

Ivadas

Knygos pagrindu yra pamokos, kurios buvo skirtos VDU informatikos fakultete šeštadieniniuose kursuose mokiniams, o vėliau KTU JKM programavimo specialybės mokiniams. Medžiaga pateikiama sudėtingumo seka, tačiau skaitytojui nebūtina atlikti visas užduotis pilnumoje, norint suprasti kito skyrelio medžiagą. Pakanka išsiaiškinti esančių pratimų programas ir suvokti naudojamas priemones. Įvairaus sudėtingumo užduotys skirtos savarankiškam darbui, kas leidžia įsitikinti, kaip pasisekė suprasti skyrelyje pristatomas programavimo priemones. Knygoje Turbo Paskalio konstrukcijos ir priemonės pristatomos vartotojo požiūriu. Plačiau apie jas reikia skaityti kalbos aprašymui skirtose knygose.

Programavimo esmę sudaro gebėjimas vartoti duomenų struktūras ir mokėjimas pasinaudoti duomenų apdorojimo algoritmais. Visą tai stengėmės demonstruoti nestandartinėmis užduotimis, panaudojant ekrano spalvas ir garsus. Devintame skyriuje pristatome Turbo Paskalio grafikos priemones. Šį skyrių galima skaityti ir suprasti įsisavinus pirmųjų keturių skyrių medžiagą. Siūlome visų skyrių pratimus pertvarkyti darbui grafiniame ekrane.

Paskutinis skyrius skirtas grafikos pavyzdžiams.

Diskelyje rasite visų skyrių programas. Jos sudėtos kataloguose pagal skyrius.

Tikimės, kad knyga pasinaudos visi, kuriems įdomu ir žingeidu patiems išmokti rašyti programas.

- Siūlomas pavadinimas **“Programavimas su Turbo Paskaliu”** netinkamas, nes knygoje:
 - nenagrinėjamos programavimo problemos;
 - neskiriama pakankamai dėmesio algoritmų sudarymui ir duomenų apdorojimui;
 - nenagrinėjamas Turbo Paskalis kaip programavimo kalba ir aplinka.
- Knyga skirta praktiniams pradinių programavimo įgūdžių formavimui ir pažinčiai su programavimo elementais.
- Bendrojo lavinimo mokykloms skirtoje literatūroje žodis *“praktikumas”* pavadinimuose dažnas.
- Siūlome viršelyje pavadinimo fone išsaugoti frazę **“Turbo Paskalis 7.0”**. Gal tai galėtų būti, pavyzdžiui, viršelio kairiame šone vertikaliai ši frazė kelis kartus atkartota. Pirkėjas atsitiktinai jau tikrai neapsiriks.
- Tinkamas pavadinimas **“Programavimo Turbo Paskaliu 7.0 praktikumas”**.

Jūsų pasiūlytas tekstas ant viršelio daugiau žadantis, negu pirkėjas ras knygoje. Gal tiktų toks?

Kompiuterį sudaro dvi tarpusavyje susietos dalys: techninė įranga ir programinė įranga.

Kaip parašyti programą?

Ką daryti, kad kompiuteris vykdytų programą?

Kaip įsitikinti, kad programa dirba gerai?

Ar programa daro tai, kas reikalinga jos vartotojui?

Šioje knygelėje praktiniais programų pavyzdžiais perteikiami programų rašymo elementai. Tam panaudotas Turbo Paskalis 7.0.

Cia rasite:

- Turbo Paskalio teksto redaktoriaus ir programavimo terpės teikiamų pagrindinių galimybių aprašymą;
- Žinias apie programavimo kalbos duomenų struktūras ir veiksmus su duomenimis jose;
- Sampratą apie algoritmus ir jų kūrimą;
- Programų rašymą, derinimą, vykdymą.

Daug dėmesio skirta Turbo Paskalio pagalbinių bibliotekų teikiamų paslaugų panaudojimui. Tai:

- ✓ Spalvos, garsai, langai, meniu organizavimo būdai;
- ✓ Grafikos elementai: taškai, linijos, figūros, raštai, langai, tekstas;
- ✓ Grafinių paveikslėlių sudarymas, judesio imitacija.

Programavimo praktikumas (Turbo Paskalis 7.0)

J.Blonskis, V.Bukšnaitis, J.Smolinskas, A.Vidžiūnas

1. Programų kūrimo procesas (AV)

- 1.1. Uždavinio parengimas kompiuteriniam sprendimui
- 1.2. Paskalio kalbos alfabetas ir gramatika
- 1.3. Programos struktūra
- 1.4. Programos teksto paruošimas redaktoriumi
- 1.5. Programos transliavimas ir tikrinimas

2. Programavimo abėcėlė (VB)

- 2.1. Vartotojo ir programos dialogas.
- 2.2. Valdančios struktūros.
- 2.3. Sumos, kiekio, sandaugos skaičiavimas. 3 tema
- 2.4. Didžiausios ir mažiausios reikšmės paieška. 4 tema.
- 2.5. Programos struktūriniai elementai (pažintis su procedūra). 12 tema
- 2.6. Užduotys.

3. Skaičiai ir simboliai (JB)

- 3.1. Spalvos, ekrano langai, data, laikas.
- 3.2. Skaičių srautų analizės uždaviniai.
- 3.3. Tekstiniai failai.
- 3.4. Sveikųjų skaičių analizė.
- 3.5. Simbolinis duomenų tipas.
- 3.6. Atsitiktinių situacijų modeliavimas.
- 3.7. Valdantys simboliai.
- 3.8. Užduotys.

4. Skaičių srautai. (VB)

- Pirmoji pažintis su masyvu. 13 tema.
- Funkcijos reikšmių lentelė. 14 tema.
- Procedūros. 15 tema.
- Funkcijos. 16 tema.
- Funkcijos reikšmių lentelių šeima. 17 tema.
- Vienmatis masyvas. 18 tema.
- Užduotys.

5. Teksto analizė ir tvarkymas (JB)

- 5.1. Simbolinio tipo duomenys.
- 5.2. Aibės duomenų tipas.

5.3. Eilutės duomenų tipas

5.4. Užduotys.

6. Duomenų organizavimas (AV)

Dvimatis masyvas ir struktūrinės konstantos. 23 tema.

Rezultatų kaupimas ir grupavimas masyvuose. 30 tema.

Įrašo duomenų tipas. 24 tema

Tekstinis failas kaip duomenų šaltinis. 27 tema.

Užduotys.

7. Turbo Paskalio bibliotekos (AV)

Spalvų keitimas ir žymeklio valdymas.

Diskų analizės priemonės. 31 tema

Kompiuterinis laikrodis.

Programos struktūrizavimas (bibliotekos). 28 tema.

Pavyzdžiai.

Užduotys.

8. Duomenų tvarkymas (VB)

Duomenų paieška. 26 tema

Duomenų rikiavimas. 25 tema

Duomenų grupavimas. 26 tema

Duomenų peržiūra ekrane.

Užduotys.

9. Ekraninė grafika (JB)

9.1. Grafinio ekrano tvarkyklė

9.2. Ekraninės grafikos elementai.

9.3. Funkcijų grafikai.

9.4. Judesio programavimas.

9.5. Paletė.

9.6. Užduotys.

10. Įvairūs uždaviniai (visi)

Priedas

Turbo Paskalio aplinkos valdymo priemonės. AV

Kodų lentelės.

Literatūra

- 👉 Pastaba
- 👉 Tai svarbu
- 💻 Programos tekstas, kuris yra diskelyje (?); pratimas
- ✧ išvardinimas, vietoje tradicinio storo taško.
- Šiame skyriuje: išvardinimui
- ✓ programos modifikacijos: vietoje tradicinio storo taško

1. Programų kūrimo procesas

1.1. Uždavinio parengimas kompiuteriniam sprendimui



Kompiuteriai automatiškai gali spręsti tik tuos uždavinius, kurių sprendimo aprašymai saugomi jų atmintyje. Tokie aprašymai vadinami kompiuterių programomis, o jų parengimas – programavimu. Programavimą komplikuoja tai, kad kompiuterio valdymo programos turi būti sudaromos jo vidine (mašinine) kalba, kurioje visi nurodymai aprašomi skaitmeniniais kodais. Tačiau žmogui šie kodai yra nepatogūs ir sunkiai įsimenami. Be to, vidine kompiuterio kalba turi būti aprašomas ne pats sprendžiamas uždavinys, o jo sprendimui vartojamų kompiuterio įrenginių valdymas. Todėl programavimą stengiamasi palengvinti, automatizuoti, pavedant atskirus programų rengimo etapus pačiam kompiuteriui. Automatizuotų programavimo sistemų pagrindą sudaro specialios programavimo kalbos, kuriomis aprašomas ne kompiuterio įrenginių valdymas, o tie informaciniai procesai, kuriuos reikia atlikti.

Tarkime, kad jau turime pasirinkę programavimo kalbą ir užrašėme šia kalba mums rūpimo uždavinio programos tekstą. Ką reikia dar padaryti, kad kompiuteris galėtų vykdyti parašytą programą? Kokius dar reikia atlikti programos parengimo darbus, kaip organizuoti jos vertimą (transliavimą) į kompiuterio vidinę kalbą, klaidų paiešką, taisymą, programos vykdymą, saugojimą ir kitas programos eksploatavimo operacijas?

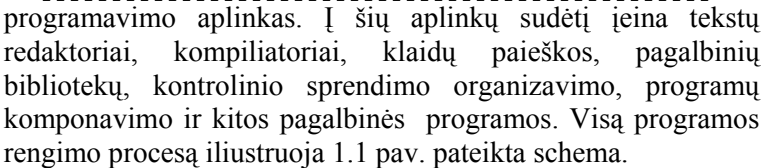
Universalaus atsakymo į šiuos klausimus nėra, nes nurodytų veiksmų turinys, jų organizavimo ir valdymo metodai priklauso nuo kompiuteryje vartojamos informacijos apdorojimo technologijos, nuo ryšio su vartotoju būdo. Vartotojui patogiausias dialoginis ryšys, kai instrukcijos kompiuteriui renkamos klaviatūra, o dialogo eiga stebima ekrane. Naudojant tokį ryšį, įvedamas į kompiuterį programos tekstas matomas ekrane, kur gali būti patikrinamas, redaguojamas ir po to

persiunčiamas į kompiuterio atmintį. Šį darbą atlieka tekstų redaktoriumi vadinama speciali programa. Programavimo kalba užrašytas ir kompiuterio atmintyje arba kokioje nors kitoje mašininiame skaitymui pritaikytoje aplinkoje saugomas programos tekstas yra vadinamas pirminiu programos modulių.

Kompiuterio atmintyje patalpintas pirminis modulis dar turi būti transliuojamas, verčiamas į kompiuterio vidinę kalbą. Vertimas gali būti atliekamas įvairiai. Paprasčiausia yra interpretavimo technologija, kai nuosekliai nagrinėjamos ir iš karto vykdomos atskiros pirminiame modulyje surašytos instrukcijos. Taip labai lengva kontroliuoti transliavimo procesą ir ieškoti klaidų, tačiau interpretavimas turi daug trūkumų. Blogiausia tai, kad interpretatoriai neparengia visos programos teksto mašinos kalba. Todėl kiekvieną kartą, kai tik norima vykdyti programą, ji turi būti interpretuojama iš naujo.

Tobulesnės, bet ir sudėtingesnės, yra kompiliuojančios sistemos, kuriose programų transliavimo ir vykdymo procesai griežtai atskirti. Čia iš pradžių visa programa parengiama vidine kompiuterio kalba ir tik po to vykdoma. Taip parengta programa gali būti daugelį kartų vykdoma nebenaudojant transliatoriaus, o tik iškviečiant ją iš saugojimo vietos kompiuterio išorinėje atmintyje paprastomis operacinės sistemos valdymo priemonėmis.

Programos transliavimo ir vykdymo procesų atskyrimas sudaro ir tam tikrų problemų. Pavyzdžiui, jeigu, vykdamas programą, pastebima klaida, sunkiau surasti jos vietą pradiniam tekste. Todėl kompiliuojančiose sistemose reikalingos įvairios pagalbinės programos, skirtos klaidų paieškai ir kitiems programų derinimo darbams atlikti. Pats kompiliavimas taip pat dažniausiai vykdomas ne iš karto, o atskirais etapais, žingsniais, kuriuos valdo atskiros programos. Pavyzdžiui, pirmo kompiliavimo žingsnio metu paprastai yra parengiamas ne vidinis programos kodas, o tarpinė jos forma, vadinama objektiniu modulių, kuri palengvina programų komponavimą iš atskirų, nepriklausomai parengtų dalių. Tokį komponavimą valdo dar viena speciali aptarnaujanti programa, vadinama sisteminių ryšių redaktoriumi.



1.1 pav. Programos tvarkymo proceso schema

Aprašytos pagalbinės programavimo aplinkos programos valdomos arba bendros paskirties operacinės sistemos komandomis arba specialiomis aplinkų valdymo programomis. Pastaruoju atveju sakoma, kad programavimo aplinka yra integruota. Populiariausios yra integruotos aplinkos su meniu tipo valdymo sistemomis. Tokio tipo integruotos aplinkos pavyzdys yra Turbo Pascal 7.0 aplinka, kuri pritaikyta programoms rengti Paskalio kalbos modifikacija (versija) Turbo Paskalis 7.0. Pagrindiniai šios aplinkos naudojimo principai aprašyti 1.4 skyrelyje.

1.2. Paskalio kalbos alfabetas ir gramatika

Paskalio kalba, kaip ir kiekviena kita kalba, turi savo alfabetą ir gramatiką. Alfabetas nurodo kalboje vartojamus simbolius, o gramatika – kaip iš šių simbolių sudaromos kitos, sudėtingesnės kalbos struktūros. Kalboje vartojamus simbolius galima suskirstyti į tris pagrindines grupes:

- raides, kurios apima didžiąsias ir mažąsias lotyniškas raides nuo A iki Z (ir nuo a iki z) ir pabraukimo simbolį _;
- skaitmeninius simbolius nuo 0 iki 9;
- specialius simbolius, kuriems priklauso aritmetinių operacijų, skyrybos ženklai ir kiti griežtai apibrėžtos paskirties simboliai.

Aprašant programos ryšį su išore, įvedamus ir išvedamus duomenis bei komentaruose raidžių sąrašą galima praplėsti, vartojant visus kompiuterio klaviatūroje esančius ir jo įrenginių atvaizduojamus simbolius. Šie simboliai sudaro kompiuterio alfabetą. Tai labai svarbu nelotynišką alfabetą vartojančioms šalims, tarp kurių yra ir Lietuva (mes vartojame išplėstą lotynišką alfabetą).

Siekiant, kad knygoje pateikti pavyzdžiai būtų lengviau analizuojami ir tikrinami kompiuteriuose, juose aprašomus programų objektus stengiasi nurodyti lietuviškais vardais, pakeičiant draudžiamas lietuviškas raides panašiomis lotyniškoms. Dėl to kai kurie lietuviški vardai knygoje pateikiamuose programų tekstuose yra iškraipyti.

Turbo Paskalio kalbos gramatikos taisyklės galima suskaidyti į sintaksę ir semantiką aprašančius rinkinius. Sintaksė aprašo kalbos struktūrų rašymo taisykles, o semantika nurodo, kaip kompiuteris supranta ir vykdo sintaksinės struktūrose aprašomas instrukcijas.

Paprastiausios sintaksinės struktūros yra žodžiais arba leksemomis vadinami simbolių rinkiniai, kurių negalima suskaidyti į smulkesnes prasmines (semantines) konstrukcijas. Žodžiai gali nurodyti įvairių kalbos ir programos objektų vardus, instrukcijų kompiuteriui pavadinimus, skaičius ir kitokių duomenų tipų reikšmes. Kai kurie Turbo Paskalio

žodžiai visuomet turi griežtai apibrėžtą prasmę ir yra vadinami baziniais. Jie, kaip ir specialūs simboliai, yra skirti įvairiems pažymėjimo tikslams (priedas A). Šiam tikslui vartojami trumpi anglų kalbos žodeliai (pavyzdžiui: *begin, end, case, if, while, repeat, do*), kurių prasmė anglų kalboje artima jų paskirčiai programose. Tačiau tai nereiškia, kad, rašant programas, būtina mokėti anglų kalbą. Bazinius žodžius galima laikyti tiesiog specialios paskirties simboliais, skirtais įvairiems žymėjimo tikslams. Žinant bazinių žodžių prasmę anglų kalboje, yra tik lengviau juos įsiminti, lengviau rašyti ir analizuoti programas.

Baziniai žodžiai Turbo Paskalyje yra rezervuoti. Tai reiškia, kad kalboje nenumatytiems žymėjimo tikslams juos vartoti draudžiama. Tokius žodžius knygų tekstuose priimta išskirti pariebintu arba pasvirusiu šriftu. Ranka rašytuose tekstuose juos rekomenduojama pabraukti.

Pagal savo paskirtį į bazinius žodžius yra labai panaši dar viena žodžių grupė, kuri vadinama standartiniais vardais (identifikatoriais). Tai taip pat kalboje apibrėžti ir specialius objektus žymintys vardai, tačiau jiems negalioja draudimas apibrėžti jų prasmę iš naujo. Bet taip daryti nerekomenduojama, nes tada yra prarandama jų pagrindinė prasmė. Standartinių vardų pavyzdžiai: *Read, WriteLn, integer, real, string*.

Programose vartotojas taip pat gali projektuoti įvairius savus objektus ir suteikti jiems vardus. Visi vardai būtinai turi prasidėti raide ir gali būti sudaromi tik iš pagrindinio alfabeto raidinių bei skaitmeninių simbolių. Varduose negali būti nei tarpų, nei specialių simbolių.

Taisyklingų vardų pavyzdžiai: *pavyzdys, prog1, mano_programa, PirmaSuma*.

Netaisyklingi vardai: *mano programa, A+B, 2prog*.

Turbo Paskalyje vardų ilgiai gali siekti iki 127 simbolių. Taip pat reikia atsiminti, kad varduose didžiųjų ir mažųjų raidžių prasmės yra vienodos. Todėl vardai *ManoAlga* ir *manoalga* yra identiški. Pirmajame variante – *ManoAlga* didžiosios raidės vartojamos pabrėžiant, kad vardas sudarytas iš

dviejų žodžių. Tai labai plačiai vartojamas programavimo Turbo Paskalio kalba technikos elementas.

Įvairūs nurodymai kompiuteriui programose yra aprašomi Paskalio kalbos sakiniais, kurie gali turėti įvairias funkcines paskirtis. Pavyzdžiui, jei sakiniais aprašomi konkretūs kompiuterio atliekami veiksmai, jie vadinami operatoriais, o jeigu jais aprašomos programose vartojamos struktūros, sakiniai vadinami aprašais. Yra ir daugiau funkcinių sakinių grupių, skirtų komentarams, programų antraštėms ir kitiems jų struktūriniais elementams aprašyti. Sakiniuose lekšemos atskiriamos tarpais, skyrybos ir operacijų ženklais bei kitokiais specialiais simboliais.

1.3. Programos struktūra

Tekstą programavimo kalba galima pavadinti programa tik tada, kai jis aprašo konkretaus uždavinio sprendimo algoritmą ir yra sudarytas laikantis kalbos rašybos (sintaksės) taisyklių. Rašybos taisyklės nusako tiek atskirų žodžių, tiek sakinių, tiek ir visos programos struktūrą. Paskalio kalba galima sudaryti labai sudėtingos struktūros programas, tačiau jos visuomet turi turėti šiuos pagrindinius elementus – antraštę, aprašus, pagrindinę dalį ir pabaigos žymę, pavyzdžiui:

```
program Pavyzdys_1_1; { Antraštė }
var
    { Kintamųjų aprašai }
    x, y : real;
begin
    { Duomenų išvedimas į ekraną }

    WriteLn ('Sveiki! Jūsų programa dirba!');
    WriteLn ('Įvesk skaičių:');
    ReadLn (x);          { Įvedimas }
    y := x*x;            { Skaičiavimas }
                        { Rezultatų išvedimas }
    WriteLn ('Kvadratas: ', y: 10: 2);
                        { Laukti Enter paspaudimo }
    ReadLn
end.                    { Pabaigos sakiny s }
```

Šioje programoje aprašyta, kaip kompiuteris kelia kvadratu standartiniame įvedimo įrenginyje (personaliniam kompiuteriui tai klaviatūra) perskaitytą skaičių. Skliaustuose { } įrašyti komentarai skirti paaiškinimams ir jokios įtakos programos darbui neturi, todėl juose galima vartoti visus kompiuterio alfabeto simbolius. Transliavimo metu komentarai yra tiesiog praleidžiami. Komentarų ribas taip pat galima nurodyti simbolių poromis (* ir *).

Kiekviena programa pradedama antraštės sakiniu, kurio struktūra (sintaksė) gali būti aprašoma taip:

```
program <Vardas> [( <Parametrai> )];
```

Aprašydami šią ir kitas sintaksines struktūras simboliais < > žymėsime paaiškinimus, įrašytus ne Paskalio kalba, o [] skliaustais nurodysime tas kalbos konstrukcijas, kurios gali būti praleidžiamos.

Antraštės sakinio struktūros aprašymas nurodo, kad jis programai suteikia vardą ir gali nurodyti parametrus. Parametrus nurodyti būtina tik senuose Paskalio kalbos variantuose, todėl plačiau jų paskirties šioje knygoje nenagrinėsime.

Visas programos tekstas Turbo Paskalio kalba sudaromas pagal tokią schemą:

```
< Programos antraštė >  
< Aprašai >  
< Pagrindinė dalis >
```

Už antraštės sakinio programose sudaroma aprašų dalis, kurioje išvardijami ir apibūdinami programoje vartojami duomenų tipai, kintamieji, procedūros bei kiti elementai. Skirtingų tipų elementų grupių aprašai pateikiami atskirose aprašų sekcijose, kurios pažymimos baziniais žodžiais. Pateiktame pavyzdyje yra tikrai viena kintamųjų aprašų sekcija, kurią žymi bazinis žodis *var*. Šioje sekcijoje būtinai turi būti aprašyti visi programoje vartojami kintamieji, nurodant jų vardus ir tipus.

Kintamuosius aprašančių sakinių (aprašų) struktūra yra tokia:

<Vardų sąrašas>: <Tipas>

Vardų sąrašas apraše sudaromas iš kableliais atskirtų vienodo tipo kintamųjų vardų, o tipas aprašomas standartiniu arba vartotojo apibrėžtu tipo vardu. Pavyzdžio programoje standartinis tipas *real* nurodo, kad kintamųjų x ir y reikšmės gali turėti trupmenines dalis. Kitas labai dažnai vartojamas standartinis duomenų tipas, kuris skirtas tik sveikiesiems skaičiams, yra *integer*.

Pagrindinėje programos dalyje, kurios pradžią žymi bazinis žodis *begin*, aprašomi visi darbai, kuriuos kompiuteris turi atlikti. Ši dalis sudaroma taip pat laikantis tam tikrų reikalavimų. Iš pradžių aprašoma, iš kur gaunamos programos argumentų reikšmės. Po to nurodoma, kokius veiksmus su šiais argumentais reikia atlikti ir kaip pavaizduoti bei išsaugoti programos darbo rezultatus. Visų šių operacijų turinys ir aprašymo būdas priklauso nuo konkretaus sprendžiamo uždavinio pobūdžio.

Pavyzdžiui, personalinių kompiuterių pradiniai duomenys dažniausiai įvedami klaviatūra, o rezultatai parodomi ekrane. Didžiųjų kompiuterių pradiniai duomenys dažniau skaitomi iš mašininių laikmenų, o rezultatai spausdinami.

Duomenų apdorojimo procesai aprašomi priskyrimo operatoriais, kurių struktūra:

<Kintamojo vardas>:= <Išraiška>

Čia simbolis := žymi patį priskyrimo veiksmą, o išraiška aprašo, kokia tvarka, su kokiais argumentais ir kokius veiksmus reikia atlikti. Kairėje operatoriaus pusėje įrašytas kintamojo vardas nurodo, kaip bus pavadintas išraiškos skaičiavimo rezultatas. Dažniausiai yra vartojamos aritmetinės išraiškos, kurios aprašo aritmetinius skaičiavimus ir atitinka mums įprastą algebrinės išraiškos sąvoką.

Pavyzdžio programoje vartojamas operatorius $y := x * x$ rodo, kad argumento x reikšmė keliama kvadratu ir rezultatui

suteikiamas vardas *y*. Priskyrimo operatoriaus užrašymo forma artima lygybių užrašymo būdai, tačiau tarp šių struktūrų yra principinis skirtumas. Lygybėmis yra nurodoma egzistuojanti priklausomybė, patvirtinamas tam tikras faktas. Tuo tarpu, priskyrimo operatoriai aprašo procesus, kuriems būdingas reikšmių kitimas laike, todėl yra taisyklingas sakinyss:

s := *s*+1;

Jis nurodo, kad senoji kintamojo *s* reikšmė turi būti padidinama vienetu, ir rezultatas vėl pavadinamas *s*. Tokį procesą galima aprašyti lygtimi:

$$S_{(nauja)} = S_{(sena)} + 1$$

Priskyrimo operatoriuose gali būti nurodomos įvairių tipų išraiškos, kurių rezultatai taip pat gali būti įvairūs. Todėl būtina sekti, kad priskyrimo operatoriaus kairėje pusėje nurodyto kintamojo tipas atitiktų išraiškos reikšmės tipą. Smulkiau apie įvairių tipų išraiškas kalbama kituose skyriuose.

Išraiškose galima nurodyti tiksliai tokius kintamuosius, kurių reikšmės jau yra apibrėžtos anksčiau įrašytuose programos sakiniuose. Kintamųjų reikšmės apibrėžiamos tiesiogiai, priskyrimo operatoriais bei specialiomis procedūromis ir funkcijomis, arba netiesiogiai, nurodant, iš kokių kompiuterio įrenginių jos gali būti gaunamos. Dažniausiai naudojamos Paskalio kalbos aritmetinių skaičiavimo funkcijos aprašytos 1.1.1 lentelėje.

1.1.1 lentelė. Pagrindinės aritmetinės funkcijos

Tradicinė forma	Paskalyje	Pastaba
arctg <i>x</i>	Arctan(<i>x</i>)	Rezultatas radianais
cos <i>x</i>	Cos(<i>x</i>)	Argumentas radianais
sin <i>x</i>	Sin(<i>x</i>)	Argumentas radianais
<i>e</i> ^{<i>x</i>}	Exp(<i>x</i>)	
\sqrt{x}	Sqrt(<i>x</i>)	
ln <i>x</i>	Ln(<i>x</i>)	

	Frac(x)	Trupmeninė argumento dalis
	Int(x)	Sveikoji argumento dalis
x	Abs(x)	

Kompiuteryje programų ryšys su jo įrenginiais valdomas pagalbinėmis programomis, kurios vadinamos įvedimo/išvedimo procedūromis.

Aprašant duomenų apdorojimo procesus, į pagalbines procedūras kreipiamasi procedūrų operatoriais, kurių sintaksė:

<Procedūros vardas> (<Parametrų sąrašas>)

Kompiuteriuose vartojama daug įvairių įrenginių, ir jų valdymas yra gana sudėtingas, nes reikia nurodyti ne tik perduodamus duomenis, bet ir įvairias įrenginių paruošimų bei duomenų sutvarkymo operacijas. Siekiant palengvinti įvedimo/išvedimo procesų aprašymą, visuose kompiuteriuose vienam įvedimo ir vienam išvedimo įrenginiui, kurie vadinami standartiniais, yra taikomos supaprastintos įvedimo-išvedimo procedūros. Personaliniuose kompiuteriuose standartinis įvedimo įrenginys yra klaviatūra.

Duomenų skaitymas iš klaviatūros valdomas operatoriais:

```
Read (<Kintamųjų sąrašas>);
ReadLn (<Kintamųjų sąrašas>);
```

Pats duomenų įvedimo procesas programos darbo metu vykdomas trimis etapais. Iš pradžių įvedimo operatorius stabdo kompiuterio darbą ir laukia, kol klaviatūroje bus surinkti duomenys. Po to klaviatūroje renkami duomenys kaupiami specialiaame buferinės atminties įtaise ir rodomi ekrane. Šiuo metu jie gali būti taisomi ekrano vaizdo redagavimo priemonėmis. Trečias etapas prasideda nuspaudus įvedimo klavišą, kuris žymimas simboliu ↵ arba žodžiu *Enter*. Jį paspaudus, buferyje saugomi duomenys paskirstomi procedūros operatoriaus sąrašė išvardytiems kintamiesiems, kurie nurodo, kiek ir kokio tipo duomenų turi būti įvesta, kaip šie duomenys turi būti paskirstomi programos kintamiesiems.

