima pasiekti norimo kitimo diapazono, -3 - kai neįmanoma pasiekti nei norimo tikslumo, nei norimo diapazono. Atvirkščius veiksmus - suteikia informaciją apie duomenį - atlieka kitos trys vidinės funkcijos: KIND(*duomuo*), RANGE(*duomuo*), PRECISION(*duomuo*). Pirmos dvi iš jų jau aptartos, o trečioji suteikia informaciją apie duomens tikslumą reikšminių skaitmenų kiekiu.

Kompleksinė konstanta susideda iš dviejų atskirų sveikojo arba realiojo tipo konstantų, kurių pirmoji reiškia realiąją kompleksinės konstantos dalį, o antroji - menamąją (žr. 3-ią skyrių). Jei viena konstantos dalis yra sveikojo tipo, tai visos konstantos rūšis sutaps su kitos dalies rūšimi (išskyrus atvejį, kai abi dalys yra sveikosios - tada konstanta bus standartinės realiosios rūšies). Jei abi dalys yra realiosios, tai konstantos rūšis sutaps su tikslesnės dalies rūšimi. Reikia neužmiršti, kad kompleksinei konstantai būtina dviguba atmintis: pavyzdžiui, kai abi dalys yra standartinės realiosios rūšies - 4-ių baitų ilgio, tai visa konstanta bus 8-ių baitų ilgio. Kompleksiniams duomenims gali būti taikomos jau minėtos funkcijos KIND, RANGE, PRECISION.

Loginiai duomenys nuo 77-ojo standarto loginių duomenų skiriasi tik didesniu duomens ilgių kiekiu.

Standartinė tekstinių duomenų rūšis - ASCII simboliai.

Loginiams ir tekstiniams duomenims gali būti taikoma funkcija KIND.

Kintamieji. Kaip ir 77-ame standarte, jų tipas nustatomas nutylėjimo principu, nurodomas operatoriumi IMPLICIT arba tipo apibrėžimo operatoriais. Tačiau pastarųjų bendrasis pavidalas kiek skirtingas:

tipas [, *atributų sąrašas*] [::] *kintamųjų sąrašas* / *reikšmių sąrašas* / .

Paskiro kintamojo reikšmė gali būti suteikta ne tik skliausteliuose / / , bet ir po lygybės ženklo reiškiniu.

Atributai bus nagrinėjami vėliau. Jei *atributai* yra, reikalingas ir dvigubas dvitaškis.

Tipas gali būti: CHARACTER [**ilgis* |*tipo rūšis ir ilgis*], COMPLEX [*tipo rūšis*], DOUBLE COMPLEX, DOUBLE PRECISION, INTEGER [*tipo rūšis*], LOGICAL [*tipo rūšis*], REAL [*tipo rūšis*], TYPE [*tipo vardas*]. Visi šie duomenų tipai mums pažįstami, išskyrus tipą TYPE, apie kurį kalbėsime šiame skyriuje vėliau. *Tipo rūšis*

nurodoma taip: ([KIND =] *tipo rūšies parametras*). Tekstiniams duomenims be *tipo rūšies* dar galima nurodyti ir duomens *ilgį* taip: LEN = *ilgis* .

Pavyzdžiai: du pirmieji operatoriai atlieka tokius pat veiksmus - suteikia kintamajam t tekstinį tipą ir ląstelės ilgį 7 baitus bei reikšmę tekstas. Trečias operatorius suteikia kintamajam r 8 baitų ilgio ląstelę.

```
CHARACTER( KIND = 1, LEN = 7 )  t / 'tekstas' /
CHARACTER*7 :: t = 'tekstas'
REAL( 8 ) :: r
```

Be vidinių duomenų, programuotojas gali naudoti ir ***išvestinius duomenis,*** kurių struktūrą jis pats ir nustato. Išvestinį duomenį gali sudaryti tiek vidiniai duomenys, tiek ir anksčiau programuotojo aprašyti kiti išvestiniai duomenys. Toks duomuo taip pat vadinamas *struktūra*. Būtent šį terminą toliau ir naudosime. Struktūra aprašoma taip:

```
TYPE  [ [ , atributų sąrašas] ::]  duomens tipo vardas
duomens komponentų apibrėžimo operatoriai
END  TYPE  [duomens tipo vardas]
```

Atributai paskiria duomeniui vienokias ar kitokias savybes. Šiame skyriuje paminėsime tik tris atributus.

Išvestinio tipo konstanta užrašoma taip:

duomens tipo vardas (*komponentų reikšmių sąrašas*) .

Pavyzdžiai . Tarkim, norim saugoti tokią informaciją apie firmos darbuotojus: vardą - tai bus tekstinio tipo iki 20 simbolių ilgio kintamasis; pavardę - bus toks pat kintamasis; amžių - standartinės rūšies sveikasis kintamasis ir identifikavimo numerį - 8 baitų ilgio sveikasis kintamasis. Patogu būtų visą šią informaciją surašyti į vieną struktūrą vardu, tarkim, person. Šio duomens pirmą komponentą pavadinkim taip pat anglišku vardu firstname, antrą - lastname, amžiui skirtą komponentą - age, o numeriui - id_number. Duomenį aprašytume taip:

```
TYPE  person
        CHARACTER*20  firstname
        CHARACTER*20  lastname
        INTEGER  age
        INTEGER( KIND = 8)  id_number
END  TYPE  person
```

Kitas pavyzdys. Visą informaciją apie trikampio viršūnių a, b, c koordinates plokštumoje x ir y galėtume sutalpinti į struktūrą triangle, kurios komponentas pats būtų kita aukščiau apibrėžta struktūra point:

```
TYPE point
        REAL x, y
END TYPE point
TYPE triangle
        TYPE (point) a, b, c
END TYPE triangle
```

Dabar, kai struktūra person apibrėžta, galime kintamiesiems, pavyzdžiui, jonas ir petras paskirti tokio išvestinio duomens tipą:

```
TYPE (person) jonas, petras
```

Galime parašyti, pavyzdžiui, tokią išvestinę konstantą

```
person( 'Jonas', 'Petraitis', 20, 123456789_8 )
```

ir jos reikšmę priskirti kintamajam jonas:

```
jonas = person('Jonas', 'Petraitis', 20, 123456789_8 ) .
```

Struktūrai reikšmę galim suteikti ir tiesiog kintamojo aprašymo metu; galim vieno kintamojo-struktūros reikšmę priskirti kitam tokio pat tipo kintamajam:

```
TYPE (person) :: petras = person('Petras', 'Jonaitis', 30, 123456790_8)
jonas = petras
PRINT '( 2A20, 2I10)'  jonas
```

- dabar visi struktūros jonas komponentai įgyja naujas reikšmes - tokias, kokias turi struktūra petras. Kaip matyti iš šio pavyzdžio, struktūrą galim įrašyti ir įvesties/išvesties operatorių sąraše, kiekvienam struktūros komponentui skirdami tinkamą formatą.

Galima operuoti kiekvienu taip apibrėžtos struktūros komponentu atskirai, nurodant struktūros ir jos komponento vardus, atskirtus *komponento selektoriumi* - procento ženklu %. Pavyzdžiui, į kintamojo jonas komponentą identifikavimo numeris galime kreiptis jonas%id_number; Jono ir Petro amžiaus skirtumą gautume

aritmetiniu reiškiniu jonas%age - petras%age ir pan. Jei kintamajam t būtų suteiktas tipas triangle, tai galėtume operuoti ir jo subkomponentais t%a%x bei t%a%y. Struktūros komponentas gali būti ir masyvas. Galimi ir išvestinio tipo masyvai. Pavyzdžiui, firmos visų darbuotojų duomenys būtų saugomi masyve employes, kurio matavimai nurodomi atributu DIMENSION:

```
TYPE (person), DIMENSION (100) :: employes
```

Atributo DIMENSION funkcijos ir forma tokios pat, kaip mums pažįstamo savarankiško operatoriaus DIMENSION. Kreipinys employes%id_number dabar išskirs iš karto 100 komponentų, o employes%id_number(1) - pirmojo darbuotojo identifikavimo numerį.
Tuo tarpu kreipinys employes(10) išskirs dabar struktūrą - 10-ąjį masyvo employes elementą. Kaip žinoma, atskiras struktūros komponentas savo ruožtu gali būti ir masyvas, tačiau FORTRANas 90 šį duomenį vis tik laikys *skaliaru*. Kad sąvokos *skaliaras* ir *kintamasis* būtų aiškios (to mums reikės vėliau), čia derėtų pateikti FORTRANe 90 galiojančias konvencijas:

- tik duomuo, kuris nėra dalis didesnio duomens ir turi vardą, yra vadinamas *įvardintu objektu*
- jo *subobjektai* turi objekto vardą ir papildomus skyriklius, pavyzdžiui,%
- *masyvas* yra bet kuris objektas, nesantis skaliaru
- *kintamasis* yra įvardintas objektas, kuris nėra nurodytas būti konstanta, ir nesantis jokiu subobjektu

Funkcijos FORTRANe 90 taip pat gali būti struktūros tipo: šiuo atveju funkcija kviečiančiajai programai grąžina ne vieną, kaip 77-ame standarte, o iš kelių komponentų susidedantį rezultatą. Tokios funkcijos vardas funkcijoje turi būti apibrėžtas kaip struktūra. Kadangi dar nežinom FORTRANo 90 subprogramų, pavyzdžius pateiksim masyvams skirtame skyriuje.
Atributų sąraše galimi atributai PUBLIC ir PRIVATE. Jie turi prasmę tik programoje, suskaidytoje į subprogramas (žr. 8.2 skyrių). Jei struktūrai nurodytas atributas PRIVATE, tai tokios struktūros vardas ir jos komponentai bus žinomi tik struktūrą talpinančiai subprogramai-moduliui. PUBLIC reiškia, kad šie duomenys bus prieinami visoms modulį naudojančioms programoms. Beje, šių atributų nepateikus, pagal nutylėjimą naudojama PUBLIC reikšmė.

8.2. SUBPROGRAMOS IR MODULIAI

Vietoj ankstesniojo termino “paprogramiai” dabar vartosim bendresnį “subprogramos”.

Yra *vidinės* ir *išorinės* subprogramos; jos gali būti grupuojamos į *modulius*. Pagrindinė programa, išorinės subprogramos ir moduliai bus dar vadinami *programiniais vienetais*. Vidinės subprogramos įrašomos kitos programos viduje, todėl ir kompiliuojamos kartu su ta programa. Išorinės subprogramos kompiliuojamos atskirai nuo kviečiančios programos.

8.2.1. VIDINĖS SUBPROGRAMOS

Yra du jų tipai - *funkcijos* ir *paprogramiai*. Daugelis jau pažįstamų 77-ojo standarto funkcijų ir paprogramių savybių išlaikytos, todėl jų nekartosim, o sutelksim dėmesį į naujus dalykus.

Vidinės subprogramos yra jas kviečiančių programų viduje tarp operatorių CONTAINS ir END. Vidinės subprogramos negali savyje talpinti kitų vidinių subprogramų.

Bendrasis ***vidinės funkcijos*** pavidalas yra:

FUNCTION *vardas*([*formaliųjų argumentų sąrašas*]) [RESULT(*rezultato vardas*)]
...
vardas | *rezultato vardas* = *rezultatas*
...
END FUNCTION [*vardas*]

Funkcijos kūne galimi (bet nebūtini) ir mums pažįstami operatoriai RETURN. Jei kviečiančiojoje programoj yra IMPLICIT NONE operatorius, tai funkcijos vardas ir argumentai turi būti aprašyti tipo apibrėžimo operatoriuose. Tą nebūtina padaryti kviečiančiojoje programoje, galima apibrėžti ir funkcijos kūne. Nebūtinas žodis RESULT nurodo, kad funkcijos rezultatas yra *rezultato vardas*. Šiuo atveju *vardas* negali būti apibrėžtas jokiame tipo apibrėžimo operatoriuje funkcijos viduje, o turi būti apibrėžtas kviečiančiojoje programoje. Funkcijos iškvietimo forma lieka ta pati. Pavyzdžiai su šia nauja funkcijos forma pateikiami 8.3.1 skyriuje.

Vidinei subprogramai automatiškai prieinami visi kviečiančios programos kintamieji (čia ir toliau šį terminą naudosim ir apibendrinta prasme: jis gali reikšti ne tik kintamąjį, bet ir masyvą, jo elementą, tekstinę eilutę). Todėl visi kviečiančiosios programos kintamieji yra *globalūs* - juos žino programos visos vidinės subprogramos. Ši vidinių subprogramų savybė “padeda” programoje padaryti sunkiai aptinkamas subtilias klaidas, todėl derėtų rašyti tik trumpas vidines subprogramas, o visas dideles

apiforminti kaip išorines. Be to, jeigu kviečiančiosios programos kintamasis vidinėje subprogramoje apibrėžtas dar kartą, tai vidinė subprograma jau “nežino” originalaus to pat vardo kintamojo iš kviečiančiosios programos.

Pavyzdys. Skaičių nuo 1 iki 10 faktorialų skaičiavimas. n skaičiaus faktorialo skaičiavimas įforminamas kaip vidinė funkcija.

```
PROGRAM  factorial
IMPLICIT  NONE
INTEGER  i
                  ! FORTRANe 90 galimos tuščios eilutės. Jas ir naudosim vietoj
DO i = 1,10     ! “tuščių” komentarų programos logiškai skirtingoms dalims atskirti
  PRINT *, ‘skaičius ir jo faktorialas’, i, fact( i )
END  DO

CONTAINS
  FUNCTION  fact( n )
    INTEGER  n, i
    INTEGER*4  fact, temp
      temp = 1
      DO i = 2, n
        temp = i*temp
      END  DO
    fact = temp
  END  FUNCTION  fact
END
```

Pastaba. Iškart pabrėšim vieną svarbiausių šiam pavyzdžiui dalykų: kintamąjį i subprogramoje būtina dar kartą aprašyti tipo apibrėžimo operatoriuj. Kitaip jis liktų globalus. Jau po pirmo funkcijos kvietimo pagrindinei programai būtų grąžinta i reikšmė 2, likusi iš funkcijos ciklo (šis ciklas nebūtų vykdomas, nes i>n) - t.y. būtų papildomai pakeista programos ciklo kintamojo reikšmė.

Dar viena neesminė pastaba. FORTRANe 90 pagrindinę programą, kaip ir subprogramas, galima užbaigti ne tik taip, kaip šiame pavyzdyje, bet ir įvardintu operatorium END:

END [PROGRAM [*programos vardas*]]

Taigi programą galėjom baigti ir operatoriais END PROGRAM arba END PROGRAM factorial.

Vidiniai paprogramiai. Tai - analogiškas vidinėms funkcijoms dalykas. Todėl vietoj kokių nors aiškinimų apsiribosim vienu pavyzdžiu.

Pavyzdys. Vidinis paprogramis swop sukeičia vietomis du skaičius a ir b:

```
IMPLICIT NONE
REAL a, b

READ *, a, b
PRINT *, 'duomenys', a, b
CALL swop( a, b )
PRINT *, 'rezultatas', a, b

CONTAINS
  SUBROUTINE swop( x, y )
    REAL temp, x, y
      temp = x
      x = y
      y = temp
  END SUBROUTINE
END PROGRAM
```

8.2.2. IŠORINĖS SUBPROGRAMOS

Vėlgi turim išorines funkcijas ir išorinius paprogramius. Tokios subprogramos kompiliuojamos atskirai nuo jas kviečiančiųjų programų. Išorinės subprogramos gali turėti savo vidines subprogramas.

Bendrasis išorinių subprogramų pavidalas yra:

```
FUNCTION vardas([formaliųjų argumentų sąrašas]) [ RESULT(rezultato vardas) ]
   ...
vardas | rezultato vardas = rezultatas
   ...
 [ CONTAINS
     vidinės subprogramos ]
END [ FUNCTION [ vardas ] ]
ir
```

```
SUBROUTINE vardas [ (formaliųjų argumentų sąrašas) ]
   ...
 [ CONTAINS
     vidinės subprogramos ]
END [ SUBROUTINE [ vardas ] ]
```

Pavyzdys. Perrašysim paskutinio pavyzdžio programą, naudodami išorinę subprogramą:

```
IMPLICIT NONE
EXTERNAL swop
REAL a, b

READ *, a, b
PRINT *, 'duomenys', a, b
CALL swop( a, b )
PRINT *, 'rezultatas', a, b

END PROGRAM


SUBROUTINE swop( x, y )
  REAL temp, x, y
    temp = x
    x = y
    y = temp
END SUBROUTINE
```

Pastaba. Operatorius EXTERNAL, aišku, yra nebūtinas. Tačiau mes rekomenduojam jį rašyti ir taip apsidrausti nuo tokios situacijos, kai mūsų išorinės subprogramos vardas atsitiktinai sutaptų su FORTRANo 90 standartinės funkcijos vardu. Jei taip atsitiktų programoje nesant EXTERNAL, tai būtų kviečiama standartinė funkcija. Vardų sutapimas yra tikėtinas, nes 90-asis standartas siūlo kelis kartus daugiau nei ankstesnis standartinių funkcijų.

8.2.3. SĄSAJŲ BLOKAI (INTERFEISAI)

Kad kompiliatorius tinkamai paruoštų subprogramų iškvietimus, jis turi žinoti subprogramos vardą, argumentų kiekį, tipą ir pan. Visa ši informacija ir yra *subprogramos sąsaja* (arba interfeisas - iš angl. interface). Visada *tiksliai* žinoma

(explicit interface) vidinės subprogramos, standartinės funkcijos ir modulinės subprogramos (žr. 8.2.4 skyrių) sąsaja. Generuojant išorinės subprogramos iškvietimą, tokia informacija neįmanoma. Paruošta sąsaja dabar bus ne tiksli, o *numatoma* (implicit interface). Kartais, kaip vėliau matysim, yra būtina tik tiksli sąsaja. FORTRANe 90 yra mechanizmas - sąsajos blokas - kuris gali numatomą sąsają paversti tikslia:

```
INTERFACE
  sąsajos kūnas
END  INTERFACE
```

Sąsajos kūne yra tiksli subprogramos deklaravimo, jos objektų aprašymo ir jos pabaigos operatoriaus kopija. Tačiau argumentų vardai gali būti pakeisti, o deklaruojamoji dalis papildyta kitais apibrėžimo operatoriais. Kai kurie apibrėžimai, pavyzdžiui, lokalių subprogramos kintamųjų, gali būti praleisti. Kūne negali būti DATA ir FORMAT operatorių.

Pavyzdys. Paskutinę programą perrašysime naudodami sąsajos bloką:

```
IMPLICIT  NONE
 INTERFACE
  SUBROUTINE  swop( x, y )
    REAL  x, y                 !  temp  galima neaprašyti  -  lokalus kintamasis
  END  SUBROUTINE
 END  INTERFACE

REAL  a, b
READ *, a, b
PRINT *, 'duomenys', a, b
CALL  swop( a, b )
   ...
```

Kada naudoti sąsajų blokus? Juos galima rašyti visoms išorinėms subprogramoms. Yra situacijų, kai šie blokai tiesiog būtini. Visos šios situacijos bus paaiškintos kituose skyriuose. Dabar tik išvardinsime šiuos atvejus:

- kai subprograma kviečiama nurodant būtinus ir pasirenkamus argumentus
- kai funkcijos rezultatas - masyvas
- kai subprogramos argumentas - numanomos formos masyvas, taikinys ar nuoroda

- kai subprograma kviečiama bendriniu vardu
- kai subprograma “perkrauna” veiksmų operatorių ar suteikia papildomą turinį prieskyros operatoriui
- kai pati subprograma yra formalusis ar faktiškasis argumentas

8.2.4. MODULIAI

Moduliai talpina apibrėžimo operatorius, kurie tampa pasiekiami visoms modulį naudojančioms programoms. Kadangi tie apibrėžimo operatoriai užima nemenką programos dalį, tai moduliai dažnu atveju padeda žymiai patogiau organizuoti informacijos mainus tarp subprogramų. Moduliai yra kompiliuojami atskirai nuo juos naudojančių programų.

Bendroji modulio forma yra:

MODULE *modulio vardas*
  [*apibrėžimo operatoriai*]
[CONTAINS
  modulinės subprogramos]
END [MODULE [*modulio vardas*]]

Modulis tampa prieinamas programiniam vienetui, jei šio pradžioje yra operatorius

USE *modulio vardas*

Modulinės subprogramos yra lygiai tokios pat, kaip ir išorinės. Vienintelis skirtumas - jų END operatoriuose negalima praleisti žodžių FUNCTION ar SUBROUTINE. Modulinės subprogramos “žino” visus modulyje aprašytus kintamuosius. Modulinės subprogramos savo ruožtu gali turėti vidines subprogramas.

Pavyzdys. Perrašysim ankstesnįjį pavyzdį, paprogramį swop padarydami moduliniu modulio my_module paprogramiu. Iliustracijos dėlei modulyje deklaruosim dar vardinę konstantą pi. Ši konstanta dėl to bus žinoma programai, todėl jos reikšmę galime suteikti kintamajam b. Programos ir modulio tekstai:

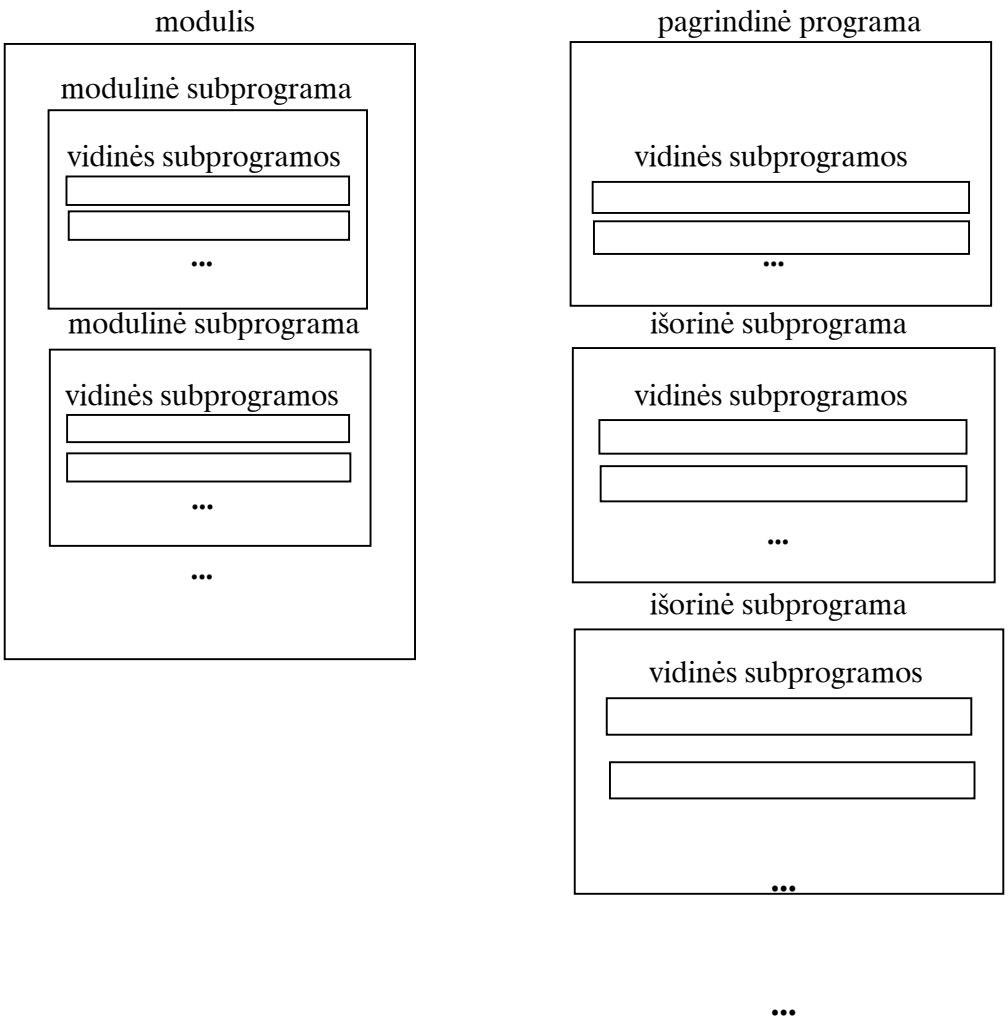
```
USE my_module
IMPLICIT NONE
REAL a, b

READ *, a
```

```
PRINT *, 'duomenys', a
b = pi
CALL  swop( a, b)
PRINT *, 'rezultatai', a, b

END  PROGRAM


MODULE  my_module
  REAL, PARAMETER  ::  pi = 3.141593   !  PARAMETER gali būti ir atributu
CONTAINS
  SUBROUTINE  swop( x, y)
    REAL  temp, x, y
      temp = x
      x = y
      y = temp
  END SUBROUTINE  swop
END MODULE  my_module
```

Pastaba 1. Šiame pavyzdy EXTERNAL jau ne tik nereikalingas, bet ir nekorektiškas, kadangi swop dabar yra ne išorinė subprograma, o vidinė.

Pastaba 2. Šiame pavyzdyje panaudotas naujas atributas PARAMETER. Jo prasmė tokia pat, kaip ir analogiško anksčiau išmokto operatoriaus. Beje, FORTRANe 90 šį atributą, kaip ir daugumą kitų, leidžiama rašyti ir kaip savarankišką operatorių (prisiminkim kad ir DIMENSION).

Dabar, kai jau žinome visus tris 90-ojo standarto programinių vienetų tipus, schemiškai pavaizduosime galimą programos struktūrą (8.1 pav.).



8.1 pav. Galima programos struktūra

Modulis gali naudoti kitus modulius, tačiau negali kreiptis į save patį (pavyzdžiui, per kelis operatorius USE skirtinguose moduliuose).

Pagrindinė modulinės struktūros savybė - modulis gali talpinti apibrėžimo operatorius, kurie pasidaro prieinami visiems modulį naudojantiems programiniams vienetams. Tai ypač patogu, kai naudojami sudėtingi išvestinio tipo duomenys.

Pavyzdys. Parašysim du modulius kintamųjų π ir e reikšmių suteikimui: vieną - kai reikalingas viengubas konstantų tikslumas, o kitą - kai dvigubas, 15-os skaitmenų tikslumas ir kitimo ribos iki 1.E±100 .

```
MODULE  single
  REAL :: pi = 3.141593
  REAL :: e = 2.718282
END MODULE  single
```

```
MODULE  double
  INTEGER, PARAMETER :: double_precision =     &
                                        SELECTED_REAL_KIND(15, 100)
    REAL(double_precision) :: pi = 3.141592653589793_double_precision
    REAL(double_precision) :: e  = 2.718281828459045_double_precision
END MODULE  double
```

Dabar, rašydami programą, kuriai užtenka viengubo tikslumo ir kuriai reikia pi ir e reikšmių, pajungsime modulį single:

```
  ...
USE  single
  ...
```

Analogiškai, prireikus didesnio tikslumo, programai pajungtume modulį double.

Dar kelios pastabos dėl operatorių USE naudojimo. Kartais, su USE pajungiant modulį, patogu naudoti kitokius nei modulyje vardus: pavyzdžiui, kai naudojami keli atskirai parašyti moduliai, jie gali turėti kelis sutampančius kintamųjų vardus. Šiuo atveju pajungiant modulį, reikia pakeisti tuos vardus:

```
USE  modulio vardas, naujas_kintamojo_vardas =>
                                        senas_kintamojo_vardas [, ...]
```

Kitas būdas, keičiantis (ir apribojantis) priėjimą prie modulio objektų, yra parametras ONLY, po kurio išvardinami tie vardai, kuriuos turi žinoti modulį naudojantis programinis vienetas. Pavyzdžiui,

```
USE  your_module, ONLY : x, y
```

leidžia modulį naudojantiems programiniams vienetams žinoti tik modulio kintamuosius x ir y . Be to, čia, kaip ir ankstesniu atveju, dar galima ir kintamųjų vardus pakeisti.

Atributai PRIVATE ir PUBLIC. Kaip matėm, visi modulio kintamieji yra prieinami pagal nutylėjimą. Tačiau priėjimas gali būti apribotas, jei kintamojo apibrėžimo operatoriuje bus nurodytas atributas PRIVATE (kuris gali būti parašytas ir savarankišku operatorium). Pavyzdžiui,

```
    REAL, PRIVATE :: x
```

reiškia, kad pajungus modulį kintamasis x liks neprieinamas jokiam moduliu besinaudojančiam programiniam vienetui. PUBLIC, aišku, turi atvirkščią prasmę.

Operatoriai PUBLIC ir PRIVATE be vardų sąrašų gali pakeisti nutylėjimo principą. Pavyzdžiui,

```
PRIVATE
PUBLIC x
```

reiškia, kad visi modulio kintamieji - privatūs, išskyrus x .

8.2.5 ŽYMIŲ IR VARDŲ VIENAREIKŠMIŠKUMO SRITIS

Tai - programinio vieneto dalis, kurioje konkreti žymė ar konkretus vardas turi būti unikalūs ir negali būti panaudoti kitam tikslui. Šia prasme anksčiau išskyrėm globalius ir lokalius vardus FORTRANe 77.

Žymių vienareikšmiškumo sritis. Kiekviena subprograma turi nepriklausomą žymių sąrašą. Todėl, pavyzdžiui, pagrindinė programa ir jos vidinės subprogramos gali naudoti tą pačią žymę.

Vardų vienareikšmiškumo sritis. Sritis programoje deklaruotam vardui prasideda operatoriumi PROGRAM ir tęsiasi iki operatoriaus END, neišsiplėsdama į jokias išorines programos kviečiamas subprogramas. Visos programos vidinės subprogramos yra apimamos.

Sritis subprogramoje deklaruotam vardui tęsiasi nuo subprogramos deklaravimo operatoriaus iki jos pabaigos operatoriaus END, taigi apima visas subprogramoje esančias vidines subprogramas.

Sritis vidinėje subprogramoje deklaruotam vardui apima tik šią vidinę subprogramą.

Sritis vidinės subprogramos vardui bei argumentams apima visą šią vidinę subprogramą ir visą kviečiančiąją programą (subprogramą).

Sritis kintamojo ar subprogramos vardo, deklaruoto modulyje, apima visus modulį naudojančius programinius vienetus. Tačiau sritis neapims tų vidinių subprogramų, kuriose vardas papildomai deklaruotas (panašų pavyzdį žiūrėkite 8.2.1 skyriuje - faktorialų skaičiavimo programa).

Iš čia išvardintų taisyklių išeina:

- skirtingose vienareikšmiškumo srityse deklaruoti objektai, nors ir turėdami vienodus vardus, turės skirtingas prasmes

- vienoje vienareikšmiškumo srityje visi objektai privalo turėti unikalius vardus. Išimtis: subprogramų bendriniai vardai (žr. 8.2.7 skyrių)
- programinių vienetų vardai programoje yra globalūs, todėl turi būti unikalūs

8.2.6. ARGUMENTAI

Argumentų perdavimas. Faktiškieji argumentai subprogramoms perduodami dviem skirtingais būdais: arba *nuoroda*, arba savo *reikšme*.