Procedūros *Read* ir *ReadLn* skiriasi duomenų atrinkimo iš pagalbinės atminties (buferio) būdais. Procedūra *Read* visuomet atrenka tiek duomenų, kiek yra jos sąraše parametrų, o procedūra *ReadLn* atrenka visus buferyje esančius duomenis, kurie yra prieš klavišu *Enter* įvedamą eilutės pabaigos kodą. Tokią pastarosios procedūros savybę nurodo ir jos vardas *ReadLn*, kuris sudarytas iš dviejų anglų kalbos žodžių: *Read* (skaityk) ir *Line* (eilutė).

Pavyzdžiui, jeigu bus vykdomas operatorius *Read(a, b)*, o mes įvesime tik vieną skaičių – 15, kintamajam *a* bus suteikta reikšmė **15** ir kompiuteris pakartos užklausa *b* reikšmei. Jei, vykdant *Read(a, b)*, įvesime tris reikšmes: 15 12 10, kintamiesiems *a* ir *b* bus paskirstytos pirmosios dvi reikšmės, o trečioji liks buferyje ir lauks naujo skaitymo operatoriaus.

Jeigu tokia pati situacija susidarytų vykdant operatorių *ReadLn*, trečioji reikšmė 10 būtų prarasta, nes šis operatorius perteklinius duomenis, kurie lieka paskirsčius perskaitytos eilutės duomenis, atmeta.

Įvedant klaviatūra vienoje eilutėje duomenų grupes, atskiros reikšmės tarpusavyje atskiriamos tarpais arba tabuliacijos simboliais.

Standartinis personalinių kompiuterių išvedimo įrenginys, į kurį duomenys nukreipiami procedūromis *Write* ir *WriteLn* yra ekranas. Kreipinių į šias procedūras sintaksė:

```
Write (<Duomenų sąrašas>);  
WriteLn (<Duomenų sąrašas>);
```

Išvedamų duomenų sąrašuose galima nurodyti kintamuosius, konstantas ir išraiškas. Be to, gali būti aprašomi ir atskiriems duomenims patalpinti skirti laukai. Duomenų sąrašo elemento struktūra:

```
<Duomuo>[:<l>[:<t>]]
```

Sveiko tipo konstanta **l** elemento apraše nurodo jam skiriamo lauko dydį, o konstanta **t** rašoma tik realiems skaičiams, turintiems sveikąsias ir trupmenines dalis. Tai trupmeninei daliai skirtų simbolių skaičius. Pavyzdžiai:

```
Write (a:5);      WriteLn (a:5, b:6:2);
```

Pradžia. Pavyzdys_1_1
x, y realūs skaičiai
Ivesti x reikšmę
$y := x * x$
Išvesti y reikšmę
Pabaiga

Jeigu, išvedant realius skaičius, trupmeninei daliai skiriamo lauko dydis nenurodomas, skaičiai yra rodomi rodyklinėje formoje, kurioje sąvoka "dešimt pakelta laipsniu" yra žymima simboliu E. Pavyzdžiui, 1.254E-05 atitinka $1.254 \cdot 10^{-5}$.

Išvedami ekrane duomenys yra pradedami rašyti nuo ekrano žymeklio (kursoriaus) nurodomos ekrano pozicijos, kuri vadinama aktyvia ekrano pozicija. Jeigu yra vykdomas operatorius *Write*, surašius ekrane operatoriuje aprašytus duomenis, aktyvioji ekrano pozicija lieka ties paskutinio operatoriaus suformuoto lauko riba. Operatorius *WriteLn* aktyviają ekrano poziciją visuomet perkelia į naujos eilutės pradžią. Šią savybę galima panaudoti aprašant tuščias ekrano eilutes:

```
WriteLn;
```

Be parametrų taip pat galima vartoti ir operatorių *ReadLn*. Ši jo forma nurodo, kad reikia stabdyti kompiuterio darbą ir jį tęsti tik paspaudus klavišą *Enter*. Laikinas kompiuterio stabdymas reikalingas derinant programas, išvedant į ekraną didelius duomenų kiekius ir kitais atvejais.

Operatorių *Write* ir *WriteLn* argumentai gali būti ne tik skaičiai ir kintamieji, bet ir bet kokie tekstiniai duomenys, kuriuos pageidaujama matyti ekrane. Tokius duomenis argumentų sąrašė būtina rašyti tarp kabučių (apostrofų). Pavyzdžiui, operatorius *WriteLn* ('Sveiki! Jūsų programa dirba!') išveda ekrane jo argumento nurodomą tekstą:

```
Sveiki! Jūsų programa dirba!
```

Aprašant duomenų įvedimą, patariama prieš operatorių *ReadLn* (arba *Read*) visuomet rašyti išvedimo operatorių, kuris paaiškintų, kokių duomenų programai reikia. Tokia išvedimo ir įvedimo operatorių pora dažnai dar vadinama duomenų užklausa. Pavyzdžio programoje vartojama tokia užklausa:

```
WriteLn ('Įvesk skaičių:'); ReadLn (x);
```

Ji nurodo, kad kompiuteriui ekrane reikia suformuoti pranešimą **Įvesk skaičių**, stabdyti kompiuterį ir laukti, kol klaviatūra bus surinktas skaičius.

Aprašant programos darbo rezultatų išvedimą, labai patogu operatoriaus *WriteLn* duomenų sąrašą nurodyti tiek duomenų reikšmes, tiek juos paaiškinančias simbolių eilutes. Pavyzdžiui, operatorius *WriteLn* ('Kvadratas: ', y: 10: 2) nurodo, kad prieš kintamojo *y* reikšmę turi būti įrašomas paaiškinimas **Kvadratas**.

Programos pagrindinės dalies ribos žymimos baziniais žodžiais *begin* ir *end*, kurie, pabrėžiant jų paskirtį, dar vadinami operatoriniais skliaustais. Jais pažymėta programos dalis vadinama sudėtinio operatoriumi arba bloku. Programos pabaigoje, už žodžio *end*, visuomet rašomas taškas, o sakiniai vienas nuo kito atskiriami kabliataškiais.

Programų rašymą patogu pradėti nuo jos struktūros projektavimo. Šiam tikslui patogu naudoti lenteles, kurios vadinamos struktūrogramomis. Skyrelyje aprašomą programą vaizduojanti struktūrograma pateikta 1.3.1 pav. Joje nurodoma programos struktūrininių elementų (skyrių ir sakinių) rašymo bei vykdymo tvarka. Tokie nurodymai gali būti sudaromi tiek Paskalio kalbos sakiniais, tiek lietuvių kalbos sakiniais. Turint tokią struktūrogramą, programos tekstą Paskalio kalba parašyti nesunku. Tereikia jos elementus pakeisti išsamiais aprašymais Paskalio kalba.

Pradžia. Pavyzdys_1_1
x, y realūs skaičiai
Įvesti x reikšmę
$y := x * x$
Išvesti y reišdme
Pabaiga

1.3.1.pav. Programos struktūrograma

Jeigu programoje nėra specialių nurodymų, jos sakinių instrukcijos vykdomos nuosekliai, surašymo tvarka. Tokios programos vadinamos tiesinėmis. Jų struktūrogamų visi elementai yra vienodos stačiakampės formos.

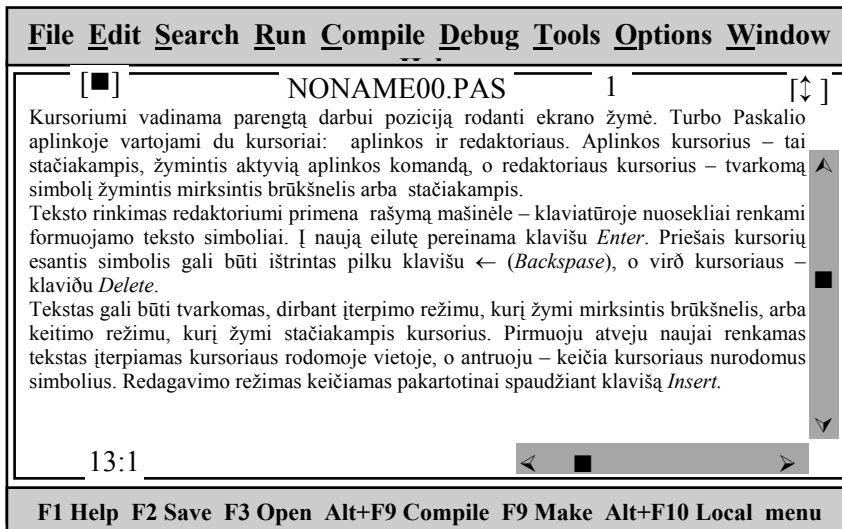
1.3.1 pratimas. Sudaryti skritulio ploto skaičiavimo programą. Argumentas - skritulio spindulys.

1.3.2 pratimas Sudaryti stačiakampio gretasienio tūrio skaičiavimo programą. Argumentai - kraštinių ilgiai.

Sudarytose programose turi būti užklaustos argumentams, skaičiavimus ir rezultatų pateikimą ekrane aprašantys operatoriai, išsamūs komentarai.

1.4. Programos teksto paruošimas redaktoriumi

Tarkime, kad jau turime parašę programą ir norime ją išbandyti kompiuteryje. Norint tai atlikti, visų pirma, reikia programos tekstą Turbo Paskalio aplinkos redaktoriumi įvesti į kompiuterio atmintį. Iškviečiant darbui Turbo Paskalio aplinką su MS DOS komanda *turbo* arba vartotojo aplinkos priemonėmis, jos redaktorius parengiamas darbui automatiškai. Apie tai iliustruoja jo darbo lange matomas kursorius. Turbo Paskalio aplinkos ir jos redaktoriaus kursorių paskirtys bei pagrindiniai teksto rinkimo principai yra aprašyti 1.4.1. pav.



1.4.1. pav. Redaktoriaus darbo langas

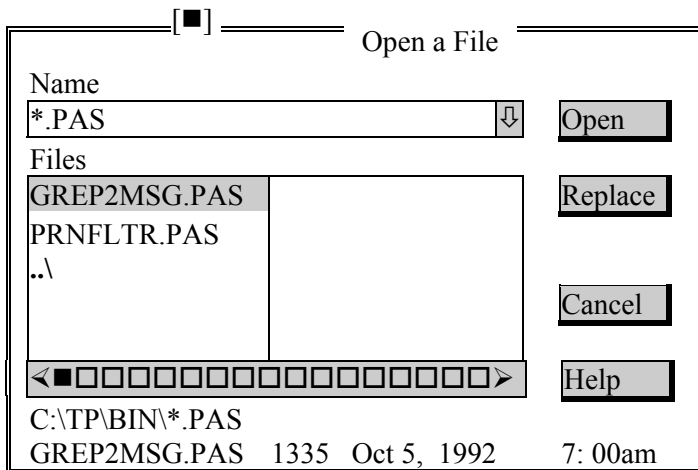
Aprašydami redaktoriaus ir kitų aplinkos sudėtinių dalių valdymo komandas, vartosime sudėtinius vardus, kurių struktūra:

<Komandos vardas>.<Komandos meniu elementas>

Pavyzdžiui, užrašas *File.Save As* rodo, kad kalbama apie komandos *File* elementą *Save As*, kuris skirtas programos rašymui į diską nurodant vardą valdyti.

Jeigu, pradėdant darbą, redagavimo lange būna matomas kitos programos tekstas, jį galima ištrinti komanda *File.New*. Po to ekrane matomas tuščias redaktoriaus langas, kuriame kursorius rodo aktyvią, paruoštą duomenų įvedimui, ekrano poziciją. Be to, darbo langą gaubiančiame rėmelyje matomas naujai programai aplinkos suteiktas pavadinimas *NONAME00.PAS* (1.4.1. pav.). Šį vardą galima pakeisti su komanda *File.Open*, kuri ekrane formuoja naujo failo vardo užklausos rėmelį vardu *Open a File* (Parenk failą), kuris yra parodytas 1.4.2. pav.

Užklausos rėmelis leidžia pasirinkti įvairius dialogo su kompiuteriu būdus: redaguojamus laukus, parinkimo lenteles, atsiderančius langelius, prasukimo juostas ir ekraninius klavišus. Naujo programos vardo nurodymui skirtas redaguojamas laukas *Name* (Vardas), kurį galima parengti darbui, spragtelėjus jo rėmelio viduje pele. Po to šiame lauke atsiranda redagavimo kursorius ir klaviatūra galima surinkti pageidaujamą vardą. Veiksmas yra baigiamas, spragtelėjus pele ties ekraniniu mygtuku *Open*.



1.4.2. pav. Komandos *File.Open* užklausos rėmelis

Ekraniniai mygtukai nuo kitų rėmelio elementų skiriasi šešėliais, kurie sudaro iškilusio ekrano paviršiuje mygtuko išpūdį. Jie vartojami parenkant komandas, kurios turi būti vykdomos panaudojant rėmelio laukuose nurodytus duomenis. Du ekrano mygtukai būna visuose dialogo rėmeliuose. Tai *Cancel* (atsisakyti, panaikinti) ir *Help* (pagalba). Pirmuoju mygtuku atsisakoma pradėtos vykdyti komandos, o antruoju yra iškviečiamas pagalbinės informacijos apie vykdomą komandą langas. Dialogo rėmelių kairiajame viršutiniame kampe yra dar vienas standartinis valdymo elementas – tarp laužtinių skliaustų patalpintas kvadratis [■], kuris skirtas dialogo langelio uždarymui. Šis veiksmas atliekamas, spragtelėjus ties kvadratiu pele.

Pradedant komandos *File.Open* dialogą, lauke *Name* matomas šablonas *.PAS, kuris nurodo, kad lentelėje *Files* turi būti rodomi tik tie failai, kurių varduose yra išplėtimai PAS. Lauko *Name* šabloną galima pakeisti konkrečiu failo vardu arba kitu šablonu, sudarytu pagal operacinės sistemos MS DOS taisykles. Pažymėjus pele pageidaujamą failą lentelėje *Files*, jam galima atidaryti naują redaktoriaus langą (*Open*) arba pakeisti šiuo failu aktyviame lange buvusį tekstą (*Replace*). Langelis su rodykle ⇩ lauko *Name* dešinėje informuoja, kad šis laukas turi atsidarantį langelį, kuriame galima pamatyti daugiau vardų šablonų ir pačių vardų.

Parinkimo lentelės *Files* apačioje yra matoma prasukimo juosta su rodyklėmis ◀ ir ▶ galuose, kuri vartojama tada, kai šabloną atitinkančių failų sąrašas yra didelis ir parinkimo lauke netelpa. Lentelėje rodomi duomenys yra pastumiami, "paspaudžiant" su pele rodykles. Pilnaviduris kvadratis juostoje rodo, kurioje sąrašo vietoje yra aktyvus failas.

Dialogo rėmelio apačioje matomos dvi informacinės eilutės, kurios informuoja apie aktyvų katalogą ir parodo parinkto failo parametrus: vardą, dydį baitais, sukūrimo datą ir laiką.

Programos teksto rinkimas redaktoriaus lange primena rašymą rašomąja mašinėle, tačiau čia galima vartoti daug papildomų specialių teksto formavimo ir redagavimo priemonių. Sudėtingesnė yra ir kompiuterio klaviatūra, kuri yra

pritaikyta tiek dideliame įvairių funkcinių komandų skaičiui valdyti, tiek visiems kompiuterio apdorojamo alfabeto simboliams įvesti.

Teksto rinkimui skirtos klaviatūros režimai valdomi klavišais *Shift*, *CapsLock* ir *NumLock*. Klavišas *CapsLock* perjungia klaviatūrą iš mažųjų raidžių į didžiųjų raidžių režimą ir priešinga kryptimi. Daugumoje klaviatūrų apie įjungtą didžiųjų raidžių režimą informuoja vardu *CapsLock* žymimas šviesos indikatorius.

Renkant raides, tą pačią paskirtį kaip ir *CapsLock* klavišas turi priešdėlio klavišas *Shift*, tačiau jis perjungia klaviatūros režimą tik laikinai, kol būna nuspaustas. Klavišas *Shift* taip pat keičia ir pagrindinėje klaviatūroje esančių skaitmeninių bei kitokių neraidinių klavišų interpretavimą. Jį nuspaudus, galioja ne pagrindinė, klavišo apačioje užrašyta, o papildoma, jo viršuje nurodyta, reikšmė. Klavišas *CapsLock* neraidinių klavišų interpretavimo nekeičia.

1.4.1.lentelė. Kursoriaus valdymo klavišai

Klavišas	Komanda
↑	Kursoriaus perkėlimas per vieną eilutę į viršų.
↓	Kursoriaus perkėlimas per vieną eilutę į apačią.
←	Kursoriaus perkėlimas per vieną poziciją į kairę.
→	Kursoriaus perkėlimas per vieną poziciją į dešinę.
<i>End</i>	Kursoriaus perkėlimas į eilutės galą.
<i>Home</i>	Kursoriaus perkėlimas į eilutės pradžią.
<i>PgUp</i>	Kursoriaus perkėlimas per vieną puslapį (ekraną) į viršų.
<i>PgDn</i>	Kursoriaus perkėlimas per puslapį (ekraną) į apačią.
<i>Ins</i>	Redagavimo režimo keitimas į įterpimą arba keitimą.
<i>Del</i>	Kursoriaus rodomo simbolio trynimasis.

Klavišu *NumLock* galima perjungti papildomą skaitmeninę klaviatūrą į skaitmeninį arba į kursoriaus valdymo režimą (↑, ↓, ←, →, *End*, *Home*, *Ins*, *Del*, *PgUp*, *PgDn*). Šios klavišų grupės darbo režimą rodo šviesos indikatorius *NumLock*, o klavišų paskirtys yra aprašytos 1.4.1 lentelėje. Daugumos kompiuterių klaviatūrose yra dar ir trečia klavišų grupė, skirta vien tik kursoriaus pozicijai valdyti.

Redaktoriaus lango plotis ekrane priklauso nuo jo darbo režimo. Dažniausiai jis lygus 77 simboliams. Jei formuojamos programos teksto eilutės ilgis viršija lango plotį, ekrano vaizdas automatiškai pastumiamas į kairę pusę.

Maksimalus redaktoriaus formuojamos eilutės ilgis gali siekti 249 simbolius, tačiau Turbo Paskalio kompiliatorius nagrinėja tik pirmuosius 128 simbolius, todėl programas sudarančių eilučių ilgiai turi neviršyti šio skaičiaus.

Siekiant padidinti redaktoriumi tvarkomų programų teksto vaizdumą ir palengvinti jo analizę, yra vartojamos spalvos, kurios išryškina tekste įvairios sintaksinės paskirties elementus. Skirtingomis spalvomis yra rašomi baziniai žodžiai, komentarai ir kiti tekstai. Šias spalvas galima pakeisti aplinkos parametrais (komanda *Options*).

Redaktoriaus lango apatiniame rėmelyje matoma skaičių pora, kuri informuoja apie redaktoriaus kursoriaus poziciją darbo lange. 1.4.1. pav. matoma pora 13:1 rodo, kad kursorius yra 13 eilutės 1-me stupelyje.

Į bet kurią pageidaujamą vietą redaktoriaus kursorių galima perkelti pele arba kursoriaus valdymo klavišais (lentelė 1.4.1.).

Nuosekliai renkant programos tekstą, papildomos redaktoriaus valdymo komandos beveik nereikalingos, tačiau analizuojant ir pertvarkant programų tekstus, jos būtinos.

Ypatingai patogios ir efektyvios yra manipuliavimo teksto blokais komandos, kurios yra valdomos su aplinkos komanda *Edit*. Tvarkomas teksto blokas iš pradžių turi būti pažymimas. Tai daroma perkeliant kursorių į žymimo teksto pradžią ir skenuojant tekstą pele arba pažymint jį kursoriaus klavišais, esant nuspaustam prefikso klavišui *Shift*.

1.4.2 lentelė. Teksto blokams taikomi veiksmai

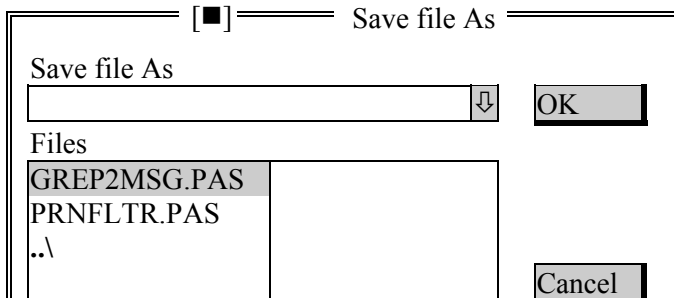
Menu elementas	Valdantys klavišai	Komandos paskirtis
Undo	<i>Alt+Bksp</i>	Atšaukti paskutinę redagavimo komandą.
Redo		Pakartoti paskutinę redagavimo komandą.
Cut	<i>Shift+Del</i>	Persiųsti pažymėtą bloką į buferinę atmintį.

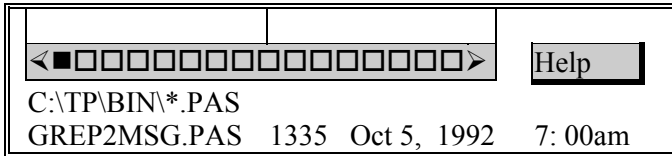
Copy	<i>Ctrl+Ins</i>	Kopijuoti pažymėtą bloką į buferinę atmintį.
Paste	<i>Shift+Ins</i>	Rašyti buferinėje atmintyje saugomą bloką kursoriaus nurodomoje vietoje.
Clear	<i>Ctrl+Del</i>	Panaikinti pažymėtą teksto bloką.
Show clipboard		Parodyti buferinėje atmintyje saugomą tekstą.

Skenavimu pele vadinamas pelės kursoriaus traukimas per žymimą tekstą, esant nuspaustam jos dešiniajam klavišui. Pažymėtas teksto blokas yra matomas ekrane pakeistos spalvos fone. Jam galima taikyti įvairias komandas *Edit* operacijas, kurių paskirtys yra aprašytos 1.4.2 lentelėje.

Komandos operacijos yra parenkami arba iš jos pagalbinio meniu pele, arba lentelėje aprašytais valdymo klavišais. Redaguojant programas, taip pat gali būti vartojamas supaprastintas aplinkos valdymo komandų meniu su pagrindinėmis redagavimo ir kitomis dažniausiai vartojamomis aplinkos valdymo komandomis, kuris iškviečiamas dešiniuojų pelės klavišu.

Prieš transliuojant ir tikrinant redaktoriumi paruoštą programą, rekomenduojama ją įrašyti į diską, kad prireikus, jos tekstą būtų galima atstatyti. Rašymui į diską skirta komanda *File.Save As*, kurios dialogo rėmelis yra analogiškas komandos *File.Open* rėmeliui ir yra atvaizduotas 1.4.3 pav. Šiame rėmelyje programos failo įrašymas į diską lauke *Save file As* nurodytu vardu yra patvirtinamas ekraniniu mygtuku OK (angliško Okay sutrumpinimas).





1.4.3. pav. Komandos *File.Save As* užklaustos rėmelis

1.5. Programos transliavimas ir tikrinimas

Surinkus redaktoriumi programos tekstą, jį būtina išversti (kompiliuoti) į mašininį kodą ir patikrinti programos darbą, įsitikinti, kad ji dirba teisingai. Tam reikia mokėti pasirinkti aplinkoje vartotojo poreikius atitinkantį kompiliavimo ir tikrinimo būdą. Tikrinant mažas programas, yra patogiausias toks kompiliavimo būdas, kai mašininis kodas yra paliekamas operatyviojoje atmintyje. Mašininio kodo paskirties vieta aplinkoje yra keičiama su komanda *Compile.Destination*, kuri gali parinkti vieną iš dviejų mašininų kodų saugojimo būdų: *Memory* (Operatyvioji atmintis) ir *Disk* (Diskas). Vartojant operatyviają atmintį, programos kompiliavimas ir tikrinimas vyksta gerokai greičiau, taupiau naudojama diskinė atmintis. Į diską mašininis kodas ilgalaikiam saugojimui rašomas tiktai po to, kai įsitikinama, kad programa dirba teisingai.

Paprasčiausias kompiliavimo būdas yra valdomas su komanda *Compile.Compile*. Ši komanda taip pat tikrina, ar nėra tvarkomoje programoje sintaksės klaidų. Aptikus klaidą, ekrano viršuje yra formuojamas pranešimas apie spėjamą klaidos tipą, o redaktoriaus kursorius rodo orientacinę jos vietą tekste. Pavyzdžiui, praleidus du operatorius skiriančią kabliataškį, ekrane matomas pranešimas:

Error 85: ";" expected.

Detalesnės informacijos apie aptiktos klaidos pobūdį, panaudojus pranešime rodomą kodą, galima ieškoti komandos *Help.Error messages* lentelėse. Ištaisius klaidą, kompiliavimą reikia pakartoti. Kai klaidų nebelieka, kompiliatorius formuoja pranešimą apie sėkmingą darbo pabaigą (*Compile successful*) ir

pateikia informacinius duomenis apie programą bei neužimtos atminties dydį. Dabar jau galima tikrinti programos darbą. Ši veiksmą valdo komanda *Run.Run*.

Turbo Paskalio aplinkoje vartojama daugialangė išorinio interfeiso struktūra. Tai reiškia, kad ekrane galima matyti keletą langų su įvairiais duomenimis. Šie langai gali būti ekrane greta vienas kito arba dengti vienas kitą. Langų rodymo būdas yra keičiamas aplinkos komandomis. Dažniausiai aplinka dirba taip, kad langai dengia vienas kitą. Renkant programos tekstą, viršuje būna redaktoriaus darbo langas. Vykdam programą, jį uždengia programos darbo langas, kuris vadinamas vartotojo langu (*User screen*), o baigus programos darbą, viršuje vėl būna redaktoriaus darbo langas. Jeigu norime atidžiau panagrinėti programos darbo rezultatus, gali būti pageidautina grąžinti vartotojo langą. Šis veiksmas valdomas komanda *Debug.User Screen*. Iš vartotojo lango į redaktoriaus langą pereinama paspaudus bet kurį klavišą. Tokio perjunginėjimo galima išvengti, įrašant programos gale jos stabdymo komandą *ReadLn*. Ši komanda saugos ekrane vartotojo langą tol, kol nebus paspaustas klavišas *Enter*.

Komandos *Compile.Compile*, *Run.Run* ir *Debug.User Screen* vartojamos labai dažnai, todėl jų greitam parinkimui yra numatyti taip pat ir klavišai, atitinkamai *Alt+F9*, *Ctrl+F9* ir *Alt+F5*.

1.5.1 pratimas. Surinkite Turbo Paskalio aplinkos redaktoriumi, įrašykite į diską ir patikrinkite 1.3 skyrelio pratimuose surinktas savo programas.

2. Programavimo abėcėlė

2.1 Vartotojo ir programos dialogas.

Jau rašydamas savo pirmąją programą programuotojas neturi pamiršti, kad programą rašo ne vien tik sau. Gal būt jo parašyta programa naudosis ir kiti jo draugai, mokytojai, o gal ir labai didelis vartotojų ratas. Todėl gatava programa turi ‘kalbėtis’ su šios programos vartotoju, t.y. turi vykti dialogas tarp dirbančios programos ir jos vartotojo. Šis dialogas turi būti nesudėtingas, aiškiai apibrėžtas ir būti vienareikšmiškas. Tokiam dialogui organizuoti yra naudojamos šiuolaikinių kompiuterių teikiamos galimybės – garsas, spalvos ir kt. Šiame skyrelyje mes ir pabandydysime pasinaudoti šiomis pačiomis elementariausiomis galimybėmis. Tam tikslui programos pradžioje nereikia pamiršti užrašyti tokį sakinį (eilutę):

```
uses Crt; { Lietuviškai tai reiškia  
– “vartojama Crt biblioteka” }
```

Crt yra vadinamas moduliu (biblioteka), kuriame yra įrankiai (procedūros, funkcijos, konstantos ir kintamieji), skirti dirbti su ekranu ir klaviatūra. Tai bus plačiau pakomentuota vėlesniuose skyreliuose.

Pirmasis veiksmas, kurį turėtų atlikti savo darbą pradedanti programa, tai būtų kompiuterio ekrano išvalymas. Tą atlieka procedūra:

```
procedure ClrScr; { Ekrano išvalymas }
```

Todėl šia procedūra reikėtų naudotis kiekvienoje būsimojoje programoje.

Darbą pradėjusi programa turėtų pasisveikinti, o pabaigus darbą atsisveikinti su jos vartotoju.