Jei faktiškasis argumentas yra kintamasis, tai jis perduodamas nuoroda. Šiuo atveju subprogramai paprasčiausiai perduodamas pirmosios iš atminties ląstelių, kuriose saugomas faktiškasis argumentas, adresas. Taigi formalusis ir faktiškasis argumentai užima tą pat atminties vietą. Visi subprogramoje formaliojo argumento patirti pokyčiai automatiškai keičia kviečiančiosios programos faktiškojo argumento reikšmę.

Jei faktiškasis argumentas yra konstanta arba reiškinys, sudėtingesnis nei kintamasis, tai jis perduodamas subprogramai reikšme. Dabar formaliojo argumento pokyčiai neturės įtakos kviečiančiajai programai. Perdavimo reikšme mechanizmas yra toks: padaroma atminties srities, kurioje saugomas perduodamas argumentas, kopija. Tais atvejais, kai perduodamas reikšme argumentas yra didelės apimties masyvas, tai gali būti itin neekonomiška.

Jeigu kintamojo vardą apimsime skliaustais, tai bus jau reiškinys, todėl kreipinyje

```
CALL  swop( (x), y )
```

pirmasis tikrasis argumentas būtų perduotas paprogramei swop reikšme, o antrasis - nuoroda. Jei paprogramė pakeistų x reikšmę, tai kviečiančiajai programai liktų nežinoma. Tokie dalykai gali būti sunkiai aptinkamų subtilių klaidų priežastis. Nuo jų apsidrausti patogiausia formaliesiems argumentams suteikiant *ketinimo atributo* INTENT reikšmes: jei numatoma argumentą naudoti kaip įėjimo argumentą - suteikti atributo reikšmę IN, jei kaip išėjimo - OUT, o jei dvejopai - INOUT.

Pavyzdys

```
SUBROUTINE  correct( x, y, z )
  REAL, INTENT( IN )     :: x
  REAL, INTENT( OUT )  :: y
  REAL, INTENT( INOUT )       :: z
```

Jei argumento ketinimo atributo reikšmė yra IN, tai formaliojo argumento subprogramoje keisti negalima. Jei reikšmė yra OUT arba INOUT, tai atitinkamas faktiškasis argumentas privalo būti kintamasis. Aukščiau parašytam pavyzdžiui kreipinį

```
CALL  correct( x, (y), z )
```

kompiliatorius pripažintų klaidingu, nes antrasis faktiškasis argumentas turi būti kintamasis, o ne reiškinys.

Ketinimo atributas gali būti rašomas ir kaip atskiras operatorius, pavyzdžiui,

```
INTENT  (INOUT)   x, y, z
```

Būtini ir pasirinktiniai argumentai. Formaliųjų argumentų sąrašas gali būti ilgas; kartais ne visi argumentai reikalingi konkrečiu subprogramos taikymo atveju. Tada yra patogu tuos kartais nereikalingus argumentus pažymėti atributu OPTIONAL (arba tokiu pat savarankišku operatorium), o subprogramos kvietimo metu tuomet bus galima apsieiti tik *būtinais* ir reikalingais *pasirinktiniais* argumentais. Pastarųjų pateikimo forma kiek kitokia. Paprasčiausia tai bus paaiškinti konkrečiu pavyzdžiu; čia tik pasakysim, kad norint naudoti pasirinktinius argumentus yra būtina tiksli sąsaja.

Pavyzdys. Tegu paprogramio correct argumentų sąrašas, kaip matyti iš sąsajos, yra iš 6 argumentų, kurių būtini yra tik pirmieji du, o kiti - pasirinktiniai. Kviečiant paprogramį galimi tokie skirtingi argumentų pateikimo būdai:

```
INTERFACE
  SUBROUTINE  correct( au, av, aw, ax, ay, az )
    OPTIONAL  aw, ax, ay, az
  END  SUBROUTINE
END  INTERFACE
   ...
CALL  correct( a, b )          !  reikalingi ir naudojami tik būtinieji argumentai
   ...
CALL  correct( a, b, ax = d, az = e, ay = f )          !  iš pasirinktinių reikalingi trys
                                                       !  argumentai. Jie pateikiami tokia
                                                       !  forma. Jų eilės tvarka nesvarbi.
   ...
CALL  correct( a, b, c, d )    !  reikalingi tik du pirmieji iš pasirinktinių argumentų.
                               !  Kadangi praleisti turi būti paskutiniai pasirinktiniai
                               !  argumentai, tai pateikimo forma gali būti įprasta.
```

Atributas SAVE lokaliams subprogramų kintamiesiems “įsako” išsaugoti savo reikšmes tarp atskirų subprogramos kvietimų. Tie lokalūs kintamieji, kuriems pradinės

reikšmės suteiktos apibrėžimo operatoriais, automatiškai turi SAVE atributą. Formaliajam argumentui SAVE atributas negali būti priskirtas.

Argumentai - masyvai. Apie tai kalbama 8.3 skyriuje.

8.2.7. BENDRINIAI SUBPROGRAMŲ VARDAI

Dar iš FORTRANo 77 žinome, kad faktiškųjų ir formaliųjų argumentų sąrašai turi atitikti pagal kiekį, prasmę ir tipą. Tuo tarpu FORTRANo 90 standartinei funkcijai, pavyzdžiui, SIN galima perduoti sveikuosius, realiuosius arba kompleksinius tikruosius argumentus. Tai pasiekiama rašant atskiras, turinčias skirtingus vardus, subprogramas kiekvienam argumentų tipui, o visoms šioms subprogramoms sąsajos bloke dar suteikiamas vienas *bendrinis* vardas. Kviečiant subprogramą nurodomas bendrinis vardas; kompiliatorius pagal faktiškųjų argumentų tipą nusprendžia, kurią konkrečiai iš po bendriniu vardu besislepiančių subprogramų aktyvuoti.

Standartinių funkcijų, turinčių bendrinius vardus, sąrašas labai ilgas, todėl pateikiamas prieduose.

Pavyzdys. Prisiminkim vėl išorinį paprogramį swop dviems skaičiams sukeisti. Pertvarkysim swop taip, kad jis pripažintų ir realiuosius, ir sveikuosius argumentus. Tam reikės parašyti du atskirus paprogramius, sakykime, swopreals ir swopintegers vardais, o vėliau sąsajos bloku joms teks suteikti bendrinį vardą swop. Tada pagrindinėje programoje jau galėsim swop vardą naudoti tiek realiesiems, tiek sveikiesiems skaičiams sukeisti.

```
SUBROUTINE swopreals( x, y )
  REAL temp, x, y
    temp = x
    x = y
    y = temp
END SUBROUTINE swopreals
SUBROUTINE swopintegers( x, y )
  INTEGER temp, x, y
    temp = x
    x = y
    y = temp
END SUBROUTINE swopintegers
PROGRAM main
  INTERFACE swop
    SUBROUTINE swopreals( x, y )
```

```
      REAL temp, x, y
    END SUBROUTINE swopreals
    SUBROUTINE swopintegers( x, y )
      INTEGER temp, x, y
    END SUBROUTINE swopintegers
  END INTERFACE swop
REAL a, b
INTEGER k, l
  ...
CALL swop( a,b )
  ...
CALL swop( k, l )
  ...
```

Konkretus kurios nors subprogramos vardas gali sutapti su bendriniu. Bendrinis vardas gali sutapti su jau turimu kitu bendriniu vardu - dabar šis vardas apims visas po juo slypinčias programas. Taip galima, pavyzdžiui, išplėsti esamas standartines funkcijas, kad jos pripažintų mums reikiamus išvestinius duomenų tipus.

Kaip suteikti bendrinį vardą modulinėms subprogramoms? Tokių subprogramų, primenam, sąsaja jau yra tiksli, todėl nekorektiška ją būtų dar kartą tiksliai užrašyti INTERFACE bloke. Šiuo atveju paprasčiausiai užtenka sąsajos bloke parašyti operatorių

MODULE PROCEDURE *subprogramų vardai* ,

kur *vardai* - vardai subprogramų, kurioms norima parinkti bendrinį vardą.

Mūsų pavyzdžio su swop paprogramiu atveju, jei swopreals ir swopintegers būtų moduliniai paprogramiai, reikėtų tik tokios sąsajos:

```
INTERFACE swop
  MODULE PROCEDURE swopreals, swopintegers
END INTERFACE swop
```

INTERFACE bloku galima ne tik paskirti bendrinius vardus, bet ir aritmetinių veiksmų operatoriams ar prieskyros operatoriui suteikti kitokį turinį. Apie tai - 8.3 skyriuje.

8.2.8. REKURSINIS SUBPROGRAMŲ KVIETIMAS

Daugelis matematinių funkcijų apibrėžiamos rekursyviai, t.y. paprastesniais jų pačių atvejais. Pavyzdžiui, faktorialo skaičiavimą galim užrašyti rekursyviai, jei iš anksto sutarsim, kad 0! = 1 :

n! = n (n - 1)!

Panašias funkcijas galima realizuoti rekursinėmis, save kviečiančiomis, funkcijomis. Prieš žodį FUNCTION čia būtinas kitas standartinis žodis RECURSIVE, o po argumentų sąrašo - žodis RESULT, paskiriantis rezultato išsaugojimui reikalingą papildomą, būtinai kitokį funkcijos vardą. Dabar funkcijos vardas, parašytas po žodžio FUNCTION, jau nebegali būti apibrėžtas jokiame subprogramos tipo apibrėžimo operatoriuje. Paprasčiau užrašomas rekursinis paprogramis. Visus šiuos teiginius iliustruosim rekursiniu faktorialo skaičiavimu.

Pavyzdys. Skaičiaus 10 faktorialo skaičiavimas. Šį pavyzdį užrašysim dviem būdais: rekursine funkcija ir rekursiniu paprogramiu.

```
IMPLICIT  NONE
INTEGER     ::  k = 10
INTEGER*4  ::  factorial
PRINT *, k, factorial( k )
CONTAINS
  RECURSIVE  FUNCTION  factorial( n )  RESULT( fact )
    INTEGER  ::   n   !  factorial  neturi būti deklaruotas; to reikia tik po
                      !  RESULT  parodytam vardui. Vardas  factorial
                      !  reikalingas tik tarpiniam funkcijos kvietimui
    INTEGER*4  ::   fact
    IF( n == 0 )  THEN
       fact = 1
    ELSE
       fact = n * factorial( n-1 )     !  rekursinis kreipinys į save patį
    END  IF
  END  FUNCTION
END  PROGRAM
```

Tas pat uždavinys su rekursiniu paprogramiu:

```
IMPLICIT  NONE
```

```
INTEGER     :: k = 10
INTEGER*4  :: f
CALL  factorial( f, k )
PRINT *, f
CONTAINS
 RECURSIVE  SUBROUTINE  factorial( fact, n ) ! reikia žodžio RECURSIVE
    INTEGER     :: n
    INTEGER*4  :: fact

    IF( n == 0 )  THEN
       fact = 1
    ELSE
       CALL  factorial( fact, n-1 )     !  rekursinis kreipinys į save patį
       fact = n * fact
    END  IF
 END  SUBROUTINE
END  PROGRAM
```

Rekursyvus funkcijos ar paprogramio kvietimas apgaulingai atrodo nesudėtingas. Iš tikro rekursyvinei grandinei realizuoti atliekama daug veiksmų ir reikalaujama labai daug kompiuterio atminties ir laiko. Programuojant rekursiją reikėtų taikyti atsargiai.

8.3. GALIMYBĖS KEISTI VEIKSMŲ OPERATORIŲ PRASMĘ

Sakydami “veiksmų operatoriai” turim omeny aritmetinių veiksmų operatorius +, -, *, / ir loginių santykių operatorius .LT., .LE., .GT., .GE., EQ., .NE. (arba jų naujoji forma <, <=, >, >=, ==, /=). Dalis jų - dviviečiai operatoriai (+, -, *, ...), vienviečio operatoriaus pavyzdys būtų loginio neigimo operatorius (.NOT.), o operatoriai + ir - gali būti ir vienviečiai, ir dviviečiai. Anksčiau jiems buvo suteikta griežtai apibrėžta prasmė. Dabar, įvedus naujus duomenų tipus, vien šių veiksmų operatorių nebepakanka. Todėl FORTRANas 90 leidžia praplėsti turimų operatorių galimybes arba sukurti visai naujus veiksmų operatorius.

Faktiškai tokie nauji operatoriai yra tik alternatyvi funkcijos iškvietos forma. Tarkim, jei turim du išvestinius duomenis point_coordinates_1 , point_coordinates_2 ir funkciją atstumui tarp tų taškų nustatyti distance, tai tas pat rezultatas gali būti gaunamas tradiciniu būdu iškviečiant funkciją

distance_p1_p2 = distance(point_coordinates_1, point_coordinates_2)
arba
distance_p1_p2 = point_coordinates_1 - point_coordinates_2 ,

kur dviviečiam operatoriui "-" iš anksto turėtų būti suteikta kita, šiam uždaviniui reikalinga prasmė. Kitais žodžiais, operatorius "-" turėtų būti *perkrautas* (angl. overloaded).

Jei funkcijos rezultatas yra standartinio tipo duomuo, tai tokia FORTRANo 90 galimybė nėra labai naudinga, o jei rezultatas yra masyvas ar išvestinis duomuo - tai leidžia gerokai sutrumpinti programos tekstą.

Nauja prasmė operatoriui suteikiama ar naujas operatorius įvedamas FORTRANo operatorium INTERFACE OPERATOR. Tai - alternatyvi FUNCTION iškvietimo sintaksės forma. INTERFACE ASSIGNMENT suteikia papildomą prasmę prieskyros operatoriui. Tai - alternatyvi SUBROUTINE iškvietos sintaksės forma. Abiem atvejais po operatoriaus INTERFACE reikalinga funkcijos arba paprogramio sąsaja. Naujai įvesto ar pakeisto veiksmų operatoriaus vienareikšmiškumo sritis - kaip ir bet kurio subprogramos vardo (žr. 8.2.5 skyrių).

8.3.1. STANDARTINIŲ VEIKSMŲ OPERATORIŲ PERKROVIMAS

Perkrauti standartinį veiksmų operatorių gali tik funkcija. Vienviečiam operatoriui perkrauti reikia funkcijos su vienu argumentu (galima - ir dviem argumentais). Dviviečiam operatoriui reikia funkcijos su dviem įėjimo argumentais, būtinai su atributu INTENT(IN), ir vienu išėjimo argumentu - pertvarkytam reiškiniui žymėti. Niekaip negalima vienviečio operatoriaus paversti dviviečiu arba atvirkščiai.

Kaip perkraunamas operatorius, tegu paaiškina programos conversion pavyzdys: ji vienviečiam operatoriui + suteikia naują prasmę - simbolių seką transformuoja į didžiųjų simbolių (upper case) seką, pavyzdžiui, a -> A. Analogiškai, programa operatoriui "-" suteikia galią simbolinę seką transformuoti į mažuosius simbolius (lower case): A -> a. Ši ir kitos šio skyriaus programos, palyginti su ankstesniais nagrinėtais pavyzdžiais, bus sudėtingos.

Pirmiausia parašysim visą darbą atliekančias funkcijas. Tam reikia žinoti kai kuriuos dalykus apie kompiuterio naudojamus simbolius (asmeniniam kompiuteriui tai bus visad ASCII simboliai) ir kai kurias standartines funkcijas darbui su simboliais. Taigi ASCII simbolių sekos tvarka tokia: iš pradžių didžiosios raidės A<B<C<...<Y<Z, po to mažosios: Z<a<b<...<z; o kitų simbolių eilės tvarka mūsų šiuo atveju nedomina. Standartinės funkcijos: LEN(string) grąžina sveiką rezultatą - simbolių grandinės string simbolių kiekį; ICHAR(c) grąžina sveiką rezultatą -

simbolio c pozicijos numerį ASCII kodų sekoje; CHAR(i) grąžina simbolinį rezultatą - simbolį, kuris yra ASCII kodų sekos i-joje pozicijoje.

Išorinės funkcijos:

```
FUNCTION  upper_case( old )   RESULT( new )
  !  pradinė simbolių seka   old   transformuojama į didžiųjų simbolių seką   new
CHARACTER( LEN = * )                              ::  new
CHARACTER( LEN = * ), INTENT( IN )                ::  old
INTEGER                                           ::  ismall, ibig, difference, i
  ismall = ICHAR( 'a' )
  ibig     = ICHAR( 'A' )
  difference = ibig - ismall   !  tai - tų pat didžiųjų ir mažųjų simbolių pozicijų
                               !  skirtumas kodų sekoje; tai - neigiamas dydis
  new = old
  DO i = 1, LEN( new )     !  ciklas per visus sekos simbolius
    IF( new(i:i) >= 'a' .AND. new(i:i) <= 'z' )  THEN      !  taigi simbolis mažasis;
      new(i:i) = CHAR( ICHAR( new(i:i) ) + difference )  !  simbolių sekoj
                                                          !  grįžtame į didžiųjų
                                                          !  simbolių tarpą
    END  IF
  END  DO
END  FUNCTION  upper_case
FUNCTION  lower_case( old )   RESULT( new )
  !  pradinė simbolių seka   old   transformuojama į mažųjų simbolių seką   new
CHARACTER( LEN = * )                              ::  new
CHARACTER( LEN = * ), INTENT( IN )                ::  old
INTEGER                                           ::  ismall, ibig, difference, i
  ismall = ICHAR( 'a' )
  ibig     = ICHAR( 'A' )
  difference = ibig - ismall
  new = old
  DO i = 1, LEN( new )
    IF( new(i:i) >= 'A' .AND. new(i:i) <= 'Z' )  THEN    !  taigi simbolis didysis;
      new(i:i) = CHAR( ICHAR( new(i:i) ) - difference )  !  simbolių sekoj judame
                                                          !  pirmyn į mažųjų
                                                          !  simbolių tarpą
    END  IF
  END  DO
```

```
END FUNCTION lower_case
```

Programoj dabar būtina tiksli sąsaja, kartu žodžiu OPERATOR nurodanti perkraunamą veiksmų operatorių. Programos apdorojamos simbolių sekos ilgis tegu būna iki 20-ies simbolių:

```
PROGRAM conversion
IMPLICIT NONE

INTERFACE OPERATOR( + )
  FUNCTION upper_case( old )  RESULT( new )
    CHARACTER( LEN = * )                    :: new
    CHARACTER( LEN = * ), INTENT( IN ) :: old
  END FUNCTION upper_case
END INTERFACE

INTERFACE OPERATOR( - )
  FUNCTION lower_case( old )  RESULT( new )
    CHARACTER( LEN = * )                    :: new
    CHARACTER( LEN = * ), INTENT( IN ) :: old
  END FUNCTION lower_case
END INTERFACE

CHARACTER*20 :: string = 'This is a MIXED case'
PRINT *, ""                          ! išvedama tuščia eilutė
PRINT *, string                              ! bus išvesta:  This is a MIXED case
PRINT *, + string                    ! bus išvesta:  THIS IS A MIXED CASE
PRINT *, - string                    ! bus išvesta:  this is a mixed case

END PROGRAM conversion
```

8.3.2. NAUJŲ VEIKSMŲ OPERATORIŲ KŪRIMAS

Kaip ir pakeičiant standartinius veiksmų operatorius, taip ir šiuo atveju būtina parašyti funkciją, kuriai keliami tokie patys kaip anksčiau reikalavimai. Naujas operatorius rašomas būtinai tokios formos: *.vardas.* ; čia *vardas* negali sutapti nė su vienos standartinės funkcijos vardu bei su loginėmis konstantomis .TRUE. ir .FALSE. .

Viskas apie naujų operatorių kūrimą taps aišku, kai ankstesnį pavyzdį perrašysime apibrėždami naujus operatorius, kurių vardus parinksime .to_upper. ir to_lower. :

```
PROGRAM conversion
IMPLICIT NONE

INTERFACE OPERATOR( .to_upper. )
  FUNCTION upper_case( old )  RESULT( new )
    CHARACTER( LEN = * )                    :: new
    CHARACTER( LEN = * ), INTENT( IN ) :: old
  END FUNCTION upper_case
END INTERFACE
INTERFACE OPERATOR( .to_lower. )
  FUNCTION lower_case( old )  RESULT( new )
    CHARACTER( LEN = * )                    :: new
    CHARACTER( LEN = * ), INTENT( IN ) :: old
  END FUNCTION lower_case
END INTERFACE

CHARACTER*20 :: string = 'This is a MIXED case'
PRINT *, ""                          ! išvedama tuščia eilutė
PRINT *, string                            ! bus išvesta:  This is a MIXED case
PRINT *, .to_upper. string       ! bus išvesta:  THIS IS A MIXED CASE
PRINT *, .to_lower. string       ! bus išvesta:  this is a mixed case

END PROGRAM conversion
```

8.3.3. PRIESKYROS OPERATORIAUS PERKROVIMAS

Tai įmanoma atlikti tik su SUBROUTINE: joje rašomas specialus operatorius INTERFACE ASSIGNMENT paprogramio atliekamus veiksmus susieja su naujuoju prieskyros operatorium. Šis prieskyros operatoriaus perkrovimas turi griežtą sintaksę. Paprogrami privalo turėti du būtinuosius argumentus, kurių pirmasis dar turi turėti atributo INTENT reikšmę OUT, o antrasis - IN. Pirmas argumentas atitinka prieskyros operatoriaus kairiąją pusę, o antrasis - dešiniąją.

Prieskyros operatoriaus naujai suteiktos savybės pasireikš kaskart, kai abi jo pusės tiksliai atitiks paprogramėje nustatytą tipą ir rangą; antraip bus atliekama įprastinė prieskyra. Dar vienas labai griežtas apribojimas - abu operatoriaus argumentai negali būti kartu aritmetinio, loginio arba tekstinio tipo (jei argumentai būtų vienodo tipo - būtų atlikta įprastinė prieskyra).

Pavyzdys 1. Programa comma praplečia prieskyros operatorių taip, kad būtų įrašomi kableliai kas tris skaičiaus skaitmenis. Pavyzdžiui, taip apdorojus skaičių 1234567 turėtume gauti 1,234,567 . Sakykim, kad maksimalus skaičiaus ilgis yra apribotas 10 skaitmenų.

Paprogramis insert atliks visus veiksmus:

```
SUBROUTINE  insert( alpha, numeric )          ! argumentai privalo būti du ir
                                              ! būtinai skirtingų tipų
CHARACTER*13,  INTENT( OUT )    ::   alpha    ! 13: rezultatas turės iki 3 kablelių
INTEGER*4,  INTENT( IN )                      :: numeric
INTEGER                                       :: extra, length, alpha_pos, text_pos
CHARACTER*10                                  :: text
WRITE( text, '(i10)' )  numeric     ! pradinis skaičius  numeric  pervedamas į
                                    ! tekstinę formą - tekstinį kintamąjį  text
text = ADJUST( text )               ! ši standartinė funkcija tekstą patraukia į kairę
text
                                    ! skirtų pozicijų lauko pusę
length = LEN_TRIM( text )           ! ši standartinė funkcija, skaičiuodama teksto
                                    ! ilgį, nepriskaičiuoja intervalų
extra = MOD( length, 3 )    ! skaičiuojamas sveikas pozicijų likutis nuo trejetukų:
                            ! length-INT( length/3 )*3
IF( extra == 0 )  extra = 3 ! jei reikšmė yra 0, tai skaitmenų kiekis yra 3-jų
                            ! daugiklis, ir kablelį reikia įterpti po pirmų 3-jų
                            ! skaitmenų
alpha( 1:extra ) = text( 1:extra )    ! perkeliame skaičiaus dalį į rezultatą
alpha_pos = 1+extra
text_pos = 1+extra
DO              ! kablelio įterpimas ir skaičiaus eilinės dalies perkėlimas - cikle,
IF( text_pos > length )  EXIT               ! iš kurio išeisim pasibaigus tekstui
  alpha( alpha_pos : alpha_pos ) = ","
  alpha( alpha_pos+1 : alpha_pos+3 ) = text( text_pos : text_pos+2 )
  alpha_pos = alpha_pos+4
  text_pos = text_pos+3
END DO
END SUBROUTINE  insert
```

Programa:

```
PROGRAM  comma
IMPLICIT  NONE

INTERFACE  ASSIGNMENT( = )
  SUBROUTINE  insert( alpha, numeric )
    CHARACTER*13,  INTENT( OUT )          :: alpha
    INTEGER*4,  INTENT( IN )              :: numeric
  END  SUBROUTINE  insert
END  INTERFACE

INTEGER*4          :: number
CHARACTER*13       :: string
  PRINT*, 'enter integer number"
  READ*, number
  PRINT*, 'initial number', number
     string = number              ! čia slepiasi paprogramio  insert  iškvietimas
  PRINT*,  'modified number',  string

END  PROGRAM  comma
```

Pavyzdys 2. Perkrausime prieskyros operatorių taip, kad jis įgytų naujas galimybes operacijoms su struktūromis. Tarkim, kad turime išvestinį duomenį, studento pavardei ir vienam pažymiui:

```
TYPE  student
  CHARACTER( 20 ) name
  REAL  mark
END TYPE  student
```

Tarkim, kad norime supaprastinti prieskyros operatoriaus sintaksę taip, kad būtų galima tiesiogiai, nesikreipiant į struktūros komponentą, priskirti name reikšmę:

```
student1 = "Jonas" ,
```

o mark komponentui dabar jokia reikšmė nebūtų suteikiama. Antra, norime, kad būtų galima vėlgi tiesiogiai gauti name komponento reikšmę iš kintamojo student:

```
student_name = student1 ,
```

kur student_name būtų tekstinio tipo kintamasis.

Turim parašyti paprogramius student_from_name - pirmajam atvejui, ir name_from_student - antrajam. Pirmas paprogramis turės du formalius argumentus: vieną student tipo, o kitą tekstinio. Antrasis atitinkamai - tekstinio ir student tipo. Patogiausia būtų šiuos paprogramius patalpinti modulyje:

```
MODULE student_module

  TYPE student
    CHARACTER( 20 ) name
    REAL mark
  END TYPE student

INTERFACE ASSIGNMENT( = )
  MODULE PROCEDURE name_from_student, student_from_name
END INTERFACE

CONTAINS

  SUBROUTINE name_from_student( string, student_n )
    CHARACTER( * ), INTENT = OUT        ::  string
    TYPE( student ), INTENT = IN        ::  student_n
      string = student_n%name
  END SUBROUTINE
  SUBROUTINE student_from_name( student_n, string )
    CHARACTER( * ), INTENT = IN         ::  string
    TYPE( student ), INTENT = OUT       ::  student_n
      student_n%name = string
  END SUBROUTINE
END MODULE
```

Pagrindinės programos, kuri naudotų parašytą modulį, pavyzdys:

```
USE student_module
TYPE( student )       ::  student1 = student( “Petras”, 8. )
CHARACTER( 20 )  ::  student_name
  student1 = “Jonas”          !  struktūros pirmojo komponento reikšmė dabar
                              !  suteikiama tokiu būdu
  student_name = student1 !  atvirkščias veiksmas: reikšmė iš struktūros
                              !  paimama tokiu būdu
END
```

8.4. MASYVAI

FORTRANas 90, lyginant su ankstesniu standartu, įveda tris dideles naujoves masyvams:

- masyvams gali būti taikomos beveik visos anksčiau tik pavieniams duomenims skirtos operacijos. Pavyzdžiui, dviejų masyvų a ir b suma s dabar gaunama paprasčiau-siai taip: s =a+b;
- funkcijos rezultatas gali būti ne tik paprastas kintamasis, bet ir masyvas. Tuo būdu praktiškai pranyksta ribos tarp FUNCTION ir SUBROUTINE; kurią subprogramos formą pasirinkti - dabar yra labiau programuotojo skonio reikalas;
- jei ankstesnis FORTRANas leido kreiptis tik į atskirus masyvo elementus arba iškart į visą masyvą (pavyzdžiui, įvesties/išvesties operatoriuose), tai dabar siūlomos plačios galimybės operuoti įvairiomis masyvo dalimis;
- ir viena maža naujovė - atsirado vadinamieji nuliniai masyvai, neturintys nė vieno elemento. Tokie masyvai prasmingi kai kuriuose rekurentiniuose skaičiavimuose. Jie yra suderinami su skaliaru, t.y. gali dalyvauti reiškiniuose kartu su skaliariniais dydžiais, tačiau faktiškai reiškinių prasmės nekeičia, nes netalpina jokių reikšmių.

8.4.1. BENDROSIOS ŽINIOS

Masyvų aprašymas. Prieš nagrinėdami masyvų aprašymo operatorius, įvesime kelias naujas masyvus charakterizuojančias sąvokas:

- masyvo elementų kiekis vienoje kurioje dimensijoje yra *ekstentas*
- masyvo *forma* yra masyvo ekstentų sąrašas
- masyvo *dydis* yra visų masyvo elementų kiekis
- masyvo *rangas* yra jo dimensijų kiekis. Skaliaro rangas - 0

Paprasčiausia masyvo deklaravimo forma yra:

tipas [, atributų sąrašas] :: masyvų vardų sąrašas

Iš visų galimų atributų dabar paminėsim tik masyvo matmenis nurodantį DIMENSION(ribos), kuriame nurodoma tiek ribų, koks yra masyvo rangas. Maksimalus leistinas masyvų rangas yra 7. Ribos nurodomos taip, kaip ir ankstesniame FORTRANe: arba maksimalia ribos reikšme - tada elementai dimensijoje numeruojami nuo pirmojo, arba dimensijos apatine riba : viršutine riba. DIMENSION gali būti ir savarankiškas operatorius.

Pavyzdžiai:

```
REAL, DIMENSION (2, 3)  ::  array          !  dvimatis masyvas  array:
                                           !  rangas - 2,dydis - 6, forma - 2, 3
INTEGER, DIMENSION (-2:2)  ::  int_array       !  rangas - 1, dydis - 5
INTEGER, DIMENSION (10)  ::  x, y(20), z       !  masyvų  x ir z  dydis - 10,
                                           !  masyvo  y  - 20
```

FORTRANe 90 yra standartinės funkcijos, nustatančios visas masyvo savybes. Kaip jos veikia, parodo šis pavyzdys:

```
REAL, DIMENSION (-3:4, -5:6)          ::  array
INTEGER, DIMENSION (2)     ::  array_lower
INTEGER, DIMENSION (2)     ::  array_shape
INTEGER                    ::  array_size
INTEGER, DIMENSION (2)     ::  array_upper
  array_shape = SHAPE( array )              !  masyvo formą nustatančios funkcijos
                                     !  rezultatas  -  8, 12
  array_size = SIZE( array )               !  masyvo  dydis  -  96
  array_lower = LBOUND( array )     !  dimensijų apatinės ribos  -  -3, -5
  array_upper = UBOUND( array )     !  dimensijų viršutinės ribos  -  4, 6
```

Šių funkcijų darbo rezultatais patogu naudotis, pavyzdžiui, programos vykdymo metu derinant FORMATo specifikacijų sąrašus ar skaičiuojant reikalingą kompiuterio atminties kiekį masyvui.

Masyvų inicializavimas. Masyvų aprašymo operatoriumi dabar galima kartu suteikti ir pradines reikšmes masyvo elementams; t.y. atlikti veiksmus, anksčiau įmanomus tik su operatoriais DATA ir PARAMETER. Be to, panašiais į mums pažįstamus neišreikštuosius ciklus *masyvų konstruktoriais* galime suteikti masyvų elementams įvairias reguliarias reikšmes. Šie visi dalykai bus aiškūs iš pavyzdžių:

```
REAL, DIMENSION (10)  ::  array = 0.0   !  visi 10 elementų įgis reikšmes  0.       C
šie trys operatoriai masyvo array1 elementams suteikia reikšmes  2., 4., 6. :
REAL, DIMENSION (3)   ::  array1 = ( / ( j, j=2, 6, 2 ) / )
REAL, DIMENSION (3)   ::  array1 = ( / ( 2*j, j=1,3 ) / )
REAL, DIMENSION (3)   ::  array1 = ( / 2., 4., 6. / )
```

Taigi masyvų konstruktoriuose konstantų sąrašas apimamas skliaustais (/ ... /), o šių skliaustų viduje gali būti neišreikštieji ciklai. Masyvų konstruktoriais formuojami tik

vienmačiai masyvai. Jei būtina konstruktorius pasitelkti keliamačio masyvo formavimui, tai dar teks panaudoti standartinę funkciją RESHAPE (žr. žemiau).

Masyvo dalių išskyrimas. Kaip tą padaryti, paaiškina pavyzdys:

```
INTEGER, DIMENSION (5,5)  :: array
INTEGER, DIMENSION (3,3)  :: array_interior
INTEGER, DIMENSION (5   )  :: array_left_column
INTEGER, DIMENSION (   5)  :: array_top_row
INTEGER, DIMENSION (   5)  :: array_diagonal
INTEGER, DIMENSION (25)   :: linear
  array_left_column = array( :, 1 )            ! išskiriamas pirmasis stulpelis ir
                                    ! talpinamas į masyvą  array_left_column
                                    ! ":" apima visą dimensijos diapazoną.
  array_top_row = array( 1, : )     ! pirmoji eilutė - į   array_top_row
  array_interior = array( 2:4, 2:4 )           ! masyvo dalis: 2, 3, 4 eilučių 2, 3, 4
                                    ! elementai  -  į   array_interior
  linear = PACK( array, .TRUE. )           ! funkcija PACK "nufiltruoja" į  linear
                                    ! visus per filtrą praeinančius elementus.
                                    ! Filtras .TRUE. praleidžia juos visus.
                                    ! Smulkiau apie  PACK  -  žemiau.
  diagonal = linear( 1:25:6 )       ! Iš  linear  išrenkami elementai  1, 7, 13,
                                    ! 19, 25  -  taigi,  array  įstrižainė. Toks
                                    ! trigubas indeksas tolygus neišreikštam
                                    ! ciklui  ( i, i= 1, 25, 6).
```

Jei minėti pertvarkymai leidžia "paimti" tik reguliarias masyvo dalis, tai *vektoriniai indeksai* - bet kokias.

1 pavyzdys.

```
INTEGER, DIMENSION (10)  ::  a = ( / (j, j=2,20,2) / )   ! a  elementai -
                                                         ! 2, 4, 6, ... , 20
INTEGER, DIMENSION (5)   ::  b = ( / 5, 4, 3, 1, 2 / )   ! masyvas b  bus
                                                         ! vektorinis indeksas
PRINT*, a( b )      !  vektorinis indeksas nurodo spausdinti iš eilės elementus
                    !  5, 4, ... , 2. Taigi spausdinimo rezultatas - 10, 8, 6, 2, 4.
```

2 pavyzdys. Standartinės funkcijos RESHAPE, leidžiančios iš vienmačio masyvo sudaryti dvimatį, panaudojimas. Po to gautą dvimatį masyvą pertvarkysime vektoriniais indeksais:

```
INTEGER, DIMENSION (3, 3)  :: a
INTEGER, DIMENSION (3    ) :: rows = (/ 3, 2, 1 /)            ! vektorinis indeksas
INTEGER, DIMENSION(    3)  :: columns = (/ 3, 2, 1 /)         ! kitas vektorinis
                                                              ! indeksas
C       Funkcijos argumentas SOURCE paskiria vienmatį masyvą, o argumentas
C       SHAPE  rodo, kad dvimatis masyvas turės 3 eilutes ir 3 stulpelius; elementų
C       išdėstymo tvarka - tokia kaip kompiuterio atmintyje - stulpeliais:

a = RESHAPE(  SOURCE = (/ 1, 4, 7, 2, 5, 8, 3, 6, 9 /), SHAPE = (/ 3, 3 /)  )
        ! Masyvo  a  elementai dabar -  1  2  3
        !                               4  5  6
        !                               7  8  9
a = a( rows, columns )
        ! Eilučių tvarka dabar: 3, 2, 1;  stulpelių - tokia pat. Masyvo  a
        ! pertvarkymo vektoriniais indeksais rezultatas -     9  8  7
        !                                                     6  5  4
        !                                                     3  2  1
```

Vektoriniame indekse masyvo elementų numeriai išdėstomi bet kokia tvarka, gali būti ir kartojami.

Elementų tvarkai masyvuose pakeisti siūlomos ir kitos sudėtingos standartinės funkcijos: CSHIFT, EOSHIFT "perstumia" elementus kiekvienoj dimensijoj per nurodytą elementų kiekį, SPREAD padidina vienetu masyvo rangą ir t.t.

8.4.2. MASYVAI - SUBPROGRAMŲ ARGUMENTAI

Masyvą subprogramai (arba iš jos - kviečiančiajai programai) galima perduoti keliais būdais. Matyt, pats patogiausias būtų toks:

Pavyzdys:

```
IMPLICIT  NONE
INTEGER                       :: n       ! masyvo  x  tikrasis elementų kiekis
REAL, DIMENSION (100)  :: x              ! pradinis masyvas, perduodamas
                                         ! subprogramai
REAL                          :: y, z    ! subprogramos darbo rezultatai
```

```
      ...
CALL  subr( x, y, z, n )
      ...
CONTAINS
  SUBROUTINE  subr( a, b, c, k )
    REAL, DIMENSION( : ), INTENT( IN )  :: a   ! subprogramoje aprašant
                                               ! masyvą, tereikia jo rango
    REAL, INTENT( INOUT )               :: b, c
    INTEGER, INTENT( IN )               :: k   ! masyvo elementų kiekis,
                                               ! matyt, bus reikalingas
                                               ! apdorojant masyvą
      ...
  END  SUBROUTINE
END
```

Apibrėžimo operatorius

```
REAL, DIMENSION( : ), INTENT( IN )  :: a
```

masyvą a apibrėžia kaip *numanomos formos masyvą* (assumed-shape array). Pilna masyvo dimensijos sintaksė yra

[*apatinė riba*] :

Jei apatinė riba praleista, pagal nutylėjimą ji laikoma esant 1. Į šį dalyką reikia kreipti dėmesį, nes ankstesniame pavyzdyje subprogramoje aprašius masyvą

```
REAL, DIMENSION( 0 : ), INTENT( IN )  :: a
```

sutapdintume tikrąjį masyvą x ir formalųjį a taip: x(i) = a(i-1), i=1, 2, ...
Dar grįžkime prie pavyzdžio. Jei paprogramis subr būtų kompiliuojamas atskirai nuo pagrindinės programos - būtų išorinis paprogramis, tai jos sąsaja turėtų būti tiksli:

```
INTERFACE
  SUBROUTINE  subr( a, b, c, k )
    REAL, DIMENSION( : ), INTENT( IN )  :: a
    REAL, INTENT( INOUT )               :: b, c
    INTEGER, INTENT( IN )               :: k
  END  SUBROUTINE
END  INTERFACE
```

8.4.3. DINAMINIAI MASYVAI

Visi kintamieji, apie kuriuos iki šiol kalbėjom, buvo *statiniai*. Statiniam kintamajam kompiliatorius iškart skiria tam tikrą kompiuterio atminties adresą vien tik radęs šio kintamojo deklaravimą. Tai gali būti nepatogu, pavyzdžiui, tais atvejais, kai kelis kartus kviečiama programa, apdorojanti didelį masyvą, kurio apimtis kiekvieną sykį yra skirtinga. Šiam atvejui FORTRANas 90 siūlo *dinaminius* kintamuosius, kuriems atmintis skiriama tik programos vykdymo metu.

Kintamasis laikomas dinaminiu, jei jam paskirtas atributas ALLOCATABLE. Pavyzdžiui, vienmatis dinaminis masyvas x gali būti apibrėžtas taip:

```
REAL, DIMENSION( : ), ALLOCATABLE :: x
```

Nors masyvo rangas ir nurodytas, masyvui neskiriama jokia atmintis tol, kol jo dydis bus apibrėžtas operatoriuje ALLOCATE. Beje, masyvo formos, jei masyvui paskirtas atributas ALLOCATABLE, nurodyti ir negalima. Jei masyvui x reiktų skirti n ląstelių atminties, tai rašytume

```
ALLOCATE( x( n ) )
```

Kai masyvas nebereikalingas - jam skirta atmintis išlaisvinama (kartu prarandant visus šioje atminties srityje turėtus duomenis):

```
DEALLOCATE( x )
```

Pilnesnė operatoriaus ALLOCATE forma yra:

ALLOCATE(*objektų sąrašas* [, STAT = *sveikas skaliarinis kintamasis*])

Čia rašome “objektų”, nes dinaminė atmintis gali būti skiriama ne tik kintamiesiems, bet ir nuorodoms (žr. kitą skyrių). Objekto pavidalas - *objektas*[(*masyvo ribų sąrašas*)]; ribų sąrašo pavidalas - [*apatinė riba*] : *viršutinė riba*. Jei operatoriuj yra STAT = *kintamasis*, tai *kintamajam* bus priskirta reikšmė 0 - sėkmingai išskyrus atmintį objektui, ir teigiama reikšmė - pritrūkus atminties. Jei parametro STAT nėra, o atminties išskirti nepavyko - programos darbas nutraukiamas.

Panaši ir operatoriaus DEALLOCATE forma. Šiame operatoriuje parametras STAT taip pat prasmingas, nes gali nepavykti išlaisvinti atminties, skirtos nuorodoms.
Formalių subprogramų argumentų apibrėžti dinaminiais subprogramos viduje negalima. Tokiems argumentams-masyvams subprogramoje derėtų tik nurodyti rangą, o apibrėžti juos dinaminiais - kviečiančiojoje programoje (žr. pavyzdį 8.4.6 skyriuje).

8.4.4. MASYVAI REIŠKINIUOSE

FORTRANo aritmetiniai operatoriai gali operuoti ne tik skaliarais, bet ir masyvais. Vienvietis operatorius veikia kiekvieną masyvo elementą: pavyzdžiui, -x pakeičia kiekvieno masyvo x elemento ženklą. Kiek sudėtingiau su dvivietėmis operacijomis, kuriose dalyvauja pora masyvų: jie privalo būti suderinti; t.y. jų rangai ir ekstentai turi būti vienodi. Dvivietė operacija veikia porą atitinkamų abiejų masyvų elementų, o operacijos rezultatas yra pradinės masyvų formos masyvas. Jei vienas operacijos operandų yra skaliaras, tai prieš atliekant operaciją jis "išplečiamas" iki kito operando - masyvo - formos.

Masyvas gali būti standartinės funkcijos argumentas. Standartinė funkcija veikia kiekvieną masyvo elementą, ir todėl jos rezultatas - tos pat formos masyvas.

Pavyzdžiai:

```
REAL, DIMENSION( 10 )                  ::  x, y, z, t(4)
REAL, DIMENSION( 5,5 )          ::  a, c
LOGICAL, DIMENSION( 10 )        ::  k
      ...
x = 1.                  ! x_i = 1          , i=1,2, ... ,10
x = 1./x                ! z_i = x_i * 3    , i=1,2, ... ,10
z = x + y               ! z_i = x_i + y_i  , i=1,2, ... ,10
z = x * y               ! z_i = x_i + y_i  , i=1,2, ... ,10
z = x * 3.              ! z_i = x_i + y_i  , i=1,2, ... ,10
z = SQRT( x )           ! z_i = x_i + y_i          , i=1,2, ... ,10
z = x * SQRT( y )             ! z_i = x_i + y_i    , i=1,2, ... ,10
k = x == y                    ! k_i elementas bus .TRUE. jei x_i = y_i ,
                        ! ir   .FALSE. jei x_i /= y_i      ; i=1,2, ... ,10
t = x( 2:5 ) + y( 4:7 ) ! t_1 = x_2 + y_4 , t_2 = x_3 + y_5 , ... ,
                        ! t_4 = x_5 + y_7
a( i,1:5 ) = c( j, 1:5 )  ! a i-oji eilutė bus c j-osios eilutės kopija
```

Taigi standartinės funkcijos gali operuoti masyvais. Jei norime, kad masyvu operuotų mūsų rašoma funkcija, tai jai teks parašyti tikslią sąsają.

Operatorius WHERE leidžia operuoti ne visais masyvo elementais, o tik tais, kurie atitinka norimą sąlygą. Bendrasis šios konstrukcijos pavidalas yra

WHERE(*loginis reiškinys-masyvas*)
  prieskyros operatorius su reiškiniu-masyvu

[ELSEWHERE]
  [*prieskyros operatorius su reiškiniu-masyvu*]
END WHERE ,

o pati paprasčiausia jos forma -

WHERE(*loginis reiškinys-masyvas*) *prieskyros operatorius su reiškiniu-masyvu*

Ši konstrukcija labai panaši į konstrukciją IF-THEN-ELSE, todėl toliau jos veikimą aiškinti nereikia. *Loginis reiškinys-masyvas* gali būti tiesiog loginis masyvas. Toks loginis masyvas dėl jo prasmės kartais dar vadinamas *masyvu-trafaretu* (array-mask).
WHERE neveiks, jei *prieskyros operatorius su reiškiniu-masyvu* yra neelementinė funkcija, t.y. jos rezultatas - ne masyvas.

Pavyzdys:

```
WHERE( a > 0. )
  a = SQRT( a )    ! šaknies funkcija taikoma tik teigiamiems masyvo  a
                   ! elementams
ELSEWHERE
  a = 0.           ! visi neteigiami elementai prilyginami nuliui
END WHERE
```

8.4.5. KAI KURIOS MASYVAMS SKIRTOS STANDARTINĖS FUNKCIJOS

ALL(xl) grąžina kviečiančiajai programai reikšmę .TRUE., jei visi loginio masyvo (masyvo-trafareto) xl elementai yra .TRUE. .
ANY(xl) grąžina kviečiančiajai programai reikšmę .TRUE., jei nė vienas masyvo-trafareto xl elementas nėra .TRUE. .
SUM(x) reikšmė yra skaliaras - visų masyvo x elementų suma. SUM - bendrinis vardas, todėl x gali būti sveikasis, realusis ar kompleksinis masyvas.
PRODUCT(x) reikšmė yra skaliaras - visų masyvo x elementų sandauga; x gali būti sveikasis, realusis arba kompleksinis.

Pavyzdžiai:

```
IF( ANY( a>0 ) )  a = 1      ! jei joks  a  elementas nėra didesnis už nulį, tai
                             ! visus juos prilyginti vienetui
IF( ALL( a==0 ) )  a = -1          ! jei visi  a  elementai nuliniai, tai juos prilyginti -1
z = SUM( x*y )               ! skaliarinė masyvų  x  ir  y  sandauga
```

COUNT(xl) grąžina masyvo-trafareto xl elementų .TRUE. kiekį.
MAXVAL(x) randa didžiausio x elemento reikšmę; x sveikasis arba realusis.
MINVAL(x) analogiška funkcija; mažiausio elemento reikšmė.
ALLOCATED(x) reikšmė .TRUE., jei x šiuo metu yra skirta atmintis.
LBOUND(x [, DIM]) reikšmė, kai DIM nėra - rango 1 masyvas, kurio elementai - apatinės kiekvienos dimensijos ribos. Kai DIM yra - skaliaras; nurodytos dimensijos apatinė riba.
UBOUND(x [, DIM]) analogiška funkcija viršutinėms riboms nustatyti.
SHAPE(x) reikšmė - rango 1 masyvas, kurio elementai - x forma. Jei x skaliaras, rezultatas yra 0.
SIZE(x [, DIM]) reikšmė, kai DIM nėra - skaliaras, x dydis. Kai DIM yra - skaliaras, nurodytos dimensijos ekstentas.
MAXLOC(x [, xl]) reikšmė, kai masyvo-trafareto xl nėra - rango 1 masyvas, kurio elementai yra maksimalaus x elemento indeksai. Kai yra masyvas-trafaretas, funkcija taikoma tik trafaretu atrenkamiems x elementams.
MINLOC(x [, xl]) analogiška mažiausio elemento indeksų funkcija.
PACK(x, xl) reikšmė - rango 1 masyvas, sudarytas iš x masyvo elementų, atitinkančių trafareto xl .TRUE. elementus. Jei visi trafareto elementai yra .TRUE., į vienmatį rezultato masyvą bus praleisti visi x elementai.
RESHAPE([SOURCE=] x, [SHAPE=] y) pertvarko vienmatį masyvą, nustatomą argumentu SOURCE, į keliamatį pagal argumentu SHAPE sveikosiomis konstantomis nustatomus matmenis. Tai - ne bendriausia funkcijos forma; galima nurodyti dar du argumentus.
MERGE(a, b, xl) grąžina tokios pat kaip a ir b formos masyvą. Jo elementai bus: a(i,j), jei trafareto xl atitinkamas elementas yra .TRUE., ir b(i,j) atvirkščiu atveju. Pavyzdžiui, c = MERGE(a, b, a>b) atrenka į masyvą c didesnius elementus iš a ir b masyvų.

Dabar - matyt, dažniausiai naudojamos funkcijos, atliekančios matricų aritmetikos veiksmus:
TRANSPOSE(x) - rango 2 masyvo x transponavimas.
MATMUL(x, y) - suderinamų rango 2 masyvų x ir y matricinė sandauga. Jei argumentai aritmetiniai, tai funkcijos rezultatas gaunamas tokių matmenų:
kai x dimensijos yra n×m, o y m×k - rezultato-masyvo dimensijos bus n×k;
kai x dimensija yra m, o y m×k - rezultato-masyvo dimensija bus k;
kai x dimensijos yra n×m, o y m - rezultato-masyvo dimensija bus n. Kai x ir y yra loginiai: nesunku pastebėti, kad masyvo-rezultato (i,j)-asis elementas gali būti

išreikštas ir taip: SUM(x(i,:) * y(:,j)). MATMUL rezultatas bus gautas, SUM pakeitus standartine funkcija ANY, o veiksmų operatorių * - loginiu operatorium .AND. . DOT_PRODUCT(x, y) rango 1 vienodo dydžio masyvų x ir y skaliarinė sandauga. Jei argumentai yra aritmetiniai, funkcijos rezultato matematinė prasmė aiški. Jei argumentai loginiai, tai rezultatui suteikiama tokia prasmė: ANY(x.AND.y).

8.4.6. PROGRAMŲ PAVYZDŽIAI

Dažniausiai atliekamas matricų aritmetikos veiksmas yra matricų daugyba. Tokius veiksmus reikėtų mokėti programuoti. Todėl nors ir yra matricų daugybos standartinė funkcija MATMUL, pabandysim parašyti savo programą šiems veiksmams. Toki, tik mažiau bendrą paprogramį jau esame parašę 7.4 skyriuje. Ten paaiškintas ir uždavinio algoritmas.

```
SUBROUTINE  multiplication( a, b, c )
C       Daugina matricas a(n×m) ir b(m×k), rezultato matrica c(n×k)
REAL, DIMENSION( :,: )              ::   a, b, c
INTEGER                             ::   n, m, mb, k, i, j
  n = SIZE( a, 1 )
  m = SIZE( a, 2 )
  k = SIZE( b, 2 )
    mb = SIZE( b, 1 )
    IF( m /= mb )  THEN          !   ar įmanoma matricų daugyba ?
      PRINT *, 'a ir b daugyba negalima'
      RETURN
    END  IF
  DO i = 1, n
    DO j = 1, k
      c(i,j) = SUM( a(i,1:m) * b(1:m,j) )
    END  DO
  END  DO
END  SUBROUTINE  multiplication
```

Kviečiančioji programa, kuri sudaugintų dvi bet kokio dydžio matricas:

```
IMPLICIT  NONE
  INTERFACE
      SUBROUTINE  multiplication( a, b, c )
      REAL, DIMENSION( :,: )              ::   a, b, c
      INTEGER                             ::   n, m, mb, k, i, j
```

```
        END  SUBROUTINE  multiplication
  END  INTERFACE
REAL, DIMENSION( :,: ), ALLOCATABLE                 ::   a, b, c
INTEGER                                             ::   i, j, n, m, k, test          C
duomenų įvestis ir atminties paskyrimas matricoms

PRINT *, 'įvedimas:  n, m, k'
READ *, n, m, k
PRINT *, 'kontrolinis spausdinimas', n, m, k
  ALLOCATE( a(n, m), STAT = test )
        IF( test>0 ) THEN
          PRINT *, 'trūksta atminties matricai a: stop'
          STOP
        END IF
  ALLOCATE( b(m, k), STAT = test )
        IF( test>0 ) THEN
          PRINT *, 'trūksta atminties matricai b: stop'
          STOP
        END IF
  ALLOCATE( c(n, k), STAT = test )
        IF( test>0 ) THEN
          PRINT *, trūksta atminties matricai c: stop'
          STOP
        END IF

PRINT *, 'įvedimas:  matrica a'
READ *, ((a(i,j), j=1,m), i=1,n)
PRINT *, 'kontrolinis spausdinimas: a', ((a(i,j), j=1,m), i=1,n)
PRINT *, 'įvedimas:  matrica b'
READ *, ((b(i,j), j=1,k), i=1,m)
PRINT *, 'kontrolinis spausdinimas: b', ((b(i,j), j=1,k), i=1,m)                  C
daugyba ir rezultatų spausdinimas
  CALL  multiplication( a, b, c )
PRINT *, 'rezultatas: c', ((c(i,j), j=1,k), i=1,n)
END
```

Jei programoje tenka dažnai naudoti matricų daugybos veiksmus, patogu tiesiog įvesti naują operatorių matricų daugybos veiksmui žymėti. Tą galima padaryti, kaip

prisimenam, tik su FUNCTION (žr. skyrius 8.3.1 - 8.3.2). Turim perrašyti paprogramį į funkciją, pavyzdžiui, palikdami visus ankstesnius vardus:

```
FUNCTION  multiplication( a, b )  RESULT( c )
REAL, DIMENSION( :,: ), INTENT( IN )                          ::  a, b
REAL, DIMENSION( SIZE( a,1 ), SIZE( b,2) ), INTENT( OUT )     ::  c
        ...
END  FUNCTION  multiplication
```

Programoje būtų parašytas INTERFACE OPERATOR blokas, priskiriantis matricų daugybą naujam operatoriui, tarkim, .x. :

```
IMPLICIT  NONE
INTERFACE  OPERATOR( .x. )
  FUNCTION  multiplication( a, b )  RESULT( c )
    REAL, DIMENSION( :,: ), INTENT( IN )                          ::  a, b
    REAL, DIMENSION( SIZE( a,1 ), SIZE( b,2) ), INTENT( OUT ) ::  c
  END  FUNCTION  multiplication
END  INTERFACE
INTEGER  ::  i, j, n, m, k, test                                                C
programos tekstas lygiai toks pat, išskyrus matricų daugybos veiksmą:
        . . .
c = a .x. b
        . . .
END
```

Dar kartą panagrinėkim šį pavyzdį. Būtų dar patogiau, jei naujai įvesto operatoriaus .x. funkcijas atliktų standartinis dvivietis veiksmų operatorius * . Tačiau jam tiesiogiai išplėsti galimybių, kad būtų apimta ir matricų daugyba, negalim, nes jis jau yra apibrėžtas matricų daugybos atvejui c = a*b taip: c(i,j) = a(i,j)*b(i,j). Standartinės operacijos gi pakeisti negalima.
Išeitis tokia: suteikti operatoriui matricų daugybos galimybes išvestiniams duomenims ir visoms matricoms suteikti šių išvestinių duomenų tipą. Apibrėžkim išvestinį tipą ir jį patalpinkim modulyje:

```
MODULE  matrix_type
  TYPE  matrix
    REAL  ::  z
  END  TYPE  matrix
```

```
END MODULE matrix_type
```

Dabar funkcija multiplication būtų:

```
FUNCTION multiplication( a, b ) RESULT( c )
USE matrix_type
TYPE( matrix ), DIMENSION( :,: ), INTENT( IN )                    :: a, b
TYPE( matrix ), DIMENSION( SIZE( a,1 ), SIZE( b,2) ), INTENT(OUT)  :: c
      . . .
  c(i,j)%z = SUM( a(i,1:m)%z * b(1:m,j)%z )
      . . .
END FUNCTION multiplication
```

Programa:

```
USE matrix_type
IMPLICIT NONE
INTERFACE OPERATOR( * )
  FUNCTION multiplication( a, b ) RESULT( c )
    USE matrix_type
    TYPE( matrix ), DIMENSION( :,: ), INTENT( IN )              :: a, b
    TYPE( matrix ), DIMENSION( SIZE( a,1 ), SIZE( b,2) ), &
                                              & INTENT(OUT)  :: c
  END FUNCTION multiplication
END INTERFACE
INTEGER :: i, j, n, m, k, test
      . . .
C         programos tekstas lygiai toks pat, išskyrus matricų daugybos veiksmą:
      . . .
c = a * b
      . . .
END
```

8.5. NUORODOS IR TAIKINIAI

Tokios FORTRANo 90 konstrukcijos atveria kelią dar vienam dinaminės atminties gavimo būdui. *Nuorodos* (pointer) ir *taikiniai* (target) leidžia programos darbo metu keisti duomenims skirtą atminties sritį. Anksčiau FORTRANui buvo būdingos dvi esminės savybės: nuoseklus atminties organizavimas (pavyzdžiui, bet kokio matiškumo masyvas saugomas atminty kaip vienmatis, masyvo elementus išdėstant tam tikra

griežta tvarka), ir griežta priklausomybė tarp kintamojo vardo ir jį atitinkančios atminties srities (tam tikra atminties sritis skiriama kintamajam kompiliacijos metu; programos darbo metu kiekvieną kartą paminėjus kintamąjį, visada kreipiamasi į tą griežtai apibrėžtą sritį). Kintamiesiems ar masyvams 90-asis standartas leidžia sukurti tiesiog dimensijų sąrašus, tačiau neskirti jokios atminties srities. Programos vykdymo metu atmintis dinaminiams masyvams gali būti paskirta bet kurioj kompiuterio atminties vietoj. Panašus efektas pasiekiamas ir antru būdu, nuorodų ir taikinių konstrukcijomis: nuorodos programos darbo metu su kintamuoju gali asocijuoti skirtingas atminties sritis. Taigi ir čia programos naudojama atmintis jau nėra nei nuosekli, nei griežtai fiksuota tam tikriems vardams.

Nuorodos ir taikiniai - sudėtingas programavimo įrankis, o su jais kuriami *saitiniai sąrašai* dar kebliau suvokiami. Kaip teigia J. Kerriganas, tai puikus įrankis visiškai nesuprantamoms programoms kurti. Mes nekelsim sau uždavinio išnagrinėti visas šių FORTRANo struktūrų subtilybes, o apsiribosim daugiau idėjiniu lygiu ir paprastesniais programų pavyzdžiais. Taigi šis skyrius galėtų būti įvadu, o išsamaus POINTER ir TARGET aprašymo reikėtų ieškoti, pavyzdžiui, W.S.Brainerdo ir kt. knygoje.

8.5.1. BENDROSIOS ŽINIOS

Tegu

```
INTEGER, TARGET  :: r = 13   ! r - taikinys, nes yra atributas TARGET
INTEGER, POINTER :: p        ! p - nuoroda, nes yra atributas POINTER
p => r          ! => - nuorodos prieskyros operatorius; juo p nukreipiam į r
r = 2 * p       ! taikinio reikšmė dabar - 26, nes ankstesnis operatorius
                ! nukreipė p į r ir tuo pačiu r reikšmė tapo p reikšme
PRINT *, p, r   ! bus atspausdintos reikšmės 26 ir 26. Kokią reikšmę turi
                ! taikinys, tokia bus ir nuorodos reikšmė
  ...
```

Taigi į nuorodą galima žiūrėti tiesiog kaip į kitą taikinio vardą. Kas atsitinka taikiniui, atsitinka ir nuorodai. Kaip matėm, nuoroda deklaruojama atributu (arba savarankišku operatorium) POINTER, o taikinys analogiškai - TARGET. Kintamojo-nuorodos ir kintamojo-taikinio tipai privalo sutapti. Nuoroda gali “rodyti” ne tik į taikinį, bet ir į kitą nuorodą. Nuorodos nukreipimo, arba prieskyros, operatorius yra =>.

Tegu

```
INTEGER, TARGET  :: r = 13
INTEGER, POINTER :: p1, p2
```

```
p1 => r                ! p1 - nuoroda į r; todėl jos reikšmė yra 13
p2 => p1                     ! p2 - nuoroda į p1; todėl jos reikšmė tokia kaip p1: 13
PRINT *, p1,p2,r       ! bus atspausdintos reikšmės 13, 13, 13
   ...
```

Sunkiau suvokiamas pavyzdys:

```
INTEGER, TARGET   ::  r1 = 13
INTEGER, TARGET   ::  r2 = 17
INTEGER, POINTER  ::  p1, p2
p1 => r1               ! p1 reikšmė 13
p2 => r2               ! p2 reikšmė 17
p1 = p2          ! p1 reikšmė tampa 17, o tai sukelia ir r1 reikšmės pakitimą į 17.
   ...           ! Tiesiog p1 ir r1 yra vienos atminties srities skirtingi vardai. Lygiai
                 ! tokį pat rezultatą gautume paprasta prieskyra  r1 = r2
```

Iš šių pavyzdžių matom, kad nuorodą programos darbo metu galima nutaikyti į kelis skirtingus taikinius ir tuo būdu gauti dinamiškai kintančią atmintį.

Nuorodų būklė. Yra trys skirtingos būklės:

- nenustatyta. Tokioj būklėj yra visi POINTER pradedant vykdyti programą;
- nulinė - nustatoma operatorium NULLIFY *nuorodos vardas*. Ši būklė reiškia, kad nuoroda rodo “į niekur”;
- nustatyta (associated): nuoroda nukreipta į kažkurį kintamąjį.

Nuorodos būklė programos darbo metu gali būti patikrinta loginio tipo standartine funkcija ASSOCIATED(*nuoroda1* [, *nuoroda2* arba *taikinys*]). Jei funkcija turi tik pirmą argumentą, tai jos rezultatas .TRUE. reiškia, jog *nuoroda1* nustatyta, o .FALSE. - jog nulinė. Jei antras argumentas yra, ir yra nuoroda (kuri privalo būti nustatyta), tai reikšmė .TRUE. reiškia, jog abi nuorodos nulinės arba rodo į tą patį kintamąjį. Jei antrasis argumentas yra taikinys, tai .TRUE. reiškia, jog nuoroda nukreipta būtent į šį taikinį.