Tai iliustruoja pavyzdžio 2.1 programa:

```
program P21;  
uses Crt;  
begin  
  ClrScr;
```

```

    WriteLn ( 'Sveiki ! Programa darba
pradedama...' );
    WriteLn ( 'Paspauskite ENTER klavišą' );
    ReadLn;
    WriteLn;
    WriteLn ( '      Šioje vietoje bus
atliekama:      ' );
    WriteLn ( '      - duomenų įvedimas;
' );
    WriteLn ( '      - duomenų
išvedimas/spausdinimas; ' );
    WriteLn ( '      - skaičiavimai;
' );
    WriteLn ( '      - rezultatų
išvedimas/spausdinimas.' );
    WriteLn;
    WriteLn ( 'Programa darba sėkmingai
pabaigė...' );
    WriteLn ( 'Sėkmės Jums visur ir visada!'
);
    WriteLn ( 'Paspauskite ENTER klavišą' );
    ReadLn;
end.

```

Kiekviena programa turi palaikyti ryšį su išore, išorine aplinka t.y. su programos vartotoju: gauti iš ten duomenis ir pateikti skaičiavimų rezultatus. Personaliniuose kompiuteriuose labai populiarus dialoginio ryšio su išore forma, kai programa rašo ekrane pranešimą apie savo pageidavimus (WriteLn), stabdo savo darbą (ReadLn) ir laukia, kol vartotojas klaviatūroje surinks atsakymą ir paspaus Enter klavišą. Ekrane rašomi programos pageidavimai yra vadinami užklausomis. Jos formuojamos rašymo operatoriais:

```
WriteLn ( '<Užklauso tekstas>' );
```

Operatoriaus WriteLn į ekraną siunčiamą tekstą būtina rašyti tarp pavienių kabučių (apostrofų). Taip sudaryta struktūra vadinama tekstiniais duomenimis, simbolių eilutėmis arba tiesiog

eilutėmis. Jose gali būti visi kompiuterio alfabeto simboliai. Kintamųjų, pačių programų ir kitų programos objektų vardus galima rašyti tik lotyniškais raidėmis.

Už užklausos operatoriaus programoje turi būti rašomas skaitymo iš klaviatūros operatorius, kuris stabdo kompiuterio darbą ir laukia ten surinktų duomenų. Skaitymo operatoriaus sintaksė:

```
ReadLn ( <Kintamųjų sąrašas> );
```

Sąrašas nurodo, kiek reikšmių klaviatūroje reikia surinkti ir kokiems kintamiesiems jas suteikti. Reikšmės galima surinkti vienoje ekrano eilutėje, atskiriant tarpais, arba galima jas rašyti atskirose eilutėse. Labai svarbi yra reikšmių surinkimo tvarka. Klaviatūroje surinkta eilutė įvedama tik po klavišo Enter paspaudimo.

Užklausų pavyzdžiai:

```
WriteLn ( 'Įveskite skritulio diametrą:' );  
ReadLn ( d );                { Programa lauks d  
reikšmės įvedimo }
```

arba

```
WriteLn ( 'Įveskite prekės kainą ir kiekį:' );  
ReadLn ( kaina, kiekis );    { Programa lauks  
kainos ir kiekio reikšmių įvedimo }
```

Programos suskaičiuotus rezultatus pamatysime ekrane, jeigu programoje pasinaudosime rašymo operatoriumi WriteLn, kurio atskiras atvejis buvo aprašytas anksčiau. Bendru atveju, jo sintaksė yra tokia:

```
WriteLn ( <Į ekraną siunčiamų duomenų sąrašas> );
```

Sąrašo elementais gali būti kintamieji, išraiškos ir įvairių tipų konstantos. Operatorius nuosekliai rašo ekrane visų savo argumentų, sąrašo elementų, reikšmes. Pavyzdžiui, jeigu buvo įvestas skritulio diametras (d) lygus 10, programai apskaičiavus

skritulio spindulį ($r=d/2$), rezultatus galima bus pamatyti, užrašius programoje toki operatorių:

```
WriteLn ( 'Skritulio spindulys yra ', r:8:2 );
```

Ivykdžius programą ekrane matysime:

Skritulio spindulys yra 5.00,

nes operatoriaus WriteLn argumentų sąrašė yra du elementai atskirti kableliu: eilutės tipo konstanta ir kintamasis r. Eilutės tipo konstantų ir kintamųjų vardų komponavimas rašymo sąrašuose yra labai patogi ir vaizdi programos darbo rezultatų aprašymo priemonė.

Nurodant išvedimo sąrašuose **real** tipo kintamuosius, rekomenduojama juos rašyti kartu su išvedimo šablonais, kurie aprašo kintamųjų reikšmėms skiriamus ekrano laukų dydžius:

<Duomenų elementas>: <Lauko ilgis>: <Trupmeninė dalis>

Šablonas aprašo, kokio dydžio ekrano laukas yra skiriamas visam skaičiui (Lauko ilgis) ir trupmeninės dalies vaizdavimo tikslumą (Trupmeninė dalis).

Pavyzdžiui, operatorius WriteLn (r:10:2); nurodo, kad kintamojo r reikšmei ekrane skiriamas 10-ies simbolių laukas, kuriame turi būti rašoma dviejų ženklų po kablelio tikslumu.

Jei programos darbui yra reikalinga grupė argumentų, galima rašyti užklausas kiekvienam argumentui atskirai, grupuoti jas arba rašyti vieną bendrą užklausą visiems argumentams. Pavyzdžiui, tarkime, kad reikia sudaryti programą, kuri apskaičiuotų stačiakampio plotą, kai žinome jo kairiojo viršutinio (x1,y1) ir dešiniojo apatinio (x2,y2) kampo koordinatės. Pavyzdyje 2.2 pateikiama programa, kuri organizuoja užklausą, atlieka skaičiavimus ir išveda (atspausdina) rezultatus.

```
program P22;  
uses Crt;
```

```

var x1, y1 : real;      { Pirmojo taško
koordinatės           }
    x2, y2 : real;      { Antrojo taško
koordinatės           }
    a, p   : real;      { Stačiakampio
aukštis ir plotis     }
    pl     : real;      { Stačiakampio plotas
}
begin
  ClrScr;
  WriteLn ( 'Sveiki ! Programa darba
pradeda...' );
  WriteLn ( 'Paspauskite ENTER klavišą' );
  ReadLn;
  WriteLn ( 'Įveskite pirmojo taško
koordinatės x1 ir y1' );
  ReadLn ( x1, y1 );
  WriteLn ( 'Įveskite antrojo taško
koordinatės x2 ir y2' );
  ReadLn ( x2, y2 );
  a := y1 - y2;
  p := x2 - x1;
  pl := a * p;
  WriteLn;
  WriteLn ( 'Stačiakampio plotas yra lygus
', pl:9:2      );
  WriteLn;
  WriteLn ( 'Programa darba sėkmingai
pabaigė...' );
  WriteLn ( 'Sėkmės Jums visur ir visada!'
);
  WriteLn ( 'Paspauskite ENTER klavišą' );
  ReadLn;
end.

```

Kaip jūs jau pastebėjote personalinis kompiuteris informaciją ekrane gali pateikti spalvotai. Iki šiol mūsų programa informaciją pateikdavo baltais simboliais juodame fone. Rašydami programą, pasinaudokime modulio Crt procedūromis:

```

        procedure TextColor ( <spalva >: byte );           { Nustato
ekrane rašomo teksto spalvą }
        procedure TextBackground ( <spalva >: byte ); { Nustato
ekrane rašomo teksto fono spalvą }

```

kurios priverstų jūsų kompiuterį rašyti spalvotai.

Pavyzdžiui, jeigu mes norime, kad tam tikras tekstas (pvz. 'Dėmesio !') būtų rašomas raudona spalva mėlyname fone turime programoje užrašyti tokią sakinių seką:

```

        TextColor ( Red );
        TextBackground ( Blue );
        WriteLn ( 'Dėmesio !' );

```

Šia spalva bus rašoma ir kitas tekstas, kol vėl nebus pakeista spalva arba fonas. Teksto spalvą galima nurodyti ir skaičiais iš intervalo nuo 0 iki 15, o fono - skaičiais iš intervalo 0 iki 7. Kaip vėliau matysime kiekvienam iš šių skaičių atitinka tam tikra spalva. Be abejonės spalvų pavadinimus naudoti yra patogiau, negu joms atitinkamus skaičius. Tereikia žinoti tik angliškų jų pavadinimus.

Pastaba: jei teksto spalva sutampa su fono spalva, tai rašomas tekstas nesimato.

Pavyzdyje 2.3 pateikta programa, kuri apskaičiuoja alaus statinės tūrį, kai žinomas statinės diametras ir aukštis.

```

program P23;
uses Crt;
const pi = 3.14;
var d : real;      { Statinės diametras }
    l : real;      { Statinės aukštis }
    t : real;      { Statinės tūris }
begin
    ClrScr;
    WriteLn ( 'Sveiki ! Programa darba
pradedama...' );
    WriteLn ( 'Paspauskite ENTER klavišą' );
    ReadLn;
    TextColor( Red );           { Teksto spalva
raudona }

```

```

    TextBackground ( Blue ); { Fono spalva
mėlyna      }
    WriteLn ( 'Įveskite statinės diametrą' );
    ReadLn ( d );
    WriteLn ( 'Įveskite statinės aukštį ' );
    ReadLn ( l );
    t := pi * ( d / 2 ) * ( d / 2 ) * l;
    WriteLn;
    TextColor ( Green );      { Teksto spalva
žalia      }
    TextBackground ( Black );{ Fono spalva
juoda      }
    WriteLn ( 'Statinės tūris yra lygus ',
t:8:3      );
    WriteLn;
    TextColor ( LightGray ); { Teksto spalva
pilka      }
    WriteLn ( 'Programa darbą sėkmingai
pabaigė...' );
    WriteLn ( 'Sėkmės Jums visur ir visada!'
);
    WriteLn ( 'Paspauskite ENTER klavišą' );
    ReadLn;
end.

```

Programos pagyvinimui yra naudojami garsai.
Modulyje **Crt** yra šios garsų valdymo procedūros:

```

    procedure Sound( <Dažnis>: word );
        { Garsiakalbio įjungimo, nurodant garso dažnį,
procedūra }
    procedure Delay( <Trukmė milisekundėmis>: word );
        { Pauzė, kurios metu kompiuteris veiksmų
neatlieka }
    procedure NoSound;          { Garsiakalbio išjungimas
}

```

Visos trys šios procedūros yra vartojamos kartu - įjungiamas pageidaujamo dažnio (tono) garsas, leidžiama jam kurti laiką

skambėti - Delay ir jis išjungiamas arba keičiamas tono aukštis. Garsiniai efektai dažnai naudojami programos darbo pradžioje ir pabaigoje. Pavyzdžio 2.4 programa iliustruoja garsų panaudojimą. Sakykim, žmogus žinodamas savo kambario išmatavimus, nori nusipirkti dažų kambario sienoms ir luboms nudažyti. Parduotuvėje ant dažų dėžės yra nurodyta kokį plotą su jais galima nudažyti. Apskaičiuokite kiek dėžių dažų žmogui reikės pirkti?

```
program P24;
uses Crt;
var ka : real;      { Kambario aukstis }
    kp : real;      { Kambario plotis }
    ki : real;      { Kambario ilgis }
    dp : real;      { Nudazomas plotas }
    dk : real;      { Deziu kiekis }
    pp : real;      { Kambario plotas }
begin
  ClrScr;
  Sound ( 650 );
  Delay ( 700 );
  Nosound;
  WriteLn ( 'Sveiki ! Programa darba
pradedama...' );
  WriteLn ( 'Paspauskite ENTER klavisa' );
  ReadLn;
  TextColor( Red );          { Teksto spalva
raudona }
  TextBackground ( Blue ); { Fono spalva
melyna }
  WriteLn ( 'Iveskite kambario auksti, ploti
ir ilgi' );
  ReadLn ( ka, kp, ki );
  WriteLn ( 'Iveskite vienos dezes nudazoma
plota ' );
  ReadLn ( dp );
  pp := 2 * (kp + ki) * ka + kp * ki; {
Visas dazomas plotas }
  dk := pp / dp;
```

```

    WriteLn;
    TextColor ( Green );      { Teksto spalva
    zalia      }
    TextBackground ( Black ); { Fono spalva
    juoda      }
    WriteLn ( 'Reikalingas deziu kiekis yra
    lygus ', dk:8:3 );
    WriteLn;
    Sound ( 500 );
    Delay ( 700 );
    Nosound;
    TextColor ( LightGray ); { Teksto spalva
    pilka      }
    WriteLn ( 'Programa darba sekmingai
    pabaige...' );
    WriteLn ( 'Sekmes Jums visur ir visada!'
    );
    WriteLn ( 'Paspauskite ENTER klavisa' );
    ReadLn;
end.

```

2.2. Valdančios struktūros

Valdančios struktūros leidžia aprašyti pageidaujamą veiksmų tvarką, alternatyvų parinkimą ir veiksmų kartojimą. Populiariausia valdanti struktūra yra alternatyvų parinkimui skirtas sąlygos operatorius, kurio sintaksė:

```

if      <Sąlyga>      then  <V1>
                               [else  <V2>]

```

Paprasčiausios sąlygos yra aprašomos santykiais, kurių sintaksė:

<i1> <Santykio operacija> <i2>

Čia **i1** ir **i2** - aritmetinės išraiškos, o santykio operacija gali būti žymima tokiais simboliais:

=	lygu	<	mažiau	<=	mažiau arba lygu
<>	nelygu	>	daugiau	>=	daugiau arba lygu

Jei išraiškų reikšmės tenkina santykio operaciją, santykiui suteikiama loginė reikšmė **True** (tiesa), o jei ne - **False** (melas).

Jeigu, vykdant operatorių, sąlygos reikšmė yra true (sąlyga tenkinama), parenkama vykdymui alternatyva **V1**, jeigu reikšmė false - **V2**. Taigi operatorius if aprašo kaip parinkti vykdymui vieną iš dviejų galimų alternatyvų - **V1** arba **V2** (alternatyvos aprašomos operatoriais). Jeigu operatoriuje if šaka else praleista, jis aprašo vieno operatoriaus (**V1**) vykdymo sąlygas.

2.2.1. Argumentų kontrolė

Dažnai tenka tikrinti argumentų priklausymo leistinai reikšmių atkarpai sąlygas. Pavyzdžiui, kvadratinė šaknis gali būti skaičiuojama tik teigiamoms argumentų reikšmėms. Todėl šaknis skaičiuojančiose programose reikėtų tikrinti, ar argumento reikšmė yra leistina. Kaip tai yra daroma, parodyta programoje Trauk.

```
program Trauk;  
var   x, y: real;           { Argumentas ir rezultatas   }  
begin  
    WriteLn ('Įveskite argumento reikšmę'); { Įvedimo užklausa }  
    ReadLn(x);                 { Įvedimas                  }  
    if (x>0)                   { Argumento kontrolė        }  
    then WriteLn ('Šaknis: ', Sqrt(x):6:2)  
    else WriteLn ('Neleistinas argumentas');  
end.
```

Jei šioje programoje argumento kontrolės nebūtų, įvedus neigiamą skaičių, programos darbas būtų nutraukiamas ir ekrane matytųsi pranešimas apie jos vykdymo klaidą.

- Savarankiškai sudarykite programą, kuri iš trijų įvestų skaičių x, y ir z atrinktų didžiausią ir jo reikšmę suteiktų kintamajam max. Tokioje programoje turi būti du operatoriai if. Pirmasis turi atrinkti didesniąją reikšmę iš dviejų, o antrasis turi lyginti atrinktąją reikšmę su trečiuoju skaičiumi:

```
if      x>y          then  max:= x  
                        else  max:= y;      {  
Didesnis iš dviejų    }  
                        }
```

```

        if      z>max then    max:= z;      { Didžiausias iš
trijų          }

```

2.2.2. Ciklai while

Programa Trauk yra nelabai gera, nes, įvedus blogą argumentą, nutraukiamas jos darbas. Geriau kai, įvedus neigiamą skaičių, argumento užklausa kartojama tol, kol įvedamas šaknies skaičiavimo argumentui leistina teigiama reikšmė. Tai galima padaryti su ciklo operatoriumi while, kurio sintaksė:

```

while <Kartojimo sąlyga> do
    <Kartojamas sakiny>

```

kol kartojimo sąlyga
<div> <div>Kartojamas sakiny</div> <div></div> </div>

Šis sakiny nurodo, kad už žodelio do įrašytas operatorius privalo būti kartojamas tol, kol tenkinama kartojimo sąlyga. Jeigu norime kartoti grupę operatorių, juos reikia įrašyti tarp bazinių žodžių begin ir end žymimų operatorinių skliaustų (programa Control). Norint, kad ciklas netaptų begalinis, būtina, kad kartojimo sąlyga priklausytų nuo kartojamo operatoriaus rezultatų.

Programoje Control ciklas sudarytas tam, kad užklausa argumentui būtų kartojama tol, kol įvedama neleistina neigiama argumento reikšmė. Surinkite ir patikrinkite šią programą:

```

program Control;
var      x, y: real;
Argumentas ir rezultatas
begin    { Įvedimo užklausa
WriteLn ('Įveskite iš pradžių neigiamą o po to teigiamą skaičių');
        ReadLn(x);
        { Įvedimas
        while (x<0) do begin      { Užklauskos kartojimas
        }
            WriteLn (' Kartokite, argumentas neigiamas');
            ReadLn(x);
        end;
        WriteLn ('Šaknis: ', Sqrt(x):6:2);
end.

```

2.3 pratimas. Tarkime, kad reikia apskaičiuoti aukštyn mesto kūno trajektoriją pagal išraiškas: $\mathbf{x} = \mathbf{a} * \mathbf{t}$; $\mathbf{y} = \mathbf{b} * \mathbf{t} + \frac{1}{2} * \mathbf{g} * \mathbf{t}$, kur koeficientai \mathbf{a} ir \mathbf{b} bei pagreitis \mathbf{g} yra įvedami užklauskos būdu iš klaviatūros. Trajektorijos taškų koordinatės skaičiuoti laiko \mathbf{t} reikšmėms 0, 1, 2, 3, ir t.t. kol kūnas nukris ant žemės.

Parašykite programą duotam algoritmui Trajektorija, suderinkite programą ir patikrinkite gaunamus rezultatus. Parinkite vaizdžią formą išvedamiems rezultatams.

2.2.4. Užduotis savarankiškam darbui.

- Sudarykite programą kvadratinės lygties $\mathbf{ax}^2 + \mathbf{bx} + \mathbf{c} = \mathbf{0}$ skaičiavimui. Čia duomenys yra lygties argumentai, o rezultatai yra jos šaknys. Prieš ieškant šaknų reikia patikrinti, ar jos egzistuoja.
- Koordinačių plokštumoje yra padėta daug taškų. Jų koordinatės (x, y) yra žinomos. Bet kuriuos tris taškus jungiant atkarpomis galime gauti trikampį. Reikia nustatyti kuriuos taškus sujungus gaunami trikampiai ir kokie taškų rinkiniai negali būti panaudoti trikampio sudarymui. Reikia parašyti programą, kuri nustatytų, ar nurodyti trys taškai gali būti trikampio viršūnėmis, ar negali. Taškų koordinatės nuomos iš klaviatūros kaip šeši skaičiai (pirmo taško x, y, po to antro ir trečio). Įvedamų skaičių srauto pabaiga nurodoma šešiais skaičiais, lygiais nuliui.
- Papildykite programą veiksmams, kuriais būtų suskaičiuojamas galimo trikampio plotas ir ekrane išvedama informacija apie tą trikampį: kraštinių ilgiai ir plotas.

2.3 tema. Sumos kaupimo uždaviniai. Skaičių srautai.

Skaičių srautus apdorojančiose programose labai dažnai taikomi sumos skaičiavimo, sandaugos skaičiavimo ir kiekio radimo algoritmai.

Pavyzdžiui tarkime, kad reikia sudaryti programą klaviatūra įvedamo skaičių srauto sumavimui, kai yra žinoma, kad duomenų pabaigą žymės nulinė reikšmė. Tokio uždavinio sprendimui yra reikalingas vienas kintamasis (**x**) įvedamos reikšmės saugojimui ir antras (**s**) - sumos kaupimui. Suma yra skaičiuojama suteikiant jai pradinę reikšmę **0** (nulis) ir kartojan įvedimo bei sumos kaupimo operaciją, kol reikšmė nelygi nuliui. Sumos aprašomas operatoriumi

Pradžia. Suma
x, s realūs skaičiai
Išvesti: darbo pabaiga kai x=0
<pre> s := 0; Ivesti x reikšmę kol x <> 0 s := s+x; Ivesti x reikšmę </pre>
Išvesti rezultatą: s
reikšmę
Pabaiga

$$s := s + x$$

Priskyrimo operatoriaus kairėje ir dešinėje pusėje nurodytos kintamųjų reikšmės atitinka skirtingus laiko momentus, todėl sumos kaupimo operatoriaus prasmė yra $\mathbf{s}_{nauja} := \mathbf{s}_{sena} + \mathbf{x}$. Surinkite ir patikrinkite skaičių srauto sumavimo programą Suma.

```

program Suma;
var      x, s : real;
{vedama reikšmė ir Suma      }
begin
WriteLn ('Sumavimo programa. Darbas nutraukiamas įvedus
0. ');
      s:= 0;
      {
Pradinė reikšmė      }
      WriteLn('Įveskite x: ');      ReadLn(x);
      while (x<=0) do begin      {      Sumos
kaupimo ciklas      }
      s:= s+ x;      { Sumos
kaupimo veiksmas      }
      WriteLn('Įveskite x:'); ReadLn(x);

```

```

end;
WriteLn('Sumos reikšmė: ', s:8:2);{ Pranešimas apie
rezultatus      }
ReadLn;
Programos stabdymas      }
end.

```

3.2 pratimas. Pakeiskite šią programą taip, kad ji skaičiuotų ne įvedamų skaičių sumą, o sandaugą. Tam reikia sumos kaupimo operatorių pakeisti sandaugos kaupimo operatoriumi $s:=s*x$, sandaugai s suteikti pradinę reikšmę 1 ir pakeisti pranešimą apie rezultatus.

3.3 pratimas. Papildykite programą, kad ji suskaičiuotų kiek buvo įvesta teigiamų ir kiek neigiamų skaičių. Tam reikia turėti papildomus du kintamuosius, pavyzdžiui **kt** teigiamų skaičių kiekiui saugoti ir **kn** neigiamų skaičių kiekiui saugoti. Kiekio skaičiavimas nuo sumos kaupimo skiriasi tik tuo, kad kiekio reikšmė didinama vienetu kiekvienu atveju, kada yra surandama tinkama reikšmė. Programą Suma ciklo viduje reikia papildyti sakiniu, kurio algoritmas toks:

3.4. Užduotis savarankiškam darbui.

Klaviatūra įvedamos skaičių poros. Pirmasis skaičius reiškia prekės kainą, o antrasis tos prekės pirkėjo pasirinktą kiekį. Įvedamų skaičių srauto pabaiga nurodoma dviem skaičiais, lygiais nuliui. Reikia sudaryti programą, kuri suskaičiuotų, kiek pirkėjas privalo sumokėti pinigų už savo pirkinius. Išvesti į ekraną suskaičiuotą sumą ir kiek skirtingų prekių pasirinko pirkėjas (kiek skaičių porų buvo įvesta).

2.4 tema. Didžiausios ir mažiausios reikšmės paieška skaičių sraute

Tai dažnai sprendžiami uždaviniai. Populiariausias yra toks jų sprendimo būdas. Deklaruojami kintamieji įvedamai (**x**) ir didžiausiai (**d**) reikšmėms saugoti; skaitoma pirmoji reikšmė ir padaroma prielaida, kad ji yra didžiausia (jei ieškoma didžiausios reikšmės):

d := x.

Po to skaitomos kitos reikšmės ir lyginamos su **d**. Jei randama didesnė, kintamojo **d** saugoma reikšmė keičiama nauja: *if x > d then d := x*. Taip apdorojus visą įvedamą srautą, kintamasis **d** saugos didžiausią įvestą reikšmę.

Pradžia Maksimumas		
x, d realūs		
Įvesti x reikšmę		
d := x;		
kol x <> 0		
t	x > d	n
d := x;		
Įvesti x reikšmę		
Išvesti didžiausią reikšmę d		
Pabaiga		

- Surinkite ir patikrinkite didžiausios reikšmės atrinkimo iš klaviatūra įvedamo srauto programą Maksimumas.

•

program Maksimumas;

var x, d : real; { Įvedama ir didžiausia reikšmės }

begin

WriteLn('Maksimumo paieškos programa.');

'Darbas nutraukiamas įvedus 0.');

WriteLn('Įveskite x: '); ReadLn(x);

d := x; { Pradinė

reikšmė }

while (x <> 0) do begin { Įvedimo ir analizės

ciklas }

if x > d then d := x; { Maksimumo

paieška }

WriteLn('Įveskite x: '); ReadLn(x);

```

end;
Pranešimas apie rezultatus      }
    WriteLn('Didžiausia reikšmė: ', d:8:2);
    ReadLn;                      }
Programos stabdymas            }
end.

```

- Pertvarkykite didžiausios reikšmės paieškos programą taip, kad ji ieškotų mažiausios reikšmės įvedimo sraute.

Ne visuomet patogiu pirmąją reikšmę priimti pradine didžiausia reikšme. Ypač tuomet, kai ta reikšmė turi būti skaičiuojama (pavyzdžiui, ieškant didžiausios funkcijos reikšmės nurodytame argumentų kitimo intervale). Tokiu atveju galima pradine didžiausia reikšme priimti pakankamai mažą reikšmę, tokią, kuri tikrai būtų mažesnė už visas galimas reikšmes, tarp kurių ieškoma didžiausia. Cikle analizuojamos visos reikšmės

. Ieškant mažiausios reikšmės, pradine reikšme reikia priimti pakankamai didelį skaičių.

Pavyzdys. Klaviatūra įvedami **n** realių skaičių. Reikia surasti mažiausią skaičių. Parašykite programą pagal duotą algoritmą
 Minimumas, suderinkite programą ir patikrinkite su duomenimis, nurodytais 4.1 lentelėje.

4.1 lentelė.

n reikšmė	x reikšmės	Rezultatas
4	8 3 -5 32	-5
3	15000 28000 39000	nebuvo
0		nebuvo
1	8	8
3	18 15230 -8	-8

4.1. Užduotis savarankiškam darbui.

Sudarykite didžiausios reikšmės paieškos programos modifikaciją, kuri ne tik rastų didžiausią reikšmę, bet ir jos eilės numerį įvedimo sraute. Tam reikia papildomai įvesti įvedamų skaičių skaitiklį n , jam suteikti pradinę reikšmę 0 ir, įvedus kiekvieną skaičių, didinti šią reikšmę vienetu ($n := n + 1$). Kiekių bei eilės numerių skaičiavimui paprastai yra naudojami sveiko tipo kintamieji, kurių tipas deklaravimo sakiniuose nurodomas baziniu žodžiu *integer*. Ieškant didžiausios reikšmės vietos, reikės įsiminti ne tik dalinius maksimumus, bet ir jų numerius. Taip pat turės būti pakeistas ir pranešimas apie rezultatus.

12 tema. Programos struktūriniai elementai. (pažintis su procedūra)

12.1. Procedūros

Procedūromis yra vadinami programos struktūriniai elementai, į kuriuos galima kreiptis daugelį kartų. Jos taip pat padeda geriau struktūrizuoti programas, padaro jas lengviau skaitomas ir analizuojamas. Procedūroms yra skiriamos dviejų tipų sintaksinės struktūros: aprašymai ir kreipiniai. Aprašymai nurodo, ką procedūra turi atlikti. Vykant programą, jie praleidžiami. Procedūrose aprašyti veiksmai yra vykdomi tiksliai tada, kai į jas kreipiamasi. Procedūrų aprašymai gali būti bet kurioje aprašų vietoje, tačiau programa bus aiškesnė, jei jie bus rašomi už kintamuosius deklaruojančių sakinių. Procedūrų aprašo sintaksė :

```
procedure <Vardas>[(<Formalių parametrų aprašai>)];  
    [<Lokalinių struktūrų aprašai>]  
begin  
    <Sudėtinis procedūros operatorius>  
end;
```

Sudėtinis procedūros sakinyss aprašo jos vykdomus veiksmus, o parametrai yra skirti informaciniais ryšiams su programa palaikyti. Lokaliniai aprašai deklaruoja procedūros viduje vartojamas struktūras.

Į procedūras kreipiamasi jų vardais, už kurių, lenktiniuose skliaustuose, gali būti surašomi faktiniai parametrai:

<Procedūros vardas>[(<Faktinių parametrų sąrašas>)];

Faktiniai parametrai nurodo, kokiais pagrindinės programos duomenimis turi būti keičiami formalūs parametrai. Parametrų keitimo būdas priklauso nuo jų tipo. Formalūs parametrai - kintamieji, kurių aprašai žymimi žodeliu *var*, yra keičiami kreipinyje nurodytais pagrindinės programos kintamaisiais. Tokie parametrai yra naudingi tada, kai pageidaujama, kad procedūra pakeistų pagrindinės programos kintamųjų reikšmes.

Kai norime apsaugoti pagrindinės programos kintamuosius nuo pakeitimų procedūrų darbo metu yra vartojami parametrai - reikšmės, kurių aprašuose žodelis *var* nenurodomas. Tada duomenys perduodami tik viena kryptimi - iš besikreipiančios struktūros į procedūrą. Įvykdžius procedūrą, tokių faktinių parametrų nurodomų kintamųjų reikšmės nepakinta. Be to, faktiniais parametrais - reikšmėmis gali būti ne tik kintamieji, bet ir konstantos bei išraiškos.

Būtina žinoti, kad failo kintamieji gali būti tik parametrais - kintamaisiais.

12.2. Pratimas.

Patikrinkite ir išanalizuokite 11.1 skyrelio programos Keitimai modifikaciją, kurioje skaitmenų perrašymui priešinga tvarka vartojama pagalbinė procedūra Apsuk:

```
program Keitimai;
var    x: longint;                      (* Įvedamas
skaičius                               *)
      c: char;                          (*
Dialogui skirtas kintamasis          *)
procedure Apsuk (var x: longint);      (* Procedūros argumentas
x nurodo tvarkomą *)
      rezultata;                       (*    skaičių ir grąžina
      var      y: longint;             (*    *)
skaičius                               (* Pertvarkytas
```

```

                                s: byte;                (* Tvarkomas
skaitmuo                                *)
                                begin
                                    y:= 0;                (* Pradinė
reikšmė                                *)
                                while x<>0 do begin        (* Ar dar yra skaitmenų?
                                    *)
                                        s:= x mod 10;      (* Jauniausios
skilties atskyrimas                *)
                                        y:= y*10+s;        (* Rezultato papildymas
                                    *)
                                        x:= x div 10;      (* Skilties atmetimas
                                    *)
                                end;
                                x := y;                    (* Gražinama
eikšmė                                *)
                                end;
(* PAGRINDINĖ (VYKDOMA) PROGRAMOS DALIS
*)
begin
    repeat
        WriteLn ('Įveskite skaičių');    ReadLn(x);
        Apsuk (x);
        WriteLn('Apsuktas skaičius:', x);

        (* Ciklo valdymo užklausa                *)
        WriteLn ('Ar tęsti (t/n)?');      ReadLn
(c);
        until (c='n') or (c='N');          (* Pabaigos sąlyga
*)
    end.

```

- Pertvarkykite procedūrą Apsuk taip, kad ji nekeistų jai perduodamo argumento x reikšmės, o rezultatą grąžintų kito parametro y pagalba:

```

procedure Apsuk (x: longint,    var y: longint);
    (*      Procedūros parametras-reikšmė x
perduodą tvarkomą *)

```

```

(* reikšmę, o rezultato formavimui ir
grąžinimui skirtas *)
(* parametras-kintamasis y
*)
var s: byte; (* Tvarkomas
skaitmuo *)
begin
    y:= 0; (* Pradinė
reikšmė *)
    while x > 0 do begin (* Ar dar yra skaitmenų?
        *)
            s:= x mod 10; (* Jauniausios
skilties atskyrimas *)
            y:= y*10+s; (* Rezultato papildymas
            *)
            x:= x div 10; (* Skilties atmetimas
            *)
        end
    end;
end;

```

- Savarankiškai pertvarkykite pagrindinę programą taip, kad ji galėtų vartoti pakeistą procedūrą Apsuk.
- Savarankiškai pritaikykite programą tekstiniam faile surašytų skaičių pertvarkymui. Iš failo skaitomi ir apskukti skaičiai turi būti parodomi ekrane. Neužmirškite parengti programai pritaikytą pradinių duomenų failą!
- Savarankiškai sudarykite programą, kuri iš pradinių duomenų failo atrinktų ir parodytų ekrane tik tuos skaičius kuriuose yra skaitmenų 5. Kiekvienam tokiame skaičiui taip pat turi būti parodomas jame esančių penketų skaičius. Programoje siūloma vartoti pagalbinę procedūrą Kiek:

```

procedure Kiek ( x: longint; var n : integer);
(* Parametras x perduoda analizuojamą
skaičių, *)
(* o n praneša apie penketų kiekį
*)
var
    s: integer; (* Analizuojamas
skaitmuo *)

```

```

begin
    n:= 0; (* Pradinė
skaitiklio reikšmė *)
    while x > 0 do begin (* Sąlyga "kol yra skaitmenų"
    *)
        s:= x mod 10; (* Jauniausios skilties
atskyrimas *)
        if s= 5 then (* Skaitmens analizė
        *)
            n:= n+1; (* Skaičiavimas
            *)
            x:= x div 10; (* Skilties atmetimas
            *)
        end;
    end;
end;

```

12.3. Savarankiško darbo užduotis.

Pertvarkykite procedūrą **Kiek** taip, kad ji skaičiuotų ne kiek yra penketų, o kiek yra mus dominančio skaitmens atkartojimų skaičiuje. Norint tai padaryti, procedūroje reikia įvesti naują parametą-reikšmę, kuris nurodytų tiriamą skaitmenį. Sudarykite skaičių failo analizės programą, kuri parodytų ekrane duomenis apie skaičius su dialogo metu nurodytais skaitmenimis. Analizės ciklai turi būti kartojami tol, kol vartotojas nurodys, kad gana, įvesdamas jūsų numatytą pabaigos žymę.

3. Skaičiai ir simboliai

Šiame skyriuje:

- pažintis su ekrano valdymo priemonėmis;
- duomenų saugojimas tekstiniam faile;
- skaičiai ir veiksmai su jais;
- simboliai.

3.1. Spalvos, ekrano langai, data, laikas

Ekranų valdymo priemonės

Kiekviena programa turi ne tik realizuoti sprendžiamo uždavinio algoritmą, bet ir būti vaizdi, lengvai valdoma. Programos valdymo ir jos pateikiamų duomenų vaizdavimo priemonės bei būdai yra vadinami programos išoriniu interfeisu. Išorinio interfeiso sudarymo priemonės priklauso nuo kompiuterio įrangos savybių ir jos nuolat yra tobulinamos, kinta. Todėl šių priemonių aprašymui skirtos konstrukcijos dažniausiai priklauso ne pačioms programavimo kalboms, bet jų išplėtimams, papildymams, kurie pateikiami pagalbinių priemonių rinkiniuose - bibliotekose. Turbo Paskalio aplinkoje yra labai efektyvi ir paprasta IBM PC tipo kompiuterių ekranų vaizdo valdymo biblioteka **Crt**. Norint ja naudotis, reikia programos pradžioje įrašyti sakinį **uses Crt;**

Dažniausiai vartojamos bibliotekos procedūros ir funkcijos:

function ReadKey : char; Klavišo simbolio apklausa.

procedure GoToXY (x, y : integer);

Žymeklis perkeliamas į koordinatėmis x,y nurodomą ekrano vietą.

procedure ClrScr; Ekrano valymas.

function WhereX: integer; Žymeklio ekrane koordinatė X.

function WhereY: integer; Žymeklio ekrane koordinatė Y.

procedure ClrEol; Valoma aktyvios eilutės pabaiga.

procedure TextColor(spalva : byte);

Teksto spalva: spalvos kodas arba konstanta.

procedure TextBackground(spalva : byte)

Fono spalva: spalvos kodas arba konstanta.

procedure Window(x1, y1, x2, y2 : integer);

Aktyvaus lango aprašymas: (x1,y1) kairiojo viršutinio
ir (x2,y2) dešinio apatinio lango kampų koordinatės.

function KeyPressed : boolean;

Praneša, kad paspaustas bet koks klavišas.

var TextAttr: word;

Kintamasis, kuris saugo duomenis apie spalvinį ekrano režimą.

Norint gauti išsamesnių žinių apie kurią nors iš išvardintų procedūrų ir funkcijų arba kitą Turbo Paskalio kalbos struktūrą, reikia redaktoriaus lange surinkti struktūros pavadinimą, prie jo perkelti kursorių ir paspausti klavišų pora **Ctrl+F1**. Po to ekrane bus matoma pagalbinė informacija apie kursoriaus nurodomą struktūrą.

Nurodant kursoriaus ir langų koordinates, yra laikoma, kad koordinatinių sistemos pradžia yra kairiajame viršutiniame ekrano kampe. X ašis nukreipta į dešinę, o Y - žemyn. Ekranas skaidomas į simbolių rašymui skirtus langelius, kurių dydis priklauso nuo ekrano darbo režimo. Standartinis ekrano dydis **80x25** langelių. Koordinatinių sistemos pradžios langelio koordinatės yra (1,1).

Ekrano darbo spalvos yra nurodomi skaitmeniniais kodais arba modulio **Crt** konstantomis:

1 lentelė. Spalvas aprašančios konstantos

Vardas	Reikšmė	Vardas	Reikšmė
Black	0	DarkGray	8
Blue	1	LightBlue	9
Green	2	LightGreen	10
Cyan	3	LightCyan	11
Red	4	LightRed	12
Magenta	5	LightMagenta	13
Brown	6	Yellow	14
LightGray	7	White	15

Spalvų pulsavimą aprašo konstanta **Blink = 128**.

Simbolių, fono spalvas ir ekrano režimą galima keisti tik tada, kai pageidaujamus režimus palaiko kompiuteryje esančio ekrano kontroleris (valdymo įtaisas). Yra ir kitų ribojimų. Pavyzdžiui, foną galima nurodyti tik pirmais 8 spalvų kodais: 0..7. Jeigu prie teksto spalvos kodo pridėsime

konstantą Blink, pavyzdžiui: `TextColor(Red + Blink)`, ekrane bus rašomas pulsuojančio ryškumo tekstas.

Dažant ekraną, turi būti aprašoma fono spalva:

```
(TextBackground(<fonas>))
```

ir vykdoma ekrano valymo procedūra `ClrScr`.

Ekrano spalvas keičiančiose programose reikia įsiminti kintamojo `TextAttr` saugomas spalvas ir programos darbo pabaigoje jas atstatyti.

1 pratimas. Langas ekrane dažomas nurodyta spalva. Tekstas rašomas nurodyta spalva. Programoje yra panaudotas duomenų tipas `string`, kuris bus vėliau aiškinamas. Čia Jūs galite jo nežinoti ir nagrinėti programos tekstą praleisdami mįslės įminimo analizę.

```
program Pratimas1;
uses Crt;
{-----}
{ Ekrane piešiamas langas ir dažomas nurodyta fono spalva      }
procedure Langas( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas );  ClrScr;
end;
{-----}
var  Ekranas, Fonas, Tekstas : integer;
      kas : string;           { Simbolių eilutė }
begin
    Ekranas := TextAttr;           {Spalvų įsiminimas }
                                   { Ekrano paruošimas darbui}
    TextBackground( White );  ClrScr;
    TextColor( Black );
    Fonas := 1; { Darbo pabaiga nurodoma Fono nuline reikšme }
    while Fonas <> 0 do  begin
        Langas( 5, 2, 75, 8, Fonas );{ Dialogo langas}
        GoToXY( 10, 2 );
        Write('Nurodykte lango fono spalvos ',
              'koda (1..7); pabaiga- 0 = ');
        ReadLn( Fonas );
        if Fonas <> 0 then  begin { Ar darbo pabaiga }
            GoToXY( 10, 5 );
            Write('Nurodykte teksto spalvą (0..15) = ');
```

```

ReadLn( Tekstas );
if Tekstas = Fonas then      begin
    WriteLn( 'Spalvos vienodos.',
              ' Teksto spalva bus:', Fonas+1);
    Tekstas := Fonas + 1;    end;
{                               Spalvų demonstracija }
Langas( 10, 10, 70, 20, Fonas);
TextColor( Tekstas );
GoToXY( 10, 5 );              { Mįslė }
WriteLn('Rainas, su uodega ir usais,');
WriteLn('vaiksto kada nori ir kur nori. ');
WriteLn(' Kas? (Parašykite žodį) ');
ReadLn( kas );                 { Įvedamas žodis }
if kas = 'Katinas'
    then WriteLn( 'Puiku! Atspejote')
    else WriteLn( 'Dar neatspejote' );
end; end;
Window( 1, 1, 80, 25 );      {Ekrano pradinės spalvos }
TextAttr := Ekranas; ClrScr;
end.

```

- Mįslės lange užrašykite pasirinktų spalvų kodus. Jeigu galite-pavadinimus.
- Fono spalvos kodas nekontroliuojamas. Papildykite programą sakiniiais, kuriais apribotumėte fono spalvą 1-7 ribose ir, nurodžius neteisingai, spalvą parinktų programa, pavyzdžiui, mėlyną spalvą.

Data ir laikas.

Naudinga programos darbo rezultatams, kurie bus saugojami dokumentų formoje, nurodyti jų formavimo datą ir laiką. Patogu ir kartu malonu, kai programa, pradėdama darbą praneša datą, parodo drabo pradžios ir pabaigos laikus, pasako kiek laiko truko darbas. Kompiuterio laikrodžio parodymus galima gauti Dos bibliotekoje esančiomis procedūromis:

```

procedure GetDate( metai, mėnuo, diena,
                   savaitės_diena : word );

```

Čia savaitės dienos reikšmė nulis nurodo sekmadienį. Duomenų tipas word nurodo sveiką teigiamą skaičių.

```

procedure GetTime( valanda, minutė, sekundė,
                   sekundės_šimtoji_dalis : word);

```

Bibliotekoje Dos yra ir daugiau naudingų priemonių, kurių dalį panaudosime vėliau. Informaciją apie jas galima gauti HELP būdu arba literatūroje.

2 pratimas. Ekrano lange parašoma programos darbo pradžios data ir laikas. Atsimenama žymeklio lange vieta, darbui pasibaigus, tame lange toliau bus parašomas darbo pabaigos laikas. Programa susumuoja skaičius nuo 1 iki 100.

```
program Pratimas2;
uses Crt, Dos;
{-----}
procedure Langas( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas );  ClrScr;
end;
{-----}
procedure Pradzia( var x, y : integer );
var a, b, c, d : word;
begin
    GetDate( a, b, c, d );
    Write('Dabar yra:', a:4, '.', b:2, '.', c:2, '.');
    if d = 1 then WriteLn(' Pirmadienis');
    if d = 2 then WriteLn(' Antradienis');
    if d = 3 then WriteLn(' Trečiadienis');
    if d = 4 then WriteLn(' Ketvirtadienis');
    if d = 5 then WriteLn(' Penktadienis');
    if d = 6 then WriteLn(' Šeštadienis');
    if d = 0 then WriteLn(' Sekmadienis');
    GetTime( a, b, c, d );
    WriteLn( 'Darbo pradžia:', a:2, ' val.',
              b:2, ' min.', c:2, ':', d:2, ' sek. ');
    x := WhereX; y := WhereY;      { Žymeklio vieta lange }
end;
{-----}
var Ekranas : integer;
    Xz, Yz,           { Žymeklio vieta datos lange }
    Sx, Sy : integer; { Skaičiaus vieta lange }
    Suma, skaicius : integer;
    val, min, sek, sek100 : word;
begin
```

```

Ekranas := TextAttr;
Langas( 1, 1, 80, 25, White );  ClrScr;
TextColor( Black );
Langas( 5, 2, 70, 8, Green );
Pradzia( Xz, Yz );
      {----- Skaičių sumavimas -----}
Langas( 5, 10, 50, 20, Magenta);
GoToXY( 5, 5 );
Write( 'Sudedu skaičius nuo 1 iki 100: ');
Sx := WhereX;  Sy := WhereY;
Suma := 0; Skaicius := 1;
while Skaicius < 100 do begin
    GoToXY( Sx, Sy);  ClrEol;
    Write( Skaicius );    Delay( 500 );
    Suma := Suma + Skaicius;
    Skaicius := Skaicius + 1;
end;
GoToXY( 7, 10 ); Write( 'Suma = ', Suma );
      {---- Darbo pabaigos laikas ----}
Window( 5, 2, 70, 8 );
GoToXY( Xz, Yz+1 );
GetTime( val, min, sek, sek100 );
WriteLn( 'Darbo pabaiga:', val:2, ' val.', min:2,
        ' min.', sek:2, ':', sek100:2, ' sek100.' );
ReadLn;
      {----- Ekrano pradinės spalvos ---}
Window( 1, 1, 80, 25);
TextAttr := Ekranas;  ClrScr;
end.

```

- Datos lange, pasibaigus darbui, užrašykite darbo trukmę sekundėmis.
- Sumuojamus skaičius lange rašykite raudona spalva.
- Skaičiams rašyti sukurkite atskirą langą.

3 pratimas. Klaviatūra įvedinėjami skaičiai, ne didesni kaip 80. Kiekvienas iš jų yra ekrane pavaizduojamas atitinkamo ilgio spalvota juosta, kurios viduryje parašomas įvestas skaičius. Juosta trijų eilučių pločio. Skaičių įvedimo pabaiga nuodoma skaičiumi nulis arba neigiamu skaičiumi. Ekrane formuojamas langas, kuriame rašoma data ir laikas. Atskiras langas dialogui. Skaičius vaizduojamas langu, kuriam skiriamos

trys eilutės, o horizontalus ilgis lygus skaičiaus reikšmei.

```
program Pratimas3;
uses Crt, Dos;
{-----}
procedure Langas ( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas );  ClrScr;
end;
{-----}
procedure Pradzia( var x, y : integer );
var a, b, c, d : word;
begin
    GetDate( a, b, c, d );
    Write('Dabar yra:', a:4, '.', b:2, '.', c:2, '.');
    if d = 1 then WriteLn(' Pirmadienis');
    if d = 2 then WriteLn(' Antradienis');
    if d = 3 then WriteLn(' Trečiadienis');
    if d = 4 then WriteLn(' Ketvirtadienis');
    if d = 5 then WriteLn(' Penktadienis');
    if d = 6 then WriteLn(' Šeštadienis');
    if d = 0 then WriteLn(' Sekmdadienis');
    GetTime( a, b, c, d );
    WriteLn( 'Darbo pradzia:', a:2, ' val.',
            b:2, ' min.', c:2, ':', d:2, ' sek');
    x := WhereX;  y := WhereY;
end;
{-----}
procedure Darbas;
var Skaicius : integer;
begin
    Skaicius := 2;                { Fiktyvi pradinė reikšmė: }
    while Skaicius > 0 do begin
        {----- Dialogo langas -----}
        Langas( 5, 10, 75, 14, Yellow);
        TextColor( Black );  GoToXY( 15, 2 );
        Write(
            'Darbo pabaiga- 0 arba neigiamas skaičius');
        GoToXY( 5, 4 );
        Write('Iveskite skaiciu (tinka<80) = ');
        ReadLn( Skaicius );
    end;
end;
```

```

if Skaicius > 80 then begin                                { Kontrolė }
    Langas( 5, 10, 75, 14, Magenta);
    TextColor( Red );
    GoToXY( 5, 2 );
    WriteLn( 'Skaicius didelis. Netinka');
    GoToXY( 15, 3 );
    TextColor( Green );
    WriteLn( 'Pakartokite ( ENTER )');
    ReadLn;
end
else if Skaicius > 0 then begin
    { Skaičiaus vaizdavimas spalvotu langu }
    Langas( 1, 16, Skaicius, 18, Cyan );
    { Žymeklis perkeliamas į lango vidurį }
    GoToXY( Trunc( Skaicius / 2 ), 2);
    Write( Skaicius );                                { Parašomas skaičius}
    ReadLn;
    Langas( 1, 16, Skaicius, 18, White);
end;
end;    end;
{-----}
procedure Pabaiga( x, y : integer);
var val, min, sek, sek100 : word;
begin
    { Datos langas padaromas aktyviu }
    Window( 5, 2, 70, 8 );
    GoToXY( X, Y+1 );
    TextColor( Black );
    GetTime( val, min, sek, sek100 );
    WriteLn( 'Darbo pabaiga:', val:2, ' val.',
        min:2, ' min.', sek:2, ':', sek100, ' sek.' );
    TextBackground( Green );
    GoToXY( 50, 7); Write( 'ENTER'); ReadLn;
end;
{-----}
var Ekranas : integer;
    Xz, Yz, Sx, Sy : integer;
    Suma, skaicius : integer;
    val, min, sek, sek100 : word;
begin
    Ekranas := TextAttr;
    Langas( 1, 1, 80, 25, White ); TextColor( Black );
    Langas( 5, 2, 70, 8, Green );

```

```

Pradzia( Xz, Yz );
Darbas;
Pabaiga( Xz, Yz );
Window( 1, 1, 80, 25);
TextAttr := Ekranas;  ClrScr;
end.

```

- Papildykite programą veiksmiais, kuriais vartotojas galėtų nurodyti vaizduojamojo skaičiaus juostos spalvą ir skaičiaus užrašymo spalvą.
- Programoje dialogo langą naikinkite, kai ekrane rodoma skaičiaus juosta.
- Skaičiaus juostą pieškite ekrano centre.
- Skaičiaus juostą keiskite artimiausiu kvadratui stačiakampiu, t.y. mažesnioji kraštinė $a = \text{Round}(\text{Sqrt}(\text{Skaičius}))$, o didesnioji $b = a$, kai $a^2 \geq \text{Skaičius}$ ir $b = a+1$ priešingu atveju.

Užduotys savarankiškam darbui.

- Sudarykite sveikųjų skaičių analizės programą, kuri skaitomus teigiamus skaičius pavaizduotų ekrane mėlynomis, o neigiamus - raudonomis juostomis. Juostų ilgis turi būti proporcingas skaičių moduliams.
- Sudarykite programą, kuri kompiuterio ekrane pieštų atsitiktinio dydžio langus ir juos dažytų atsitiktine spalva. Numatykite darbo pabaigos sąlygą. Ekrane matysite įvairiaspalvę stačiakampių mozaiką.

3.2. Skaičių srautų analizės uždaviniai.

Srautų analizė.

Skaičių srautų analizės uždaviniams priskiriami uždaviniai, kuriais nusakomos srautų charakteristikos ir jų sudėtis. Pavyzdžiui, gali būti skaičiuojama, kiek skaičių telpa nurodytoje atkarpoje, kokia yra jų suma, koks yra šios sumos santykis su visų skaičių suma, ieškoma didžiausio skaičiaus ir kiek tokių yra, teigiamų skaičių aritmetinis vidurkis ir panašiai. Sprendžiant tokius uždavinius, kiekvienai skaitomai srauto reikšmei yra taikomi du veiksmi: filtravimas ir apdorojimas. Filtruojant yra nustatoma, ar reikšmė priklauso tiriamai duomenų grupei, o skaičiuojant yra

nustatomos įvairios tiriamos grupės charakteristikos.

Paprastos filtravimo sąlygos yra aprašomos santykiais, kurie rašomi if tipai sakiniuose. Sudėtingesnėms sąlygoms aprašyti tokių priemonių nepakanka, tenka iš santykių ir kitokių loginių dydžių formuoti logines išraiškas. Jose loginiams dydžiams gali būti taikomos šios operacijos:

not (*neigimas*),

or (veiksmas *arba*),

and (veiksmas *ir*),

xor (griežtas *arba*).

Loginių išraiškų reikšmės gali būti suteikiamos loginiams kintamiesiems, kurie aprašomi baziniu žodžiu boolean. Logines operacijas apibūdina 1-a lentelė. Sudarant logines išraiškas, reikia žinoti, kad loginio neigimo, kaip ir kitų unarinių operacijų prioritetas yra aukščiausias, operacijos **and** prioritetas atitinka * ir / prioritetus, o operacijos **or** - veiksmų + ir - prioritetus. Natūralią veiksmų tvarką galima pakeisti skliaustais.

Pavyzdžiui, kintamojo x priklausymas atkarpai [5, 15] gali būti aprašomas taip: $(x \geq 5)$ and $(x \leq 15)$. Šios atkarpos išorę aprašo išraiška $(x < 5)$ or $(x > 15)$.

1 lentelė. Loginės operacijos

a	b	not a	a or b	a and b	a xor b
true	true	False	true	true	false
true	false	False	true	false	true
false	true	True	true	false	true
false	false	True	false	false	false

4 pratimas. Programa skaičiuoja, koks klaviatūra įvedamų skaičių procentas patenka į atkarpą [5, 15].

Duomenų pabaigą nurodo nulinė reikšmė. Skaičių kiekių skaičiavimui yra skiriami integer tipo kintamieji (skaitikliai), kurie gali turėti tik sveikąsias reikšmes.

```
program Pratimas4;  
uses Crt;
```

```

const a = 5;    b = 15;
{-----}
procedure Langas( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas );  ClrScr;
end;
{-----}
procedure Pranesimas;
    var Spalvos : byte;
begin
    Spalvos := TextAttr;                { Išsaugojamos spalvos}
    Langas( 20, 2, 75, 9, Blue);
    TextColor( Green );
    WriteLn(' Įvedamų skaičių analizė:');
    WriteLn('Rasime, kiek skaičių bus',
            ' intervale [' , a:3, ', ', b:4, ' ]');
    WriteLn(' ir koks jų kiekis procentais');
    TextColor( Red );
    WriteLn('Srauto pabaiga - nuline reiksmė');
    TextColor( Magenta );
    WriteLn('Iveskite skaičius ',
            ' (galima kelis eilutėje:');
    WriteLn(' pirmasis nelygus nuliui');
    TextAttr := Spalvos;                { Atstatomos spalvos }
end;
{-----}
procedure Darbas( var visi, kiek: integer);
var Skaicius : integer;
    Xt, Yt, Xs, Ys : integer;
begin
    visi := -1;                        { Skaičių skaitliukai}
    kiek := 0;
    Langas( 45, 11, 75, 20, Magenta);
    WriteLn( ' Tinkami skaičiai: ');
    Xt := WhereX;  Yt := WhereY;
    Langas( 5, 11, 40, 20, Green);
    WriteLn(' Jūsų įvedami skaičiai:');
    Skaicius := 2;
    while Skaicius <> 0 do begin
        Read( Skaicius );
        visi := visi + 1;
    end;
end;

```

```

Xs := WhereX;  Ys := WhereY;
if (Skaicius >= a) and (Skaicius <= b)
then begin
    kiek := kiek + 1;
    Window( 45, 11, 75, 20);
    TextBackground( Magenta );
    GoToXY( Xt, Yt );  Write( Skaicius:5 );
    Xt := WhereX;  Yt := WhereY;
end;
Window( 5, 11, 40, 20); TextBackground( Green );
GoToXY( Xs+1, Ys );
end;
ReadLn;
end;
{-----}
procedure Pabaiga( Sk, k : integer);
begin
    Langas( 3, 2, 77, 9, Black);
    TextColor( White );
    WriteLn( 'Analizes rezultatai:');
    WriteLn( 'Buvo ivesta skaiciu:', Sk:5);
    if k > 0 then begin
        WriteLn( 'Atkarpos [', a:3, ',', b:3,
            ' ] viduje skaiciu yra:', k:3);
        Write ( ' Tai sudaro ',
            ( k / Sk ) * 100 :6:2, ' % ivestu skaiciu');
        end
        else
        WriteLn( 'Atkarpos [', a:3, ',', b:3,
            ' ] viduje skaiciu nera');
    WriteLn;
    TextColor( Green );
    Write( 'Darbo pabaiga: ENTER '); ReadLn;
end;
{-----}
var  Ekranas : integer;
     Visi, Kiek : integer;
begin
    Ekranas := TextAttr;
    Langas( 1, 1, 80, 25, White );  ClrScr;
    TextColor( Black );
    Pranesimas;

```

```

Darbas( Visi, Kiek);
Pabaiga( Visi, Kiek );
Window( 1, 1, 80, 25);
TextAttr := Ekranas; ClrScr;
end.

```

- Pertvarkykite pratimo programą taip, kad ji skaičiuotų į atkarpos [5,15] vidų patenkančių skaičių sumą ir vidutinę reikšmę.
- Savarankiškai sudarykite programą, kuri įvedamo srauto duomenis pertvarkytų pagal formulę $y := \text{Sqrt}(100 - x * x)$. Sudarykite filtrą, kuris skaičiavimams nukreiptų tik tas reikšmes, kurioms pertvarkymo formulė yra apibrėžta. Įvedus kitokias reikšmes turi būti formuojamas pranešimas, kad šioms reikšmėms skaičiavimų formulės negalima taikyti.