Argumentai-taikiniai. Jei nuoroda nukreipiama į faktinį subprogramos argumentą-taikinį, tai šitai niekaip nesusieja nuorodos su atitinkamu formaliuoju argumentu; sąryšis lieka tik su faktiniu argumentu. Jei formalusis argumentas yra taikinys, tai bet kokia su juo susieta nuoroda lieka nenustatyta subprogramai baigus darbą.

Dinaminiai kintamieji. Dinaminę atmintį galima paskirti ne tik kintamiesiems, turintiems atributą ALLOCATABLE, bet ir nuorodoms:

```
REAL, POINTER   :: p1
ALLOCATE( p1 )    ! p1 tampa vardu atminties srities, galinčios saugoti realųjį
                  ! kintamąjį
p1 = 13.          ! p1 dabar galima naudoti kaip paprastą kintamąjį
  . . .
DEALLOCATE( p1 )      ! atminties sritis išlaisvinama; p1 būklė lieka
nenustatyta.
  . . .
```

Beje, tokiomis FORTRANo galimybėmis reikia naudotis apdairiai, nes, pavyzdžiui, tokiu atveju

```
REAL, POINTER   ::   p1, p2
ALLOCATE( p1 )
p1 = 13.
p2 => p1
DEALLOCATE( p1 )
  . . .
```

jei p1 bus nenustatytas, p2 būklė lieka visiškai neapibrėžta: objektas, į kurį p2 taikė, išnyko. Kiekvienas kreipinys į p2 dabar turės nenuspėjamus padarinius.

Funkcijos-masyvai su POINTER atributu. Tai leidžia pasiekti tokį pat efektą, kokį duotų funkcijoms negalimas naudoti atributas ALLOCATABLE.

Pavyzdys. Funkcija sort_vector išrūšiuoja sveikųjų skaičių vienmačio masyvo elementus didėjimo tvarka. Algoritmas: pirmas masyvo elementas lyginamas iš eilės su visais likusiais masyvo elementais; jei jis didesnis už kitą elementą - elementai keičiami vietomis. Ta pati procedūra kartojama antram, ... , priešpaskutiniam elementams. Pagrindinėje programoje pradinį masyvą apribokime 100-u elementų. Funkciją sort_vector iškviesime spausdinimo operatoriuje:

```
IMPLICIT  NONE
INTEGER                          ::  n, i       ! n - elementų kiekis masyve
INTEGER, DIMENSION( 100 )  ::  x        ! pradinis masyvas
PRINT *, 'įvedimas: n'
READ *, n
PRINT *, 'įvedimas: masyvas'
READ *, (x(i), i=1, n)
PRINT *, 'kontrolinis spausdinimas', n, (x(i), i=1, n)
```

```
PRINT *, 'rezultato spausdinimas', n, sort_vector( x )
CONTAINS
  FUNCTION  sort_vector( a )
  INTEGER, DIMENSION( : ), POINTER   ::  sort_vector
  INTEGER, DIMENSION( : )            ::  a
  INTEGER                            ::  i, j, t
  ALLOCATE( sort_vector( SIZE( a ) ) )
  sort_vector = a

  DO i=1, SIZE( a ) - 1
    DO j=i+1, SIZE( a )
      IF( sort_vector( i ) > sort_vector( j ) )  THEN
        t = sort_vector( j )
        sort_vector( j ) = sort_vector( i )
        sort_vector( i ) = t
      END  IF
    END  DO
  END  DO
  END  FUNCTION  sort_vector
END
```

Šis pavyzdys kartu parodo, kaip reikėtų kurti nuorodas - masyvus. FORTRANas 90 neleidžia deklaruoti nuorodų masyvus:

```
INTEGER, DIMENSION( 100 ), POINTER   ::  x   !  neteisinga
```

Teisingas sprendimas būtų taip deklaruoti nurodą, o po to paskirtį atmintį masyvui:

```
INTEGER, DIMENSION( : ), POINTER   ::  x
ALLOCATE( x( n ) )
```

Nuorodos ir išvestiniai duomenų tipai. Tai - patogi programavimui konstrukcija. Išvestiniai duomenys leidžia logiškai sugrupuoti duomenis, o jų komponentai-nuorodos leidžia, jeigu reikia, komponentams skirti skirtingą atminties kiekį. Taigi galima operuoti duomenų struktūromis, kurių dydis nėra aiškus iki programos vykdymo.

Šias idėjas pabandysim paaiškinti pavyzdžiu. Rašoma programa, kuriai bus reikalingi duomenys apie periodinę cheminių elementų lentelę. Kiekvienam iš 106 elementų reikalingi tokie duomenys: vardas (tipas - CHARACTER*12), atominis numeris (INTEGER), masė (DOUBLE PRECISION), izotopų skaičius (INTEGER),

kiekvieno izotopo svoris (DOUBLE PRECISION). Problema čia ta, kad kiekvienas elementas turi skirtingą izotopų kiekį: vandenilis - 3, helis - 5, o didžiausios atominės masės elementai - ir šimtus. Norėdami taupiai naudoti atmintį, kiekvieno elemento izotopų svoriui saugoti turim skirti tiek atminties, kiek iš tikrųjų reikia.

Taigi programos pradžioje aprašom išvestinį duomenį periodic_table ir apibrėžiam tokio tipo masyvą iš 106 elementų, o po to duomenims įvesti, tarkim, iš klaviatūros parašome tokį ciklą:

```
INTEGER, PARAMETER  ::  elements = 106
TYPE  periodic_table
  CHARACTER*12          ::  name
  INTEGER               ::  atomic_number
  DOUBLE PRECISION      ::  mass
  INTEGER               ::  isotopes
  DOUBLE PRECISION, DIMENSION( : ), POINTER     ::  weight
END  TYPE  periodic_table
TYPE( periodic_table ) periodic( elements )    !  masyvo  iš  106  elementų
aprašymas
   . . .
DO i=1, elements                                  ! duomenų įvedimo ciklas
   READ *, periodic(i)%name   &
           periodic(i)%atomic_number   &
           periodic(i)%mass   &
           periodic(i)%isotopes
   ALLOCATE( periodic(i)%weight( periodic(i)%isotopes ))  !  paskiriam vietą
                                                          !  svorių masyvui
   READ *, (periodic(i)%weight(j), j=1, periodic(i)%isotopes)
END  DO
   . . .
```

8.5.2. SAITINIAI SĄRAŠAI

Saitiniai sąrašai (linked lists) - labai lanksti duomenų struktūra, kurios dydis programos vykdymo metu gali augti ar mažėti. Tokia struktūra gaunama naudojant išvestinius duomenis, kurių vienas ar keli elementai yra nuorodos. Duomenims skiriama dinaminė atmintis, o nuoroda nukreipiama į kitą duomenį, kuris pasirodys vėliau. Jei struktūroje yra viena nuoroda - yra vienakryptis saitinis sąrašas, kuriuo galima eiti viena iš dviejų krypčių: nuo pradžios link galo arba atvirkščia kryptimi. Jei

yra dvi nuorodos, tai tokiu dvikrypčiu sąrašu duomenis galima sekti tiek į vieną, tiek į kitą pusę.

Sąrašo elementus priimta vadinti mazgais (nodes). Toks mazgas turi turėti ir reikšmę, ir kartu turi būti nutaikytas į kitą grandinės elementą. FORTRANas leidžia tokią keistą struktūrą, saugančią reikšmę ir nuorodą:

```
TYPE  node
INTEGER  value                          !  reikšmės komponentas
TYPE( node ), POINTER   ::   next              !  nuorodos komponentas
END  TYPE  node
```

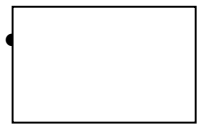
Pavyzdžio dėlei pabandysim sudaryti vienakryptį saitinį sąrašą, atspausdinti sąrašo mazgų reikšmes ir tą sąrašą panaikinti. Pradžiai - sąrašas iš vieno mazgo; tegu jo vieno komponento skaitinė reikšmė bus 1. Deklaruojame du kintamuosius jau aprašyto tipo node: current_node bus bendrinis mazgo vardas, o end_node - sąrašo paskutiniojo mazgo vardas.

```
TYPE( node ), POINTER   ::  current_node, end_node
NULLIFY( end_node )              !  iš pradžių sąrašas tuščias, todėl end_node yra
                                 !  kartu ir sąrašo pradžia, ir pabaiga. Kad vėliau
                                 !  galėtume surasti sąrašo pradžią ar pabaigą,
                                 !  turim suteikti šiam mazgui kokią nors reikšmę -
                                 !  pavyzdžiui, kaip čia - nulinę būklę
ALLOCATE( current_node )         !  išskiriam vietą sąrašo bendriniam mazgui
current_node%value = 1
end_node => current_node!  čia jau sunkiai suprantamas operatorius.
                                 ! Sąraše yra kol kas vienas mazgas current_node,
                                 ! todėl sąrašo galinis mazgas turi rodyti į jį
```

Kad toks sąrašas būtų suprantamesnis, pabandykim jį pavaizduoti grafiškai. Priimta tokius kintamuosius kaip node grafiškai vaizduoti iš dviejų dalių (8.1 a pav.): viršutinė dalis - skaitinei reikšmei, apatinė - nuorodai; nuorodos kryptis žymima rodykle. Nulinės būklės nuorodą priimta žymėti taip, kaip įžeminimą elektros grandinėse (8.1 b pav.).

8.2 a ir b pav. Sąrašo mazgų vaizdavimas

Naudodami tokius žymėjimus, ką tik sudarytą sąrašą iš vieno mazgo galėtume pavaizduoti 8.3 pav.:

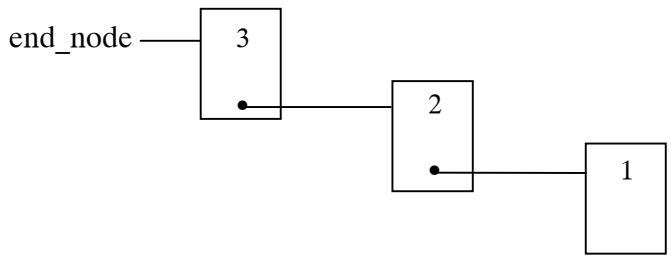


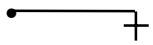
8.3 pav. Vieno mazgo sąrašas

Dabar pabandykim sudaryti bet kokio ilgio vienakryptį sąrašą. Tegu programa įveda skaičius iš klaviatūros ir tuos skaičius talpina į sąrašą. Taip tęsiasi tol, kol įvedamas skaičius 0 :

```
TYPE( node ), POINTER   ::   current_node, end_node
INTEGER  number
NULLIFY( end_node )
DO  WHILE ( number /= 0 )                ! ciklą tęsti tol,
  READ *, number                  ! kol skaičiaus reikšmė ne 0
  IF( number /= 0 )  THEN         ! visus nenulinius skaičius įtraukiame į sąrašą
     ALLOCATE( current_node )
     current_node%value = number
     current_node%next => end_node    ! nuoroda: pirmame ciklo žingsnyje
                                      ! -  į 0, kituose - į ankstesnį mazgą
      end_node => current_node        ! sąrašo galinis mazgas nutaikomas
                                      ! į paskutinį sąrašo mazgą
  END  IF
END  DO
   . . .
```

Toks programos fragmentas, jei iš klaviatūros iš eilės įvestume skaičius 1, 2, 3 ir 0, sudarytų sąrašą, pavaizduotą 8.4 pav. Aišku, sunku patikėti, kad toks sąrašas kompiuterio atmintyje yra, nes juk vidiniai sąrašo mazgai neturi vardų.





8.4 pav. Programos kuriamas sąrašas

Grįžkim prie programos. Bandykim peržvelgti sąrašą ir atspausdinti jame esančius skaičius. Pradėkim nuo sąrašo galo:

```
   . . .
current_node => end_node
PRINT *, current_node%value                ! paskutinio sąrašo mazgo reikšmė
current_node => current_node%next          ! nutaikom mazgą į kitą mazgą
DO  WHILE ( ASSOCIATED( current_node ) )   ! tik taip galim aptikti
                                           ! sąrašo pradžią: pirmas mazgas
                                           ! yra nulinės būklės. Todėl kol
                                           ! funkcija ASSOCIATED
                                           ! randa nutaikytą mazgą -
  PRINT *, current_node%value              ! galim spausdinti jo reikšmę
  current_node => current_node%next        ! nutaikom mazgą į kitą mazgą
END  DO
   . . .
```

Dabar sąrašą panaikinsim. Atrodo, paprasčiausia būtų nukreipti current_node į sąrašo galą: current_node => end_node , ir išlaisvinti current_node skirtą atmintį. Tačiau, kaip matyti iš 8.4 pav., tokiu atveju suardomas ryšys su priešpaskutiniu sąrašo mazgu, ir tos sąrašo dalies bus neįmanoma aptikti, o kompiuterio atmintį ji užims. Sąrašą panaikinti būtų galima taip:

```
   . . .
current_node => end_node                   ! nuoroda į galinį mazgą
DO  WHILE ( ASSOCIATED( end_node ) )       ! kartoti, kol galinis mazgas
                                           ! ne nulinės būklės
end_node => current_node%next              ! end_node atjungiamas nuo pasku-
                                           ! tinio mazgo ir nutaikomas į kitą
  DEALLOCATE( current_node )               ! galima išmesti paskutinį mazgą
  current_node => end_node                 ! nutaikom į paskutinį likusį mazgą
END  DO
   . . .
```

Taigi šis skyrius - tik įvadas į saitinio sąrašo duomenų struktūrą. Norintys tokias struktūras naudoti programose gali remtis J.Kerrigano arba W.S.Brainerdo ir kt. knygomis.

8.5.3. APRIBOJIMAI ATRIBUTAMS ALLOCATABLE, POINTER IR TARGET

Naudojant FORTRANo 90 galimybes dinaminei atminčiai, tenka laikytis daugelio taisyklių. Formalūs apribojimai dinaminės atminties atributams išvardyti šio skyriaus gale. Tačiau lengviausia su abiem dinaminės atminties būdais “susitvarkyti”, jeigu laikomasi kelių paprastų taisyklių:

- dinaminį masyvą visada “alokuoti” ir “dealokuoti” toje pat subprogramoje;
- visada nukreipti nuorodą ir prilyginti ją nuliui toje pat subprogramoje;
- jei duomenys turi išlikti, kai nuoroda išlaisvinama - naudoti NULLIFY; jei duomenys nebereikalingi - galima išlaisvinti su DEALLOCATE.

Jei šių taisyklių nesilaikoma, nuorodos lengvai gali tapti nenustatytos būklės, o to padariniai - nenuspėjami.

Apribojimai atributams:

- nuorodomis negali būti objektai su atributais ALLOCATABLE, EXTERNAL, INTENT, INTRINSIC, PARAMETER, TARGET;
- taikiniais negali būti objektai su atributais EXTERNAL, INTRINSIC, PARAMETER, POINTER;
- nuoroda:
  negali būti nukreipta į konstantą;
  negali būti nukreipta į masyvo dalį, išskirtą vektoriniu indeksu;
  negali būti nukreipta į masyvą, esantį COMMON sąraše;
  negali būti nukreipta į kintamąjį, esantį EQUIVALENCE sąraše;
  negali pasirodyti įvesties/išvesties sąraše, kol nesusieta su taikiniu;
  jei išvestinio tipo duomuo turi komponentą-nuorodą, jis negali pasirodyti įvesties/išvesties sąraše;
  jei formalusis subprogramos argumentas yra nuoroda ir jį atitinkantis faktiškasis argumentas turi būti nuoroda. Jei faktiškasis argumentas yra nuoroda, tai jį atitinkantis formalusis argumentas gali būti nuoroda.

*9. FORTRANO GRAFIKA

9.1. GRAFINĖS PROCEDŪROS

Šiuolaikinis FORTRANas suteikia plačias galimybes programuoti grafinius vaizdus, t.y. pateikti skaičiavimo rezultatus grafiškai. Tai padaryti leidžia grafinių procedūrų (funkcijų ir paprogramių) biblioteka, įeinanti į FORTRANo programavimo sistemos rinkinį. Skirtingų platformų FORTRANuose, kaip, pavyzdžiui, Fortran for HP U-X, XL Fortran for AIX, Microsoft Fortran PowerStation, Digital Fortran for MS Windows NT/95, šios bibliotekos gali šiek tiek skirtis. Mes toliau kalbėsime apie Digital Visual Fortran for MS WINDOWS NT/95 grafinę sistemą.

Grafinės procedūros gali būti naudojamos tik *Standard Graphics* arba *Quickwin* tipo programoms kurti (apie programų tipus, jų projektus rasite 10 skyriuje). Grafinės procedūros braižo displėjaus ekrane linijas, stačiakampius, elipses ar kitus paprastus grafinius elementus, rašo įvairaus šrifto ir dydžio tekstą, nustato ekrano grafinį režimą ir t.t. Šių procedūrų biblioteka yra modulis DFLIB, todėl grafinės programos pradžioje būtina užrašyti operatorių

```
USE   DFLIB
```

Pavyzdžiui, paprogramio su grafikos elementais tekstas turėtų prasidėti taip:

```
SUBROUTINE grafika
USE   DFLIB
INTEGER*2  status, dummy, maxx, maxy
ir t.t.
```

Microsoft Fortran PowerStation aplinkoje rašomoms programoms reikėtų nurodyti įterptinius failus INCLUDE FGRAPH.FI ir INCLUDE FGRAPH.FD. Priminsime, kad INCLUDE FGRAPH.FI rašomas prieš operatorius PROGRAM, SUBROUTINE ar FUNCTION, o INCLUDE FGRAPH.FD po jų.

Grafinės procedūros leidžia atlikti šiuos veiksmus:

- Parinkti displėjaus parametrus.
- Nustatyti ekrano koordinates.
- Nustatyti spalvų paletę.
- Nustatyti linijų stilių, geometrinių figūrų spalvą, ornamentą bei kitus požymius.
- Nubrėžti grafinius elementus.
- Parodyti įvairaus dydžio ir šriftų tekstą.
- Išsaugoti ir atstatyti ekrano atvaizdą. Užrašyti jį į BMP tipo failą, pateikti šį failą ekrane.

Kituose skyreliuose plačiau aptarsime šiuos veiksmus ir juos atliekančius paprogramius bei funkcijas. Kadangi daugumos jų vardai viršija 6 simbolius reikia vengti grafinėse programose naudoti metakomandą $TRUNCATE, kuri draudžia vartoti tokius vardus.

9.2. GRAFINIŲ CHARAKTERISTIKŲ NUSTATYMAS

Prieš pradėdami braižyti grafinius elementus, mes turime nustatyti kompiuterio grafinės įrangos tipą, norimą ekrano skiriamąją gebą, naudojamų spalvų kiekį. Kitaip tariant, turime nustatyti tam tikras grafines charakteristikas - video režimą, kuriame kursime vaizdus. Šiame ir kituose skyreliuose naudosime tokias sąvokas:

- "*Koordinačių pradžia*" (taškas 0,0) yra viršutinis kairysis ekrano kampas. Tai x ir y ašių pradžios taškas. Kai kuriose koordinačių sistemose koordinačių pradžią galima keisti.
- Horizontali kryptis išreiškiama "*x ašimi*". Didžiausias vaizdo elementų (taškelių, pikselių) skaičius horizontalia kryptimi svyruoja nuo 320 iki 1280, pagal video adapterio galimybes ir nustatytą video režimą.
- Vertikalią kryptį išreiškia "*y ašis*". Vertikalių taškelių ribos nuo 200 iki 1024.
- Kai kurie video adapteriai pateikia "*spalvų gamą*", kuri gali būti pakeista.
- Galima pateikti 2, 4, 8, 16, 32 ar 256 "*spalvų indeksus*" (tam tikrą skaičių, susietą su atitinkama spalva), pagal video adapterio ir video režimo. Spalvos indeksas yra INTEGER*2 tipo skaičius

Prieš pradėdami konkrečius braižymo veiksmus, turime nusistatyti ekrano grafinį režimą. Svarbiausi jo parametrai yra maksimalus taškelių skaičius horizontalia ir vertikalia kryptimis, spalvų kiekis. Šie parametrai paprastai nustatomi instaliuojant operacinę sistemą ar WINDOWS aplinką. Jie priklauso nuo konkretaus kompiuterio įrenginių (video adapterio, displėjaus). FORTRAN PowerStation grafinė biblioteka palaiko daugelį adapterių, taip pat ir dabar populiarius SVGA adapterius ir pagal jų galimybes leidžia nustatyti įvairius grafinius režimus.

Informacija apie video režimą, adapterį ir monitorių pateikiama INTEGER*2 tipo parametrais, kurių simboliniai vardai deklaruojami faile FGRAPH.FD. Video režimą galima nusakyti 24 konstantomis. 9.1 lentelėje pateikiamos dažniau vartojamų režimų reikšmės.

Video režimui nustatyti taip pat naudojami parametrai $MAXCOLORMODE arba $MAXRESMODE. Šiuo atveju video režimas bus priimtas atsižvelgiant į esamą video adapterį. Parametro $MAXCOLORMODE reikšmė sutaps su $MRES256COLOR, o $MAXRESMODE su $VRES16COLOR prie VGA adapterio ir abiejų parametrų reikšmės sutaps su $VRES256COLOR prie SVGA adapterio. Šiuos parametrus patartina naudoti, jei nežinome, kokie video adapteriai yra programą vykdančiuose

kompiuteriuose. Originalus kompiuterio video režimas atstatomas parametru $DEFAULTMODE.

9.1 lentelė. Video režimų parametrai

Simbolinis parametras	Video režimas	Video adapteris
$MRES256COLOR	320x200, 256 spalvos	VGA
$VRES2COLOR	640x480, juodai/baltas	VGA
$VRES16COLOR	640x480, 16 spalvų	VGA
$VRES256COLOR	640x480, 256 spalvos	SVGA
$SRES16COLOR	800x600, 16 spalvų	SVGA
$SRES256COLOR	800x600, 256 spalvos	SVGA
$XRES16COLOR	1024x768, 16 spalvų	SVGA
$XRES256COLOR	1024x768, 256 spalvos	SVGA
$ZRES16COLOR	1280x1024, 16 spalvų	SVGA
$ZRES256COLOR	1280x1024, 256 spalvos	SVGA

Video režimui nustatyti naudojame funkciją

SETVIDEOMODE(*mode*)

Čia *mode* - I*2 tipo įėjimo argumentas. Juo gali būti vieno iš aukščiau minėtų parametrų simbolinis vardas. Pavyzdžiui, jei mes norime ekrane turėti 1024x768 taškelių ir 256 spalvas, programoje rašysime

```
status = SETVIDEOMODE( $XRES256COLOR )
```

Jei kompiuteris, kuriame vykdoma programa, nepalaiko nurodyto režimo, funkcijos reikšmė lygi 0 ir teksto eilučių skaičiui, jei režimas nustatytas sėkmingai.

Paprogramė GETVIDEOCONFIG pateikia informaciją apie kompiuterio grafinės aplinkos įrenginių parametrus. Ji iškviečiama operatoriumi

CALL GETVIDEOCONFIG(*s*)

Čia *s* – išvestinio tipo duomens (struktūros) ***videoconfig*** išėjimo argumentas. Ši struktūra aprašyta faile FGRAPH.FD:

```
TYPE /videoconfig/
        INTEGER*2 numxpixels        ! Taškelių skaičius x-ašyje
        INTEGER*2 numypixels        ! Taškelių skaičius y-ašyje
        INTEGER*2 numtextcols       ! Galimas teksto stulpelių skaičius
        INTEGER*2 numtextrows       ! Galimas teksto eilučių skaičius
        INTEGER*2 numcolors         ! Spalvų skaičius
```

```
        INTEGER*2 bitsperpixel      ! Bitų kiekis taškeliui
        INTEGER*2 numvideopages     ! Galimų video puslapių skaičius
        INTEGER*2 mode              ! Esamas video režimas
        INTEGER*2 adapter           ! Adapterio tipas
        INTEGER*2 monitor           ! Monitoriaus tipas
    INTEGER*2 memory                ! Video atmintis kilobaitais
END TYPE
```

Naudodami funkciją DISPLAYCURSOR(*toggle*) galime valdyti žymeklio matomumą. Nustatytas režimas galios iki kito kreipimosi į funkciją. I*2 tipo įėjimo argumentas *toggle* gali turėti reikšmę $GCURSOROFF arba $GCURSORON

Jei kompiuterio video įranga palaiko kartotinius video puslapius, galima naudoti funkciją SETACTIVEPAGE aktyviam darbiniam puslapiui (tai atminties sritis, kurioje duotu momentu programa formuoja grafinį vaizdą) ir SETVISUALPAGE - puslapiui, kuris bus parodytas ekrane (visualizuojamam puslapiui), nustatyti. Šie moduliai naudojami tada, kai norime rodyti vieną puslapį, kol atmintyje formuojamas kitas. Taip programuojama nesudėtinga animacija.

SETACTIVEPAGE(*page*) SETVISUALPAGE(*page*)

page - I*2 tipo įėjimo argumentas, nurodantis atminties puslapio numerį. Grąžinama I*2 funkcijos reikšmė lygi ankstesnio puslapio numeriui. Puslapių numeriai prasideda nuo 0, jų kiekis gaunamas iš funkcijos GETVIDEOCONFIG argumento s%numvideopages reikšmės.

Funkcijos GETACTIVEPAGE ir GETVISUALPAGE pateikia aktyvaus ir visualizuojamo puslapių identifikacinius numerius. Šių funkcijų sintaksė yra:

GETACTIVEPAGE( ) GETVISUALPAGE( )

Grąžinama I*2 tipo reikšmė nusako atitinkamą atminties puslapio numerį, t.y. kuris puslapis formuojamas ir kuris rodomas ekrane.

9.3. KOORDINAČIŲ SISTEMOS

FORTRANo grafikoje galima naudoti skirtingas koordinačių sistemas. Kiekviena sistema atlieka jai būdingas funkcijas. Atsižvegiant į ekrano būseną (tekstinę ar grafinę), koordinačių sistema bus tekstinė arba grafinė. Tekstinė koordinačių sistema dalija ekraną į 25 eilutes ir 80 stulpelių. Eilutės numeruojamos iš viršaus į apačią nuo 1 iki 25, stulpeliai iš kairės į dešinę nuo 1 iki 80. Šias eilučių ir stulpelių koordinates naudoja paprogramiai, skirti informacijai išvesti tekstiniu režimu (*displaying character based text*). Plačiau apie tai 9.7 skyrelyje.

Geometriniams elementams braižyti, tekstui tam tikru šriftu išvesti (*displaying font based characters*) naudojamos grafinės koordinačių sistemos. Jos nusako vaizdo elemento (*pixel*) vietą ekrane. Naudojamos trijų rūšių grafinės koordinatės:

- Fiksuotos "fizinės koordinatės" (*physical coordinates*), kurias apibrėžia kompiuterio įranga ir naudojamas video režimas.
- "Braižymo srities koordinatės" (*viewport coordinates*), kurios nustatomos tam tikrai ekrano daliai (langui).
- "Lango koordinatės"(*window coordinates*), apskaičiuojamos iš slankiojo kablelio duomenų reikšmių, parenkant atitinkamą mastelį.

Fizinės koordinatės nurodo vaizdo elemento (pixel'io) vietą ekrane (jo eilės numerį horizontalia ir vertikalia kryptimis). Fizinių koordinačių sistemoje taškas (0,0) yra ekrano viršutinis kairysis kampas. X-ašies teigiama kryptis iš kairės į dešinę, Y-ašies - iš viršaus į apačią. Taškeliai numeruojami nuo 0 (o ne nuo 1). Jei, tarkime, displėjaus skiriamoji geba nustatyta 640x480, tai X koordinatė galės įgyti reikšmę nuo 0 iki 639, o Y koordinatė nuo 0 iki 479. Taigi fizinės koordinatės išreiškiamos sveikaisiais I*2 tipo dydžiais.

Grafinių charakteristikų apibrėžimo funkcija SETVIDEOMODE fizinių koordinačių pradžią nustato taške (0, 0). Tačiau, jei reikia, galima keisti koordinačių pradžios vietą. Nustatyti koordinačių sistemos pradžios tašką galima paprogramiu SETVIEWORG(*x, y, s*). Čia *x* ir *y* I*2 tipo įėjimo argumentai, nurodantys naują koordinačių pradžios tašką, *s* - struktūros ***xycoord*** tipo išėjimo argumentas, išsaugantis seną koordinačių pradžios tašką. Pavyzdžiui, operatorius

```
CALL SETVIEWORG( 50, 100, org)
```

perkels koordinačių pradžią į ekrano tašką (50, 100). Dabar X koordinatės galės įgauti reikšmes nuo -50 iki 589 ir Y koordinatės nuo -100 iki 379. Šios koordinatės tiks braižymo moduliams MOVETO, LINETO, RECTANGLE, ELLIPSE, POLYGON, ARC ir PIE.

Ekrane galime taip pat nustatyti stačiakampę sritį (*clipping region*), kurioje bus leidžiama braižyti. Ji nustatoma paprogramiu SETCLIPRGN. Pavyzdžiui, ekrane 640x480 norime nustatyti, kad brėžinys būtų ne arčiau kaip 10 vaizdo elementų nuo ekrano krašto. Tai padarysime operatoriumi

```
CALL SETCLIPRGN(10, 10, 629, 469).
```

Braižymo sritis (*viewport*) yra stačiakampė ekrano dalis (langas), kurioje yra braižoma, ar kitais žodžiais tariant, tai grafinio objekto vaizdavimo erdvės sritis (*drawing area*), kurioje vaizduojamas ir peržiūrimas modeliuojamas objektas ar jo dalis. ***Braižymo srities koordinatės*** leidžia mums rašyti atskirų objektų braižymo paprogramius, neatsižvelgiant į konkrečią jo pavaizdavimo ekrane vietą. Ekrane mes galime išskirti

keletą langų ir juose vaizduoti skirtingus objektus. Braižymo sritis nustatoma paprogramiu

SETVIEWPORT(x1,y1,x2,y2),

kurio argumentais nurodome viršutinio kairiojo (x1,y1) ir apatinio dešiniojo (x2,y2) kampo koordinates fizinių koordinačių sistemoje. Braižymo srities koordinačių sistemos pradžia - viršutinis kairysis kampas. Iš esmės šios koordinatės tai tos pačios fizinės koordinatės, išreiškiančios vaizdo elementų numerius, tačiau ne visame ekrane, o jo dalyje, t.y. jos negali viršyti x2-x1 ir y1-y2 reikšmių.

Ligi šiol kalbėjome apie koordinates, išreikštas sveikaisiais skaičiais, tačiau dauguma objektų ir funkcijų, kurias reikia pavaizduoti grafiškai, išreiškiamos realiaisiais skaičiais, ypač tai būdinga inžineriniams skaičiavimams. Šiuo atveju patogu naudoti ***langa koordinates***, kurios išreiškiamos dvigubojo tikslumo realiaisiais skaičiais. Pirmiausiai funkcija SETWINDOW nustatomas realiųjų skaičių langas, kuris atitinkamu masteliu susiejamas su fizinėmis koordinatėmis (braižoma visame ekrane) arba braižymo srities koordinatėmis (braižoma tam tikroje ekrano dalyje). Šios funkcijos sintaksė yra:

SETWINDOW(*finvert, wx1, wy1, wx2, wy2*), čia

finvert - logical*2 įėjimo argumentas, nustatantis koordinačių kryptį. Jei *finvert* yra .TRUE., Y ašis didėja iš apačios į viršų (kartezinės koordinatės), jei .FALSE., iš viršaus į apačią (kaip fizinės koordinatės).

wx1, wy1, wx2, wy2 - real*8 įėjimo argumentai, nustatantys viršutinio kairiojo ir apatinio dešiniojo lango kampų koordinates.