Savarankiško darbo užduotis

Klaviatūra įvedamų sveikų skaičių sekos pabaiga yra nulis. Reikia parašyti programą, kuri suskaičiuotų kiek buvo vienženklių, dviženklių, triženklių, keturženklių ir kitokių skaičių. Vėliau papildykite programą veiksmiais, kuriais būtų suskaičiuojamas įvestų skaičių aritmetinis vidurkis.

3.3. Tekstiniai failai

Sprendžiant uždavinius, dažnai yra pageidaujama atskirti pradinių duomenų parengimo, jų apdorojimo ir rezultatų analizės procesus. Tai galima padaryti surašant pradinius duomenis į failus ir parengiant šių failų apdorojimo programas ir rezultatus vėl rašant į failus. Programos aprašų dalyje kiekvienam failui yra skiriamas failo kintamasis, kurio aprašo sintaksė:

<Vardas>: <Failo tipas>

Universalieji yra tekstiniai failai, kurių tipas žymimas baziniu žodžiu `text`. Šie failai yra sudaromi iš į eilutes sugrupuotų simbolių. Eilučių pabaigos žymimos specialiais eilutės pabaigos simboliais, o failo pabaigą žymi failo pabaigos simbolis. Siunčiant parengtą failą į diską, šis simbolis įrašomas automatiškai. Ekrane eilučių ir failo pabaigos simboliai nerodomi.

Programos pagrindinėje dalyje kiekvienas failo kintamasis turi būti susiejamas su konkrečiu operacinės sistemos (OS) failu. Tai nurodoma procedūra:

```
procedure Assign( FailoKintamasis, 'FailoVardas' )
```

Čia FailoVardas yra rašomas tarp apostrofų, kaip tekstas, arba nurodomas konstantos vardu, pavyzdžiui:

```
const Fv = 'Duum.dat';  
var F : text;  
Assign( F, 'duom.dat' );    Assig( F, Fv );
```

Failo vardas yra sudaromas pagal aptarnaujančioje operacinėje sistemoje numatytas taisykles ir yra suteikiamas failui kuriant jį redaktoriumi. Programoje operacinės sistemos failo vardas tiesiogiai nurodomas tik procedūroje Assign, o po to jis yra atsovaujamas failo kintamojo. Po to failas turi būti parengiamas konkrečiai apdorojimo operacijai:

```
skaitymui    procedure Reset ( FailoKintamasis );  
rašymui      procedure Rewrite ( FailoKintamasis );  
  
Duomenys iš failų yra skaitomi panašiai kaip iš klaviatūros:  
Read( FailoKintamasis, KintamųjųSąrašas );  
ReadLn( FailKintamasis, KintamųjųSąrašas );
```

Iš tekstinių failų skaitomi skaičiai turi būti atskirti tarpais, tabuliacijos arba eilutės pabaigos simboliais. Procedūra Read iš tekstinio failo visuomet skaito tiek reikšmių, kiek yra elementų jo kintamųjų sąrašas. Procedūra ReadLn skiriasi tuo, kad jis iš pradžių perskaito visą eilutę ir tik po to skirsto jos duomenis kintamiesiems. Jei eilutėje duomenų būna per daug, pertekliniai duomenys atmetami. Jei jų trūksta, skaitoma kita failo eilutė ir reikšmių skirstymas kintamiesiems tęsiamas.

Išvedimui į failus naudojamos procedūros:

```
procedure Write( FailoKintamasis, DuomenųSąrašas );  
procedure WriteLn( FailoKintamasis, DuomenųSąrašas );
```

Jie savo savybėmis yra analogiški atitinkamoms išvedimo į ekraną procedūroms. Procedūra WriteLn nuo Write skiriasi tuo, kad jis išvedamų duomenų sąrašą papildo specialiais eilutės pabaigos simboliais.

Kai failų apdorojimas baigiamas, juos būtina uždaryti:

```
procedure Close( FailoKintamasis )
```

Skaitant failus, loginėmis funkcijomis:

```
function EoLn( FailoKintamasis ): boolean;
```

```
function Eof( FailoKintamasis): boolean
```

galima tikrinti atitinkamai eilutės ir failo pabaigos žymes. Šioms funkcijoms yra suteikiama reikšmė true nuskaičius paskutinį duomenų elementą eilutėje arba faile atitinkamai.

5 pratimas. Reikia susumuoti atskirose tekstinio failo 'Duomenys.dat' eilutėse įrašytus skaičius. Rezultatus parodyti ekrane ir surašyti į failą. Prieš rašant programą, duomenų failas su laisvai pasirinktais skaičiais turi būti sukuriamas Turbo Paskalio aplinkos redaktoriumi ir įrašomas į darbinį katalogą.

```
program Pratimas5;
var Duomenys,                { Duomenų ir rezultatų failai }
    Rezultatai: text;
    x,                        { Skaitytina reikšmė }
    s, n: integer;           { Skaičių suma ir eilutės numeris }
begin
    WriteLn (** Failo duomenų skaitymas ir apdorojimas **);
    { Failų parengimas skaitymui ir rašymui }
    Assign( Duomenys, 'Prat5.dat'); Reset( Duomenys );
    Assign( Rezultatai, 'Prat5.rez');
    Rewrite(rezultatai);
    n:= 0;                    { Pradinė reikšmė }
    while not Eof(duomenys) do begin { Failo skaitymas }
        s:= 0; n:= n+1;        { Suma eilutei ir numeris }
        while not EoLn(duomenys) do { Eilutės apdorojimas }
            begin
                Read(duomenys, x);          { Skaiciaus skaitymas }
                s:= s+x;                     { Sumos kaupimas }
            end;
        ReadLn( Duomenys);                  { Perejimas į naują eilutę }
        { Rezultatų išvedimas į ekraną }
        WriteLn('Eilutės nr. ', n, ' Suma: ', s:8);
        { Rezultatų rašymas į failą }
        WriteLn( Rezultatai, 'Eilutės nr.', n,
            ' Suma: ', s:8);
    end;
    Close( Duomenys );                      { Failų uždarymas }
    Close( Rezultatai );
    ReadLn;
end.
```

Failo 'Prat5.dat' pavyzdys	Failo 'Prat5.rez' pavyzdys
15 2 45	Eilutes nr. 1 Suma: 62
-5 8 5 4	Eilutes nr. 2 Suma: 12
12 5 -12	Eilutes nr. 3 Suma: 5
5 6 7 8	Eilutes nr. 4 Suma: 26

Atsiminkite, kad duomenų failas turi arba būti tame pačiame kataloge, kaip ir jį apdorojanti programa, arba jo vieta turi nurodyti aplinkos parametrai, arba kelią į jį turi aprašyti programoje sakinyss Assign.

Rezultatų failo duomenis galima pamatyti ir po to, kai programa baigia darbą. Naudodami integruotos aplinkos priemones, patikrinkite tai savarankiškai.

Papildykite ir pakeiskite programą, panaudodami ekrano langus ir spalvas programos darbui iliustruoti:

- Pakeiskite programą taip, kad ji ekrane parodytų skaitomus iš pradinių duomenų failo skaičius.
- Pakeiskite programą, kad ji papildomai skaičiuotų kiekvienoje eilutėje esančių skaičių kiekį, ir visame faile esančių skaičių sumą bei kiekį.
- Savarankiškai sudarykite sveikųjų skaičių failo analizės programą, kuri nustatytų, kokią procentą faile sudaro lyginiai skaičiai. Nustatant, ar skaičius yra lyginis, patogų vartoti liekanos skaičiavimo operaciją, kuri dar yra vadinama dalyba moduliu. Jos aprašymo sintaksė: <dalinamasis> mod <daliklis>. Operacijos rezultatas yra lygus liekanai. Pavyzdžiui, sąlyga, kuri tikrina, ar skaičius x yra lyginis, gali būti aprašoma taip: $(x \bmod 2) = 0$.
- Modifikuokite sveikųjų skaičių failo analizės programą taip, kad ji skaičiuotų, kokią procentą sudaro skaičiai, kurie baigiasi skaitmeniu 5. Tokių skaičių filtravimo sąlyga yra $(x \bmod 10) = 5$.

Užduotis savarankiškam darbui.

Sudaryti sveikųjų skaičių failo analizės programą, kuri nustatytų, kiek procentų skaičių yra mažesnių už didžiausios reikšmės pusę. Pradinių duomenų failą teks skaityti du kartus: ieškant didžiausios reikšmės ir filtruojant reikšmes pagal duotą sąlygą.

3.4. Sveikųjų skaičių analizė

Analizuojant skaičių savybes, dažnai būna naudingos sveikųjų skaičių dalybos (**div**) ir liekanos radimo (**mod**) operacijos. Operacijos **div** rezultatas yra dalybos rezultato sveikoji dalis, o operacijos **mod** rezultatas yra gauta liekana, pavyzdžiui:

veiksmas	rezultatas	veiksmas	rezultatas
15 div 3	5	15 mod 3	0
15 div 4	3	15 mod 4	3
15 div 10	1	15 mod 10	5
15 div 20	0	15 mod 15	0
15 div 15	1	15 mod 20	15

Šių operacijų pagalba galima atskirti atskirus skaičiaus skaitmenis, sukeisti juos vietomis ir atlikti daugelį kitų skaičių pertvarkymo veiksmų.

Pavyzdžiui, sakiniu **s := x mod 10**; yra priskiriama kintamajam **s** jauniausioji skaičiaus **x** skiltis, o sakiniu **x := x div 10** jauniausioji skiltis skaičiuje **x** atmetama.

Sveikasis skaičius **x** nauju skaitmeniu **s** papildomas taip: **x := x*10+s**.

Dauginant iš 10, skaičiaus gale prirašomas nulis, kuris po to yra pakeičiamas pridedamu skaitmeniu **s**.

Sveikieji skaičiai (**integer**) dažniausiai yra vartojami pagalbiniais tikslams - įvairių skaitiklių organizavimui, todėl jų reikšmių atkarpa griežtai ribojama. Turbo Paskalio aplinkoje ši atkarpa yra pagrindinė, tačiau yra keletas išvestinių, kaip parodyta 3 lentelėje.

3 lentelė. Sveikųjų skaičių tipai.

integer	-32768..32767	word	0..65535
longint	-2147483648..2147483647	byte	0..255
shortint	-128..127		

6 pratimas. Programa **Prat6** perrašo įvesto skaičiaus skaitmenis priešinga tvarka. Čia **Sk** įvestas sveikas skaičius, o **SkAp** yra skaičiaus užrašas priešinga tvarka. Skaičiavimai naudojamas veiksmas **a*10+b**, kur **a** turimas skaičius, kurio gale reikia prirašyti skaitmenį **b**.

```

program Pratimas6;
uses Crt;
const x1d = 5; y1d = 2;          { Duomenų langas          }
      x2d = 35; y2d = 10;
      x1r = 40; y1r = 2;        { Rezultatų langas       }
      x2r = 70; y2r = 10;

```

```

        Greitis = 800;          { Skaičių rašymo ekrane pauzė }
{-----}
procedure Langas ( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas );  ClrScr;
end;
{-----}
procedure Vaizdas( var xd, yd, xr, yr : integer);
begin
    Langas( x1d, y1d, x2d, y2d, Blue ); { Lango paruošimas }
        { Pirmos eilutės vidurys - 4 (pusė žodžio) }
    GoToXY( ((x2d - x1d) div 2)- 4, 1 );
    WriteLn( 'Duomenys' );           { Žodis eilutės viduryje }
    xd := WhereX; yd := WhereY;      { Žymeklio vieta lange }

    Langas( x1r, y1r, x2r, y2r, Green ); { Lango paruošimas }
        { Pirmos eilutės vidurys - 5 (pusė žodžio) }
    GoToXY( ((x2r - x1r) div 2)- 5, 1 );
    WriteLn( 'Rezultatai' );         { Žodis eilutės viduryje }
    xr := WhereX; yr := WhereY;      { Žymeklio vieta lange }

    Langas( 5, 13, 70, 16, Magenta );
    WriteLn('Duomenų failo "Prat6.dat" skaičiai užrašomi',
        ' Duomenų lange. ' );
    WriteLn('Tie skaičiai Rezultatų lange rašomi atvirkščiai.' );
    WriteLn('Jei eilutėje skaičius netelpa, tai jo galas',
        ' kitoje eilutėje. ');
end;
{-----}
procedure Raso ( x1, y1, x2, y2 : integer
                fonas : integer; ; { Lango koordinatės }
                var  xz, yz : integer; { Žymeklio vieta lange }
                Sk : integer ); { Skaičius }

var Sp : integer;
begin
    Sp := TextAttr;
    Window( x1, y1, x2, y2 );      { Lango paruošimas }
    GoToXY( xz, yz );
    TextBackground( fonas );
    Write( Sk, ' ');               { Skaičiaus rašymas }
    xz := WhereX; yz := WhereY;    { Nauja žymeklio vieta }
    Delay( Greitis );              { Pauzė }
    TextAttr := Sp;
end;
{-----}

```

```

procedure Apversti( Sk : integer; var SkAp : integer);
var s : integer;
begin
  SkAp := 0;
  while Sk <> 0 do begin
    s := Sk mod 10;
    SkAp := SkAp * 10 + s;
    Sk := Sk div 10;
  end; end;
{-----}
var F : text;
    xd, yd,
    xr, yr : integer;
    Spalvos : integer; Sk, SkAp : integer;
begin
  Spalvos := TextAttr;
  Langas( 1, 1, 80, 25, Black ); TextColor( Black );
{-----} Failo duomenų skaitymas
  Assign( F, 'Prat6.dat'); Reset( F );
  Vaizdas( xd, yd, xr, yr );
  while not Eof( F ) do begin
    Read( F, Sk );
    Raso( xld, yld, x2d, y2d, Blue, xd, yd, Sk);
    Apversti( Sk, SkAp );
    Raso( xlr, ylr, x2r, y2r, Green, xr, yr, SkAp);
  end;
  Close( F );
{-----} Darbo pabaiga
  Langas( 60, 22, 75, 22, Brown );
  Write( 'Pabaiga: '); TextColor( Red ); Write(
'ENTER');
  ReadLn;
  TextAttr := Spalvos; Window( 1, 1, 80, 25 ); ClrScr;
end.

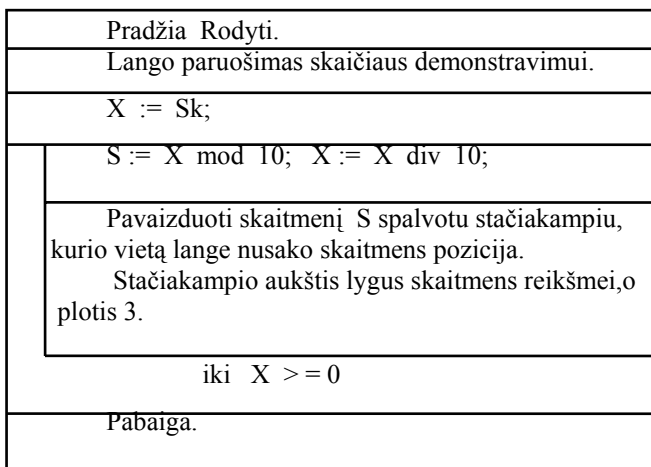
```

• Pakeiskite programą taip, kad ji patikrintų, ar skaičius yra vienodai skaitomas iš abiejų galų (laimingas). Norint tai padaryti, reikės įvesti dar vieną kintamąjį, kuris išsaugotų įvesto skaičiaus pradinę reikšmę tam, kad ją būtų galima palyginti su apskutu skaičiumi.

• Procedūrą `Raso` papildykite skaičiaus spalva. Duomenų ir rezultatų languose skaičius rašykite juodai, o tuos, kurie laimingi - raudonai.

7. Pratimas. Klaviatūra įvedinėjami sveiki teigiami skaičiai ekrane pavaizduojami grafiškai: kiekvienas skaitmuo išreiškiamas spalvotu

stačiakampiu. Skaitmenų gali būti ne daugiau kaip penki (žr. *integer* duomenų tipo reikšmių intervalą). Skaičių srauto pabaiga nurodoma skaičiumi nulis.



Ekranas padalinamas į du sektorius: duomenų įvedimo ir skaičiaus atvaizdavimo. Užduoties vykdymo algoritmas parodytas struktūrograme Prati7. Čia pradžia ir pabaiga programoje atitiks ne tik formalius `program` ir `end` žodžius, bet ir ekrano paruošimą darbui bei jo spalvų atstatymą programos darbo pabaigoje.

```

program Pratinas7;
uses Crt;
{-----}
procedure Langas ( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas ); ClrScr;
end;
```

```

{-----}
procedure Vaizdas;
begin
  Langas( 5, 2, 60, 5, Magenta );
  WriteLn('Ivesti klaviatūra sveiki teigiami',
    ' skaičiai vaizduojami ');
  WriteLn('spalvotais stačiakampiais. ');
  WriteLn('Darbo pabaiga - nulis. ');
end;
{-----}
procedure Daro ( Sk : integer ); { Skaičius }
var x1, s, spalva: integer;
begin
  Langas( 4, 10, 30, 20, White); { Lango paruošimas }
  x1 := 30; { Pirmojo skaitmens lango pabaigos koordinatė }
  while Sk <> 0 do begin
    s := Sk mod 10; Sk := Sk div 10; { Skaitmenų atskyrimas }
    x1 := x1 - 5; { Skaitmens lango pradžios koordinatė }
    { Pilka spalva mažai skiriasi nuo baltos }
    if s = 7 then spalva := 10
      else spalva := s;
    Langas( x1, 20 - s, x1+3, 20, spalva); {Skaitmens Langas }
  end; end;
{-----}
var Spalvos : integer; Sk : integer;
begin
  Spalvos := TextAttr;
  Langas( 1, 1, 80, 25, Black ); TextColor( Black );
  Vaizdas;
{----- Duomenų skaitymas }
  Langas( 45, 4, 75, 6, Green);
  WriteLn( 'Iveskite sveiką teigiamą');
  Write ( 'skaičių (0-32767): ');
  TextColor( Red ); ReadLn( Sk );
  while Sk > 0 do begin
    Daro( Sk );
    Window( 45, 4, 75, 6); TextBackground( Green );
    TextColor( Red ); GoToXY( 20, 2 );
    ClrEol; Read( Sk );
  end;
{----- Darbo pabaiga }
  TextAttr := Spalvos; Window( 1, 1, 80, 25 ); ClrScr;
end.

```

• Papildykite programą sakiniais, užrašanciais vaizduojamo skaitmens reikšmę stačiakampio apačioje.

- Papildykite programą garsiniais signalais.

Savarankiško darbo užduotis.

Iš klaviatūros yra įvedamas sveikas skaičius. Jo skaitmenis sudėkite. Jeigu rezultatas bus ne skaitmuo, tai jo skaitmenis vėl sudėkite. Skaičiavimus baikite, kai gausite vienženklį skaičių (skaitmenį). Gautą skaitmenį pavaizduokite ekrane spalvotu kvadratu, kurio kraštinės būtų lygios dvigubai skaitmens reikšmei. Kvadrato viršuje užrašykite įvestą skaičių, o viduje gautą skaitmenį. Spalvą parinkite atsitiktinę. Veiksmus palydėkite skirtingo tono garsais.

- Parinkite duomenis ir išbandykite programą su visais galimais skaičių variantais.

- Ar visais atvejais Jus tenkina rezultatas? Papildykite programą taip, kad visuomet gaunamas rezultatas būtų korektiškas.

- Pertvarkykite programą taip, kad lango spalvos ir garso tonas kistų priklausomai nuo gauto skaitmens.

- Papildykite programą taip, kad būtų galima daug skaičių apdoroti. Kiekvieno skaičiaus rezultatą parodykite panaudodami procedūrą Delay. Numatykite darbo pabaigos požymį.

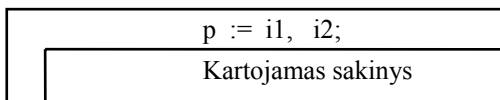
3.5. Simbolinis duomenų tipas

Ciklas **for**.

Daugelyje programų labai patogus atskira `while` ciklo modifikacija `for`. Šis ciklas galimas, kai ciklo parametras yra diskretinio tipo ir jo didėjimo (mažėjimo) reikšmė yra 1(vienetas). Ciklas `for` yra naudojamas tada, kai iš anksto žinomas ciklo kartojimų skaičius. Ciklo kartojimo metu ciklo parametrai iš eilės yra suteikiamos ciklo apraše nurodytos atkarpos reikšmės. Ciklo kartojimų skaičius yra lygus jo parametro reikšmių skaičiui šioje atkarpoje. Sakinio sintaksė:

```
for p := i1 to i2 do KartojamasSakinys;
```

Apraše **p** žymi ciklo parametą, o išraiškos **i1** ir **i2** nurodo jo reikšmių atkarpos ribas. Jei **i1>i2** (parametro reikšmių atkarpa yra mažėjanti),tai vietoje bazinio žodžio **to** rašomas žodis **downto**.



Jeigu kartojami veiksmai aprašomi ne vienu sakiniu, tai jų pradžioje parašomas žodis *begin*, o gale *end*; Tokia sakinių grupė yra vadinama sudėtinu sakiniu.

Simboliai

Mažiausia kiekvieno teksto sudėtinė dalis yra simbolis. Kiekviena kalba turi savo raidžių rinkinį – abėcėlę. Žmonės dar vartoja tam tikrą ženklų rinkinį bei skaitmenis. Visa tai yra simboliai. Kompiuteriai vartoja įvairius simbolių rinkinius. Skiriasi netgi programavimo kalbose vartojama simbolika. Simbolinio tipo reikšmės rašomos tarp apostrofų (pavyzdžiui, ‘a’, ‘d’, ‘5’, ‘?’, ‘+’) arba nurodomos kodais (#27, #13). Kompiuteryje visi simboliai yra koduojami atkarpos [0..255] skaičiais, pvz., A kodas yra #65. Dalis simbolių neturi grafinio atvaizdo ir yra vartojami valdymui. Pavyzdžiui, #7 įjungia garsinį signalą, o #13 žymi eilučių pabaigas.

Kompiuterio vartojami simboliai yra surašyti tam tikra tvarka ir turi savo eilės numerį. Tokios lentelės yra standartizuotos ir jų nėra daug.

Kintamieji, skirti simboliniams duomenims saugoti bei apdoroti, turi būti atitinkamo tipo. Paskalio kalboje toks tipas yra **char** (angl. **character** – simbolis). Kuriant algoritmus, dirbančius su simboliais, būtina žinoti simbolių rinkinio savybes. Nepriklausomai nuo programavimo kalbos bei programavimo aplinkos, galima išskirti tokias bendras savybes:

Rinkinį sudaro raidės, skaitmenys ir grafiniai simboliai. Turbo Paskalyje yra didžiosios ir mažosios lotyniškos raidės, dešimt arabiškų skaitmenų ir grafiniai simboliai (skyrybos ženklai, matematinių operacijų ženklai, tarnybiniai ženklai ir kt.).

Simbolių rinkinys yra sutvarkytas ir susietas. Tai reiškia, kad visi simboliai sudaro vieną nuoseklų sąrašą. Jame visi skaitmenys pagal kodus yra išdėstyti didėjimo tvarka. Raidės, pradžioje didžiosios, toliau mažosios, yra alfabeto seka. Grafiniai ženklai sugrupuoti ir turi sekoje savo vietas.

Simbolių rinkinyje yra “nematomų” simbolių, kaip kad eilutės pabaigos, failo pabaigos, garso, valdymo (Esc, funkciniai klavišai, kt.).

Rinkinyje yra tarpo simbolis, kuris neturi savo grafinio pavidalo, tačiau vartojamas kaip skyriklis ir vizualiai matomas kaip intervalas tarp kitų simbolių. Programuojant kartais labai svarbu nurodyti, kiek reikia tarpo

simbolių. Literaturoje sutinkame daug skirtingų sutartinių pažymėjimų. Tarpo simbolių žymėjime ‘ ‘; kartais sutinkamas žymėjimas raide *b* arba \cup .

```
var a, b, k : char;
```

Su simboliniais kintamaisiais galima atlikti šiuos veiksmus:

priskyrimą: `a := 'C'; b := a; k := #50;`

įvedimą: `Read (a, b);`

spausdinimą: `Write (a, b : 3);`

lyginimą: `if a > b then ...`

`if a = b then ...`

`...`

Lyginimo operacijose simboliai palyginimui pakeičiami jų eilės numeriu iš simbolių kodų lentelėje ir po to lyginami.

Su simboliais galima atlikti ir daugiau operacijų. Tam skirtos funkcijos:

`Pred ('C')` – rezultatas bus 'B'. Tai pirmesnis kodų lentelės simbolis. Galima rašyti `Pred (a)`, kur *a* yra **char** tipo kintamasis. Jei kintamojo *a* reikšmė yra pirmasis simbolių kodų lentelės elementas – rezultatas neapibrėžtas.

`Succ ('C')` – rezultatas bus 'D'. Tai kitas lentelės simbolis. Ši funkcija analogiška ankstesnei. Paskutiniajam sąrašo simboliui ji netaikytina.

`Ord ('C')` – rezultatas bus nurodyto simbolio eilės numeris simbolių lentelėje.

`Chr (15)` – rezultatas bus simbolis, simbolių kodų lentelėje einantis nurodytuju eilės numeriu.

`Uppcase ('a')` – pakeičia mažąsias lotyniškas raides didžiosiomis. Kitų simbolių ši funkcija nekeičia.

Pastaba. Simbolinės konstantos rašomos tarp apostrofų, kaip tai buvo daroma jau ankstesniuose skyreliuose. Be abejo, **char** tipo kintamajam galima priskirti tik vieną simbolių, o ne simbolių seką:

```
a := 'R';
```

```
b := 'AUGA'; { blogai!!! }
```

Pavyzdys:

```

program Simbolis;
const R = 'KAUNAS';
var x, a, b : char;
begin
  Read( x );           { Įvedame simbolį 'C' }
  WriteLn( R, x:3 ); { Spausdina KAUNAS C }
  b := 'K';
  if x = b then a := x { x ir b kintamųjų reikšmės}
                else a := 'D'; { nelygios. }
  WriteLn( a:2, b:2, x ); { Spausdina D KC }
end.

```

8 pratimas. Programa Ptrat8 į ekraną išveda simbolius ir jų kodus. Simboliai išvedami dalimis, kadangi ekrane vienu metu visi netelpa.

```

program Pratimas8;
uses Crt;
{-----}
procedure Simboliai( nuo, iki : integer);
var n : integer;
begin
  ClrScr;
  WriteLn(' Kompiuterio affabeto simboliai ir kodai');
  for n := nuo to iki do begin; { Simbolis ir jo kodas }
    Write( Chr( n ), ': ', n:3, ' ':3
    if n mod 9 = 0 then WriteLn; { Po 9 simbolius eilutėje }
    end;
  end;
{-----}
var Spalvos : integer; n : integer;
begin
  Spalvos := TextAttr;
  TextBackground( White ); ClrScr; TextColor( Black );
  Simboliai( 1, 100 ); { Pirmasis simbolių šimtukas }
  ReadLn;
  Simboliai( 101, 255 ); ReadLn; { Likusieji simboliai }
  TextAttr := Spalvos; Window( 1, 1, 80, 25 ); ClrScr;
end.

```

• Simbolius ir jų kodus ekrane atskirkite spalvotu tarpu. Gausite vaizdesnę lentelę.

9 pratimas. Leistinas spalvas galima demonstruoti ekraniniais langais. Labai patogu ir vaizdu tai padaryti panaudojant simbolius. Programa prat9 demonstruoja spalvas, parašydama jų kodus. Juoda spalva nedemonstruojama:

ji panaudota fonui.

```
program Pratimas9;
uses Crt;
{-----}
procedure Juosta( sim : char;      { Simbolis demonstracijai}
                  kiek : integer; { Simbolių skaičius      }
                  Sp : integer ); { demonstruojama spalva }
var i : integer;
begin
  TextColor( White ); Write(' Spalvos kodas:', Sp:3);
  TextColor( Sp );      {Spalva }
  for i := 1 to kiek do Write( sim );{ Simbolių juosta}
  WriteLn;
end;
{-----}
var Spalvos : integer; i, j : integer;
begin
  Spalvos := TextAttr; TextBackground( Black ); ClrScr;
  for i := 1 to 15 do      { Spalvų seka }
    Juosta( Chr( 219 ), 20, i );
  ReadLn;
  TextAttr := Spalvos; Window( 1, 1, 80, 25 ); ClrScr;
end.
```

- Išbandykite programą su įvairiais simboliais (pavyzdžiui, 178, 5, 177).

- Padarykite procedūrą, kuri ekrane formuotų mėlynos spalvos rėmelį, kuriam simbolius parinkite savarankiškai. Tikslinga rėmelio šonams, viršui ir apačiai, kampams parinkti tokius simbolius, kurie sudarytų malonų vaizdą.

10 pratimas. Paprasčiausi teksto analizės uždaviniai yra sužinoti kiek kokių simbolių yra, koks jų pasikartojimo dažnis, kiek skaitmenų yra ir pan. Programa Prat10 suranda ir ekrane parašo, kiek tos programos tekste yra sutinkamas nurodytas dialogo metu simbolis. Didžioji ir mažoji raidės analizės metu sutampa (neskiriamos). Programai duomenų tekstu tarnauja pati programa.

```
program Pratimas10;
uses Crt;
const Failas = 'Prat10.pas';      { Failo vardas }
{-----}
procedure Langas( x1, y1, x2, y2, fonas : byte);
begin
  Window( x1, y1, x2, y2 ); TextBackground( fonas );
```

```

ClrScr;
end;
{-----}
procedure Kiek( R : char; var K : integer);
var F : text;      sim : char;
begin
  K := 0;
  Assign( F, Failas ); Reset( F ); {Failas paruošiamas darbui}
  while not Eof( F ) do begin
    Read( F, sim );                { Skaitomas simbolis }
    if Upcase( sim ) = R then K := K+1; { Koks simbolis? }
    end;
    Close( F );
  end;
  {-----}
procedure Rezultatas( R : char; K : integer);
begin
  Langas( 10, 10, 40, 13, White );
  TextColor( Black ); Write( 'Jusu raide:' );
  TextColor( Red ); WriteLn( R: 3 );
  if K > 0 then begin
    TextColor( Black ); Write( 'Surasta tekste:' );
    TextColor( Red ); WriteLn( K: 3 );
    end
    else begin
    TextColor( Red ); WriteLn( 'Nesurasta ' );
  end;
  end;
  {-----}
var R : char; K : integer; Spalva : integer;
begin
  Spalva := TextAttr;
  Langas( 1, 1, 80, 25, Black );
  Langas( 10, 2, 40, 3, Blue );
  TextColor( Green );
  Write( ' Kokia raide Jus domina? '); ReadLn( R );
  R := Upcase( R ); { Raidė bus paverčiama didžiąja }
  Kiek( R, K );
  Rezultatas( R, K );
  ReadLn;
  TextAttr:= Spalva; Window( 1, 1, 80, 25 ); ClrScr;
end.

```

• Papildykite programą veiksmiais, leidžiančiais atlikti teksto simbolinę analizę daug kartų. Tam pagrindinėje programoje paklausimo ir kreipinių į paprogramės sakiniai turi būti cikle, kurio pradžios bei pabaigos sąlygas sugalvokite ir parašykite.

- Pertvarkykite procedūrą **Kiek**, kuri surastų kiek tekste yra eilučių, neturinčių nurodyto simbolio.

- Rezultatų lange išveskite abi raidė formas, jeigu ieškomas simbolis buvo raidė.

- Atskirkite didžiąją raidę nuo mažosios.

- Kokią dalį procentais sudaro tekste ieškomasis simbolis?

- Pakeiskite failo vardą kitu.

Savarankiško darbo užduotis.

Tekstiniame faile esančius simbolius eilės tvarka išveskite į ekraną: raides žaliai, skaitmenis raudonai, likusius simbolius juodai. Tekstą išvedinėkite po 20 eilučių ekrane. Įvertinkite, kad paskutiniam išvedimui gali būti mažiau eilučių. Taip gali būti ir kai tekste yra mažiau 20 eilučių. Ekraną apačioje sukurkite langą, kuriame būtų parašoma, kiek kokių simbolių jau buvo išvesta. Čia turėtų būti pranešimas, ar jau visos failo eilutės peržiūrėtos.

3.6. Atsitiktinių situacijų modeliavimas

Dauguma programų modeliuoja įvairius mūsų aplinkos reiškinius, kur dažnai tenka turėti reikalų su atsitiktiniais dydžiais. Pavyzdžiui, žaidžiant kortomis, atsitiktinius skaičių šaltinis yra kortų kaladė. Tik siekiant, kad maloniau būtų žaisti, kortos papuošiamos paveikslukais. Žaidėjai, imdami kortas ir darydami ėjimus formuoja situacijas, kurios lemia laimėtoją.

Žaidimų modeliavimas

Panagrinėkime, kaip galima sumodeliuoti paprasčiausią žaidimą kortomis 'Akis'. Šiame žaidime žaidėjai ima pakaitomis kortas iš kaladės tol, kol surenka sumą pakankamai artimą skaičiui 21. Laimi tas žaidėjas, kurio suma būna artimesnė skaičiui 21. Vienas iš žaidėjų yra kompiuteris.

Žaidimo modeliavimui reikalingi kintamieji, kurie vaizduotų žaidėjų imamas kortas, jų surinktas taškų sumas, aktyvų žaidėją ir sprendimus apie žaidimo nutraukimą. Kortų kaladę galima imituoti atsitiktinių skaičių generavimo funkcija `Random(n)`, kuri generuoja atkarpos `[0, n-1]` atsitiktinius skaičius. Prieš panaudojant šią funkciją, rekomenduojama iškviesti procedūrą `Randomize`, kuri susieja funkcijos `Random` generuojamos atsitiktinių skaičių sekos pradžią su kompiuterio laikrodžio

parodymu.

Prieš pradėdant rašyti programą, taip pat turi būti sudaromas jos scenarijus (algoritmas), kuris turi tiksliai numatyti visų programos valdomų veiksmų seką. Tokio scenarijaus metmenis galima aprašyti paprastais lietuvių kalbos sakiniais. Pavyzdžiui, kortų žaidimo Akis modeliavimui gali būti parenkamas toks scenarijus:

- Žaidėjų sumoms suteikiamos pradinės nulinės reikšmės;
- Formuojama užklausa apie pradėdantį žaidėją;
- Kol bent vienas žaidėjas sutinka žaisti, kartojami veiksmai:
 - ✧ Modeliuojamas kortos ėmimas;
 - ✧ Kortos taškai pridedami prie aktyvaus žaidėjo sumos;
 - ✧ Keičiama aktyvaus žaidėjo žymė;
 - ✧ Informuojama apie žaidybinių situaciją;
 - ✧ Priimamas sprendimas apie aktyvaus žaidėjo norą tęsti žaidimą;
- Skelbiami žaidimo rezultatai.

Turint scenarijų, galima parinkti jo realizavimui skirtus kintamuosius ir numatyti atskirų veiksmų programinio realizavimo būdus. Pavyzdžiui, iš kaladės imamos kortos ir žaidėjų surinktų sumų modeliavimui tinka integer tipo kintamieji, sprendimams apie norą tęsti žaidimą - boolean tipas, aktyvaus žaidėjo nurodymui - char tipas.

Iš scenarijaus aprašymo struktūros matyti, kad jį galima aprašyti sakiniu while, kuris turi tikrinti žaidėjų apsisprendimą aprašančias logines (boolean) žymes. Apie kortos ėmimo iš kaladės modeliavimą jau buvo minėta - šiam tikslui tinka atsitiktinių skaičių generavimo funkcija Random. Užklauskos ir informaciniai pranešimai realizuojami įvedimo/išvedimo sakiniais, o žaidėjų apsisprendimas apie žaidimo nutraukimą - sąlyginiais operatoriais if.

11 pratimas. Programa Prat11 modeliuoja žaidimą Akis.

```
program Pratimas11;  
uses Crt;  
var  
    x,                               { Žaidėjo imama korta      }  
    n1, n2: integer;                 { Sukauptos sumos        }
```

```

e, { Pagalbinis kintamasis }
c: char; { Aktyvus žaidėjas }
dar1, dar2: boolean; { Apsisprendimo žymės }
spalva : integer;
begin
    spalva := TextAttr; TextBackground( Blue );
    TextColor( Black ); Window( 1, 1, 80, 25 ); ClrScr;
    WriteLn ( ' ***** Ž A I D I M A S A K I S ***** ');
    { Žaidžiant siekiama surinkti sumą, kuri būtų galimai }
    { artimesnė skaičiui 21. Kintamieji dar1 ir n1 modeliuoja }
    { kompiuterį, o dar2 ir n2 - žaidėją. }
    dar1:= true; dar2:= true; { Pradinis apsisprendimas }
    n1:= 0; n2:=0; { Pradinės sumos }
    WriteLn('Kas pradeda žaidimą',
            '(k- kompiuteris, z- žaidėjas)');
    ReadLn(c);
    Randomize; { Generatoriaus nustatymas }
    while dar1 or dar2 do begin
        x:= Random(11)+1; { Imama korta }
        if dar1 and
            (c='k') then begin { Kompiuterio korta }
            n1:= n1+x; { Kompiuterio suma }
            c:='z'; { Aktyvaus žaidėjo keitimas}
            if (21-n1)<=2 then
                dar1:= false; { Sprendimas, ar tęsti }
            end
            else begin { Žaidėjo korta }
                WriteLn('Ar dar bus ėjimas (t/n)');
                ReadLn(e); { Žaidėjas atsako }
                if e='t' then begin {Sprendimo analizė }
                    n2:= n2+x; c:='k';
                    WriteLn ('Jums teko: ', x, ' Jūsų suma: ', n2)
                end
            end
        else begin { Atsisakymas tęsti }
            dar2:= false; c:= 'k' end;
        end;
    end;
    WriteLn ('Rezultatai:');
    WriteLn('Kompiuterio: ', n1, ', Jūsų: ', n2);
    ReadLn;
    TextAttr := spalva; Window( 1, 1, 80, 25 ); ClrScr;
end.

```

• Programą papildykite ekraniniais langais, skirtais žaidėjams, taisyklėms, rezultatams skelbti.

• Papildykite programą, kad jos pradžioje būtų pateikiamos žaidimo taisyklės;

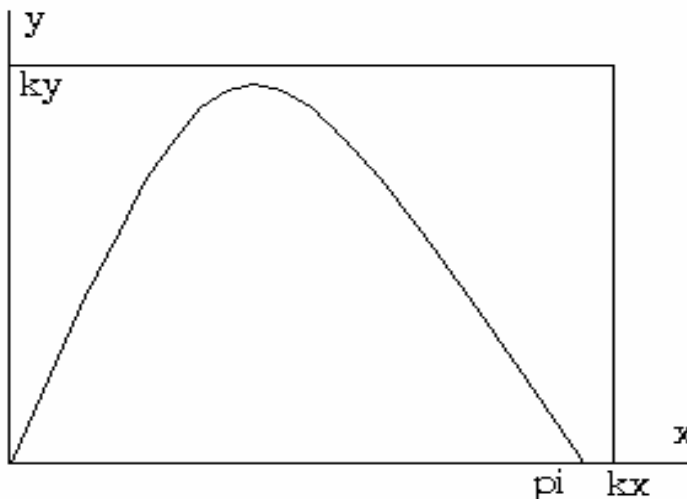
- Pakeiskite kompiuterio sumos kaupimo sąlygą taip, kad joje būtų atsitiktinis skaičius ir žaidėjui būtų sunkiau orientuotis, kada nutraukti žaidimą;

- Papildykite programą rezultatų analizės dalimi, kurioje būtų nustatoma, kas laimėjo ir būtų pranešama apie laimėtoją.

Atsitiktinių skaičių panaudojimas skaičiavimuose

Atsitiktinių skaičių sekos taip pat sėkmingai yra vartojamos sprendžiant įvairius skaičiavimo uždavinius: ieškant daugelio kintamųjų funkcijų maksimalių ir minimalių reikšmių, skaičiuojant sudėtingų figūrų plotus, tūrius ir daugeliu kitų atvejų. Pavyzdžiui, panagrinėkime, kaip gali būti apskaičiuotas x ašies ir linijos $y = \sin(x)$ pirmojo pusperiodžio ribojamos figūros plotas (1 paveiksle).

Figūra, kurios plotas yra skaičiuojamas, apibrėžiama stačiakampiu, kurio dvi kraštinės sudaro koordinatų ašys, o kitos (kx ir ky) yra parenkamos taip, kad kraštinių ilgiai būtų sveiki skaičiai ir visa nagrinėjama figūra tilptų stačiakampio viduje. Po to turi būti generuojamos atsitiktinės taškų stačiakampio viduje koordinatės (x, y) ir skaičiuojama, kiek tokių taškų patenka į figūros vidų ($y \leq \sin(x)$). Jeigu generuojamos koordinatų reikšmės bus tolygiai pasiskirsčiusios, šio skaičiaus ir visų generuojamų taškų skaičiaus santykis nurodys, kokią stačiakampio dalį sudaro nagrinėjamos figūros plotas.



1 paveikslas Ploto skaičiavimo modelis

Tokio skaičiavimo būdo tikslumas priklauso nuo generuojamų taškų skaičiaus. Skaičiavimų tikslumą galima įvertinti kartojant skaičiavimus ir sulyginant jų rezultatus. Realūs tolydinio pasiskirstymo atkaroje $[0..1]$ skaičiai yra generuojami parametrų neturinčia funkcija Random.

12 pratimas. Išbandykite programą Pratl2, kuri 10 kartų skaičiuoja funkcijos $y = \sin x$ pirmojo pusperiodžio ir y ašies ribojamą plotą ir įvertinkite skaičiavimo paklaidą (ji lygi didžiausios ir mažiausios reikšmių skirtumo pusei).

```
program Plotas;
uses Crt;
const n= 100;          { Tiriamų taškų skaičius }
var
  x, y: real;           { Taško koordinatės }
  i, j,                 { Pagalbiniai kintamieji }
  m: integer;           { Taškai figūros viduje }
  kx, ky, s: real;      { Stačiakampio kraštinės, plotas }
  ss: real;              { Skaičiuojamas figūros plotas }
  spalva : integer;
begin
  spalva := TextAttr; TextBackground( Blue );
  TextColor( Black ); Window( 1, 1, 80, 25 ); ClrScr;
  kx:= 4; ky:= 1;      { Stačiakampio kraštinių dydžiai }
  s:= kx*ky;           { Stačiakampio plotas }
  Randomize;
  for j:=1 to 10 do begin { Plotų skaičiavimai }
    m:= 0;
    for i:= 1 to n do begin { Taškų skaičiavimas }
      x:= Random* kx;      { Atsitiktinės koordinatės }
      y:= Random* ky;
      if y <= Sin(x) then m:= m+1; {Taškai figūros viduje }
    end;
    ss:= s * m / n;
    Write (ss: 6: 2 );      { Figūros plotas }
  end;
  ReadLn;
  TextAttr := spalva; Window( 1, 1, 80, 25 ); ClrScr;
```

end.

- Papildykite programą informaciniais pranešimais taip, kad jos darbo rezultatai ekrane būtų pateikti vaizdžiau. Panaudokite langus, spalvas, garsus.

- Pakeiskite programos generuojamų taškų skaičių ir patikrinkite, kaip nuo to priklauso skaičiavimų paklaida.

- Papildykite programą veiksmiais, kurie leistų vizualiai ekrane pavaizduoti skaičiuojamus taškus: jeigu kreivės viduje, tai žalias simbolis, o išorėje raudonas. Taškus vaizduokite simboliu ‘*’, kurio fonas atitiktų nurodytas spalvas. rezultatus pavaizduokite balto fono lange, kurio dydis 20x10 (čia x=20, y=10). Tam reikia sinuso skaičiavimo reikšmės 10 kartų padidinti.

Savarankiško darbo užduotis

1. Maiše sumesti trijų skirtingų spalvų rutuliukai. Du žaidėjai iš maišo traukia po vieną rutuliukus. Rutuliukų ima vienodą skaičių. Laimi tas, kuris pirmas surenka kurios nors vienos spalvos penkis rutuliukus. Reikia parašyti programą, kuri imituotų maiše esančius rutuliukus ir atsitiktinai išrinktų po vieną kiekvienam žaidėjui tol, kol kuris nors iš jų laimės. Programos algoritmas gali būti toks:

Pradžia. Rutuliukai
Kintamųjų aprašymas.
sa1 :=0; sa2 := 0; sa3 := 0; Pirmojo žaidėjo sb1 :=0; sb2 := 0; sb3 := 0; Antrojo žaidėjo
kol nei vienas iš rutuliukų skaitliukų dar nelygūs 5
Spalva := Random(3); Pirmojo žaidėjo Pridedamas vienetukas prie vieno iš skaitliukų
Spalva := Random(3); Antrojo žaidėjo Pridedamas vienetukas prie vieno iš skaitliukų
Skelbiamas nugalėtojas arba lygiosios
Pabaiga

3.7. Valdantys simboliai

Kompiuterio klaviatūroje yra daug klavišų, kurie skirti teksto (raidės, skaitmenys, ženklai) rašymui ir kompiuterio programų darbo valdymui. Rašant dialogines programas, būtina mokėti atpažinti klaviatūra duodamus nurodymus. Tam galima naudoti įprastus tekstinius simbolius arba žodžius, tačiau dialogas būna nevaizdus, programos vartotojas daro daug klaidų, labai lėtas darbas su programa. Paprasčiau ir patogiau, kai programa pateikia ekrane siūlomų paslaugų (veiksmų) sąrašą, o vartotojas pasirenka. Tam patogus naudoti valdančiuosius simbolius.

Klaviatūra paspausto klavišo reikšmė priimama procedūromis

```
ReadLn( Sim ) arba Read( Sim ), { Sim : char }.
```

Šios priemonės nepatogios, nes jų darbo rezultatas, kaip aidas, rodomas ekrane. Tai sujaukia ekrane esantį vaizdą. Modulyje Crt yra parametrų neturinti funkcija ReadKey, skirta paspausto klavišo kodui nuskaityti. Jos darbo rezultatas ekrane neparodomas. Dauguma valdančiųjų klavišų (arba jų kombinacijų) yra koduojami dviem simboliais, kurių pirmojo kodas visuomet yra 0 (nulis). Kadangi funkcija ReadKey skaito tik vieną simbolį, tai valdančiojo simbolio atpažinimui reikia jį panaudoti du kartus: nulinio kodo skaitymui ir antrojo nenulinio. Dažniausiai vartojamų valdymo klavišų antrojo simbolio kodai pateikiami 2 lentelėje.

2 lentelė. *Antrojo valdančiojo sekos simbolio kodai*

Kodas	Klavišas	Kodas	Klavišas	Kodas	Klavišas
72	↑	73	PageUp	82	Insert
75	←	81	PageDown	83	Delete
77	→	71	Home	59-68	F1-F10
80	↓	79	End	15	Shift + Tab

13 pratimas. Rodyklėmis “aukštyn”, “žemyn”, “dešinėn”, “kairėn” valdomas žymeklis ekrane palieka raudoną nueito kelio pėdsaką.

```
program Pratimas13;
uses Crt;
const Greitis = 700;
{-----}
procedure Langas( x1, y1, x2, y2 : byte; fonas : byte );
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas ); ClrScr;
```

```

end;
{-----}
procedure Klavisas( var h, v, b : integer );
var sim : char;
begin { Vienas žingsnelis: žymeklio koordinatčių pokytis }
  b := 0; h := 0; v := 0;
  while ( b = 0 ) and ( h = 0 ) and ( v = 0 ) do begin
    sim := ReadKey;
    if sim = #27 then b := 1; { Esc klavišas }
    if sim = #0 then { Valdantis simbolis }
      begin
        sim := ReadKey; { Antrasis kodas }
        if sim = #72 then v := -1; { Aukštyn }
        if sim = #80 then v := 1; { Žemyn }
        if sim = #77 then h := 1; { Dešinèn }
        if sim = #75 then h := -1; { Kairèn }
      end;
  end; end; {-----}
procedure Kelias( Sim : char; fonas : byte);
var x, y : integer;
    b : integer;
    dx, dy : integer;
begin
  x := 1; y := 1; b := 0; dx := 0; dy := 0; { Pradžia }
  TextBackground( fonas ); GoToXY( x, y ); Write( Sim );
  while b = 0 do begin
    if KeyPressed then Klavisas( dx, dy, b );
    x := x + dx; y := y + dy; { Žymeklio naujos koordinatės }
    if ( x >= 1 ) and ( x <= 80 ) and { Ekrano ribose }
      ( y >= 1 ) and ( y <= 20 )
    then begin GoToXY( x, y );
              Write( Sim ); Delay( Greitis ); end
    else begin
      Sound( 600 ); Delay( 20 ); NoSound;
      if x < 1 then x := 1;
      if x > 80 then x := 80;
      if y < 1 then y := 1;
      if y > 20 then y := 20;
    end; end; end;
{-----}
begin
  Langas( 1, 1, 80, 25, Black );
  Langas( 5, 22, 75, 24, Green ); TextColor( Black );
  WriteLn( 'Takas formuojamas žymeklio valdymo
klavisais' );
  GoToXY( 40, 2 ); Write( ' Sustoti: ' );

```

```

TextColor( Red ); Write( 'Esc' );
  Langas( 1, 1, 80, 20, Blue ); Kelias( ' ', Red );
  ReadLn; Langas( 1, 1, 80, 25, Black );
end.

```

- Takas brėžiamas nurodytame ekrano lange, kurio koordinatės pastovios. Pakeiskite programą taip, kad tako langas būtų valdomas per parametrus.

- Žymeklio klavišais keičiame tako kryptį. Pakeiskite programą taip, kad žymeklis pajudėtų ekrane tik per vieną poziciją, paspaudus jo valdymo klavišą. Kitaip jis stovi vietoje.

- Pakeiskite programą taip, kad žymeklis klaidžiotų ekrane pagal Jūsų valią nepalikdamas pėdsako.

- Padarykite taip, kad ekrane žymeklis paliktų ne savo pėdsaką, o nurodyto simbolio taką, pavyzdžiui “*”.

- Pasiekus ekrano ribą kompiuteris signalizuoja garsu. Sukurkite savo melodiją ir ją įdėkite vietoje to nemalonaus garso.

14 pratimas. Reikia kompiuterio ekrane sukurti langą ir jame parodyti tiek teksto eilučių, kiek telpa. Leisti tekstui lange “vaikščioti” pirmyn-atgal po vieną eilutę (klavišais ↑↓) arba per pusę lango (PageUp ir PageDown klavišais). Taip pat kairėn-dešinėn per vieną simbolį (klavišais ← →). Vaizduojant teksto eilutes būtina įvertinti tai, kad jos gali būti ilgesnės už lango plotį. Čia teksto eilutės simboliai skaitomi, panaudojant kintamąjį tipo string, kuris bus nagrinėjamas 5 skyriuje. Procedūros Rodo paskirtis yra iš tekstinio failo perskaityti ekrano lange telpantį simbolių skaičių: n1 rodo pirmos eilutės lange numerį (iš failo skaitomos eilutės iki n1 yra praleidžiamos), o p1 - kiekvienos eilutės pirmasis simbolis, kuris parodomas lange (kiti iš kraštų netelpantys lange simboliai nerodomi). Lange telpa 20 eilučių, eilutės ilgis 50 simbolių.

```

program Pratimas14;
uses Crt;
const failas = 'Prat14.pas';
{-----}
procedure Langas( x1, y1, x2, y2 : byte; fonas : byte );
begin
  Window( x1, y1, x2, y2 );
  TextBackground( fonas ); ClrScr;
end;
{-----}

```

```

procedure Informacija;
begin
  Langas( 2, 2, 28, 10, Green ); TextColor( Black );
  WriteLn( 'Teksto peziura: ' );
  WriteLn( 'zymeklio valdymo klavisai' );
  WriteLn( #24#25#26#27, ' ir PageUp, PageDown' );
  WriteLn; WriteLn( 'Failas: ', failas );
  GoToXY( 5, 8 ); Write( ' Darbo pabaiga: ' );
  GoToXY( 10, 10 ); TextColor( Red ); Write( 'Esc' );
end;
{-----}
procedure Klavisas( var h, v, b : integer );
var sim : char;
begin
  b := 0; h := 0; v := 0;
  while ( b = 0 ) and ( h = 0 ) and ( v = 0 ) do begin
    sim := ReadKey;
    if sim = #27 then b := 1; { Klavišas Esc }
    if sim = #0 then begin
      sim := ReadKey;
      if sim = #72 then v := -1; { Klavišas ↑ }
      if sim = #80 then v := 1; { Klavišas ↓ }
      if sim = #77 then h := 1; { Klavišas → }
      if sim = #75 then h := -1; { Klavišas ← }
      if sim = #73 then v := -10; { Klavišas PageUp }
      if sim = #81 then v := 10; { Klavišas PageDown }
    end; end;
  end;
{-----}
procedure Rodo( pl, nl : integer );
var F : text; E : String; n, i, p2 : integer;
begin
  Assign( F, failas ); Reset( F );
  n := 0;
  while not Eof( F ) do begin
    n := n + 1; ReadLn( F, E ); { Skaitoma teksto eilutė }
    if n in [ nl..nl+19 ] then begin
      p2 := Length( E ); { Eilutės simbolių skaičius }
      if p2 > ( pl + 49 ) then p2 := pl + 49;
      for i := pl to p2 do Write( E[i] ); WriteLn;
    end;
  end; Close( F ); end;
{-----}
procedure Tekstas;
var x, y : integer;
    b : integer;
    dx, dy : integer;

```

```

begin
  x := 1;  y := 1;  { Tekstas išvedamas nuo pirmos eilutės }
  b := 0; dx := 0;  dy := 0; { b- pabaiga; dx, dy - pokytis }
  while b = 0 do begin
    Langas( 30, 2, 80, 23, Blue ); TextColor( Black );
    Rodo( x, y );                    { Teksto fragmentas lange }
    Klavisas( dx, dy, b );          { Teksto fragmento pokytis }
    x := x + dx;    y := y + dy; { Naujo fragmento pradžia }
    if x < 1 then x := 1;  { Teksto ribų kontrolė }
    if x > 80 then x := 80;
    if y < 1 then y := 1;
  end; end;
  {-----}
begin
  Langas( 1, 1, 80, 25, Black );    Informacija;
  Tekstas;
  Langas( 1, 1, 80, 25, Black );
end.

```

• Darbe su failo eilutėmis yra apsauga tik dėl mažiausio eilutės numerio. Kaip padaryti, kad būtų apsauga dėl maksimalaus galimo eilutės numerio? Papildykite programą tokia apsauga.

• Teksto rodymo lango parametrai programoje fiksuoti. Padarykite tokius pakeitimus paprogramėse, kad teksto rodymo langą galima būtų nurodyti per parametrus.

• Praplėskite teksto rodymo galimybes klavišais “Home” ir “End”: eilutes rodyti nuo pradžios arba eilutes rodyti taip, kad ilgiausios pabaiga būtų lange rodoma.

Daugiavariantinės alternatyvos analizės uždaviniuose

Aprašant duomenų analizės ir grupavimo sąlygas, kurian dialogines programas patogiu vartoti daugiavariantines alternatyvas. Tam naudojami tokio tipo sakiniai:

```

case  P of
    p1 :  S1;
    p2 :  S2;
    ...
    pn :  Sn;
  else  S;
end;

```

Čia P kintamasis arba išraiška, kurios reikšmė yra diskretinio tipo; p1, p2, ... pn yra tos reikšmės, kurios vykdomi atitinkami sakiniai S1, S2, ...

Sn; sakinys S vykdomas, kai P reikšmė nėra išvardintame sąraše. Sakinio pabaiga nurodona žodžiu end.

15 pratimas. Patikrinkite programą Pratinimas15, kuri skaičiuoja, kiek tekstiname faile yra raidžių, skaitmenų ir kitokių (specialių) simbolių:

```
program Pratinimas15;
uses Crt;
{-----}
procedure Langas( x1, y1, x2, y2 : byte; fonas : byte );
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas ); ClrScr;
end;
{-----}
const failas = 'Prat15.pas'; { Pradiniu duomeniu failas }
var   F : text;              { Pradiniai duomenys      }
      sim : char;             { Analizuojamas simbolis  }
      sr,                               { Raidziu skaicius    }
      ss,                               { Skaitmenu skaicius  }
      sk,                               { Kitu simboliu skaicius }
      st : integer;            { Tarpo simboliu skaicius }
begin
    Assign( F, failas); Reset( F );
    Langas( 1, 1, 80, 25, Black );
    Langas( 10, 3, 50, 10, Blue ); TextColor( Black );
    WriteLn(' Simboliu tipai tekste:', failas );
    sr := 0; ss := 0; sk := 0; st := 0;
    while not Eof( F ) do begin
        while not Eoln( F ) do begin { Eiluciu skaitymas }
            Read( F, sim );
            case sim of
                'a'..'z', 'A'..'Z' : sr := sr+1; { Raides          }
                '0'..'9'           : ss := ss+1; { Skaitmenys          }
                ' '                : st := st+1; { Tarpai             }
            else
                sk := sk+1; { Kiti simboliai }
            end;
            ReadLn( F ); { Pereiti i nauja eilute }
        end; Close( F ); { Failo analizes pabaiga }
        Langas( 12, 4, 48, 9, Green);
        WriteLn( 'Faile simboliu yra: ', sr+ss+st+sk );
        WriteLn(' raidziu      ', sr);
        WriteLn(' skaitmenu    ', ss);
        WriteLn(' tarpu       ', st);
        WriteLn(' kitu simboliu ', sk);
        WriteLn(' Nelotynisko alfabeto raides yra');
```

```

        WriteLn(' laikomos kitais simboliais');
    ReadLn;
end.