Grąžinama integer*2 reikšmė nelygi 0, jei funkcija įvykdyta sėkmingai ir lygi 0 kitais atvejais (pavyzdžiui, ekranas nėra grafiniame režime). Pavyzdžiui, operatoriai

```
CALL  SETVIEWPORT(5, 5, 254, 204)
status = SETWINDOW( .TRUE., 0.0, -50.0, 25.0, 350.0)
```

nustatys ekrane braižymo sritį, turinčią horizontalia kryptimi 250 ir vertikalia 200 taškelių, kuri "užklojama" realiųjų reikšmių langu. Dabar taškas ekrane (5, 5) sutaps su realiųjų skaičių lauko tašku (0.0, 350.0) ir taškas (254, 204) su tašku (25.0, -50.0). Kadangi koordinačių krypties argumento reikšmė .TRUE. nustato Y ašies kryptį į viršų, tai ir didesnioji kampus apibrėžianti y reikšmė perkeliama į viršų.

Braižymo veiksmams atlikti realiųjų koordinačių lange taikomas analogiškos fizinėms koordinatėms procedūros, tik jų vardai papildomi galūne _W (Pavyzdžiui, LINETO_W, MOVETO_W, RECTANGLE_W ir t.t.).

Perėjimui iš vienos koordinačių sistemos į kitą galima naudoti specialius koordinačių sistemų konvertavimo paprogramius. Darbui su koordinačių sistemomis skirti šie grafiniai moduliai:

Modulis	Paskirtis
GETCURRENTPOSITION	Apibrėžia esamą poziciją braižymo srities koordinatėse
GETCURRENTPOSITION_W	Apibrėžia esamą poziciją lango koordinatėse
GETPHYSCOORD	Pakeičia braižymo srities koordinates fizinėmis koordinatėmis
GETVIEWCOORD	Pakeičia fizines koordinates braižymo srities koordinatėmis
GETVIEWCOORD_W	Pakeičia lango koordinates braižymo srities koordinatėmis XYCOORD struktūroje
GETWINDOWCOORD	Pakeičia braižymo srities koordinates lango koordinatėmis
SETCLIPRGN	Apriboja grafinį išvedimą ekrano dalyje
SETVIEWORG	Nustato braižymo srities koordinačių sistemos pradžios tašką fizinėse koordinatėse
SETVIEWPORT	Apibrėžia braižymo srities dydį ir vietą ekrane
SETWINDOW	Nustato lango koordinačių sistemą

Fizinių koordinačių sistemoje operuojame vaizdo elementais. Vaizdo elementų skaičius kiekvienos ašies kryptimi priklauso nuo grafinio režimo. Šie skaičiai yra gaunami iš paprogramio GETVIDEOCONFIG kaip kintamųjų NUMXPIXELS ir NUMYPIXELS reikšmės. Galimų spalvų kiekį nustatytame grafiniame režime nurodo kintamasis NUMCOLORS. Fizinių koordinačių tipas yra I*2, nes jos yra vaizdo elementų eilės numeriai horizontalia bei vertikalia kryptimis.

Braižymo srities koordinačių sistemos pradžios tašką perkelti fizinių koordinačių atžvilgiu galima paprogramiu SETVIEWORG. Žymeklio buvimo poziciją visada galime sužinoti GETCURRENTPOSITION ir GETCURRENTPOSITION_W paprogramiais.

Langų koordinačių sistema padeda lengvai parinkti atitinkamą mastelį įvairiai duomenų aibei. Bet kokių duomenų kitimo intervalas (pavyzdžiui, nuo 0 iki 5000) gali būti priimtas kaip lango koordinačių ašių diapazonas. Šios koordinatės priderinamos prie tam tikro ekrano ploto atitinkamu masteliu pagal norimą grafinio vaizdo dydį. Paprogramis SETWINDOW apibrėžia lango koordinačių ribas pagal nurodytas reikšmes.

GETPHYSCOORD perveda lango koordinates į fizines koordinates, o GETVIEWCOORD atlieka priešingą veiksmą. Panašiai GETVIEWCOORD_W pakeičia lango koordinates braižymo srities koordinatėmis, o GETWINDOWCOORD priešingai.

Paprogramis SETCLIPRGN nustato apribotą aktyvų ekrano plotą (iškarpą). SETVIEWPORT atlieka tą patį ir dar nustato braižymo srities koordinačių pradžios tašką iškirptos srities viršutiniame kairiajame kampe.

Dėl vietos stokos nepateikiame išsamaus šiame skyrelyje paminėtų funkcijų bei paprogramių sintaksės aprašymų. Skaitytojas tai gali rasti konkrečios FORTRANo realizacijos Help sistemoje bei atitinkamoje dokumentacijoje.

9.4. SPALVŲ GAMOS (PALETĖS) NUSTATYMAS

Kiekviena ekrane matoma spalva turi savo unikalų kodą, vadinamą spalvos indeksu. Spalvų gamos paprogramiai paskiria spalvas grafinių elementų braižymo funkcijoms ir paprogramiams. Jos gali taip pat pakeisti jau nubraižytų ekrane elementų spalvą.

Spalvų indeksus su braižomomis ekrane spalvomis susieja trys paletės funkcijos:

Funkcija	Paskirtis
REMAPALLPALETTE	Susieja visą spalvų rinkinį su jų indeksais
REMAPPALETTE	Susieja vieną spalvos indeksą su spalva
SELECTPALETTE	Nustato iš anksto apibrėžtą paletę

Pagal konkretų displėjaus grafinį režimą kiekvienam ekrano elementui (pixel'iui) pavaizduoti naudojamos vieno, dviejų, keturių arba aštuonių bitų reikšmės, išreiškiančios spalvos indeksą. Šis indeksas dažnai figūruoja grafinėse funkcijose kaip vienas iš jų argumentų.

Dauguma grafinių režimų palaiko daugiau nei vieną spalvų paletę. SELECTPALETTE funkcija parenka nurodytą iš galimų rinkinio paletę.

EGA, VGA ir SVGA standarto displėjai gali pakeisti paletės susiejimą, nes jie leidžia bet kokią spalvos reikšmę susieti su bet kokiu spalvos kodu. Dvi grafinės procedūros teikia šią galimybę. REMAPPALETTE ir REMAPALLPALETTE naujai susieja vieną spalvos indeksą ir visą paletę.

Tik REMAPPALETTE, SETBKCOLOR ir GETBKCOLOR atpažįsta spalvų reikšmes, apibrėžtas EGA, VGA ir SVGA standarto. Visos kitos procedūros naudoja spalvų indeksus.

9.5. FIGŪROS SAVYBIŲ NUSTATYMAS

Išvesties procedūros, kurios braižo linijas, lankus, elipses ar kitas paprastas figūras, nepateikia tikslios informacijos apie jų spalvą ar linijų formą bei tipą. Tai reikia atlikti žemiau išvardintomis procedūromis:

Procedūra	Paskirtis
GETBKCOLOR	Pateikia esamą fono spalvą
GETCOLOR	Pateikia esamą spalvos indeksą
GETFILLMASK	Pateikia esamą užpildymo trafaretą
GETLINESTYLE	Pateikia esamą linijos tipą
GETWRITEMODE	Pateikia esamą loginį linijos braižymo būdą
SETBKCOLOR	Nustato fono spalvą
SETCOLOR	Nustato braižomo objekto spalvą
SETFILLMASK	Nustato užpildymo trafaretą
SETLINESTYLE	Nustato linijos tipą
SETWRITEMODE	Nustato loginį linijos braižymo būdą

GETCOLOR ir SETCOLOR gauna ir nustato spalvos indeksą, kurį naudoja FLOODFILL, OUTGTEXT ir kitos grafinių elementų braižymo procedūros. Panašiai GETBKCOLOR ir SETBKCOLOR sugrąžina ir nustato fono spalvą.

GETFILLMASK ir SETFILLMASK pateikia ir nustato užpildymo trafaretą (fill mask). Užpildymo trafaretas - tai 8 vienbaičių sveikųjų skaičių masyvas, kur kiekvieno skaičiaus kiekvienas bitas išreiškia vaizdo elementą. Vienbaitis sveikasis skaičius dažnai išreiškiamas dviem 4 bitų ilgio šešioliktainiais simboliais. Tai primena 8 x 8 bitų masyvą. Bito reikšmė 0 nusako, kad atitinkamam vaizdo elementui priskiriama fono spalva, o -1 nustatyta spalva. Trafaretas yra kopijuojamas per visą figūros plotą. Jei užpildymo trafaretas nėra nustatytas arba jo visi bitai lygūs 1, užpildymo operacijose naudojama nustatyta spalva. Šias procedūras galima naudoti ir šešėliams kurti.

GETWRITEMODE ir SETWRITEMODE grąžina ir nustato "loginį rašymo būdą", naudojamą braižyti linijas. Loginis rašymo būdas, kuris nustatomas konstantomis $GANG, $GOR, $GPRESET, $GPSET ir $GXOR, galioja LINETO, RECTANGLE ir POLYGON procedūroms.

GETLINESTYLE ir SETLINESTYLE pateikia ir nustato linijos tipą. Linijos forma yra apibrėžiama 16 bitų ilgio šablonu, kuriame kiekvieną bitą atitinka vaizdo elementas. Jei bitas yra 0, tai vaizdo elementas išlaiko fono spalvą (jis atrodo kaip permatomas). Jei bitas yra 1, vaizdo elementas yra nustatytos spalvos. Šablonas kartojamas per visą linijos ilgį. Šis 16 bitų šablonas paprogramiui SETLINESTYLE gali būti pateiktas kaip I*2 skaičius (16 bitų = 2 baitai) arba kaip 4 šešioliktainiai simboliai, nes kiekvienas šešioliktainis simbolis išreiškiamas 4 bitų kodu. Antras būdas dažniau naudojamas, nes jis yra akivaizdesnis. Pavyzdžiui, ištisinės linijos šablonas bus #FFFF, nes šešioliktainio F kodas yra 1111, o punktyrinės - #F0F0, nes šešioliktainio nulio kodas yra 0000. Kaitaliodami šešioliktainius simbolius galime pateikti pačius įvairiausius linijos tipus. Šie tipai nustatomi prieš braižant įvairias linijas LINETO, RECTANGLE ir POLYGON procedūromis.

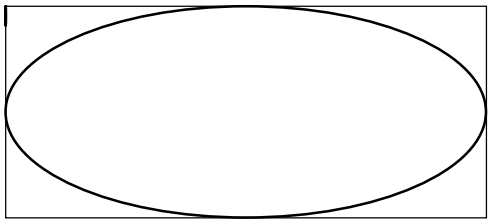
9.6. GRAFINIŲ ELEMENTŲ BRAIŽYMAS

Grafinės bibliotekos procedūros braižo geometrines figūras, naudodamos nurodytą koordinačių sistemą, linijos formą, rašto užpildymą, fono ir braižomo elemento ar rašomo simbolio spalvą. Jei norime naudoti kitą, nei nustatyta standartiškai, linijos tipą (vientisa), raštą (be rašto), fono spalvą (juoda) ar braižomo elemento ar simbolio (balta) spalvą, turime pirmiau iškviesti atitinkamas šių parametrų paskyrimo procedūras, išvardytas ankstesniuose skyreliuose. Grafinių elementų išvesties procedūros naudosis tais pačiais parametrais, kol nebus nustatyti kiti ar pakeistas video režimas. Braižymo procedūrų vardai dažnai nusako, ką jos daro ar kokią figūrą braižo.

Procedūra	**Paskirtis**
ARC, ARC_W	Braižo lanką
CLEARSCREEN	Išvalo ekraną, matymo lauką ar teksto langą
ELLIPSE, ELLIPSE_W	Braižo elipsę ar apskritimą
FLOODFILL, FLOODFILL_W	Užpildo apribotą ekrano sritį spalva ir raštu
GETARCINFO	Nurodo paskutinio nubraižyto lanko galų taškus
GETCURRENPOSITION, GETCURRENPOSITION_W	Pateikia grafinės išvesties pozicijos koordinates
GRSTATUS	Pateikia paskutinės vykdytos grafinės procedūros būklę (sėkmingai ar nesėkmingai)
LINETO, LINETO_W	Brėžia liniją nuo esamos grafinės išvesties pozicijos iki nurodyto taško
MOVETO, MOVETO_W	Perkelia grafinės išvesties poziciją į nurodytą tašką
PIE, PIE_W	Braižo disko formos figūrą
POLYGON, POLYGON_W	Braižo daugiakampį
RECTANGLE, RECTANGLE_W	Braižo stačiakampį
SETPIXEL, SETPIXEL_W	Nustato nurodytovaizdo elemento spalvą

Procedūros, kurių vardai baigiasi "_W" naudoja lango koordinačių sistemą ir dvigubojo tikslumo (DOUBLE PRECISION) argumentus. Procedūros, be šios galūnės, naudoja braižymo srities koordinates.

Kreivalinijinės figūros (lankai, elipsės) centruojamos apibrėžtu stačiakampiu, kuriam nurodomi viršutinis kairysis ir apatinis dešinysis kampai. Stačiakampio centras sutampa su figūros centru, o jo kraštinės nulemia figūros dydį (9.1 pav).



9.1 pav. Ribojantysis stačiakampis

Elementarių grafinių elementų braižymą pademonstruosime parašydami keletą

paprogramių ir sujungdami juos į vieną programą. Toks programos rašymo būdas yra patogesnis, nes jis padeda redaguoti ir derinti atskiras programos dalis nepriklausomai viena nuo kitos, komponuoti jas norima tvarka. Pateikiama programa ekrane nubraižo sinusoidę, stačiakampius, elipses, demonstruoja grafinės programos inicijavimą, braižymą ir uždarymą. Čia panaudotos procedūros būdingos daugeliui grafinių programų. Pagrindinę programą SINE sudaro kreipiniai į penkis paprogramius:

```
C               SINE.FOR  -  pagrindinių grafinių procedūrų iliustracija
      USE DFLIB
      EXTERNAL graphicsmode, drawlines, sinewave, drawshapes, end_pr
      CALL graphicsmode
      CALL drawlines
      CALL sinewave
      CALL drawshapes
      CALL end_pr
      END
```

Prieš pradėdami formuoti grafinius vaizdus displėjaus ekrane, turime nustatyti grafinį režimą, atitinkantį konkretaus kompiuterio vaizdo įrenginius. Tai atliekama paprogramyje *graphicsmode:*

```
      SUBROUTINE graphicsmode
      USE DFLIB
      INTEGER*2  modestatus, maxx, maxy
      TYPE (videoconfig)  myscreen
      COMMON  maxx, maxy
      modestatus = SETVIDEOMODE( $MAXRESMODE )
      IF(modestatus .EQ. 0) STOP ' Klaida: nustatant grafinį režimą'
C             Nustatome maksimalų vaizdo elementų skaičių X ir Y kryptimis
      CALL GETVIDEOCONFIG( myscreen )
      maxx = myscreen%numxpixels - 1
      maxy = myscreen%numypixels - 1
      END SUBROUTINE graphicsmode
```

Paprogramis pradedamas kintamųjų deklaravimu, įskaitant ir kintamojo *myscreen,* turinčio išvestinį struktūros tipą *videocofig*, deklaravimu operatoriumi TYPE. Šis kintamasis yra paprogramio GETVIDEOCONFIG išėjimo argumentas, turintis duomenis apie vaizdo režimą. Jis padeda apskaičiuoti ekrano matmenis, išreikštus vaizdo elementais:

```
      CALL GETVIDEOCONFIG( myscreen )
      maxx = myscreen%numxpixels - 1
```

```
        maxy = myscreen%numypixels - 1
```

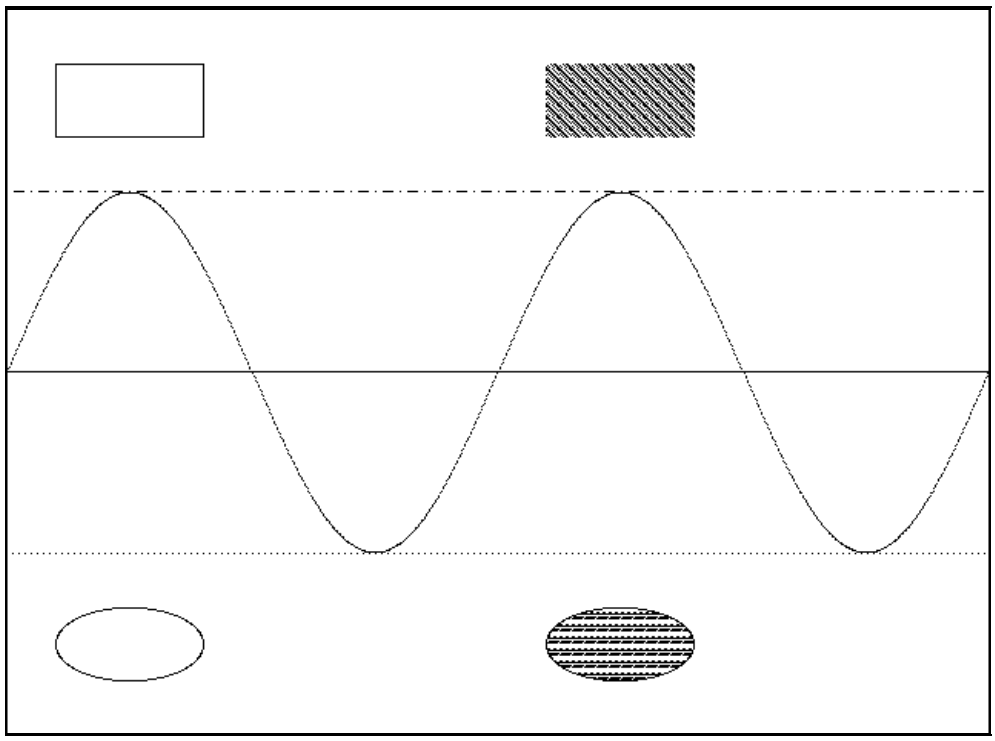
Fizinės koordinatės prasideda nuo nulio, todėl, sakykim, ekrano, turinčio 640 vaizdo elementų horizontalia kryptimi, maksimali x koordinatės reikšmė bus 639. Todėl maxx - didžiausia galima x koordinatės reikšmė turi būti vienetu mažesnė už bendrą vaizdo elementų skaičių. Tas pats taikytina ir maxy. Kaip matome, konkretaus kompiuterio ekrano matmenys vaizdo elementais gali skirtis ir jie nustatomi tik programos darbo metu. Kyla natūralus klausimas - kaip parašyti grafinę programą, nepriklausančią nuo konkretaus kompiuterio. Yra keletas būdų. Vienas iš jų - naudoti lango koordinačių sistemą, aprašytą 9.3 skyrelyje. Kitas būdas - rašyti programą sutartiniam ekranui, tarkime, turinčiam 1000x1000 vaizdo elementų, ir naudoti dvi trumpas funkcijas, konvertuojančias šio sutartinio ekrano koordinates į konkretaus kompiuterio ekrano koordinates:

```
C               NEWX - nustato faktiškąją x koordinatę
        INTEGER*2 FUNCTION newx ( xcoord )
C
        INTEGER*2   xcoord, maxx, maxy
        COMMON      maxx, maxy
        tempx = maxx / 1000.0
        tempx = xcoord * tempx + 0.5
        newx = tempx
        END FUNCTION newx

C               NEWY - nustato faktiškąją y koordinatę
        INTEGER*2 FUNCTION newy ( xcoord )
C
        INTEGER*2   xcoord, maxx, maxy
        COMMON      maxx, maxy
        tempy= maxy / 1000.0
        tempy = xcoord * tempy + 0.5
        newy = tempy
        END FUNCTION newy
```

Paprogramis *drawlines* rodo **linijų braižymą**. Jis nubraižo stačiakampį ekrano kraštuose ir tris horizontalias linijas, dalijančias ekraną į keturias lygias dalis (9.2 pav.).

```
C               DRAWLINES - linijų braižymo paprogramis
        SUBROUTINE  drawlines
        USE DFLIB
        EXTERNAL    newx,newy
```

```
      INTEGER*2     status, newx, newy, maxx, maxy
      TYPE (xycoord) xy
      COMMON        maxx, maxy
C     Braižome stačiakampį ekrano kraštuose
      status = RECTANGLE( $GBORDER, 0, 0, maxx, maxy )
C     Perkeliame koordinačių pradžią į kitą tašką
      CALL SETVIEWORG( 0, newy( INT2( 500 ) ), xy )
C     Braižome linijas
      CALL MOVETO( 0, 0, xy )
      status = LINETO( newx( INT2( 1000 ) ), 0 )
      CALL SETLINESTYLE( #87F0 )
      CALL MOVETO( 0, newy( INT2( -250 ) ), xy )
      status = LINETO(newx(INT2( 1000 ) ),
     +                newy( INT2( -250 ) ) )
      CALL SETLINESTYLE( #8888 )
      CALL MOVETO( 0, newy( INT2( 250 ) ), xy )
      status = LINETO( newx( INT2( 1000 ) ),
     +           newy( INT2( 250 ) ) )
      END SUBROUTINE  drawlines
```



9.2 pav. Programos SINE darbo rezultatas

Pirmasis funkcijos RECTANGLE argumentas yra stačiakampio užpildymo požymis, išreiškiamas parametru $GBORDER arba $GFILLINTERIOR. Jei norime tik apibrėžti stačiakampį keturiomis linijomis, pasirenkame $GBORDER, jei reikia nubraižyti nurodytos spalvos ir ornamento pilnavidurį stačiakampį, pasirenkame $GFILLINTERIOR. Kiti RECTANGLE argumentai yra viršutinio kairiojo ir apatinio dešiniojo kampų x ir y koordinatės.

Paprogramiu SETVIEWORG pakeičiamas koordinačių pradžios taškas. Pirmi du argumentai nurodo naujo koordinačių pradžios taško vietą, o trečiasis struktūrinis *xycoord* tipo argumentas išsaugo ankstesniojo taško koordinates. Struktūrinis *xycoord* duomenų tipas aprašomas DFLIB modulyje apibrėžiamas taip:

```
TYPE   xycoord
       INTEGER*2 xcoord                    ! x koordinatė
       INTEGER*2 ycoord                          ! y koordinatė
END TYPE
```

Operatoriumi

```
CALL SETVIEWORG( 0, newy( INT2( 500 ) ), xy )
```

nustatoma nauja koordinačių pradžia kairiajame ekrano krašte horizontalia kryptimi (x koordinatė nepasikeitė) ir ekrano viduryje vertikalia kryptimi (500 išreiškia sutartinio 1000 vaizdo elementų ekrano vidurį, o funkcija newy apskaičiuoja atitinkamą realaus ekrano vidurio koordinatę). Dabar koordinatės 0, 0 išreikš šį tašką ir y koordinatės reikšmė galės kisti nuo -500 iki +500.

Paprogramiu SETLINESTYLE keičiamas ištisinės linijos stilius (priimtas pagal nutylėjimą). Šio paprogramio I*2 tipo įėjimo argumentas, dažnai pateikiamas 4 šešioliktainiais simboliais, nustato linijos stilių 16 bitų šablonu, kur 0 išreiškia fono spalvos vaizdo elementą, o 1 - linijos spalvos vaizdo elementą. Apie tai jau kalbėjome 9.5 skyrelyje. Pirmiausiai nubraižoma ištisinė linija ekrano viduryje, po to stiliumi taškas-brūkšnys viršutinė linija. Ji suformuojama šešioliktainiu argumentu Z'87F0' (simbolis Z nurodo, kad tai šešioliktainis skaičius, galimas ir alternatyvus užrašas #87F0), kurio dvejetainis kodas yra 1000 0111 1111 0000 (taškas-tarpas-brūkšnys-tarpas ir t.t.). Trečioji linija turi išretintų taškų stilių, kurios šabloną pateikia argumentas Z'8888'. Jo dvejetainis kodas 1000 1000 1000 1000. Šiuos šablonus galima pateikti ir atitinkamais dešimtainiais skaičiais.

Prieš pradėdami brėžti liniją, į jos pradžios tašką "nukeliaujame" operatoriumi

```
CALL MOVETO ( 0, 0, xy),
```

pirmais dviem argumentais nurodydami jo koordinates. Trečias struktūrinis *xycoord* tipo argumentas išsaugo ankstesniojo taško koordinates. Tada liniją nubrėžiame funkcija LINETO, kurios argumentai yra linijos galinio taško koordinatės. Kadangi

dauguma šių funkcijų argumentų yra I*2 tipo, naudojame pakeitimo šiui tipu funkciją INT2. Taigi nubrėžėme stačiakampį ir tris horizontalias skirtingo stiliaus linijas. Dabar paprogramiu SINEWAVE nubrėšime du sinusoidės ciklus. Šį paprogramį galima naudoti kaip prototipą ir kitoms funkcijoms pavaizduoti. Jos tekstas:

```
C                SINEWAVE - skaičiuoja ir braižo sinusoidę
      SUBROUTINE sinewave+
      USE DFLIB
      INTEGER*2      dummy, newx, newy, locx, locy, I
      REAL, PARAMETER::  PI = 3.141593
      EXTERNAL       newx, newy
C          Skaičiuojame sinuso reikšmes ir atitinkamas grafiko koordinates
      DO i = 1 , 999
          sinus  = -SIN( PI * i / 250.0 )
          locx  = newx( i )
          locy  = newy( INT2( sinus * 250.0 ) )
          dummy = SETPIXEL( locx, locy )
      END DO
      END SUBROUTINE sinewave
```

Apskaičiuoti grafiko taškai ekrane vaizduojami funkcija SETPIXEL, nurodant jų koordinates. Čia svarbu teisingai perskaičiuoti vaizdo elemento x koordinatę (šiuo atveju tai ciklo kintamasis i) į funkcijos argumentą (4*PI/1000 *i) → (PI * i / 250.0) ir gautą funkcijos reikšmę į vaizdo elemento y koordinatę (šiuo atveju: newy(INT2(sinus * 250.0))).

Pavaizdavusi sinusoidę, programa SINE nubrėžia du nedidelius stačiakampius ir dvi elipses.Tai atlieka paprogramis DRAWSHAPES. Keičiant užpildymo parametro reikšmę apibrėžiami tik figūrų kontūrai ($GBORDER) ir užpildoma visa figūros sritis tam tikru ornamentu ($GFILLINTERIOR). Paprogramio DRAWSHAPES tekstas:

```
C                DRAWSHAPES - braižo du rėmus ir dvi elipses
      SUBROUTINE drawshapes
      USE DFLIB
      INTEGER*2 dummy, newx, newy
C          Sukuriame užpildymo ornamentą (pattern)
      INTEGER*1, DIMENSION(8)  :: diagmask,  linemask
      DATA diagmask / Z93, ZC9, Z64, ZB2, Z59, Z2C,
     +                Z96, Z4B /
      DATA linemask / ZFF, Z00, Z7F, ZFE, Z00, Z00,
     +                Z00, ZCC /
C    Braižomi stačiakampiai.
```

```
      CALL SETLINESTYLE( 'FFFF' )   ! Ištisinė linija
      CALL SETFILLMASK( diagmask )
      dummy = RECTANGLE( $GBORDER,
     +            newx( INT2( 50 ) ),  newy( INT2( -325 ) ),
     +            newx( INT2( 200 ) ), newy( INT2( -425 ) ) )
      dummy = RECTANGLE( $GFILLINTERIOR,
     +         newx( INT2( 550 ) ), newy( INT2( -325 ) ),
     +         newx( INT2( 700 ) ), newy( INT2( -425 ) ) )
C     Braižomos elipsės
      CALL SETFILLMASK( linemask )
      dummy = ELLIPSE( $GBORDER,
     +      newx( INT2( 50 ) ), newy( INT2(325 ) ),
     +      newx( INT2( 200 ) ), newy( INT2(425 ) ) )
      dummy = ELLIPSE( $GFILLINTERIOR,
     +      newx( INT2( 550 ) ), newy( INT2( 325 ) ),
     +      newx( INT2( 700 ) ), newy( INT2( 425 ) ) )
      END SUBROUTINE drawshapes
```

Ankstesnėje paprogramyje buvo nustatytas brūkšniuotas linijos stilius, todėl figūrų kontūrams apibrėžti vientisa linija vėl iškviečiamas paprogramis SETLINESTYLE su argumentu Z'FFFF'. Stačiakampio ir elipsės užpildymo ornamentai sukuriami priskiriant masyvams diagmask(8) ir linemask(8) tam tikras reikšmes, suformuojančias atitinkamus 8x8 bitų masyvus, kuriuose 0 išreiškia fono spalvos vaizdo elementą ir 1 braižymo spalvos vaizdo elementą. Operatoriumi DATA masyvui diagmask suteiktos reikšmės sukuria tokį ornamentą:

	Bitų šablonas								diagmask(i) reikšmė
Bitų Nr.	7	6	5	4	3	2	1	0	
	●	o	o	●	o	o	●	●	diagmask(1) = Z'93'
	●	●	o	o	●	o	o	●	diagmask(2) = Z'C9'
	o	●	●	o	o	●	o	o	diagmask(3) = Z'64'
	●	o	●	●	o	o	●	o	diagmask(4) = Z'B2'
	o	●	o	●	●	o	o	●	diagmask(5) = Z'59'
	o	o	●	o	●	●	o	o	diagmask(6) = Z'2C'
	●	o	o	●	o	●	●	o	diagmask(7) = Z'96'
	o	●	o	o	●	o	●	●	diagmask(8) = Z'4B'

nes Z'93' dvejetainis kodas 1001 0011, Z'C9' - 1100 1001 ir t.t. Norintiems kurti savo šablonus primename šešioliktainių ir dvejetainių simbolių atitikimo lentelę:

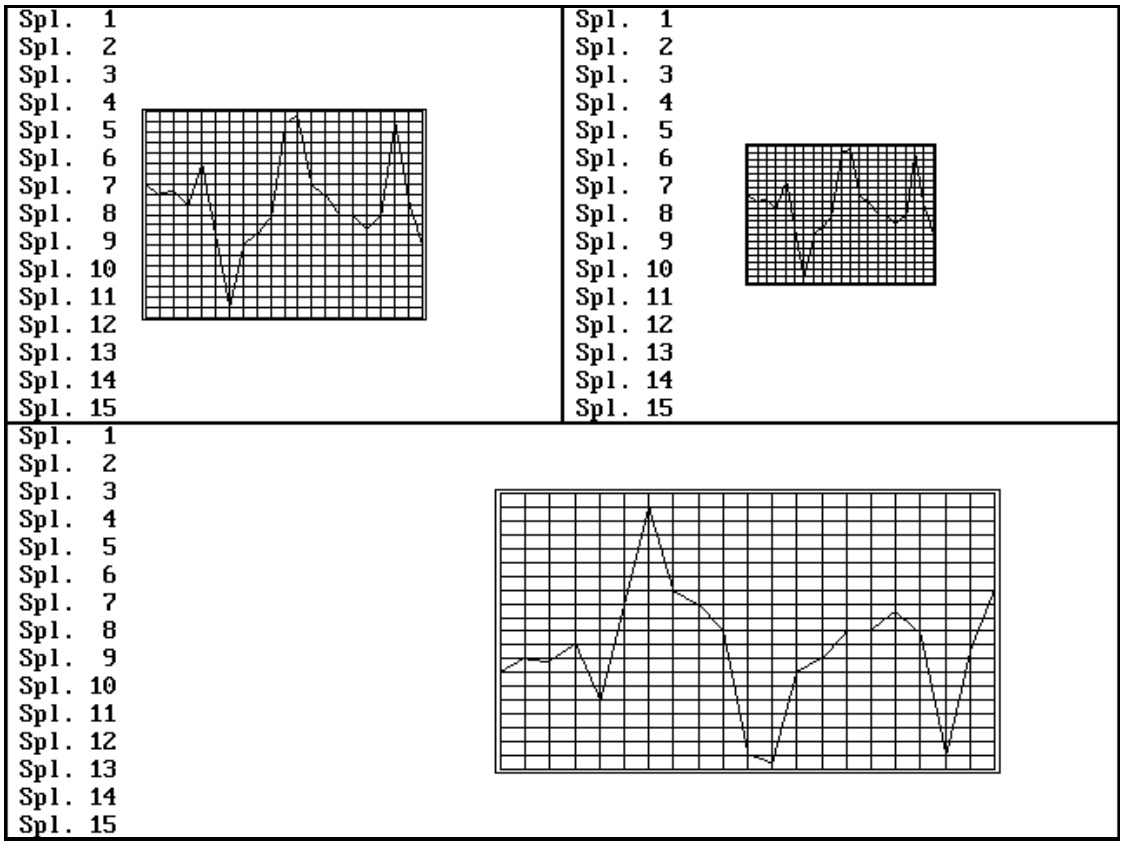
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Paskutinis programos SINE paprogramis *end_pr* laukia, kol bus paspaustas bet koks klavišas, kurio simbolinę reikšmę priims funkcija GETCHARQQ, ir po to atstatomas standartinis vaizdo režimas funkcija SETVIDEOMODE.

```
C                END_PR- grafinės programos baigimas
      SUBROUTINE end_pr
      USE DFLIB
      INTEGER*2   dummy
      CHARACTER*1 dumch
      dumch = GETCHARQQ()          ! Laukia bet kokio klavišo paspaudimo
      dummy = SETVIDEOMODE( $DEFAULTMODE )
      END SUBROUTINE end_pr
```

Panagrinėsime dar vieną pavyzdį, kuriame naudojamos realiojo tipo koordinatės. Tokia situacija pasitaiko pakankamai dažnai, kai norime skaičiavimo rezultatus pateikti grafiškai, tarkime, nubraižyti kokios nors funkcijos grafiką. Žemiau pateikta programa padalija ekraną į tris dalis ir nubraižo jose funcijos grafikus (9.3 pav.). Pagrindinė programa *realg* yra labai trumpa. Joje nustatome grafinį režimą ir kreipiamės į grafikų braižymo paprogramį *threegraphs*.

```
C    Ši programa iliustruoja realiųjų koordinačių panaudojimą
      PROGRAM  realg
      USE DFLIB
      INTEGER*2                  dummy
C    Nustatome grafinį režimą
      dummy = SETVIDEOMODE( $MAXCOLORMODE )
C    Iškviečiame grafikų braižymo paprogramį
      CALL threegraphs
      END PROGRAM  realg
```



9.3 pav. Realiųjų skaičių grafikai. Programos REALG darbo rezultatas

```
C   THREEGRAPHS - šis paprogramis braižo tris grafikus
C
          SUBROUTINE threegraphs
          USE DFLIB
C
          INTEGER*2      dummy, halfx, halfy
          INTEGER*2      xwidth, yheight, cols, rows
          LOGICAL*2  ::  finvert =.FALSE.
          TYPE  (videoconfig) screen
          SAVE  screen
C
          CALL GETVIDEOCONFIG( screen )
          CALL CLEARSCREEN( $GCLEARSCREEN )
          xwidth  = screen%numxpixels
          yheight = screen%numypixels
          cols    = screen%numtextcols
          rows = screen%numtextrows
          halfx = xwidth / 2
          halfy = (yheight / rows) * (rows / 2)
```

```
C
C    Pirmas langas - viršutinis kairysis ekrano ketvirtis
C
          CALL SETVIEWPORT( 0, 0, halfx - 1, halfy - 1 )
          CALL SETTEXTWINDOW( 1, 1, rows / 2, cols / 2 )
          dummy = SETWINDOW( finvert, -2.D0, -2.D0, 2.D0, 2.D0 )
          CALL gridshape( INT2( rows / 2 ) )
          dummy = RECTANGLE( $GBORDER, 0, 0, halfx - 1, halfy - 1 )
C
C    Antras langas - viršutinis dešinysis ekrano ketvirtis
C
          CALL SETVIEWPORT( halfx, 0, xwidth - 1, halfy - 1 )
          CALL SETTEXTWINDOW( 1, (cols / 2) + 1, rows / 2, cols )
          dummy = SETWINDOW( finvert, -3.D0, -3.D0, 3.D0, 3.D0 )
          CALL gridshape( INT2( rows / 2 ) )
          dummy = RECTANGLE_W( $GBORDER, -3.0, -3.0, 3.0, 3.0 )
C
C    Trečias langas - apatinė ekrano pusė
C
          CALL SETVIEWPORT( 0, halfy, xwidth - 1, yheight - 1 )
          CALL SETTEXTWINDOW( (rows / 2 ) + 1, 1, rows, cols )
          dummy = SETWINDOW( finvert, -3.D0, -1.5D0, 1.5D0, 1.5D0 )
          CALL gridshape( INT2( (rows / 2) + MOD( rows, 2 ) ) )
          dummy = RECTANGLE_W( $GBORDER, -3.0, -1.5, 1.5, 1.5 )
C
          READ (*,*)        ! Laukia, kol bus paspastas ENTER klavišas
          dummy = SETVIDEOMODE( $DEFAULTMODE )
          END SUBROUTINE threegraphs
```

Nors grafinio režimo nustatymas automatiškai valo ekraną, galima tai pakartoti paprogramiu CLEARSCREEN. Argumentas $GCLEARSCREEN nurodo, kad bus valomas visas fizinis ekranas. Argumentai $GVIEWPORT ir $GWINDOW naudojami išvalyti bražymo sritį ir realiųjų koordinačių langą.

Pirmas ekrane langas grafikui braižyti ir tekstui rašyti sukuriamas operatoriais:

```
CALL SETVIEWPORT( 0, 0, halfx - 1, halfy - 1 )
CALL SETTEXTWINDOW( 1, 1, rows / 2, cols / 2 )
dummy = SETWINDOW( finvert, -2.D0, -2.D0, 2.D0, 2.D0 )
```

Čia pirmas operatorius apibrėžia braižymo sritį (viewport) kairiajame viršutiniame ekrano ketvirtyje. Toliau apibrėžiamas teksto langas, praktiškai užimantis tą pačią ekrano sritį. Galiausiai apibrėžtoje lango srityje sukuriamas realiųjų koordinačių langas, kurio abi x ir y koordinatės kinta nuo -2.0 iki 2.0. Pirmas loginio tipo argumentas *finvert* nurodo y ašies kryptį. Jei jis turi reikšmę .FALSE., tai y koordinatė didėja iš viršaus į apačią, ir atvirkščiai.

Paprogramis *gridshape* suformuoja tinklelį, braižo laužytų linijų grafiką ir užrašo tam tikrą tekstą. Po to aplink langą nubrėžiamas rėmelis:

```
CALL gridshape( INT2( rows / 2 ) )
dummy = RECTANGLE( $GBORDER, 0, 0, halfx - 1, halfy - 1 )
```

Funkcijoje RECTANGLE naudojamos ne lango, o fizinės braižymo srities koordinatės. Kiti du langai yra panašūs į pirmąjį. Visi jie naudoja paprogramį *gridshape*, kuris braižo tinklelį nuo taško (-1.0, -1.0) iki (1.0, 1.0). Tinklelio dydis visuose languose skirtingas, nes nurodytos skirtingos kiekvieno lango koordanačių ribos. Kuo jos didesnės, tuo tinklelis ir grafikas bus mažesni. Paprogramio *gridshape* tekstas:

```
C   GRIDSHAPE - braižo tinklelį ir duomenų grafiką
C
        SUBROUTINE gridshape( numc )
        USE DFLIB

        INTEGER*2               dummy, numc, i
        CHARACTER*2             str
        REAL*8                  bananas(21), x
        TYPE (videoconfig)   screen
        TYPE (wxycoord)         wxy
        TYPE (rccoord)       curpos
C
C    Grafiko duomenys
C
        DATA bananas /-0.3  , -0.2 , -0.224, -0.1, -0.5  ,
   +             0.21 ,  2.9 ,  0.3  ,  0.2,  0.0  ,
   +            -0.885, -1.1 , -0.3  , -0.2,  0.001,
   +             0.005,  0.14,  0.0  , -0.9, -0.13 , 0.31 /
C
C    Spausdina ekrane spalvotus žodžius.
C
        IF( screen%numcolors .LT. numc ) numc = screen%numcolors - 1
```

```
            DO i = 1, numc - 1
                CALL SETTEXTPOSITION( i, 2, curpos )
                dummy = SETTEXTCOLOR( i )
                WRITE (str, '(I2)') i
                CALL OUTTEXT( 'Spl ' // str )
            END DO
C
C    Apibrėžia aplink tinklelį dvigubą rėmą
C
            dummy = SETCOLOR( 1 )
            dummy = RECTANGLE_W( $GBORDER, -1.00, -1.00, 1.00, 1.00 )
            dummy = RECTANGLE_W( $GBORDER, -1.02, -1.02, 1.02, 1.02 )
C
C    Braižo grafiką
C
            x = -0.90
            DO i = 1, 19
                dummy = SETCOLOR( 2 )
                CALL    MOVETO_W( x, -1.D0, wxy )
                dummy = LINETO_W( x,  1.0 )
                CALL    MOVETO_W( -1.D0, x, wxy )
                dummy = LINETO_W(  1.0, x )
                dummy = SETCOLOR( 14 )
                CALL    MOVETO_W( x - 0.1D0, bananas( i ), wxy )
                dummy = LINETO_W( x, bananas( i + 1 ) )
                 x     = x + 0.1
            END DO
C
            CALL    MOVETO_W( 0.9D0, bananas( i ), wxy )
            dummy = LINETO_W( 1.0, bananas( i + 1 ) )
            dummy = SETCOLOR( 3 )
            END SUBROUTINE gridshape( numc )
```

Paprogramis, kuris baigiasi galūne _W, veikia kaip ir paprogramis be jos, tik jo argumentai, nurodantys taško koordinates lange, yra dvigubojo tikslumo realieji duomenys. Nubraižius ekrane įvairius grafinius elementus kyla natūralus noras papildyti juos įvairiais užrašais. Grafinį vaizdą labai pagyvina įvairaus šrifto ir dydžio antraštės, komentarai ar kiti žymėjimai. Kitame skyrelyje išsamiau aptarsime teksto pateikimą ekrane įvairiais šriftais.

9.7. TEKSTO PATEIKIMAS EKRANE

Tekstą ekrane galime išvesti tiek tekstiniu, tiek ir grafiniu ekrano režimu. Jis gali būti formuojamas standartiniais simboliais arba įvairiais šriftais. Be įprastinių įvesties/išvesties operatorių, tekstą ekrane atskiruose teksto languose galime parodyti šiomis grafinėmis procedūromis:

Procedūra	**Paskirtis**
CLEARSCREEN	Išvalo ekraną ar atskirą langą
DISPLAYCURSOR	Įjungia/išjungia žymeklį
GETBKCOLOR	Pateikia fono spalvą
GETGTEXTVECTOR	Pateikia teksto išvedimo orientaciją
GETTEXTCOLOR	Pateikia esamą teksto spalvą
GETTEXTCURSOR	Pateikia esamą žymeklio formą
GETTEXTPOSITION	Pateikia esamą išvedamo teksto vietą
GETTEXTWINDOW	Pateikia aktyvaus teksto lango ribas
OUTTEXT	Išveda tekstą į ekraną nuo esamos pozicijos
SCROOLTEXTWINDOW	Padeda peržiūrėti tekstą, netelpantį jo lange
SETBKCOLOR	Nustato fono spalvą
SETGTEXTVECTOR	Nustato išvedamo teksto orientaciją
SETTEXTCOLOR	Nustato teksto spalvą
SETTEXTCURSOR	Nustato žymeklio formą
SETTEXTPOSITION	Nustato teksto pradžio vietą
SETTEXTWINDOW	Nustato teksto lango vietą ir dydį
WRAPON	Valdo eilučių išdėstymą teksto lange

Šios procedūros nesuteikia teksto formatavimo galimybių. Jei norime atspausdinti sveikuosiuss ar realiuosius skaičius, turime, prieš naudodami šias procedūras, juos pervesti į išorinį pateikties būdą, t.y. užrašyti juos simbolių sekomis į vidinį failą operatorimi WRITE. Teksto įvesties procedūros apibrėžia visas ekrano vietas simbolių eilučių ir stulpelių koordinatėmis.

SETTEXTWINDOW yra tekstinis SETVIEWPORT ekvivalentas, išskyrus tai, kad jis apriboja ekrano plotą, skirtą tekstui išvesti paprogramiu OUTTEXT. Ji neveikia standartiniams išvesties operatorimas (WRITE, PRINT).

GETTEXTWINDOW pateikia teksto lango ribas, prieš tai nustatytas paprogramiu SETTEXTWINDOW. Paprogramis SCROOLTEXTWINDOW pastumia tekstinio lango turinį. OUTTEXT parodo tekstą, užrašytą tekstiniame lange.

Įsiminti ir pakeisti teksto spalvą galima atitinkamai funkcijomis GETTEXTCOLOR ir SETTEXTCOLOR. Sužinoti išvedamo ekrane teksto vietą ar ją naujai nustatyti padeda GETTEXTPOSITION ir SETTEXTPOSITION. Nepatartina maišyti grafinių teksto išvesties procedūrų SETTEXTWINDOW ir OUTTEXT su standartiniais išvesties operatoriais (WRITE, PRINT), nes jie turi specialius karietėlės grįžties (CR) ir perėjimo į naują eilutę (LF) simbolius, kurie teksto pozicijų nustatymą

ekrane pavers neprognozuojamu. To išvengti dar galima naudojant formato deskriptorių (\ ), kuris sustabdo CR-LF poveikį.

Funkcija SETGTEXTVECTOR nustato išvedamo teksto orientaciją, o funkcija GETGTEXTVECTOR pateikia esamą išvedamo teksto orientaciją, kuri naudojama teksto išvedimo paprogramyje OUTGTEXT.

WRAPON leidžia arba uždraudžia teksto išdėstymą lange keliomis eilutėmis. Jei to daryti neleidžiama, tai ilgas tekstas nukertamas.

FORTRANo grafinių procedūrų biblioteka pateikia ir įvairių šriftų, kuriais galime rodyti ekrane tekstą. Žemiau pateikiamos procedūros valdo teksto išvedimą įvairiais šriftais:

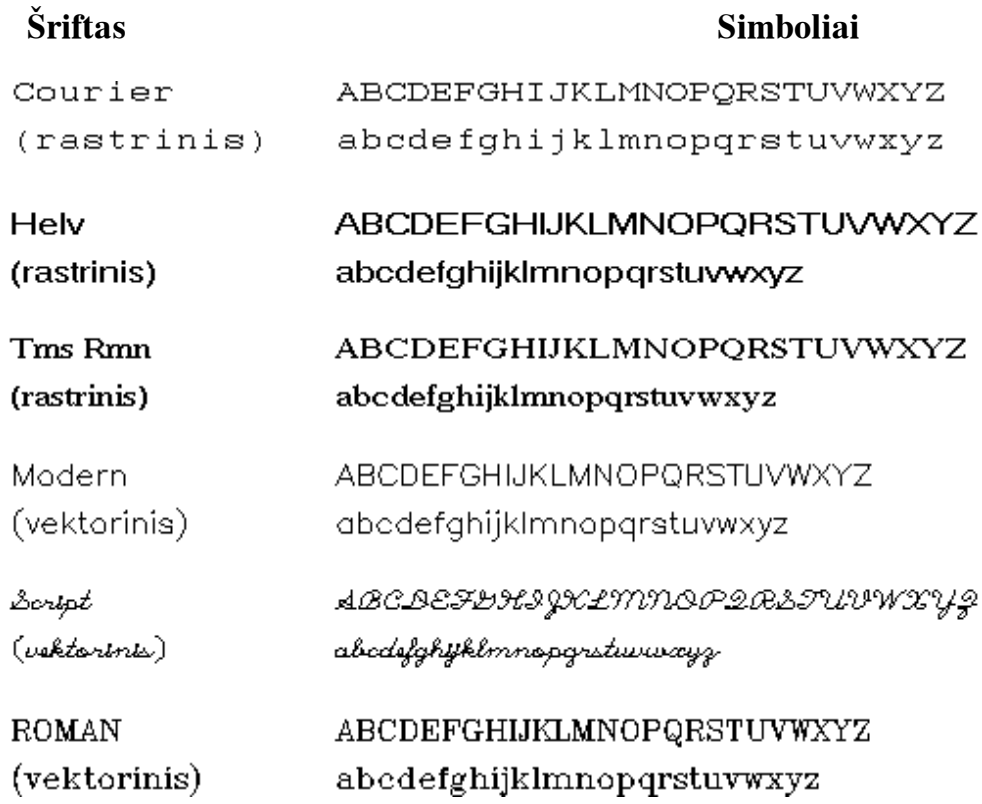
Procedūra	**Paskirtis**
GETFONTINFO	Pateikia šrifto charakteristikas
GETGTEXTEXTENT	Pateikia pažymėto teksto plotį
GETGTEXTROTATION	Pateikia išvedamo su OUTGTEXT teksto orientaciją
OUTGTEXT	Užrašo tekstą nurodytu šriftu nuo aktyvios grafinio išvedimo pozicijos
INITIALIZEFONTS	Inicijuoja šriftų biblioteką
SETFONT	Suranda šriftą, labiausiai atitinkantį nurodytam charakteristikų rinkiniui, ir jį nustato funkcijai OUTGTEXT
SETGTEXTROTATION	Nustato teksto pasukimo kampą laipsnio dešimtosiomis dalimis

Simboliai gali būti braižomi dviem būdais: kaip taškų raidės (bitmapped letters), t.y. raidę vaizduoja atitinkamas taškų “paveikslėlis”, arba kaip vektorių raidės (vector-mapped letters), sudarytos iš tiesių ir lankų rinkinio.

Standartinė FORTRANo grafikos biblioteka turi šešis šriftų tipus. Jų pavyzdžiai pateikiami 9.4 pav. Šrifto dydis nusako, kokį ekrano plotą užima simbolis. Tradiciškai tekstų redaktoriuose šrifto dydis nusakomas punktais (kėgliais). Vienas punktas yra 1/72 colio arba 0,035 cm. FORTRANo grafikoje šrifto dydis nusakomas vaizdo elementais (pixels). Pavyzdžiui, šriftas "Courier 12 9" išreiškia, kad šrifto simboliai bus 12 vaizdo elementų aukščio ir 9 pločio. Kiek tai bus, sakykime, centimetrais, priklausys nuo konkretaus grafinio režimo. Kuo daugiau vaizdo elementų ekrane, tuo fiziškai jie yra mažesni. Terminu *šriftas* dažniausiai išreiškiamas ir tipas, ir jo dydis.

Šriftai yra kuriami dviem būdais. Pirmuoju atveju Courier, Helv ir Tms Rmn šriftų simboliai formuojami bitinio (rastrinio) vaizdavimo (bitmapping or raster-mapping) metodu. Čia kievienas simbolis vaizduojamas tam tikru stačiakampiu taškų šablonu, kur kiekvieną tašką atitinka bito reikšmė (1-šrifto spalva, 0-fono spalva). Kitų trijų - Modern, Script ir Roman - šriftų simboliai formuojami vektoriniu būdu (vector-mapping), t.y. kiekvienas simbolis vaizduojamas kaip tam tikras linijų atkarpų

(vektorių) rinkinys. Abu metodai turi savo privalumų ir trūkumų. Bitinio vaizdavimo simboliai yra lygesni, nes jie apibrėžiami vaizdo elementais. Tačiau jų dydis negali būti laisvai keičiamas. Vektorinio vaizdavimo simbolių dydį galime nustatyti laisvai, bet jiems gali pritrūkti bitinio vaizdavimo simbolių vientisumo.



9.4 pav. FORTRANo šriftų pavyzdžiai

Žemiau pateiktoje lentelėje nurodyti galimi šriftų dydžiai. Bitinio vaizdavimo simbolių matmenys duoti vaizdo elementais. Tikrieji jų dydžiai priklauso nuo konkretaus ekrano skiriamosios gebos ir displėjaus tipo:

Šriftas	Vaizdavimas	Dydis (vaizdo elementais)	Intervalas
Courier	Bitais	10 8, 12 9, 15 12	Fiksuotas
Helv	Bitais	10 5, 12 7, 15 8, 18 9, 22 12, 28 16	Proporcinis
Tms Rmn	Bitais	10 6, 12 6, 15 8, 16 9, 20 12, 26 16	Proporcinis
Modern	Vektoriais	laisvo mastelio	Proporcinis
Script	Vektoriais	laisvo mastelio	Proporcinis
Roman	Vektoriais	laisvo mastelio	Proporcinis

Duomenys apie šiuos šriftus saugomi šriftų failuose, kurių vardai atitinka šriftą, o jų standartinis išplėtimas yra .fon, pavyzdžiui, MODERN.FON, ROMAN.FON ir t.t. Šie failai turi būti kataloge, kuriame vykdoma programa, arba programoje turi būti nurodoma jų buvimo vieta.

Naudojant programoje šriftinį tekstą, reikia:

1. Inicijuoti šriftus.
2. Parinkti aktyvų šriftą iš esamų sąrašo.
3. Pavaizduoti šriftinį tekstą operatoriumi OUTGTEXT.

Naudojanti šriftus programa visų pirma inicijuoja kompiuterio atmintyje jų sąrašą ir nustato jų skaičių. Šis procesas ir vadinamas šriftų registracija. S ąrašas suteikia kompiuteriui informaciją apie esamus *.FON failus. Šriftų registracija atliekama iškvietus funkciją INITIALIZEFONTS( ), kurios grąžinama I*2 tipo reikšmė nurodo užregistruotų šriftų skaičių.

Po registracijos konkretus šriftas, kuriuo bus išvedamas tekstas, nustatomas funkcija SETFONT(*options*). Įėjimo argumentą *options* sudaro tam tikras simbolių rinkinys, aprašantis nustatomą šriftą. Svarbesni iš jų yra:

t'fontname' - šrifto vardas, kuris, paimtas į kabutes, gali būti:

courier	helv	tms rmn
modern	script	roman

hy - simbolių aukštis, kur y aukštis vaizdo elementais.

Wx - simbolių plotis, kur x plotis vaizdo elementais.

Nustačius šriftą tekstas išvedamas tokiais žingsniais:

1. Nustatoma pradinė teksto pozicija procedūra MOVETO.
2. Jei reikia, nustatoma teksto orientacija (kampas) su SETGTEXTROTATION.
3. Išvedamas tekstas į ekraną su OUTGTEXT.

Primename, kad OUTGTEXT išveda į ekraną tik CHARACTER tipo duomenis, todėl skaičius reikia pervesti į tekstinį tipą ir užrašyti juos į vidinį failą operatoriumi WRITE (žr. 5.13 skyrelį).

Pateikiame nedidelę programą, demonstruojančią teksto užrašymą šriftais. Jos išvesti į ekraną tekstai parodyti 9.5 pav.

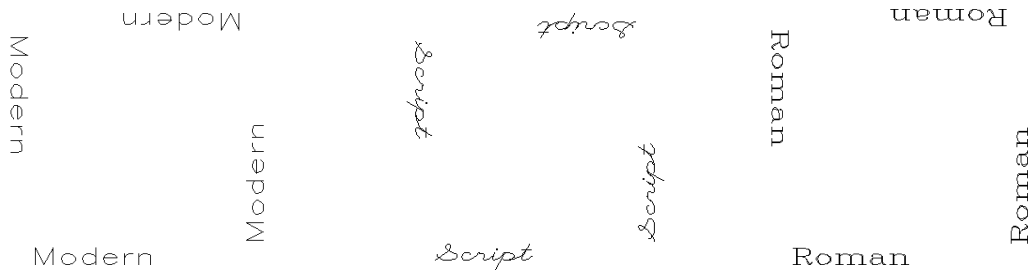
C

```
      PROGRAM fonts
      USE DFLIB
      INTEGER(2)                  ::    dummy, x, y, iend, offset, numfonts
      INTEGER(4)                  ::    ifont
      INTEGER, PARAMETER                ::    NFONTS = 6
```

```
      CHARACTER(11)                    ::    face(NFONTS)
      CHARACTER(10)                    ::    optns(NFONTS)
      CHARACTER                        ::    list*20, fontpath*64
      TYPE (videoconfig)          ::    vc
      TYPE (xycoord)              ::    xy
      TYPE (fontinfo)             ::    fi
C
      DATA face    / "Courier"   , "Helvetica"  , "Times Roman",
     +             "Modern"    , "Script"     , "Roman"      /
      DATA optns / "t'courier'", "t'helv'"    , "t'tms rmn'" ,
     +             "t'modern'" , "t'script'"  , "t'roman'"   /
C
      CALL CLEARSCREEN( $GCLEARSCREEN )
C           Gaunamas esamų šriftų skaičius
      numfonts = INITIALIZEFONTS()
      IF( nfonts > numfonts) STOP ' Yra < nei 6 šriftai'
C                 Nustatomas grafinis režimas
      IF( SETVIDEOMODE( $MAXRESMODE ) .EQ. 0 )
     +  STOP 'Klaida:  Negalima apibrėžti grafinio režimo'
      CALL GETVIDEOCONFIG( vc )
C         Užrašomi įvairiai orientuoti 6 šriftų pavadinimai, kurie
C         centruojami ekrano viduryje
      DO  ifont = 1, nfonts
C     Apibrėžiami šrifto požymiai: aukštis - 30 ir plotis - 24 vaizdo elementai
        list = optns(ifont) // 'h30w24b'
C
        CALL CLEARSCREEN( $GCLEARSCREEN )
        IF( SETFONT( list ) > 0 ) THEN
C         Naudojamas teksto ilgis ir šrifto aukštis įįvedamam tekstui centruoti
          offset = GETGTEXTEXTENT(face(ifont))/2
          x = vc%numxpixels/2 - offset
          IF( GETFONTINFO( fi ) /= 0 ) THEN
            CALL OUTTEXT('Klaida:Negauta informacija apie šriftą')
            READ (*,*)
          END IF
          y = (vc%numypixels - fi%ascent) / 2 + offset
          CALL MOVETO( x, y, xy )
          IF( vc%numcolors  >  2 ) dummy = SETCOLOR( ifont )
```

```
            CALL SETGTEXTROTATION( 0 )
            CALL OUTGTEXT( face(ifont))
            CALL SETGTEXTROTATION ( 900 )
            CALL OUTGTEXT( face(ifont))
            CALL SETGTEXTROTATION ( 1800 )
            CALL GETCURRENTPOSITION( xy )
            CALL MOVETO (xy%xcoord-offset*2_2,
     +        xy%ycoord-offset*2_2, xy)
            CALL OUTGTEXT( face(ifont))
            CALL SETGTEXTROTATION ( 2700 )
            CALL GETCURRENTPOSITION( xy )
            CALL MOVETO (xy%xcoord-offset*4_2,
     +        xy%ycoord, xy)
            CALL OUTGTEXT( face(ifont))
        ELSE
            CALL OUTTEXT( 'Klaida:  Negalima nustatyti srifto' )
        END IF
        READ (*,*)
    END DO
    dummy = SETVIDEOMODE( $DEFAULTMODE )
    END
```

Ši programa ekrane užrašo skirtingai orientuotus 6-ių šriftų pavadinimus, kurie keičiami nuspaudus Enter klavišą. Išvedamo į ekraną teksto orientacija apibrėžiama SETGTEXTROTATION (deg) paprogramiu, kurio I*4 tipo argumentas deg išreiškia teksto pasukimo kampą prieš laikrodžio rodyklę laipsnio dešimtosiomis dalimis, pavyzdžiui, jei deg=-900, tekstas bus rašomas vertikaliai žemyn. Teksto pradžios vieta dar pakoreguojama GETCURRENTPOSITION ir MOVETO paprogramiais.



9.5 pav. Programos FONTS darbo rezultatas

9.8. DARBAS SU ATVAIZDAIS (IMAGES)

Taikomosiose programose, naudojančiose grafikos elementus, dažnai prireikia užfiksuoti esamą atvaizdą ekrane, jį išsaugoti kompiuterio atmintyje ir, jei reikia, jį vėl perkelti į ekraną. Tai padaroma taikant grupę grafinių procedūrų, skirtų darbui su atvaizdais.

Procedūrų GETIMAGE ir PUTIMAGE grupė perkelia atvaizdą iš ekrano į atmintį, ir atvirkščiai. Programa išskiria šiam atvaizdui atitinkamą atminties sritį, kurios dydis apskaičiuojamas procedūra IMAGESIZE.

Procedūros, kurių vardai baigiasi "_W", naudoja lango koordinates, visos kitos braižymo srities koordinates.

Procedūra	**Paskirtis**
GETIMAGE, GETIMAGE_W	Išsaugo ekrano atvaizdą atmintyje
IMAGESIZE, IMAGESIZE_W	Pateikia atvaizdo dydį baitais
PUTIMAGE	Atstato iš atminties ekrano atvaizdą

Jei kompiuterio aplinkos sistema palaiko kelių puslapių grafikos režimą, galima grafinį vaizdą siųsti į puslapį atmintyje, o ekrane vaizduoti kitą grafinį puslapį. Kai jis atmintyje bus užbaigtas, galime jį parodyti ekrane ir pradėti piešti kitą puslapį atmintyje. Šio proceso kartojimas leidžia sukurti ekrane objekto animaciją.

Procedūrų LOADIMAGE ir SAVEIMAGE grupė perkelia atvaizdą iš ekrano į rastriniu būdu (bitmap) koduojamą grafinį failą, ir atvirkščiai.

Procedūra	**Paskirtis**
LOADIMAGE, LOADIMAGE_W	Skaito rastrinį failą (*.bmp) iš disko ir pavaizduoja jį ekrane nurodytomis koordinatėmis
SAVEIMAGE, SAVEIMAGE_W	Užrašo ekrano atvaizdą į grafinį (*.bmp) failą rastriniu būdu

Kartu su ekrano atvaizdu užrašomos į failą ir esamos grafinio režimo charakteristikos. Perkeliant šį atvaizdą iš failo į ekraną, reikia užtikrinti monitoriaus režimo sutikimą su šiomis charakteristikomis.