```

Ekrano darbo režimo keitimas.

Tekstinis ekrano darbo režimas yra keičiamas su funkcija `TextMode(<Kodas>)`, kurioje gali būti vartojami tokie tekstinio ekrano darbo režimo kodai:

BW40 = 0 (black-white: juodai baltas, formatas 40*25),
 BW80 = 2 (black-white: juodai baltas, formatas 80*25),
 MONO = 7 (monochromatinis, formatas 80*25),
 CO40 = 1 (color: spalvotas, formatas 40*25),
 CO80 = 3 (color: spalvotas, formatas 80*25),
 FONT8x8 = 256; (keičia eilučių skaičių ekrane, tinka tik EGA (43 eilutės) ir VGA (50 eil.) tipų monitoriams).

Kodas `Font8x8` yra vartojamas kartu su kitais darbo režimo kodais. Pavyzdžiui, `CO80+FONT8x8` nurodo 80 kolonelių ir 50 eilučių darbo režimą (VGA). Prieš iškviečiant procedūrą `Textmode`, ekrano darbo režimo kodą rekomenduojama išsaugoti, kad jį būtų galima atstatyti. Esamo darbo režimo kodas yra saugomas modulio `Crt` kintamajame `LastMode`. Pakeiskite programose ekrano režimą į **CO40**, po to į **CO80+FONT8x8** ir patikrinkite, kaip pasikeis vaizdas ekrane.

Savarankiško darbo užduotis.

Sudarykite programą žaidimo kauliukais modeliavimui. Šiame žaidime kiekvienas žaidėjas meta po tris kubo formos kauliukus, ant kurių šonų yra atkarpos [1..6] skaičiai. Todėl žaidėjui tenka trijų atsitiktinių šios atkarpos skaičių suma. Laimi tas žaidėjas, kurio suma didesnė.

3.7. Užduotys

Suplanuokite ir programose realizuokite ekrano panaudojimą dialogui, duomenims ir rezultatams, pranešimams. Panaudokite Jums tinkamas paprogrames iš pratimų, naujas sukurkite taip, kad jos tiktų kitoms programoms. Nedarykite sudėtingų ir didelės apimties programinių blokų.

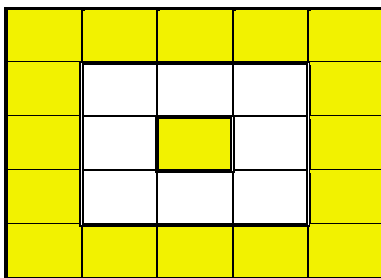
✧ Pitagoriniais skaičiais vadinami skaičiai 3, 4 ir 5. Jiems teisingas užrašas: $3^2 + 4^2 = 5^2$. Patikrinkite, ar pirmajame sveikų skaičių šimtuکه yra sveikų skaičių, kurie tiktų tokiam reiškiniui. Jeigu yra, tai ar tarp jų yra tokie, kurie skirtųsi vienetu (pitagoriniais galėtume vadinti).

✧ Turime P pinigų sumą litais. Kokiais banknotais juos paimsime iš banko, jeigu norime, kad jų būtų kaip galima mažiau.

✧ Turime daug kvadratėlių, kurie paskirstyti į n ($n \leq 10$) eilučių ir po m ($m \leq 20$) kiekvienoje eilutėje. Atvaizduokite kvadratėlius ekrane spalvotais langais, kurių plotis du simboliai, o aukštis viena eilutė. Visi langeliai vienodos spalvos.

Padaroma programa, kuri leidžia žymeklio valdymo klavišais keliauti per langelius. Aktyvus langelis pažymimas kita spalva. Jeigu nuspaudžiamas simbolinis klavišas, tai jo reikšmė užrašoma aktyviame lange. Perėjus į kitą langelį, to langelio spalva pakeičiama į naują (užimtumą žyminčią) spalvą. Jeigu nuspaudžiamas tarpo klavišas, tuomet buvęs simbolio užrašas valomas (jeigu nebuvo – nieko nedaroma) ir perėjus į kitą langelį, to langelio spalva keičiama į tuščio langelio spalvą. Darbo pabaiga, kai atėjus į dešinį apatinį langelį paspaudžiamas klavišas **Enter** arba bet kada **Esc**.

✧ Turime n kvadratinę plytų. Kokio dydžio piramidę galima pastatyti, jeigu kiekviename lygyje pagrindas turi būti kvadratinis ir laiptelio žingsnis yra viena plyta. Surasti kiek panaudota plytų kiekviename lygmenyje ir kiek jų liko. Pvz., kai $n = 37$, tai $L_1 = 25$, $L_2 = 9$, $L_3 = 1$ ir lieka 2 plytos nepanaudotos:



5. Teksto analizė ir tvarkymas

Šiame skyriuje:

- sudėtingesni teksto analizės uždaviniai; masyvų panaudojimas;
- aibės ir jų taikymas teksto analizės uždaviniuose, skaičiavimuose;
- eilutės ir darbo su jomis priemonės;

5.1. Simbolinio tipo duomenys.

Trečiame skyriuje buvo nagrinėta simbolinė teksto analizė. Sudėtingesniems tokio tipo uždaviniams spręsti reikalingos sudėtingos duomenų struktūros, galinčios vienu metu saugoti daugiau negu vieną simbolį. Masyvai ar kitos struktūros, sudarytos iš simbolių, aprašomos panašiai kaip analogiškos skaičių struktūros:

```
type Mas = array [ 1 .. 10 ] of char;  
var A : Mas;
```

Veiksmus su simboliais masyve galima atlikti nurodant jų indeksą:

```
A[ 5 ], A[ i+3 ] ir pan.
```

Panašiai galima sukurti aukštesnės eilės masyvus. Jie sudaromi ir naudojami tokia pat tvarka, kaip ir skaičių masyvai.

Taikant simbolių masyvus yra lengviau parašyti programas, skirtas teksto analizei ar redagavimui žodžių lygyje. Pavyzdžiui, reikia surasti kiek tekste yra žodžių. Žodžiai skiriami bent vienu tarpo simboliu. Eilutėse prieš pirmą ir po paskutinio žodžio gali būti arba nebūti tarpai. Žodžiai iš eilutės į eilutę nekeliami.

Prieš rašant programą būtina išsiaiškinti, kaip atpažinsime žodį. Žinome, kad žodžių skirtuku eilutėje yra tarpo simbolis. Visi kiti simboliai (net ir vienas) priklauso žodžiui. Jeigu tekstą nagrinėsime eilutėmis, tuomet eilutės pabaigos simbolio ieškoti nereikės. Jeigu visą tekstą surašytume į simbolių tipo masyvą, tai kaip žodžių skirtukus reikia nagrinėti eilutės pabaigos simbolius #10 ir #13 (eilutės pabaiga ir perėjimas į naują eilutę).

Žodžiui atpažinti reikia nagrinėti du gretimus simbolius. Jeigu tarpo

simbolių pažymėsime **t**, o netarpo **n**, tai galimos tokios situacijos:

tt	esame tarp žodžių	tn	žodžio pradžia
nn	esame žodyje	nt	žodžio pabaiga

Norint sužinoti, kiek yra žodžių tekste, pakanka suskaičiuoti, kiek yra simbolių porų **tn** arba **nt**. Būtina įvertinti, kad prieš pirmąjį žodį eilutėje ir po paskutinio gali nebūti tarpo. Jiems negalioja žodžio pradžios arba pabaigos požymis. Atskirai programuoti tokios situacijos analizę nepatogu. Galima teksto eilutę analizės metu papildyti tarpo simboliais pradžioje ir gale. Gausime situaciją, kai visų žodžių pradžios ir pabaigos požymiai bus vienodi.

1 pratimas. Programa **Pratimas1** tekstą nagrinėja eilutėmis, kurių simboliai saugomi masyve. Kiekvienos eilutės simboliai surašomi į masyvą. Čia pradžioje ir gale yra papildomai įrašomi tarpo simboliai.

```

program Pratimas1;
uses Crt;
const L = 300;      { Leistinas simbolių skaičius masyve }
      Failas = 'Prat1.pas';
type  mas = array[ 0..L+1 ] of char;
{-----}
procedure Eilute( var A : mas;  n : integer;
                  var K : integer);
var i : integer;    { Randamas žodžių skaičius masyve A(n) }
begin
  K := 0;
  for i := 1 to n do
    if ( A[ i-1 ] = ' ' ) and ( A[ i ] <> ' ' )
      then K := K+1;
end;
{-----}
procedure Kiek( var Zd : integer);
var F : text;
    A : mas;      n : integer;
    EilZod : integer;
begin
  Assign( F, Failas);  Reset( F );
  Zd := 0;
  while not Eof( F ) do begin
    n := 0;          { Skaitoma viena failo eilutė }
    while not EoLn( F ) and ( n < L ) do begin

```

```

        n := n + 1;  Read( F, A[ n ] );           end;
    ReadLn( F );
    A[ 0 ] := ' ';  A[ n+1 ] := ' '; { Papildymas tarpais }
    Eilute( A, n, EilZod );           { Žodžių skaičius eilutėje }
    Zd := Zd + EilZod;                { Žodžių skaičius tekste }
end;  Close( F );
end;
{-----}
var Zod : integer;  spalvos : integer;
begin
    spalvos := TextAttr;
    TextBackground( Blue ); ClrScr; TextColor( Black );
    Kiek( Zod );
    GoToXY( 20, 10 );
    Write( 'Zodziu tekste => ', Failas, ' <= yra' );
    GoToXY( 30, 12 ); TextColor( Red ); Write( Zod:5 );
    ReadLn;
    TextAttr := spalvos; ClrScr;
end.

```

✓ Papildykite programą procedūra, kuri išvestų rezultatus ekrano viduryje nurodyto dydžio ir spalvos lange.

✓ Papildykite programą veiksmiais, kuriais rastumėte kiek eilučių buvo tekste.

✓ Išsiaiškinkite, kam procedūroje **Kiek** vidiniame cikle (skaitant vienos eilutės simbolius) reikalinga sąlyga ($n < L$), kodėl negalima rašyti ($n \leq L$).

✓ Kokiems duomenims esant programa neteisingai suskaičiuos žodžius?

■ 2 pratimas. Programoje `Pratimas1` galima atsisakyti masyvo. Procedūra eilutės žodžių skaičiui surasti turėtų būti tokia:

```

procedure Eilute( var F : text;  var K : integer );
var  s1, s2 : char; { Randamas žodžių kiekis eilutėje }
begin
    K := 0;  s1 := ' ';  s2 := ' ';
    while not EoLn( F ) do begin
        s1 := s2;  Read( F, s2 );
        if ( s1 = ' ' ) and ( s2 <> ' ' ) then K := K+1;
    end;  ReadLn( F );
end;

```

✓ Pakeiskite procedūrą **Kiek** taip, kad ji teisingai dirbtų su naująja procedūros **Eilute** versija.

✓ Kur ir kaip reikėtų pertvarkyti programą, norint sužinoti, kiek yra

žodžių, kurių pirma raidė yra 'A'? Padarykite tai.

✓ Kur ir kaip reikėtų pertvarkyti programą, norint sužinoti, kiek yra žodžių, kurių pirma ir paskutinė raidės yra vienodos? Padarykite tai.

3 pratimas. Žaidimas. Ekrane rodoma viena programos teksto eilutė. Klaviatūra nurodome simbolį, kurio visi atkartojimai iš teksto "nukrenta" į ekrano apačią.

```
program Pratimas3;
uses Crt;
type Mas = array[ 1.. 80 ] of char;
{-----}
procedure Eilute( var E : Mas; var n : integer);
var Sim : char;
begin
  Randomize; { Generatoriaus pradinė reikšmė }
  n := 0; { Generuojama eilutė }
  while ( n < 80 ) do begin
    Sim := Chr( Random( 156 )); { Atsitiktinis simbolis }
    if ( Uppcase( Sim ) >= 'A' ) and { Ar raidė ? }
      ( Uppcase( Sim ) <= 'Z' ) then
      begin n := n+1; E[ n ] := Sim; end; { Papildoma eilutė }
    end;
  end;
{-----}
procedure Mesti( E : Mas; n : integer; sim : char);
var i, j : integer;
begin
  for i := 1 to n do { Eilutės peržiūra }
    if E[ i ] = sim then begin { Pasirinktas simbolis }
      TextColor( 15 );
      for j := 2 to 20 do begin { Simbolio rašymas }
        GoToXY( i, j ); Write( sim ); Delay( 200 );
      end;
    end; end;
{-----}
var Raid : char; Eil : Mas; n : integer;
begin
  Eilute( Eil, n );
  TextBackground( Blue ); ClrScr; TextColor( 7 );
  Raid := ' '; GoToXY( 25, 24 );
  WriteLn('Kokią raidę išmesti iš teksto esančio viršuje ???');
  WriteLn( 'Pabaiga - klavišas ESC ');
  GoToXY( 1,1 ); Write( eil ); TextBackground( Red );
  while Raid <> #27 do begin { Pabaiga: Esc }
    GoToXY( 1,1 ); Raid := ReadKey; { Skaitomas simbolis }
```

```

Mesti( Eil, n, Raid );      { Simbolis ekrane "krinta" }
end;                        end.

```

✓ Išsiaiškinkite ką programa daro, o po to patikrinkite, ar Jūs teisingai supratote programos tekstą.

✓ Pakeiskite programą taip, kad ekrane būtų dvi teksto eilutės, su kuriomis būtų atliekami tie patys veiksmai.

Savarankiško darbo užduotis

Tekstiniame faile surašytos žmonių pavardės ir vardai. Viena eilutė - vienam žmogui. Žodžiai skiriami tarpo simboliais. Surasti ir surašyti į rezultatų failą vardus po vieną eilutėje, nurodant šalia, kiek žmonių iš sąrašo jį turi. Ekrane ir failo gale parašyti, kiek buvo vyriškų vardų (baigiasi raide s) ir kiek moteriškų (visi likusieji), koks moteriškas ir koks vyriškas vardai populiariausi.

5.2. Aibės duomenų tipas

Aibėmis Paskalio kalboje vadinami diskretinio tipo duomenų rinkiniai. Jiems taikomos sąjungos (+), sankirtos (*), skirtumo (−) ir santykio (=, <, <=, >=, in) operacijos. Paskalio kalbos aibės atitinka matematikoje vartojamą nesutvarkytų aibių poaibį. Elementų, iš kurių sudaroma aibė, tipą vadinsime baziniu. Aibės tipo aprašo sintaksinė struktūra tokia:

<Vardas> = set of <bazinis tipas>

Aibės tipo aprašymo pavyzdžiai:

```

type  raides = set of 'A' .. 'Z';
      numeriai = set of 1..200;

```

Aibės tipo kintamiesiems galima suteikti priskyrimo sakiniu bet kokius jų bazinių tipų reikšmių rinkinius, kurie yra žymimi laužtiniuose skliaustuose:

```
[1,3..10,12], [x,y], ['A', 'C', 'D'], [x,y], [ ]
```

Struktūra [] žymi tuščią aibę. Naujų aibės reikšmių formavimas aprašomas operacijomis +, −, *. Aibių sąjunga (+) formuoja naują aibę su visų argumentų elementais, o sankirta (*) atskiria bendrus argumentų elementus. Skirtumo operacijoje (−) yra atrenkami tie turinio elementai, kurių nėra atėminyje. Aibėms taikomus veiksmus iliustruoja 1 lentelė.

1 lentelė. Veiksmai su aibėmis

Išraiška	Rezultatas
$['a', 'b', 'd'] + ['c']$	$['a'..'d']$
$[0..5] + [0..9]$	$[0..9]$
$[0..5] - [0..9]$	$[]$
$[0..9] - [0..5]$	$[6..9]$
$[0..9] * [0..5]$	$[0..5]$

2 lentelė. Santykio operacijų taikymas

Santykis	Reikšmės true sąlyga
$a=b$	Aibės turi tuos pačius elementus
$a \supset b$	Elementų sudėtis skiriasi
$a \subseteq b$	Visi aibės a elementai yra ir aibėje b
$a \supseteq b$	Visi aibės b elementai yra ir aibėje a
$c \text{ in } a$	Elemento c reikšmė yra aibėje a (elementas c aibei a bazinis)

Santykio operacijose (2 lentelė) galima lyginti tik vienodo tipo elementų aibes. Lygiomis yra vadinamos iš tų pačių elementų sudarytos aibės, pavyzdžiui, santykio $['a', 'b'] = ['b', 'a']$ rezultatas yra true.

Aibių santykiais patogų aprašyti sudėtingas logines sąlygas. Pavyzdžiui, tikrinant, ar simbolis s yra didžioji raidė, galima vartoti santykį: $s \text{ in } ['A'..'Z']$.

Įvairios Paskalio kalbos versijos taip pat riboja maksimalų aibės elementų skaičių. Pavyzdžiui, Turbo Paskalio kalboje šis skaičius lygus 256.

4. Pratimas. Programa ekrane parodo raides, kurios buvo panaudotos programos tekste: ekrane surašomos panaudotos didžiosios raidės, o po to mažosios. Aibėje **R** yra surašomi visi tekste panaudoti simboliai.

```
program Pratimas4; { Programa yra faile 'Prat4.pas' }
uses Crt;
const Failas = 'Prat4.pas';
type aibe = set of char;
{-----}
procedure Skaito( var R : aibe );
var F : text; sim : char;
begin
    Assign( F, Failas); Reset( F );
```

```

R := [];
while not Eof( F ) do begin
    Read( F, sim );    R := R + [sim];
end;
Close( F );
end;
{-----}
procedure Rodo( s1, s2 : char;  R : aibe );
var sim : char;
begin
    for sim := s1 to s2 do
        if sim in R then Write( sim:2 );
        WriteLn;
    end;
{-----}
var R : aibe;
begin
    ClrScr;
    WriteLn( 'Programoje buvo tokios raidės: '); WriteLn;
    TextColor( Green );
    Skaito( R );
    Rodo( 'A', 'Z', R );
    Rodo( 'a', 'z', R );
    ReadLn;
end.

```

5. Pratimas. Dvi draugės turi lipdukų albumus ir turi atliekamų lipdukų mainams. Reikia parašyti programą, kuri padėtų mergaitėms atlikti mainus. Visi lipdukai turi savo numerius, sutampančius su jų vietos albume numeriais. Pažymėkime LA, LB albumuose esančių lipdukų rinkinius, o KA, KB rinkinius lipdukų, skirtų mainams. Albume telpa 150 lipdukų, kurie numeruojami eilės tvarka, pradedant vienetu.

```

program Lipdukai;
uses Crt;
const  Pirma = 'Rasa';    Antra = 'Rita';
        L = 150;
type  aibe  = set of 1..L;
{-----}
procedure Langas( x1, y1, x2, y2, fonas : integer);
begin
    Window( x1, y1, x2, y2 );
    TextBackground( fonas ); ClrScr;
end;
{-----}
{ Klaviatūra įvedamų lipdukų numerių aibės formavimas. }

```

```

{ Įvedamo sąrašo srauto pabaiga nurodoma skaičiumi nulis}
procedure Ivesti ( var A : aibe );
var sk : integer;
begin
    sk := 1;  A := [];
    while sk <> 0 do begin
        TextColor( Black );
        Write('Lipduko numeris /įvedimo pabaiga- nulis/: ');
        TextColor( Red );
        ReadLn( sk );  A := A + [ sk ];
    end;
    A := A - [0];
end;
{-----}
{ Aibėje esantys skaičiai parodomi ekrane }
procedure Rodyti( A : aibe );
var k : integer;
begin
    for k := 1 to L do
        if k in A then Write( k:4 );
    end;
    {-----}
var  LA, KA,  LB, KB : aibe;
    Ekranas : byte;
begin
    Ekranas := TextAttr;  Langas( 1, 1, 80, 25, Black );
                        { Pirmos mergaitės lipdukų albume sąrašas }
    Langas( 10, 5, 60, 20, White ); TextColor( Green );
    WriteLn( Pirma, ' turi tokius lipdukus:');
    Ivesti( LA );
        { Antros mergaitės keitimui skirtų lipdukų sąrašas }
    Langas( 10, 5, 60, 20, White ); TextColor( Green );
    WriteLn( Antra, ' turi keitimui tokius lipdukus:');
    Ivesti( KB );  { Antrosios kolekcionierės lipdukai }
                        { Patarimai pirmai mergei }
    Langas( 1, 1, 80, 25, Black );

    Langas( 5, 2, 70, 7,  White ); TextColor( Green );
    WriteLn( Pirma, ' turi tokius lipdukus:');
    Rodyti( LA );  ReadLn; { Pirmosios sąrašas }

    Langas( 5, 8, 70, 13,  Green ); TextColor( Black );
    WriteLn( Pirma, ' gali gauti tokius lipdukus:');
    Rodyti( KB-LA );  ReadLn; { Aibių skirtumas }

    Langas( 5, 15, 70, 20, Green ); TextColor( Black );
    WriteLn( Pirma, ' kolekcijoje bus lipdukai:');

```

```
Rodyti( LA+ (KB-LA) ); ReadLn; { Rezultatas }
```

```
Langas( 1, 1, 80, 25, Black );
```

```
TextAttr := Ekranas;
```

```
ClrScr;
```

end.

✓ Papildykite programą sakiniais, organizuojančiais tokią pat paslaugą antrai mergaitei.

✓ Padarykite paslaugą organizuojančią procedūrą ir ją pritaikykite abiem mergaitėms.

■ 6 Pratimas. Reikia surasti visus pirminius skaičius, esančius pirmame šimtuke.

```
                { Pirmojo šimtuko pirminiai skaičiai }
program Pirminiai;
uses Crt;
type Sk = set of 1..99;
{-----}
procedure Rasti( var P : Sk ); { P- pirminiai skaičiai }
var A : Sk; dal, k : integer;
begin
    A := [ 2..99 ]; P := [ 1 ];
    while A <> [] do begin { Nagrinėjamų skaičių aibė }
        dal := 2;          { Pirminio paieška }
        while not( dal in A ) and ( dal < 99 ) do
            dal := dal + 1;
        P := P + [ dal ]; { Pirminių skaičių aibės papildymas }
        for k := 2 to 99 do { Pašalinami dalūs iš dal }
            if k mod dal = 0 then A := A - [ k ];
    end; end;
{-----}
procedure Rodo( P : Sk );
var k : integer;
begin
    for k := 1 to 99 do begin
        if k in P then TextColor( Red )
        else TextColor( Black );
        if k mod 10 = 0 then WriteLn( k:3 )
        else Write ( k:3 );
    end; end;
{-----}
var Pirm : sk;
begin
    TextBackground( Blue ); ClrScr; TextColor( Yellow );
```

```

WriteLn;
WriteLn('Pirmo šimtuko pirminiai skaičiai raudoni:');
Window( 10, 5, 40, 15 ); TextBackground( White );
ClrScr;
  Rasti( Pirm );  Rodo( Pirm );
  ReadLn
end.

```

Savarankiško darbo užduotis.

Failo pirmoje eilutėje yra surašyti loto žaidimo laimingų rutuliukų numeriai (visi numeriai mažesni už šimtą). Kiekvienoje kitoje eilutėje yra žaidėjų bilietuose esančių numerių sąrašai. Reikia surasti laimėtojus, jeigu tokių yra. Laimėtojo numerių rinkinys turi sutapti su žaidime iškritusių rutuliukų numerių rinkiniu. Reikia surasti, kokie numeriai iš laimingųjų yra visuose žaidėjų rinkiniuose. Jeigu nėra, pranešti ekrane.

5.3. Eilutės duomenų tipas

Tai simbolių seka, turinti ne daugiau kaip 255 simbolius. Veiksams su ja yra paprogramių rinkinys.

Tekstas, saugomas tekstiname faile, yra neribotos apimties. Jo apimtį riboja techninės informacijos saugojimo priemonių galimybės. Sudėtingus veiksmus atlikti, naudojant tik char tipo kintamuosius, sudėtinga, o kartais ir neįmanoma. Naudoti simbolių masyvus nevisuomet racionalu, nes:

- ☞ masyvas nevisada gali talpinti visus teksto simbolius;
- ☞ kuriami algoritmai darbui su teksto dalimis tampa sudėtingi;
- ☞ programuotojas visus veiksmus su simboliais masyve aprašo kiekvienu atveju, t.y. sudaro savo priemonių darbui su tekstu rinkinį.

Turbo Paskalis turi eilutės duomenų tipą string. Šis duomenų tipas gali būti paaiškintas tokia iliustracija:

```
type string = array [ 0..255 ] of char;
```

Visi string tipo kintamieji turi simbolių masyvo savybes, todėl juos galima indeksuoti. Kartu tai yra duomenų tipas, kuriam yra suteiktos atitinkamos savybės, kaip pavyzdžiui, eilutės ilgis saugomas vietoje, nusakomoje nuliniu indeksu; simbolių seka (eilutė) nagrinėjama kaip reikšmė, kuriai yra leistini veiksmai: priskyrimas, įvedimas, išvedimas, palyginimas, sujungimas. Eilutės duomenų tipas ir leistini veiksmai su šio tipo kintamaisiais yra pateikiami kalbos aprašyme. Programuotojui būtina

žinoti, kokias priemones Turbo Paskalis turi darbui su eilutėmis. Jos palengvina programų, skirtų teksto analizei ir tvarkymui, rašymą. Lentelėse 3 ir 4 yra tų priemonių sąrašas. Detalesnių paaiškinimų reikėtų ieškoti Turbo Paskaliui skirtuose leidiniuose.

☞ Visos priemonės, skirtos darbui su eilutės tipo kintamaisiais, naudoja ir koreguoja eilutės ilgį, užrašytą nulinėje eilutės vietoje. Eilutės simboliai saugomi pradedant pirmąja vieta. Esant simbolių daugiau, negu gali saugoti eilutės tipo kintamasis, pertekliniai atmetami.

☞ Dirbant su string tipo kintamuoju kaip su masyvu, programuotojas pats turi koreguoti eilutės ilgį ir pasirūpinti, kad visi veiksmas būtų korektiški, ypač kai norima naudoti ir Paskalio priemones darbui su eilute. Naudojant paprogrames, skirtas darbui su eilutėmis, eilutės ilgis koreguojamas automatiškai.

Pavyzdžiui, kai `var A : string,`

tai veiksmu `A := 'KAUNAS'`

šeši simboliai užrašomi pradedant pirma vieta, o `A[0]` užrašomas eilutės ilgis: šeštas simbolis pagal standartinę kodų lentelę.

Jeigu dabar užrašysime `A[7] := 'T'`, tai to simbolio niekas nepastebės. Pavyzdžiui, `Write (A)` spausdins tik pirmus šešis simbolius. Turime koreguoti eilutės ilgį: `A[0] := Chr(7)`.

Naudojant darbo su eilutėmis paprogrames, eilutės ilgis koreguojamas automatiškai.

3 Lentelė Procedūros darbui su eilutėmis.

Kreipinys	Procedūros paskirtis
Delete(E, p, n)	Eilutėje E pašalina n simbolių, pradedant p.
Insert(D, E, p)	Įterpia duotą eilutę D į turimą eilutę E, pradedant pozicija p.
Str(R, E)	Formuoja aritmetinio reiškinio R reikšmės simbolinį ekvivalentą eilutėje E.
Val(E, R, c)	Formuoja kintamojo R reikšmę, lygią eilutės E skaitmeniniam ekvivalentui. Rezultato tipas (real arba integer) atitinka kintamojo R tipą. Kintamasis c nurodo pirmojo klaidingo simbolio poziciją eilutėje E. Jeigu klaidų nėra, tuomet <code>c = 0</code> .