10. PROGRAMAVIMO APLINKOS

Parašyti programą viena ar kita aukšto lygio algoritmine kalba galime ir popieriuje, nesinaudodami kompiuteriu. Tačiau veikiančiai programai paruošti reikės ne tik kompiuterio, bet ir atitinkamos programinės įrangos - programavimo kompleksų, susidedančių iš transliatorių, kompiliatorių, standartinių procedūrų bibliotekų, tekstų redaktorių, kitų papildomų failų. Šiuolaikiniai programavimo kompleksai sukuria patogias programuotojui programavimo aplinkas, palengvinančias programų tekstų redagavimą bei jų derinimą. Šios aplinkos skiriasi priklausomai nuo naudojamų kompiuterių tipo (personaliniai kompiuteriai, darbo stotys) bei juose įdiegtų operacinių sistemų. Kadangi Lietuvos universitetuose plačiai naudojami personaliniai kompiuteriai ir Microsoft Windows 9X/NT operacinės sistemos, išsamiau aptarsime programavimo aplinką Developer Studio, skirtą programuoti su Microsoft Visual C++, Digital Visual Fortran (DVF) ir kt. programavimo kompleksais personaliniuose kompiuteriuose su minėtomis operacinėmis sistemomis.

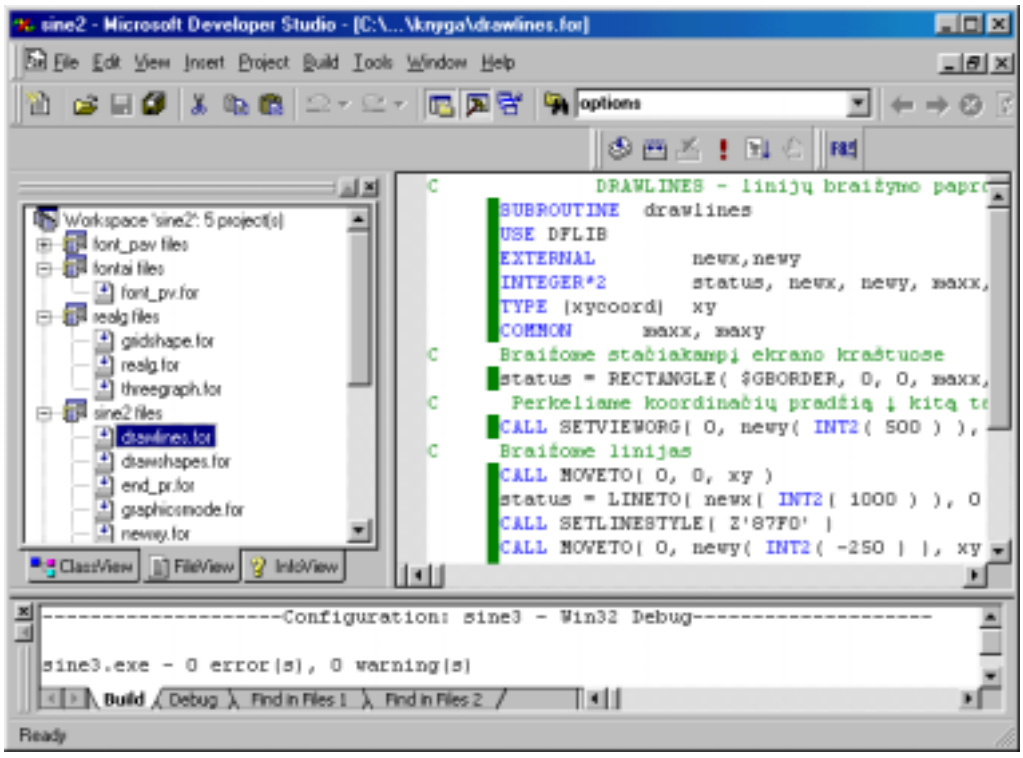
10.1. DIGITAL VISUAL FORTRAN PROGRAMAVIMO APLINKOS LANGAS

Digital Visual Fortran programavimo aplinkos langas iškviečiamas suaktyvinant Developer Studio programos piktogramą (10.1 pav.).

10.1 pav. Developer Studio programavimo aplinkos standartinė piktograma

DVF programavimo aplinkos langas yra standartinis MS WINDOWS langas, turintis visas šiems langams būdingas savybes Jo darbo lauke atidaromi informacinis projekto darbo srities (Project Workspace), programos teksto ir kompiliavimo bei komponavimo rezultatų išvesties (Output) langai (10.2 pav.). Šiuos langus prireikus galime sutraukti, išskleisti, pakeisti dydį, uždaryti standartinėmis Windows priemonėmis. Įvairioms redagavimo ar kitoms operacijoms atlikti skirta meniu eilutė bei įrankių juostos mygtukai, dubliuojantys dažniau naudojamas meniu komandas.

Informaciniame lange suaktyvinus *InfoView* kortelę gaunamas informacinės sistemos pagrindinis meniu, kuriame galime rasti nuorodas į norimą informaciją apie programavimo kalbą, programavimo aplinkos technologiją, įvairius jos įrankius, tad tolimesniuose skyreliuose paliesime tik svarbiausius darbo Digital Visual Fortran aplinkoje aspektus.



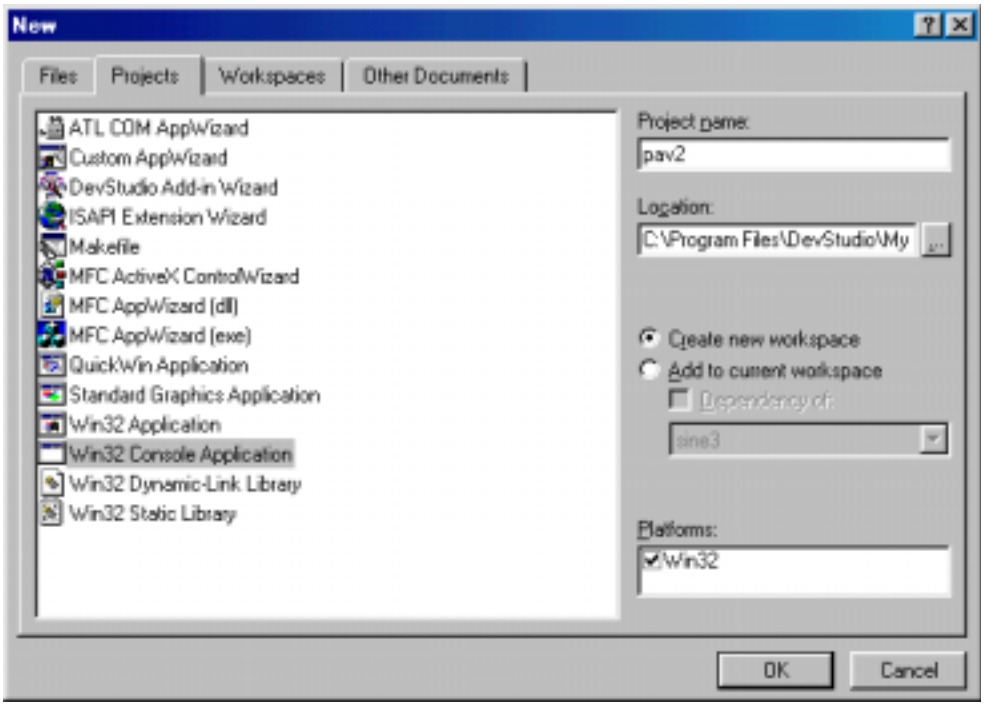
10.2 pav. Digital Visual Fortran programavimo aplinkos langas

10.2. DIGITAL VISUAL FORTRAN PROJEKTAI IR DARBO SRITYS

Projektai (Projects) ir darbo sritys (Workspaces) yra labai svarbūs DVF elementai, kaupiantys visą informaciją apie kuriamas programas. Projektų darbo sritį sudaro 3 failai, saugantys informaciją apie sukurtus projektus, kompiuterio sisteminės aplinkos bei įrenginių charakteristikas. Svarbiausias iš šių failų yra *.dsw failas, kurį reikia nurodyti dialogo lange atidarant anksčiau sukurtą darbo sritį komandomis *File*/*Open Workspace*.... Vienoje darbo srityje gali būti keletas skirtingų tipų projektų. Projektas yra specialus failas (*.dsp), turintis informaciją apie kuriamos programos tipą, pradinių programos tekstų failus (*.for, *.f90), reikalingus objektinių modulių bibliotekoms (*.lib. *.dll) sudaryti, arba apie pradinių programos tekstų failus bei objektinių modulių bibliotekas, reikalingas atitinkamo tipo programos vykdomajam failui (*.exe) sukurti. Jame taip pat yra kompiliatoriui bei saitų redaktoriui (Linker) skirta informacija. Į projektą gali būti įtrauktas vienas pradinio programos teksto failas arba keletas ar daugiau tokių failų bei objektinių modulių ir jų bibliotekų.

Prieš pradėdami rašyti programos tekstą turime nuspręsti, kokio tipo programą

kursime, ir sukurti tam atitinkamą projektą. Paprastoms programoms, naudojančioms standartinę duomenų įvestį/išvestį (operatorius READ, WRITE, PRINT) reikia pasirinkti Console Aplication tipą, jei programoje naudosime grafikos elementus, kursime atskirus langus reikės pasirinkti QuickWin Aplication, Standard Graphics Aplication arba Win32 Aplication tipą. Šių tipų programų pradiniuose tekstuose būtina operatoriumi USE nurodyti atitinkamų paprogramių modulius (DFLIB, DFPORT, DFLOGM ar kt.) Naują projektą sukuriame meniu *File* / *New* lange pasirinkdami kortelę *Projects* (10.3 pav.).

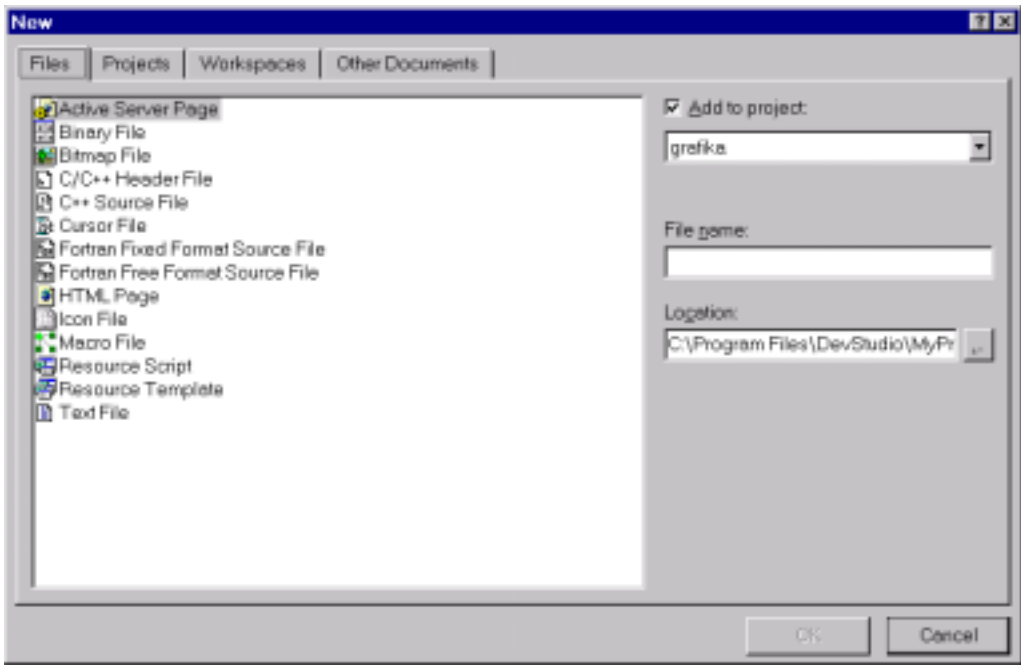


10.3 pav. Projekto kūrimo langas

Lauke *Project name* užrašome projekto vardą (šį vardą gaus ir vykdomoji programa), lauke *Location* parenkame kitą ar paliekame esamą katalogą, kuriame bus sukurtas projekto katalogas ir parenkame jungiklio padėtį kurti naują arba įdėti projektą į esamą darbo sritį.

Naujas programos teksto langas iškviečiamas panašiai kaip ir projektas, tik pasirenkama kortelė *File* (10.4 pav.) Kaip buvo minėta 3-me skyriuje, FORTRAN'o programos tekstą galime užrašyti tradicine fiksuota forma arba būdinga FORTRAN90 laisva forma pasirinkdami atitinkamai *Fortran Fixed Format Source File*, kuris turės plėtinį *.for, arba *Fortran Free Format Source File* su plėtiniu *.f90. Tekstų laukuose *Add to project*, *File Name* ir *Location* nurodome projekto bei failo vardus ir jo saugojimo vietą.

Bakstelėjus mygtuką OK atsidaro programos teksto redagavimo langas, kuriame rašomas FORTRANo programos tekstas. Čia galime vykdyti būdingas Microsoft tekstų redaktoriams kopijavimo, perkėlimo ir kitas operacijas meniu *Edit* komandomis arba atitinkamais funkciniais klavišais bei įrankių juostos mygtukais. Parengtas programos tekstas išsaugomas diske kaip *.for arba *.f90 tipo failas meniu *File* komanda *Save, Save as...* arba įrankių juostos mygtuku .



10.3 pav. Naujo failo kūrimo langas

Dabar galime kompiliuoti pradinį programos tekstą arba iš karto bandyti kurti vykdomosios programos failą *.exe. Tai atliekama meniu *Build* komandomis *Compile File *.for* arba *Build *.exe*. Ekrano apačioje esančiame *Output* lange pateikiamas programos kompiliavimo bei komponavimo protokolas. Jo pabaigos tekstas (AAA - programos teksto failo vardas)

----------Configuration: AAA - Win32 Debug----------

AAA.exe - 0 error(s), 0 warning(s)

rodo, kad programoje sintaksės bei komponavimo klaidų nėra ir galima ją vykdyti meniu *Project* komanda *Execute *.exe*. Bet tai nereiškia, kad programa skaičiuos teisingai. Joje gali būti loginių klaidų, kurios nustatomos tik testinių pavyzdžių skaičiavimais.

Jei programos tekste bus neteisingai parašytų operatorių ar kitų FORTRANo kalbos elementų, t.y. sintaksės klaidų, kompiliatorius informuos apie tai atitinkamu pranešimu. Pavyzdžiui, jei kintamųjų aprašymo operatoriuje INTEGER po paskutinio kintamojo vardo X paliksime kablelį, gausime tokį pranešimą:

```
--------------------Configuration: AAA - Win32 Debug--------------------
Compiling Fortran...
……\AAA.for
……\AAA.for(4) : Error: Syntax error, found END-OF-STATEMENT when expecting one of: %FILL <IDENTIFIER>
 INTEGER(2) status, x,
----------------------------^
Error executing df.exe.

AAA.exe - 1 error(s), 0 warning(s)
```

Tai reikš, kad faile AAA.FOR esančio programos teksto 4-je eilutėje yra sintaksės klaida. Jei jų bus daugiau, tai ir atitinkamų pranešimų bus daugiau. Taisyti klaidas patogiausia *F4* klavišu, kurį paspaudus iš karto pereinama į redagavimo langą, o žymeklis fiksuojamas klaidingoje eilutėje. Į klaidingai parašytą kitą eilutę pereinama tuo pačiu *F4* klavišu.

FORTRANo kompiliatorius pateikia ne tik pranešimus apie sintaksės klaidas, bet taip pat ir perspėjimus (*warnings*) apie įvairius netikslumus bei nesutikimus su FORTRANo kalbos taisyklėmis. Gana dažnai kintamieji deklaruojami jų tipo aprašymo operatoriuose, o tolesniame programos tekste jie nenaudojami, arba atvirkščiai, reiškinyje sutinkamas kintamojo vardas, o jam prieš tai nėra suteikta jokia konkreti reikšmė. Šiais ir panašiais atvejais kompiliatorius pateiks atitinkamus perspėjimus.

Be kompiliavimo galimos ir komponavimo (*linking*) klaidos, kada ryšių redaktorius neranda programos tekste paminėtų procedūrų vardų (*unresolved externals*). Tai atsitinka, jei parašome neteisingą procedūros (paprogramio, funkcijos) vardą, naudojame modulines procedūras, nenurodydami jų modulio vardo operatoriumi USE, neįtraukiame į projektą failų su naudojamomis išorinėmis procedūromis ir pan. Tokią klaidą, pavyzdžiui, sukels neteisingai parašytas grafinės funkcijos IMAGESIZE (praleista pirma I raidė) vardas. Kompiliatorius sintaksės klaidų neras, nes sintaksės požiūriu galimas ir toks vardas, tačiau ryšių redaktorius pateiks tokį pranešimą:

```
--------------------Configuration: grafika - Win32 Debug--------------------
Compiling Fortran...
C:\Program Files\DevStudio\MyProjects\bandymai\grf1.for
Linking...
grf1.obj : error LNK2001: unresolved external symbol _MAGESIZE@16
Debug/grafika.exe : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.
grafika.exe - 2 error(s), 0 warning(s)
```

Tai reiškia, kad procedūros MAGESIZE, kurios argumentų bendras ilgis yra 16 baitų,

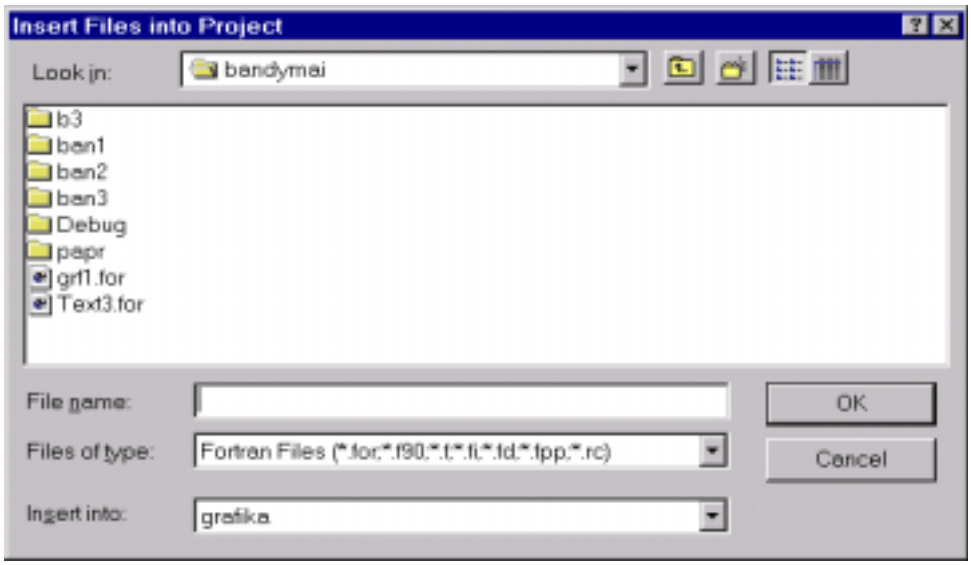
vardas nėra žinomas ir vykdomasis programos failas grafika.exe nėra sukurtas.

Kompiliuoti pradinį programos tekstą, kurti *.exe failą ar vykdyti programą galima ne tik meniu *Project* komandomis *Compile, Build,* ir *Execute,* bet ir naudojant funkcinius klavišus ar įrankių juostos mygtukus (žr. 10.1 lentelę).

10.1 lentelė. Programos parengimo ir vykdymo komandos

Veiksmas	Meniu *Project* komanda	Funkciniai klavišai	Įrankių juostos mygtukai
Kompiliuoti	Compile	Ctrl+F7	
Kurti *.exe failą	Build	F7	
Vykdyti programą	Execute	Ctrl+F5	

Dažnai programą sudaro keletas ar daugiau paprogramių bei funkcijų, kurių tekstai užrašyti atskiruose failuose. Šiuo atveju reikia įtraukti į projektą visus su kuriama programa susijusius pradinių programos tekstų, sutransliuotų objektinių modulių ar jų bibliotekų failus. Tai atliekama meniu *Project* / *Add to Project* / *Files…* lange (10.4 pav.). Lauke *Look in* nustatome katalogą, kuriame yra saugomi mums reikalingi failai, lauke *Files of Type* nustatome reikalingą failų tipo filtrą. Lauke *File Name* pažymėtas failas įtraukiamas į projektą mygtuku OK arba dvigubu kairiojo pelės klavišo paspaudimu ant failo vardo. Įtrauktų į projektą failų sąrašas pateikiamas darbo srities lange (10.2 pav.) suaktyvinus kortelę *FileView*. Pažymėti šiame lange failai šalinami iš projekto mygtuku *Delete*. Atlikę visus projekto papildymo veiksmus programą kompiliuojame (compile), saistome (link, build) bei vykdome (execute, run) meniu *Build* arba 10.1 lentelėje pateiktomis priemonėmis.



10.4 pav. Projekto papildymo langas

*10.3 DVF PERŽIŪROS PROGRAMA (BROWSER)

Didesnės apimties programų (susidedančių iš dešimčių, o kartais ir šimtų funkcijų bei paprogramių) kūrimas ir derinimas yra pakankamai sudėtingas procesas. Viena iš šio proceso dalių yra ryšių tarp atskirų programos vienetų, simbolių nustatymas. Dažnai reikia žinoti, kokiose programos vietose yra iškviečiama viena ar kita funkcija bei paprogramis. Taip pat yra svarbu turėti galimybę greitai pereiti nuo vieno vieneto prie kitų, esančių keliuose skirtinguose failuose. Čia programuotojui gali padėti peržiūros programa (Browser).

Naudodamasi kompiliatoriaus generuojama informacija peržiūros programa padeda rasti susietus tarpusavyje programos teksto simbolius ir parodyti jų ryšius. Ji sukuria specialią duomenų bazę, turinčią informaciją apie paprogramių tarpusavio ryšius bei kintamuosius (kur jie apibrėžiami ir kur naudojami). Šiame skyrelyje ir aptarsime, kaip sukurti, atidaryti peržiūros duomenų bazę bei pateikti jai užklausas.

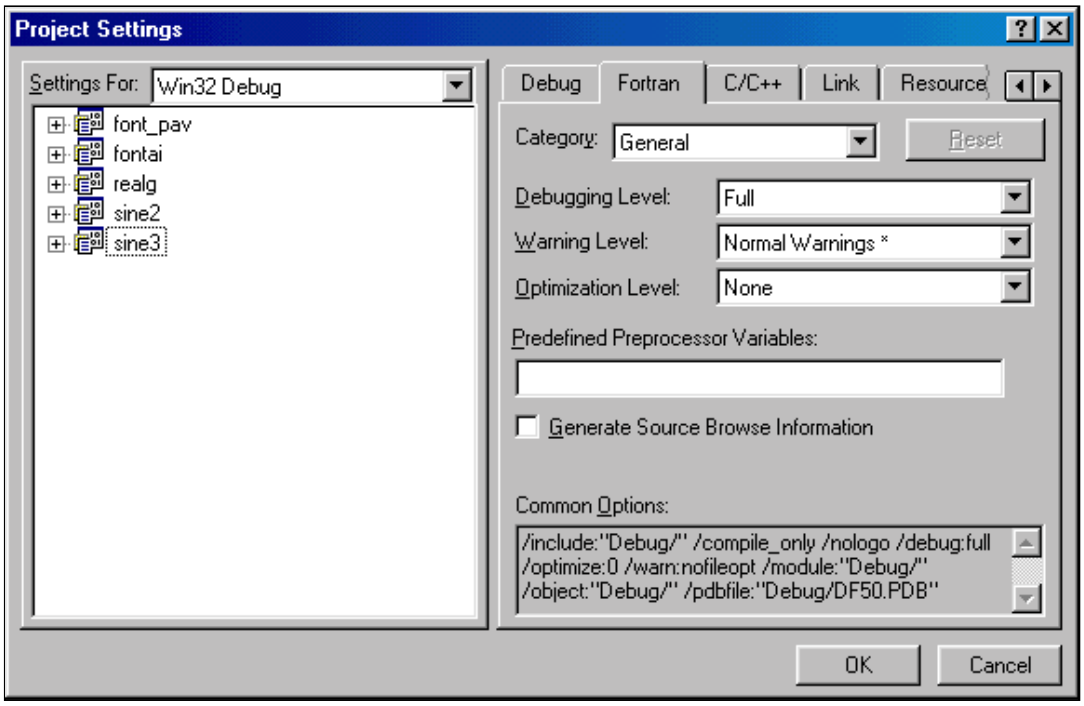
Prieš naudojantis peržiūros duomenų baze reikia ją sukurti. DVF tai atlieka automatiškai, kaip komponavimo (*Build)* proceso dalį, jei tai yra nurodyta projekto nustatymuose. Standartiškai (b*y default*) šis veiksmas nėra vykdomas, nes peržiūros duomenų bazės generavimas lėtina komponavimą, pats duomenų bazės failas gali būti labai didelis.

Peržiūros duomenų bazės failui (.BSC) sukurti reikia atlikti šiuos veiksmus:

1. Meniu *Project* pasirinkti nuorodą *Settings...* Ekrane atsiras *Project Settings* dialogo langas.
2. Suaktyvinti kortelę *Fortran*.
3. Atsiradusiame lange (10.5 pav.) pažymėti pelės spustelėjimu jungiklį *Generate Source Browse Information*. Jame atsiras ženklas ✓.
4. Suaktyvinti kortelę *Browse Info* ir pažymėti jungiklį *Build Browse Info File* .
5. Spragtelėti *OK* mygtuką.
6. Spausti įrankių juostoje *Rebuild* mygtuką  arba pasirinkti meniu *Build* komandą *Rebuild All*. Dabar EXE failo komponavimo metu bus sukurta ir peržiūros duomenų bazė, t.y. failas *.bsc. Programa apie tai informuos rezultatų išvesties lange pranešimu Creating browse info file.

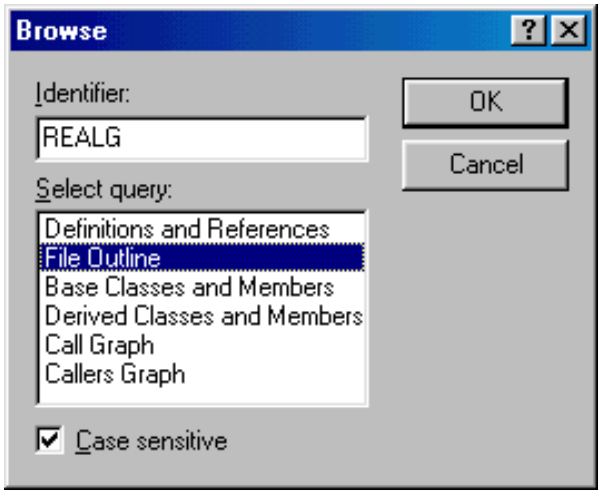
Prieš šiuos veiksmus reikia arba sukurti naują projektą arba atidaryti jau esamą, kuriam ir bus kuriama peržiūros duomenų bazė.

Aktyviam projektui peržiūros duomenų bazė atidaroma automatiškai. Kitų projektų duomenų bazę atidarome meniu *File*/*Open* lange nurodydami atitinkamą *.bsc failą. Atlikti užklausas peržiūros duomenų bazėje pradedame pasirinkdami meniu *Tools* komandą *Source Browser (Alt-F12)*. Ekrane atsiras langas, parodytas 10.6 paveiksle.



10.5 pav. Projekto nustatymų kortelės *Fortran* dialogo laukas

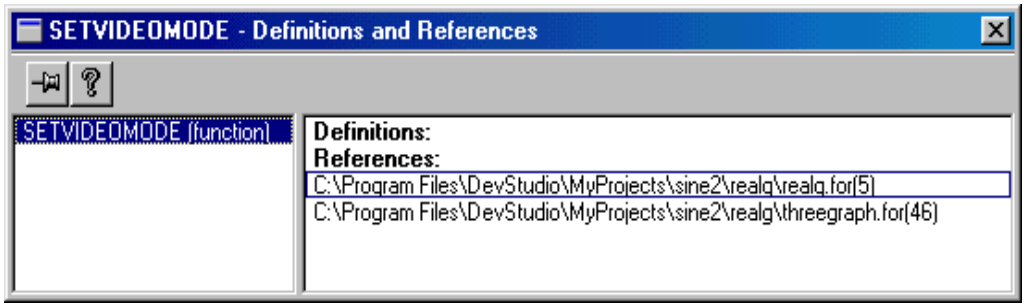
Šiame lange nurodomas objektas (*Identifier*), apie kurį norime gauti informaciją bei pasirenkamas užklausos tipas (*Select Query*).



10.6 pav. Programos REALG peržiūros (Browser) langas

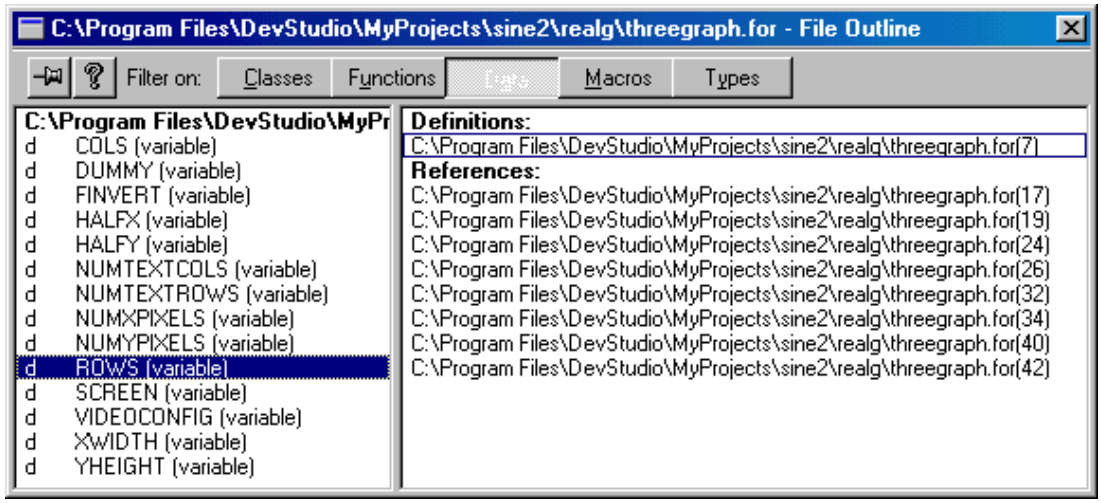
Jei peržiūrimos programos pradinio teksto lange iš anksto pažymėsime norimą objektą (failo, paprogramio, funkcijos, modulio ar kintamojo vardą), jis lauke *Identifier* atsiras automatiškai. Iš *Select Query* lauke esančio užklausos tipų sąrašo išsirenkame norimą tipą. *Definitions and References* (apibrėžtys ir nuorodos), *File Outline* (bendri failo bruožai), *Call Graph* ar *Caller Graph* (grafinis paprogramių ryšio vaizdavimas).

10.7 pav. parodytas 9 skyriuje pateiktos programos REALG funkcijos SETVIDEOMODE užklausos *Definitions and References* langas. Jis parodo, kad į šią funkciją kreipiamasi pagrindinės programos REALG.FOR 5-je ir paprogramio THREEGRAPH.FOR 46-je eilutėse.



10.7 pav. Užklausos *Definitions and References* langas

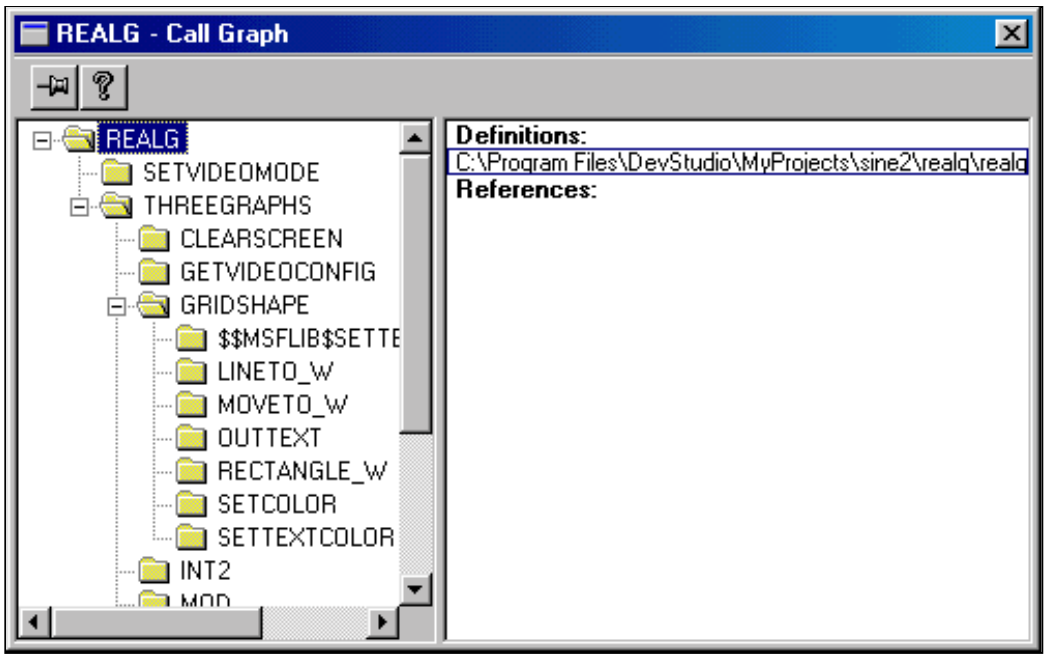
Užklausa *File Outline* galime gauti informaciją apie nurodytame faile esančio FORTRANo programos teksto elementus: funkcijas, duomenis, tipus. 10.8 pav. matome informaciją apie tos pačios programos failo THREEGRAPH.FOR duomenis. Kairiajame lange pateiktas duomenų sąrašas su pažymėtu kintamuoju *rows*, o dešiniajame parodyta, kur šis kintamasis apibrėžtas ir kur yra nuorodos į jį.



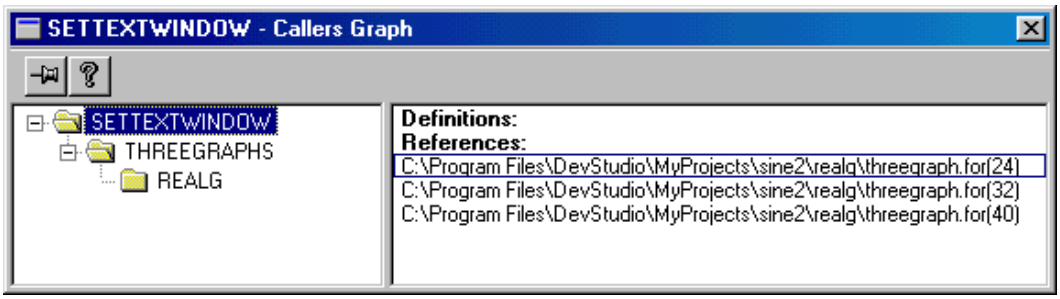
10.8 pav. Failo THREEGRAPH.FOR *File Outline* užklausos langas

Užklausa *Call Graph* parodo tiesioginį programos REALG paprogramių ryšio medį (10.9 pav.). Tačiau dažnai reikia žinoti kelią nuo konkretaus paprogramio iki pagrindinės programos. Užklausa *Caller Graph* pateikia tokį atvirkščią paprogramių ryšio medį (10.10 pav.). Čia matome, kad paprogramis SETTEXTWINDOW iškviečiamas paprogramio THREEGRAPHS 24, 32 ir 40 eilutėse, o pastarasis

iškviečiamas pagrindinėje programoje REALG.



10.9 pav. Paprogramių sąrašo langas



10.10 pav. Atvirkščias paprogramių ryšio medis

*10.4. FORTRANO PROGRAMOS DERINIMAS (DEBUGGING)

FORTRANo programos derinimą (debugging) galime padalyti į dvi fazes. Iš pradžių turime ištaisyti visas kompiliavimo (compiling) ir saistymo (linking) klaidas, norėdami gauti *.exe failą. Tačiau sudėtingose programose atsiranda programos vykdymo (run-time) klaidų arba gaunamas neteisingas rezultatas dėl įvairių loginių ar algoritmo klaidų. Jas rasti pakankamai sudėtinga. Tam reikia bent jau žinoti tarpinius rezultatus įvairiose programos vykdymo stadijose. Tai galima padaryti papildomais

duomenų išvedimo operatoriais. Šis kelias pakankamai paprastas, tačiau nėra racionalus. Efektyviau naudotis specialiomis derinimo programomis, angliškai vadinamomis "debugger". Šis terminas kilęs iš programuotojų žargono programos klaidas vadinti blakėmis, vabalais (angl. bug). Taigi, debuger'is yra vabalų (klaidų) naikintojas, nes priešdėlis de turi neigimo reikšmę. Tokį programų derinimo įrankį turi ir DVF programavimo aplinka.

Programų derintojas leidžia mums sekti programos vyksmą, stabdyti ją ir pamatyti bei keisti tuo metu esamas kintamųjų reikšmes, peržiūrėti atminties registrų, procedūrų iškvietimo steko turinį, gauti programos tekstą asemblerio kalba, bei atlikti kitus specialius veiksmus.

Derinimo režimas nustatomas projekte. Tai atliekama taip:

1. Atidaromas meniu *Project* / *Settigs...* langas
2. Suaktyvinama kortelė *Fortran* (10.5 pav.)
3. Lauke *Debuging Level* parenkamas *Full*, *Partial* arba *Minimal* variantas.
4. Langas uždaromas OK mygtuku.

Jei kompiliuojama ir saistoma sėkmingai, sukuriamas *.exe failas su reikalinga derintojo informacija. Dabar belieka pažymėti kontrolines programos sustojimo vietas (breakpoints) ir vykdyti programą. Tai galima padaryti naudojantis įrankių juostos mygtukais bei atitinkamomis funkcinių klavišų kombinacijomis.

Kontrolinė programos sustojimo vieta pažymima taip:

1. Perkelti žymeklį į norimą programos eilutę.
2. Spustelėti pele mygtuką  arba paspausti klavišą *F9*.

Pažymėtos eilutės pradžioje atsiras spalvotas apskritimas. Pažymėjimas anuliuojamas atliekus tuos pačius veiksmus.

Programa derinimo režimu paleidžiama vykdyti meniu *Build* komanda *Start Debug*/*Go*, klavišu *F5* arba įrankių juostos mygtuku  . Ji bus vykdoma iki pirmos pažymėtos sustojimo vietos. Programai sustojus, atitinkamoje eilutėje atsiras sustojimo ženklas, o meniu skyrių *Build* pakeis skyrius *Debug* su įvairiomis derinimo procedūrų komandomis. Dabar galime patikrinti esamas kintamųjų reikšmes, jei reikia jas ir pakeisti. Atsiradusiame derintojo lange suaktyvinę kortelę *Locals* matysime visų lokaliųjų kintamųjų reikšmes. Pažymėto kintamojo reikšmę pamatysime *QuickWatch* lange spustelėdami pele mygtuką  , meniu *Debug* komanda *QuickWatch* arba klavišais *Shift+F9*.

Komanda *Go* ar klavišu *F5* programa vykdoma iki kitos sustojimo vietos. Iki eilutės su žymekliu programa vykdoma komanda *Step to cursor* arba klavišai *Ctrl+F10*. Atitinkamais įrankių mygtukais, meniu *Debug* komandomis arba funkciniais klavišais

galima įeiti į paprogramį (*Step Into* ar *F11*), išeiti iš jo (*Step Out ar Shift+F11*), peršokti (*Step Over ar F10*). Komanda *Restart* (*Ctrl+Shift+F5*) grįžtama į programos pradžią ir *Stop Debugging* (*Shift+F5*) nutraukiamas programos veikimas derinimo režimu.

Informaciją apie programos būseną galima gauti ir meniu *View* atidaromuose languose *Watch, Locals, Registers* ir *Output.* Lange *Watch,* nurodę kintamojo vardą, gausime jo reikšmę. Tai galima padaryti tiesiogiai - surinkti vardą klaviatūra ir paspausti Enter arba panaudoti tarpinę Windows atmintį - pažymėti norimą vardą ir komandomis *Copy* (*Ctrl+C*) ir *Paste* (*Ctrl+V*) perkelti jį į *Watch* langą. Lange *Memory* taip pat pateikiami kintamųjų adresai atmintyje, o *Registers* lange - procesoriaus registrų turinys. Komanda *Call Stack...* galima kontroliuoti paprogramių parametrų reikšmių perdavimą.

Platesnę ir išsamesnę informaciją apie konkrečios FORTRANo realizacijos programavimo aplinką, jos galimybes, programų kompiliavimo, saistymo parametrų valdymą skaitytojas gali rasti atitinkamuose dokumentuose bei plačioje HELP sistemoje, iškviečiamoje meniu *Help* komandomis. *Developer Studio* aplinkoje šią informaciją rasime suaktyvinę informacinio lango *InfoView* kortelę.

P R I E D A I

1 PRIEDAS. STANDARTINIŲ FUNKCIJŲ SĄRAŠAS

Priede pateiktas didžiosios dalies vidinių standartinių funkcijų (žr. 8.2 skyrių) sąrašas. Be funkcijos atliekamų veiksmų, taip pat nurodoma funkcijos klasė: elementinė, informacinė, masyvinė arba funkcija - standartinis paprogramis; ir leistini argumentų tipai. Dalis sudėtingų masyvinių, masyvais operuojančių ir reikalaujančių išsamesnio komentaro standartinių funkcijų aprašyta 8.4.5 skyriuje. Neaprašyta keletas sudėtingų atskirais duomenų bitais operuojančių funkcijų bei informacinių funkcijų tekstinio tipo duomenims.

Funkcijų klasės. Elementinė funkcija - tokia funkcija, kurios argumentai gali būti ir skaliarai, ir masyvai. Jei bent vienas argumentų yra masyvas, tai visi išėjimo argumentai turi būti tokios pat formos, kaip didžiausio rango argumentas-masyvas (žr. 8.4 skyrių). Informacinės funkcijos rezultatas priklauso tik nuo funkcijos argumentų savybių, bet ne nuo argumentų reikšmių. Masyvinės funkcijos įėjimo ir išėjimo argumentai dažniausiai yra masyvai. Dalis vidinių standartinių funkcijų apiformintos kaip paprogramiai-SUBROUTINE (žr. 7.3 skyrių). Jų klasę žymėsime žodžiu “SUBROUTINE”.

Argumentų tipus žymėsime tipo pirmąja raide. Jei argumentas gali būti kelių tipų, tai reiškia, kad funkcija - bendrinė (žr. 8.2 skyrių), ir jos rezultatas bus to pat tipo kaip ir argumentai. Nurodytas tipas apima visus leistinus šiam tipui ląstelės ilgius (žr. 3.6 ir 8.1 skyrius). Pavyzdžiui, funkcijos argumento tipas I reiškia, kad argumentas gali būti vienos iš rūšių I*1, I*2, I*4, I*8. Kai kurioms funkcijoms galima nurodyti ir nebūtiną pasirinktinį argumentą - argumento rūšį (tą visada žymėsime žodžiu “[kind]”).

Funkcija	***Funkcijos prasmė***	***Funkcijos klasė***	***Argumentų tipas***
ABS(x)	Modulis	Elementinė	I,R,C
ACHAR(n)	Eilės numerį n ASCII koduose atitinkantis simbolis	Elementinė	I
ACOS(x)	Arkkosinusas	Elementinė	R
ADJUSTL(s)	Paeilutėje s visus simbolius - neintervalus pastumia prie kairiojo paeilutės krašto	Elementinė	CH
ADJUSTR(s)	Tas pat, prie dešiniojo krašto	Elementinė	CH

AIMAG(z), IMAG(z)	Kompleksinio duomens z menamoji dalis; rezultato tipas - R	Elementinė	C
AINT(a, [kind])	Apvalina iki mažesnio (absoliu-čiuoju dydžiu) sveikojo duomens	Elementinė	a - R, kind - I
ALLOCATED(array)	Loginė funkcija: .T. - masyvui šiuo metu atmintis išskirta; .F. - ne	Informacinė	Dinaminis masyvas
ANINT(a, [kind])	Apvalina iki sveikojo duomens	Elementinė	a - R, kind - I
ASIN(x)	Arksinusas	Elementinė	R
ASSOCIATED(p, t)	Loginė funkcija: .T. - nuoroda p nutaikyta į taikinį t; .F. - ne	Informacinė	p - nuoroda, t - taikinys
ATAN(x)	Arktangentas	Elementinė	R
BIT_SIZE(i)	Bitų kiekis sveikajam kintamajam i	Informacinė	I
BTEST(i, k)	Reikšmė .T., jei kintamojo i k-ojo bito reikšmė yra “1”, ir .F. - jei”0”	Elementinė	I
CEILING(a)	Mažiausia iš didesnių arba lygių argumentui a sveikųjų reikšmių	Elementinė	R
CHAR(i, kind)	Eilės numerį i atitinkantis simbo-lis koduose, nurodomuose rūšies parametru kind	Elementinė	I
CMPLX(x [,y] [,kind])	Paverčia kind rūšies duomenis x ir y vienu C tipo duomeniu.	Elementinė	x - I, R, C y - I, R kind - I
CONJG(z)	Funkcijos reikšmė - C tipo duomuo (x, -y), jei argumento z reikšmė yra (x, y)	Elementinė	C
COS(x)	Kosinusas	Elementinė	R, C
COSH(x)	Hiperbolinis kosinusas	Elementinė	R

DATE_AND_ TIME(d,t,z,v)	Data ir laikas standartinėje formoje. Visi argumentai - išėjimo, pasirinktiniai	SUBROUTIN E	
DBLE(a)	Paverčia duomenį D tipo duome-niu	Elementinė	I, R, C
DIGITS(x)	Reikšminių skaitmenų kiekis duo-menyse tokio pat tipo kaip ir x	Informacinė	I, R
DIM(x, y)	Skirtumas x - y, jei teigiamas; antraip - 0	Elementinė	I, R; abu - vienodo tipo
DPROD(x, y)	DP tipo sandauga x*y	Elementinė	R*4
EPSILON(x)	Teigiama mažiausia reikšmė, kurią gali įgyti tokio kaip x tipo duomuo	Informacinė	R
EXP(x)	Eksponentė	Elementinė	R, C
FLOOR(x)	Didžiausia iš sveikųjų reikšmių, mažesnių arba lygių argumentui	Elementinė	R
HUGE(x)	Didžiausia reikšmė, kurią gali įgyti tokio pat kaip x tipo duomuo	Informacinė	I, R
IACHAR(c)	Simbolio c eilės numeris ASCII koduose	Elementinė	CH*1
IAND(i,j)	Loginė i ir j daugyba	Elementinė	I; abu - vie-nodos rūšies
IBCLR(i,k)	Duomens i k-ąjį bitą prilygina 0	Elementinė	I
IBITS(i,k,len)	Duomens i bitų nuo k-ojo iki k+len sekos sumos reikšmė	Elementinė	I
IBSET(i,k)	Analogiška IBCLR; prilygina 1	Elementinė	I
ICHAR(c)	Simbolio c eilės numeris koduose, atitinkančiuose c rūšį	Elementinė	CH*1
IEOR(i,j)	Loginė i ir j suma; reikšmių	Elementinė	I; abu - vie-

	kombi-nacijai .T. ir .T. sumos reikšmė .F.		nodos rūšies
INT(a [,kind])	a paverčia sveikojo tipo kind rūšies duomeniu	Elementinė	a - I, R, C; kind - I
IOR(i,j)	Loginė i ir j suma	Elementinė	I; abu - vienodos rūšies
KIND(x)	Duomens x rūšies parametro reikšmė	Informacinė	Bet kokio standartinio tipo
LEN(string)	CH tipo duomens string ilgis	Informacinė	CH
LEN_TRIM(string)	Tas pat, neįskaičiuojant galinių intervalų	Elementinė	CH
LOG(x)	Natūrinis logaritmas	Elementinė	R, C
LOG10(x)	Dešimtainis logaritmas	Elementinė	R
MAX(a1,a2,a3,...)	Maksimali reikšmė	Elementinė	Visi - pasirinktiniai; I,R
MAXEXPONENT (x)	Didžiausias laipsnis duomenims tokio tipo kaip x	Informacinė	R
MIN(a1,a2,a3, ...)	Minimali reikšmė	Elementinė	Visi - pasirinktiniai; I,R
MINEXPONENT (x)	Mažiausias laipsnis duomenims tokio tipo kaip x	Informacinė	R
MOD(a, p)	Reikšmė: a-INT(a/p)*p, jei $p \neq 0$; neapibrėžta, jei $p = 0$	Elementinė	I, R; abu - vienodo tipo
NEAREST(x, s)	Artimiausias procesoriaus skiriamas skaičius į teigiamą (teigiamam s) arba neigiamą (neigiamam s) pusę	Elementinė	R; $s \neq 0$
NINT(a [,kind])	Artimiausias sveikasis duomuo	Elementinė	a - R, kind - I
NOT(i)	Loginis neigimas kiekvienam i bitui	Elementinė	I

PRECISION(x)	Tokio tipo kaip x duomenų tikslu-mas reikšminių skaitmenų kiekiu	Informacinė	R, C
RANDOM_ NUMBER(x)	Atsitiktinė reikšmė iš intervalo 0<x<1	SUBROUTIN E	R
RANGE(x)	x tipo duomenų kitimo intervalas dešimties laipsnio rodikliu	Informacinė	I, R, C
REAL(a[,kind])	Paverčia a realiuoju kind rūšies duomeniu	Elementinė	a - I, R, C; kind - I
REPEAT(string,n)	Atlieka n string kopijų konkatena-ciją	Masyvinė	CH
SCALE(x,i)	x daugina iš dviejų laipsniu i	Elementinė	x - R, i - I
SELECTED_ INT_KIND(r)	Nustato rūšies parametrą duome-nims iš intervalo -1Er< <1Er	Informacinė	I
SELECTED_ REAL_KIND(p, r)	Nustato rūšies parametrą duome-nims iš intervalo -1.Er< <1.Er ir turintiems tikslumą p reikšminių skaitmenų	Informacinė	R
SIGN(a, b)	a absoliutiniu dydžiu kart sign b	Elementinė	I, R
SIN(x)	Sinusas	Elementinė	R, C
SINH(x)	Hiperbolinis sinusas	Elementinė	R
SQRT(x)	Kvadratinė šaknis	Elementinė	R, C
SYSTEM_ CLOCK(c,r,m)	Procesoriaus laikrodžio laikas sveikaisiais duomenimis	SUBROUTIN E	I
TAN(x)	Tangentas	Elementinė	R
TANH(x)	Hiperbolinis tangentas	Elementinė	R
TINY(x)	Mažiausia teigiama reikšmė, kurią gali įgauti tokio tipo kaip x duome-nys	Informacinė	R
TRIM(string)	Galiniai string intervalai pašalinami	Masyvinė	CH
VERIFY(string,	Jei back yra .F. arba praleistas -	Elementinė	string - CH,

set [,back])	rezultatas yra kairiausio string simbolio, nesančio aibėje set, pozicijos numeris; rezultatas - I tipo. Jei back yra, ir jo reikšmė yra .T. - rezultatas yra dešiniausio string simbolio, nesančio aibėje set, pozicijos numeris; I tipo. Jei kiekvienas string simbolis yra aibėje set, arba string ilgis yra nulinis - rezultato reikšmė yra I tipo 0		set - CH, back - L

2 PRIEDAS. KAI KURIOS MS FORTRAN POWERSTATION YPATYBĖS

Microsoft FORTRAN PowerStation kompiliatoriuje nėra tokių į FORTRAN 90 standartą įtrauktų operatorių:

- visų su modulinėmis subprogramomis susijusių operatorių: MODULE, MODULE PROCEDURE, CONTAINS, USE; subprogramose naudotinų atributų (arba savarankiškų operatorių) INTENT, OPTIONAL, PRIVATE, PUBLIC
- išvestinių duomenų deklaravimo operatoriaus TYPE ... END TYPE bei operatoriaus SEQUENCE

Pastaba. Šiuos operatorius pagal prasmę atitinka kompiliatoriaus operatoriai STRUCTURE ... END STRUCTURE; išvestinio duomens komponentams skirti juose naudojamas taškas "."

- sąsajos bloko operatorių INTERFACE ... END INTERFACE.

Pastaba. Šiuos operatorius pagal prasmę atitinka kompiliatoriaus operatoriai INTERFACE TO ... END

- masyvinių standartinių funkcijų bei pilnais masyvais operuojančių operatorių WHERE ir ELSEWHERE
- nuorodų bei taikinių konstrukcijų ir atitinkamų operatorių POINTER, TARGET, NULLIFY

Microsoft FORTRAN PowerStation parengta programa ne jo aplinkoje dirba tik su DOS išplėtimo programa, t.y. aktyviame kataloge turi būti failas DOSXNT.EXE.

Microsoft FORTRAN PowerStation kompiliatoriuje yra tokie į FORTRAN 90 standartą neįtraukti operatoriai:

- su išvestiniais duomenimis susiję operatoriai STRUCTURE ... END STRUCTURE, MAP ... END MAP, UNION ... END UNION, RECORD
- kintamųjų saugojimo būdo apibrėžimo operatorius AUTOMATIC
- duomenų apsaugos operatorius LOCKING
- tekstinių failų įterpimo operatorius INCLUDE
- sąsajos operatorius INTERFACE TO ... END

Šie FPS neatitikimai FORTRAN 90 standartui yra įtraukti ir į Digital Visual Fortran kompiliatorių, todėl su FPS parašytas programas galima naudoti DVF aplinkoje.

3 PRIEDAS. DIGITAL VISUAL FORTRAN TIESIOGINIO DUOMENŲ ĮVEDIMO IŠ KLAVIATŪROS IR FAILŲ SISTEMOS VALDYMO PROCEDŪROS

Digital Visual Fortran turi grupę procedūrų, kurių dėka galime įvesti duomenis klaviatūra tiesiogiai, o taip pat atlikti įvairius veiksmus su failų sistema programos vykdymo metu operacinės sistemos lygmeniu, nenaudodami FORTRANo Į/I sistemos operatorių. Jos leidžia kurti, pervardinti, ištrinti failus bei katalogus, nustatyti loginius diskus ir pan. Šios procedūros priklauso vadinamajai "run-time" procedūrų grupei, jos yra patalpintos DFPORT, DFLIB ir kt. moduliuose. Jei programoje naudojamos šiame skyrelyje aptariamos procedūros, reikia nurodyti atitinkamą modulį operatoriumi USE:

```
USE DFPORT
USE DFLIB  ir t.t.
```

Tiesiogiai duomenims įvesti klaviatūra skirtos 3 procedūros (modulis DFLIB):

GETCHARQQ(). Pateikia paspausto klavišo reikšmę

Funkcijos reikšmė yra character*1 tipo. Jei paspaustas klavišas nėra simbolis, o funkcijos ar krypties klavišas, funkcijos reikšmė yra šešioliktainis 00 arba E0.

GETSTRQQ(*buffer*) Pateikia įvedamų klaviatūra simbolių seką

buffer - character*(*) tipo argumentas, išreiškiantis įvestų simbolių seką. Grąžinama funkcijos I*4 reikšmė nusako įvesto teksto ilgį. Teksto įvedimas užbaigiamas klavišu Enter arba Return.

PEEKCHARQQ() Tikrina, ar buvo paspaustas bet koks klavišas

Funkcijos reikšmė yra logical*4 tipo. Jei klavišas buvo paspaustas, jos reikšmė .TRUE. Jei norime sužinoti, koks konkrečiai buvo paspaustas klavišas, turime naudoti funkciją GETCHARQQ.

Duomenų įvesties/išvesties operacijas galima atlikti funkcijomis (modulis DFPORT) :

FGETC (*unit, char*) Perskaito simbolį iš išorinio failo

unit – integer*4 tipo įėjimo argumentas, išreiškiantis susieto su išoriniu failu kanalo numerį;

char – character*1 tipo išėjimo argumentas su perskaityto simbolio reikšme.

Sugrąžinama funkcijos reikšmė integer*4 tipo lygi 0, jei funkcija įvykdyta sėkmingai, -1, jei aptikta failo pabaiga.

FPUTC(*unit, char*) Užrašo simbolį į išorinį failą

Funkcijos argumentai analogiški FGETC funkcijos argumentams.

GETC(*char*) Argumentui *char* priskiria paspausto klavišo reikšmę

PUTC(*char*) Išveda argumentą *char* į standartinį išvesties įrenginį

Veiksmams su failais ir katalogais skirtos šios procedūros (modulis DFLIB):

CHANGEDIRQQ(*dir*) Nurodytą katalogą padaro aktyvų

dir - character*(*)- reikalingo padaryti aktyviu katalogo vardas. Sugrąžina logical*4 reikšmę .TRUE., jei funkcija įvykdyta sėkmingai, ir reikšmę .FALSE. priešingu atveju.

CHANGEDRIVEQQ(*drive*) Nurodytą loginį diską padaro aktyvų

drive - character*(*)- reikalingo padaryti aktyviu loginio disko vardas. Sugrąžina logical*4 reikšmę .TRUE., jei funkcija įvykdyta sėkmingai, ir reikšmę .FALSE. priešingu atveju.

DELFILESQQ(*files*) Pašalinina failą iš disko

files - character*(*) tipo argumentas, identifikuojantis šalinamus failus. Galima naudoti failų šablonų simbolius ? ir *, jei norime pašalinti daugiau nei vieną failą. Sugrąžinama funkcijos integer*2 tipo reikšmė parodo pašalintų failų skaičių. Katalogai ir sisteminio, paslėpto ar tik skaitymo požymius turintys failai nešalinami;

FINDFILEQQ(*filename, varname, pathbuf*) Ieško nurodyto vardo failą

filename - character*(*) - ieškomo failo vardas.
varname - character*(*) - kintamojo vardas, nurodantis paieškos kelią.
pathbuf - character*(*) - pilnas surasto failo kelias.
Sugrąžinama funkcijos reikšmė integer*4 lygi surasto failo kelio teksto ilgiui kintamajame *pathbuf* arba lygi nuliui, jei failas nerastas.

GETFILEINFOQQ(*files, buffer, handle*) Pateikia informacijos apie failą

files - character*(*) - failo ar jų grupės vardas (galima nurodyti visą kelią bei šabloną simboliais “*” ir “?”).

buffer - struktūros tipo kintamasis, pateikiantis informaciją apie failą.

handle - integer*4 valdymo parametras. Jei norime gauti informaciją apie visus šablonu *files* nurodytus failus, *handle* reikia suteikti reikšmę FILE$FIRST ir kviesti funkciją cikle, kol *handle* bus lygus FILE$LAST arba FILE$ERROR.

GETDRIVESIZEQQ(*drive, total, avail*) Pateikia diskų įrenginio talpą

drive - character*(*) - diskų įrenginio vardas. Reikalinga tik pirma raidė. Gali būti tiek mažoji, tiek ir didžioji raidė. Galima naudoti konstantą FILE$CURDRIVE, jei norime gauti informaciją apie aktyvų loginį diską.

total - integer*4 - iš viso talpa baitais.

avail - integer*4 - laisvos talpos kiekis baitais.

Grąžinama funkcijos logical*4 reikšmė .TRUE., jei funkcija įvykdyta sėkmingai.

GETDRIVESQQ() Pateikia esamų loginių diskų sąrašą

Sąrašas pateikiamas grąžinama character*26 tipo reikšme, turinčia savyje esamų loginių diskų vardų raides. Pavyzdžiui, jei kompiuteryje yra A, D ir C diskai, tai šios funkcijos grąžinama reikšmė bus ‘A C D ‘.

MAKEDIRQQ(*dirname*) Sukuria nurodytą katalogą

dirname - character*(*) - norimo sukurti katalogo vardas. Grąžinama logical*4 reikšmė lygi .TRUE., jei katalogas sukuriamas, ir .FALSE., priešingu atveju.

Yra dar keletas funkcijų ir paprogramių, skirtų failams valdyti operacinės sistemos lygmeniu, apie kuriuos informaciją galima rasti specialioje literatūroje ar Help sistemose.

Detalų bet kokios procedūros aprašymą bei panaudojimo pavyzdžius skaitytojas gali gauti Developer Studio aplinkoje. Tam reikia atlikti tokius veiksmus:

1. Meniu *Help*/*Search* lange suaktyvinti kortelę *Index*
2. Lauke *Type in the keyword to find* užrašyti procedūros vardą
3. Dvigubu pelės kairiojo klavišo spragtelėjimu pažymėti šį vardą lange *Select keyword to list related topics*
4. Spragtelėti mygtuką *Display*.

Ekrane atsiras langas su pilnu nurodytos procedūros aprašymu ir panaudojimo pavyzdžiu.

Digital Visual Fortran aplinka turi žymiai daugiau “run-time” procedūrų, nei paminėta šiame priede. Daugiau bei naujesnės informacijos galite rasti įvade nurodytuose WWW puslapiuose. Pilnesnis šių procedūrų sąrašas yra VGTU skaičiavimo centro puslapyje www.vtu.lt/sc

L I T E R A T Ū R A

1. W.S. Brainerd, et. al *A Programmer's Guide to Fortran 90*. McGraw-Hill, 1990. 410 p.
2. J.Kerrigan *Migrating to Fortran 90*. O'Reilly&Associates, Inc., 1993. 361 p.
3. Br.D.Hahn. *Fortran 90 for Scientists and Engineers*. Edward Arnold, 1994. 351 p.
4. *Mocrosoft FORTRAN*. Developmen System. Version 5.1. Microsoft Corporation, 1993. 534 p.
5. *XL Fortran for AIX. Language Reference. Version 3 Release* 2. International Business Machines Corporation, 1994. 489 p.
6. Соловьев П.В. *Fortran для персонального компьютера*. М.: Арист, 1991. 223 с.
7. Самохин А.Б. Самохина А.С. *Численные методы и программирование на Фортране для персонального компьютера.* М.: Радио и связь, 1996, 224 с.
8. *DIGITAL Fortran. Language Reference Manual.* Digital Equipment Corporation, Maynard, Massachusetts, 1997
9. *Фортран 90*. Международный стандарт / Перевод с англ. С.Г. Дробышевич. Редактор перевода А.М. Горелик. М.: „Финансы и статистика“, 1998. 416 с.