4 Lentelė Funkcijos darbui su eilutėmis.

Kreipinys	Funkcijos paskirtis
Length (E)	Skaičiuoja eilutės ilgį.
Copy (E, p, n)	Iš eilutės E išskiria dalį eilutės, pradedant pozicija p. Simbolių kiekis nurodomas n.
Pos (D, E)	Nurodo duotos eilutės D pradžios poziciją eilutėje E. Jeigu eilutė D neįeina į eilutę E, tai rezultatas yra reikšmė nulis.
Concat (E1, E2, ... ,En)	Sujungia nurodytas eilutes į vieną. Tai tolygu veiksmui: $S := E1 + E2 + \dots + En$

7 pratimas. Reikia padaryti programą, kuri duotame tekste nurodytą simbolių seką pakeistų nauja. Veiksmą nurodo komanda, įvedama klaviatūra. Jos formatas toks:

<Komanda> /<sena seka>/<nauja seka>/

Koreguojamojo teksto ir rezultato failų vardai nurodomi klaviatūra. Duomenys pateikiami korektiškai ir programoje jie netikrinami.

```

program Keitimas;
uses Crt;
const   Kom = 'Keisti';  { Komandos atpažinimui }
{-----}
procedure Skaidymas(      E   : string;
                        var S1, S2, S3 : string);
var  p : integer;
begin
    { Pašalinami tarpo simboliai }
    while E[1] = ' ' do Delete( E, 1, 1);
    p := Pos( '/', E );
    S1 := Copy( E, 1, p-1);  { Komandos pavadinimas }
    Delete( E, 1, p );       { Pašalinama komanda iš E }
                                { Komandos gale šalinami tarpai }
    while S1[ Length( S1 )] = ' ' do
        Delete( S1, Length( S1 ), 1 );
    p := Pos( '/', E );
    S2 := Copy( E, 1, p-1 ); { Senoji simbolių seka }
    Delete( E, 1, p );
    S3 := Copy( E, 1, Pos( '/', E )-1 ); { Naujoji seka }
end;
{-----}
procedure Koregavimas( var F, R : text;
                      S, N : string );
var
    E : string;
    p , t : integer;

```

```

begin
  t := Length( S );      { Senosios sekos ilgis  }
  while not Eof( F ) do begin
    ReadLn( F, E );      { Eilutė iš duomenų failo}
    while Pos( S, E ) <> 0 do begin
      p := Pos( S, E ); { Senosios sekos pradžia }
      Delete( E, p, t ); { Senoji seka šalinama  }
      Insert( N, E, p ); { Naujoji seka įterpiama }
    end;
    WriteLn( R, E ); { Sutvarkyta eilutė užrašoma}
  end;
end;
{-----}
var
  K : string;      { Komandinė eilutė  }
  S1, S2, S3 : string; { Komandos dalys  }
  FD, FR : string[ 12 ]; { Failų vardai  }
  F, R : text;
begin
  ClrScr;
  WriteLn( 'Iveskite komandinę eilutę. Formatas:');
  WriteLn(
    ' Keisti /sena_simbolių_seka/nauja_simbolių_seka/' );
  ReadLn( K ); { Komandinės eilutės įvedimas}
  Skaidymas( K, S1, S2, S3 );
  if S1 <> Kom then begin { Ar tinkama komanda? }
    WriteLn( 'Nevykdoma komanda: ', S1); Halt;
    end;
    { Failų vardų įvedimas ir paruošimas darbui }
  Write( 'Duomenų failo vardas = '); ReadLn( FD );
  Assign( F, FD ); Reset( F );
  Write( 'Rezultatų failo vardas = '); ReadLn( FR );
  Assign( R, FR ); Rewrite( R );
  Koregavimas( F, R, S2, S3 ); { Koregavimas  }
  Close( F ); Close( R );
end.

```

Komandinės eilutės pateikino pavyzdys:

Keisti /end/KATINAS/

☞ Programa nedirbs (“amžinas” ciklas), jeigu senoji ir naujoji eilutės sutaps arba naujojoje bus senoji, pavyzdžiui: /sako/**Pasako**/.

Eilutėms taikomos santykio operacijos. Lyginant eilutes, iš eilės yra lyginami jose esančių simbolių kodai. Didesne yra laikoma ta eilutė, kurioje anksčiau aptinkamas didesnio kodo simbolis. Primename, kad raidžių kodai alfabeto tvarka didėja, o mažųjų raidžių kodai yra didesni už

didžiųjų raidžių kodus. Eilučių lyginimo pavyzdžiai:

veiksmas ir rezultatas	
'A'<'B'	true
'B'<'b'	true
'Turbo' >'Turbo'	true

veiksmas ir rezultatas	
'2'<'12'	false
'Mano'>'mano'	false
'and'>'a'	true

Sudedant eilutes, yra gaunamas jų junginys. Eilučių tvarkymo procedūrų taikymo pavyzdžiai:

```
{ var a, c: string[20]; }  
a:='katytė';  
c:= a+ 's';           { 'katytės' }
```

Ypač reikia būti atidiems vartojant eilutes įvedimo procedūrose, nes eilučių reikšmės įvedamos skirtingai nuo skaičių. Viena ReadLn sakiniu įvedant grupę skaičių, jų reikšmės viena nuo kitos atskiriamos tarpais, o eilutėms toks reikšmių atskyrimo būdas netinka, nes tarpo simboliai gali būti pačių eilučių elementais. Įvedant eilutės reikšmę iš klaviatūros, jos pabaiga yra nurodoma klavišu Enter. Jei simbolių yra perdaug, pertekliniai yra atmetami.

Eilutės labai glaudžiai susijusios su simboliu tipu, kurio pagrindu jos yra sudarytos. Dargi numatyta galimybė kreiptis į pavienius eilučių simbolius, nurodant juos indeksais, pavyzdžiui:

```
{ var s: char; a: string[20]; }  
a:='namas';           s:=a[3];           { s = 'm' }
```

■ **8 pratimas.** Turime tekstą faile **‘Prat8.dat’**. Mus domina, kiek tekste kartų yra sutinkamas žodis **‘Kaunas’**. Žodžiai skiriami tam tikru skirtukų rinkiniu, iš eilutės į eilutę nekeliami.

Teksto apimtis neribojama. Patogiausia tekstą nagrinėti eilutėmis. Perskaičius eilutę papildome tarpo simboliais pradžioje ir gale. Dabar visų žodžių pradžios ir pabaigos požymiai tampa vienodi. Nurodyto sąlygoje žodžio paieškai eilutėje funkcija **Pos** netinka, nes ji vykdo eilutės analizę nuo pradžios, ieško pirmojo duotos simbolių sekos (ne žodžio) pakartojimo. Reikia padaryti savo analogišką funkciją, kuri atliktų simbolių sekos (arba žodžio) paiešką eilutėje nuo nurodytos pozicijos. Funkciją **Pos** galima naudoti, jeigu prieš tai surastą žodį pašalinsime iš eilutės. Yra dar viena pavojinga situacija: ieškomas žodis yra kito žodžio sudėtinė dalis (pvz., *Kaunasalis* . *KaunasKaunas*. *Kaunasi kauniečiai gerai*.). Išanalizavus įvairias situacijas, pastebime, kad pašalinus surastą žodį iki

galo, išvengsime pasitaikančių netikslumų. Šalinti žodžius iš duomenų eilutės galime, nes po analizės jos nerašome atgal į failą.

```

program Pratimas8;
uses Crt;
const Failas = 'Prat8.dat';
type skirtukai = set of char;
{-----}
procedure MestiZodi( var E : string;  n : integer;
                    A : skirtukai);
begin
    while ( n < Length( E )) and not( E[ n ] in A )
        do      Delete( E, n, 1 );
end;
{-----}
function Eilute( E : string; A : skirtukai;
                Z : string): integer;
var k, i : integer;
begin
    E := ' ' + E + ' ';      k := 0;
    while Pos( z, E ) <> 0 do begin
        i := Pos( z, E );
        if ( E[ i-1 ] in A ) and ( E[ i+ Length( z ) ] in A
)
            then begin
                k := k + 1;  Delete( E, i, Length( z ))
            end
            else
                MestiZodi ( E, i, A );
        end;
        Eilute := k;
    end;
{-----}
var F : text;  E : string;  Kiek : integer;
    A : skirtukai;
begin
    ClrScr;
    Assign( F, Failas );  Reset( F );
    Kiek := 0;      A := [ ',', ' ', '.', '?', ':', ';' ];
    while not Eof( F ) do begin
        ReadLn( F, E );
        Kiek := Kiek + Eilute( E, A, 'Kaunas' );
    end;
    Close( F );
    WriteLn('Tekste => ', Failas,
        ' <= yra zodziu ''Kaunas'':', Kiek:5 );
    ReadLn;

```

end.

Duomenų failo pavyzdys:

```
program Kaunas;      Kaunas
uses Crt;
    Kiek := Kiek + Eilute( E, A, ' Kaunas ' );
    WriteLn( 'Tekste => ',
    Failas, ' <= yra zodziu ' ' Kaunas '':', Kiek:5 );
    ReadLn;
end.
```

✓ Parašykite duomenų tekstą programai patikrinti. Pasistenkite sukurti visas galimas žodžio ‘Kaunas’ pozicijas tekste.

✓ Pakeiskite programą taip, kad ji atliktų teksto analizę žodžiui, nurodomam klaviatūra.

■ 9 pratimas. Aprašant įvedamų duomenų kontrolę, labai naudinga yra procedūra **Val(E, R, C)**. Ji formuoja kintamajame R formuoja eilutės E skaitmeninį ekvivalentą. Rezultato tipas (real arba integer) atitinka kintamojo R tipą, o integer tipo kintamasis C nurodo pirmo klaidingo simbolio poziciją eilutėje E (jei eilutė neturi skaitmeninio ekvivalento). Jei toks ekvivalentas yra, C = 0.

```
program Pratimas9;
uses Crt;
var   E : string;    X : integer;    C : integer;
begin
    ClrScr;
    E := '145';      Val( E,  X, C );
    if C = 0 then    WriteLn( E, ' Skaičius:',  X:5 )
                    else WriteLn( E, ' Klaida');
    E := '14a5';      Val( E,  X, C );
    if C = 0 then    WriteLn( E, ' Skaičius:',  X:5 )
                    else WriteLn( E, ' Klaidingas simbolis
nr.: ',
                                C ); { C reikšmė 3 }

    ReadLn;
end.
```

■ 10 pratimas. Patikrinkite, kaip dirba programa, skirta klaviatūra įvedamų sveikųjų skaičių kontrolei. Skaičiai įvedami po vieną. Jei skaičiuose pastebimi neleistini simboliai, į ekraną išvedamas pranešimas apie klaidą. Teisingai surinkti skaičiai siunčiami į rezultatų failą. Programoje taip pat yra skaičiuojami teisingai ir klaidingai įvestų skaičių kiekiai. Suformuotas skaičių failas gali būti naudojamas kaip duomenų

šaltinis kitose programose.

```
program Pratimas10;
uses Crt;
var      F : text;      E : string;
        g, b : integer;
        c, x : integer;
begin
  TextBackground( Black );  ClrScr; TextColor( Green );
  Assign( F, 'Result.dat');  Rewrite( F );
  g:= 0;  b:= 0;
  WriteLn('Programos darbas nutraukiamas įvedant "q"');
  repeat
    Write('Iveskite skaičių:');  ReadLn( E );
    Val( E, x, c );
    if c= 0 then begin      g := g+1; Write( F, x:8); end
  else if E <> 'q' then      begin
      b := b+1; TextColor( Red );
      WriteLn( 'Klaida skaičiuje: ', E );
      TextColor( Green );      end
  until E = 'q';
  WriteLn ( 'Teisingai įvedėte ', g, ' skaičius');
  WriteLn ( 'Neteisingai įvedėte ', b, ' skaičius');
  Close( F ); ReadLn;
end.
```

✓ Pakeiskite programą taip, kad ji galėtų tikrinti skaičius su trupmeninėmis dalimis.

✓ Pakeiskite programoje ciklą repeat ciklu while.

✓ Pakeiskite programą taip, kad klaidingai įvesti skaičiai būtų surašyti į atskirą klaidų failą vardu blogi.dat, o pranešime apie klaidą būtų įterptas šauktukas už pirmo neleistino simbolio.

Žodžių analizė.

Tvarkant tekstinius duomenis reikia mokėti atpažinti žodžius, gebėti juos keisti, šalinti, įterpti naujus. Tekste žodžiai viena nuo kito skiriami tam tikrais simboliais, kuriuos galime pavadinti skirtukais. Jais gali būti tarpas, kablelis, taškas, klausukas, šauktukas ir pan.

Tarkime, kad žodžiai skiriami žinomais skirtukais ir nekeliami iš eilutės į eilutę. Prieš pirmąjį ir po paskutiniojo žodžio eilutėje taip pat gali būti skirtukai. Reikia surasti ilgiausią žodį ir jo vietą tekste (eilutės numeris ir žodžio pirmo simbolio vieta toje eilutėje).

Žodį tekste nusako trys dydžiai: pradžios vieta, pabaigos vieta, ilgis (simbolių skaičius). Pakanka visuose veiksmuose turėti tik dvi charakteristikas, o trečiąją visuomet bus galima suskaičiuoti.

Žodžius atpažinti galime nagrinėdami du gretimus teksto simbolius, pavyzdžiui s1 ir s2:

s1	s2	prasmė
Tarpas (skirtukas)	netarpas (ne skirtukas)	žodžio pradžia s2
Tarpas (skirtukas)	tarpas (skirtukas)	intervalas tarp žodžių
netarpas (ne skirtukas)	netarpas (ne skirtukas)	žodžio vidus
netarpas (ne skirtukas)	tarpas (skirtukas)	žodžio galas s1

11 pratimas. Duotame tekste reikia surasti ilgiausią žodį ir jo vietą faile: eilutės numerį ir pradžios poziciją eilutėje. Programos pavyzdys parašytas nenaudojant duomenų failo simboliams masyvų. Masyvas naudojamas tik surasto žodžio simboliams saugoti. Skirtuku panaudotas tik tarpo simbolis.

```

program Pratimas11;
uses Crt;
const failas = 'Prat11.pas';
      L = 80;
type Masyvas = array[ 1..L ] of char;
{-----}
procedure Rasti( var F : text;
  var dzpr, dzil, dzeil : integer; { Pradžia, ilgis, eilutė }
      var Dz : Masyvas); { Žodis }
  { Analizuojamo žodžio pradžia, ilgis ir žodis }
var  zpr, zil : integer;      Z : Masyvas;
    e, s : integer; { Eilutės numeris, simbolio numeris }
    s1, s2 : char;      { Gretimi du failo simboliai }
begin
  dzeil := 0; dzpr := 0; dzil := 0; e := 0;
  while not Eof( F ) do begin
    zpr := 0; zil := 0; s := 0;
    s1 := ' '; s2 := ' '; e := e+1;
    while not EoLn( F ) do begin
      s1 := s2; Read( F, s2 ); s := s+1; {Skaitomas simbolis }
      if (s1 = ' ') and (s2 <> ' ') then { Žodžio pradžia }
        begin zpr := s; zil := 1; Z[ zil ] := s2;
      end
      else if (s1 <> ' ') and (s2 <> ' ') then { Žodyje }
        begin zil := zil + 1; Z[ zil ] := s2;
      end
    end
  end
end

```

```

end
  else if (s1 <> ' ') and (s2 = ' ') then {Zodzio pabaiga }
    if zil > dzil then      begin { Ilgesnis zodis }
      dzeil := e;   dzpr  := zpr;
      dzil  := zil; Dz   := Z;
      end;      end;
    { Po paskutinio zodzio eiluteje nebuvo tarpo }
    if (s2 <> ' ') and ( zil > dzil) then  begin
      dzeil := e;   dzpr  := zpr;
      dzil  := zil;  Dz   := Z;          end;
    ReadLn( F );                          { Pereiti i kita eilute }
  end;  end;
{-----}
var      { Ilgiausio zodzio eilute, pradzia, ilgis ir zodis }
  dzpr, dzil, dzeil : integer;  Dz : Masyvas;
  F : text;  i : integer;  Sp : integer;
begin
  Sp := TextAttr;
  TextBackground( Blue ); ClrScr; TextColor( Black );
  Assign( F, failas);  Reset( F );
  Rasti( F, dzpr, dzil, dzeil, Dz );
  Close( F );
  WriteLn( 'Ilgiausias zodis yra      ', dzeil:4, ' eiluteje');
  WriteLn( 'Ilgiausio zodzio pradzia', dzpr :4, ' vietoje');
  WriteLn( 'Ilgiausio zodzio ilgis  ', dzil :4, ' simboliu');
  WriteLn( 'Zodis:');  TextColor( Green );
  for i := 1 to dzil do Write( Dz[ i ]); WriteLn;
  ReadLn;  TextAttr := Sp; ClrScr;
end.

```

✓ Papildykite programą veiksmiais, kuriais būtų randamas ilgiausias žodis, esant duotam skirtukų rinkiniui.

✓ Pertvarkykite programą taip, kad rastų ilgiausią žodį, sudarytą tik iš skaitmenų (ilgiausias sveikas skaičius be ženklo).

✓ Pertvarkykite programą taip, kad ji surastų kiek duotame tekste yra žodžių, turinčių bent vieną Jūsų vardo raidę.

✓ Pertvarkykite programas darbui su eilutės duomenų tipu. Palyginkite jų algoritminį sudėtingumą.

5.4. Užduotys.

✧ Turime faile surašytus vaikų vardus po vieną eilutėje. Kitame faile yra surašyta skaičiuoklė (posmelis, naudojamas išskaičiavimui žaidžian

slėpynes). Reikia parašyti programą, kuri išskaičiuotų, kokia tvarka vaikai bėga slėptis ir kas lieka nežiūrėti.

✧ Turime du didelius sveikus skaičius. Kiekvienas iš jų gali turėti iki 80 skaitmenų. Jie užrašyti atskirose failo eilutėse. Reikia trečioje failo eilutėje parašyti tų skaičių sumą, o ketvirtoje – skirtumą. Programoje skaičius vaizduokite simbolių eilutėmis. Padarykite sudėties ir atimties veiksmams atskiras paprogrames.

✧ Faile surašyti duomenys apie stačiakampius: kairio viršutinio kampo koordinatės plokštumoje (x,y), aukštis a, plotis b ir spalva s. Kiekvieno stačiakampio duomenys surašyti atskirose eilutėse. Reikia stačiakampius atvaizduoti ekrane, kur kampo koordinatės sutaptų su ekrano koordinatėmis. Jeigu stačiakampis netelpa ekrane, tuomet jis nepiešiamas ir jo duomenys surašomi į kitą failą (nenupieštų stačiakampių sąrašas). Darbo pabaigoje pateikti statistiką: kiek kokios spalvos stačiakampių ekrane piešta ir kiek kokios spalvos stačiakampių nebuvo piešta.

✧ Turime tekstiniam faile tekstą, kurį reikia užšifruoti. Šifras: kiekvieną eilutę užrašyti atvirkščia tvarka, o teksto eilutes surašyti nuo galo. Padaryti dešifratorių.

✧ Turime tekstiniam faile tekstą, kurį reikia užšifruoti. Šifras: pagal nurodytą šifro faile lentelę, balsė keičiama priebalse, o priebalsė – balse. Nesantys lentelėje simboliai nekinta. Pvz., a-p, e-t, o-k. Tai galioja ir didžiosioms raidėms. Padarykite dešifratorių. Patikrinkite, ar Jūsų programa tiks bet kokiai duotai simbolių porai, jeigu lentelė teisinga, - nėra pasikartojančių simbolių.

7. Duomenų organizavimas

7.1. Dvimačiai masyvai

Dvimačiais masyvais arba matricomis vadinami tokie duomenų rinkiniai, kurių elementų vieta nurodoma dviem indeksais. Tokių masyvų elementus galima surašyti lentelėse su numeruojamais stulpeliais ir eilutėmis. Pirmasis elemento indeksas nurodo jo vietos lentelėje eilutę, o antrasis - stulpelį. Paskalio kalboje dvimatis masyvas gali būti aprašomas kaip masyvų masyvas:

type

```
masyvas = array [<Indekso aprašas>] of <Elementų tipas>;  
matrica = array [<Indekso aprašas>] of masyvas;
```

Turbo Paskalyje įvesta aprašymo modifikacija, kurioje iš karto aprašomi abu indeksai ir elementų tipas:

type

```
masyvas = array [<Pirmo indeksso aprašas>,  
                 <Antro indeksso aprašas>] of <Elementų tipas>;
```

Kreipinio į dvimačio masyvo elementą sintaksinė struktūra yra tokia:

<Masyvo vardas> [<Eilutės indeksas>, <Stulpelio indeksas>]

Pavyzdžiui, tarkime, kad turime tokius aprašus:

```
type      Mas  = array [ 1..6 ] of real;  
          Matr = array [ 1..5 ] of Mas;  
var       A, B : Matr;   C : Mas;
```

Tada kintamieji A ir B galės saugoti tokios matricos duomenis:

$$\begin{pmatrix} 1, & 2, & 3, & 4, & 5 \\ 6, & 7, & 8, & 9, & 10 \\ 11, & 12, & 13, & 14, & 15 \\ 16, & 17, & 18, & 19, & 20 \end{pmatrix}$$

Užrašas **A[2,3]** nurodo antros eilutės trečio stulpelio elementą, kurio reikšmė yra 8. Indeksus gali nurodyti išraiškos, kurių reikšmės yra suderinamos su indekso tipu.

Pastaba. Masyvų aprašymuose indeksams gali būti parenkamas ne tik sveikasis, bet ir bet kuris kitas diskretinis tipas (loginis, simbolinis, vardinis).

Kaip ir vienmačiams masyvams, dvimačiams masyvams galiojo galioja priskyrimo veiksmas: $B := A$, tačiau tokio veiksmo argumentai privalo būti vienodo tipo matricos.

1 pratimas. Tekstinio duomenų failo eilutėse surašytos atkarpų galų taškų koordinatės (x,y). Sudaryti programą kuri suskaičiuotų visų atkarpų ilgį ir surastų ilgiausią.

Šiame uždavinyje patogų dvimačio masyvo eilutes naudoti tiek analizuojamų duomenų, tiek rezultatų (atkarpų ilgių) saugojimui. Jeigu tokį masyvą aprašysime kaip vienmačių masyvų masyvą, turėsime papildomą galimybę vienu vardu nurodyti ištisą masyvo eilutę. Atsižvelgiant į šiuos samprotavimus, uždavinio sprendimui parinkti tokie duomenų tipai:

```
const    Daug = 100;    {Pagalbinio duomenų masyvo dydis}
type
                                {Atkarpos galų koordinatinių ir ilgių masyvas }
    Viena = array[ 1..5 ] of real;
                                {Atkarpų aprašymų rinkinys }
    Visos = array[ 1..Daug ] of Viena;
```

Programą siūloma konstruoti iš tokių pagalbinių procedūrų:

```
procedure Skaito ( var A : Visos; var n : integer);
```

{Užpildo masyvą A iš tekstinio failo skaitomomis taškų koordinatėmis.
Parametras n informuoja apie perskaitytų atkarpų aprašymų skaičių}

```
procedure Ilgiai( var A : Visos; var n : integer);
```

{Formuoja matricos A penktąjį stulpelį su atkarpų ilgių reikšmėmis}

```
procedure Ilgiausia( var A : Visos; var n, k : integer);
```

{Parametras n grąžina ilgiausios atkarpos indeksą masyve}

Pradinių duomenų skaitymo iš tekstinio failo procedūroje *Skaito* reikia atsižvelgti į kritinius atvejus, kai failas yra tuščias ir kai jo duomenys netelpa pagalbiniame masyve. Apie pirmąjį atvejį informuoja procedūros

grąžinamas perskaitytų eilučių skaičius (0). Antruoju atveju, procedūra perteklinius duomenis atmeta.

Pagal 1 pratimo užduotį sudarytos programos tekstas yra toks:

```
program Pratimas1;
const  Daug = 100;
       Duomenys = 'Atkarpa.dat';
type   Viena = array[ 1..5  ] of real;
       Visos = array[ 1..Daug ] of Viena;
{-----}
procedure Skaito (var A : Visos; var n : integer);
var  F : text;  i : integer;
begin
    Assign ( F, Duomenys );  Reset ( F );
    n := 0;
    while not Eof ( F ) and ( n < Daug ) do  begin
        n := n+1;
        for i := 1  to  4 do
            Read ( F, A[ n, i ] );
        ReadLn ( F );
    end;
    Close ( F );
end;
{-----}
procedure Ilgiai( var A : Visos; var n : integer);
var  i : integer;
begin
    for i := 1 to n do
        A[i,5] := Sqrt( Sqr( A[i,1] - A[i,3]) +
                        Sqr( A[i,2] - A[i,4] ) );
    end;
{-----}
procedure Ilgiausia(var A : Visos; var n, k : integer);
var  i : integer;
begin
    if n > 0 then k := 1  else  k := 0;
    for i := 1 to n do
        if A[i,5] > A[i,k] then k:= i;
    end;
{-----}
var
    A : Visos;
    n, d, s : integer;
```

```

begin
  Skaito( A, n );
  Ilgiai( A, n );
  Ilgiausia( A, n, d );
  if d > 0 then begin
    WriteLn( 'Ilgiausia atkarpa', d:3,
              ' Ilgis:', A[d,5]:6:2 );
    WriteLn( 'Jos koordinatės:');
    WriteLn( A[d,1]:6:1, A[d,2]:6:1,
              A[d,3]:6:1, A[d,4]:6:1 );
  end
  else WriteLn( 'Atkarpų sąrašas tuščias');
end.

```

Papildykite programa Pratimas1 taip, kad ji papildomai galėtų skaičiuoti tokias reikšmes:

- Ilgiausios atkarpos galų taškų atstumus nuo koordinatinių pradžios taško.
- Koordinatinių plokštumos, kuriame yra atkarpos pradžia numerį (jam rašyti duomenų masyvą papildykite 6-tu stulpeliu)
- Kiekviename koordinatinių plokštumos ketvirtyje prasidedančių atkarpų skaičius.
- Kiekvieno skirtingo ilgio atkarpų skaičius.

Skaičiuojant skirtingo ilgio atkarpų skaičius, reikia atkreipti dėmesį į realių skaičių lyginimo problemą. Taškų koordinatės galima išmatuoti ir jų ilgius skaičiuoti tik su tam tikra paklaida. Todėl reikia nuspręsti, į kokius ilgius skirtumus galima nekreipti dėmesio. Pavyzdžiui, procedūroje Kiek, kuri skaičiuoja atkarpų turinčių toki pat ilgį, kaip indekso k atkarpa, skaičių, laikoma, kad leistina ilgio skaičiavimo paklaida yra 0.001. Atkarpos, kurių ilgiai skiriasi mažesniu dydžiu, laikomos lygiomis. Skaičiavimo rezultatą šioje procedūroje grąžina parametras s.

```

procedure Kiek( var A : Visos; var n, k, s : integer);
var i : integer;
begin
  s:=0;
  for i := 1 to n do
    if Abs(A[i,5] - A[k,5]) <= 0.001 then
      s := s+1;
  end;
end;

```

7.2 Masyvo tipo konstantos

Paprastos Paskalio kalbos konstantos, kurių vardus reikšmėmis programos tekste keičia pirminis procesorius gali būti tiktai elementarių tipų ir eilutės tipo. Masyvų ir kitų struktūrinių tipų konstantas galima aprašyti tiktai tipizuotomis konstantomis vadinamomis struktūromis. Jų deklaravimo sintaksė:

`<Vardas> : <Tipas> = <Konstantos reikšmė>`

Taip gali būti apibrėžiamos tiek skaliarinės, tiek struktūrinės konstantos. Tipizuotos konstantos skiriasi nuo paprastų tuo, kad joms skiriama vieta atmintyje. Be to, jų reikšmės programos darbo metu galima keisti. Todėl jos dar vadinamos inicializuojamais kintamaisiais.

Vienmačių masyvų – konstantų elementų reikšmės išvardinamos lenktiniuose skliaustuose:

```
type langas = array[1..4] of byte;  
const k: langas = (1, 1, 1, 1);
```

Dvimačių masyvų tipo konstantose skliaustais nurodoma kiekvienam indeksui skiriama reikšmių grupė, pavyzdžiui:

```
type kurs = array[1..2,1..2] of byte;  
const k: kurs = ((1,1), (1,1));
```

Masyvo tipo konstantos yra patogi giminingų programos tvarkomų objektų grupavimo priemonė. Taip pat jose galima organizuoti programos testavimui skirtus duomenis.

2 pratimas. Sudaryti programą, kuri rastų duotame plokštumos taškų rinkinyje du taškus, tarp kurių atstumas yra didžiausias.

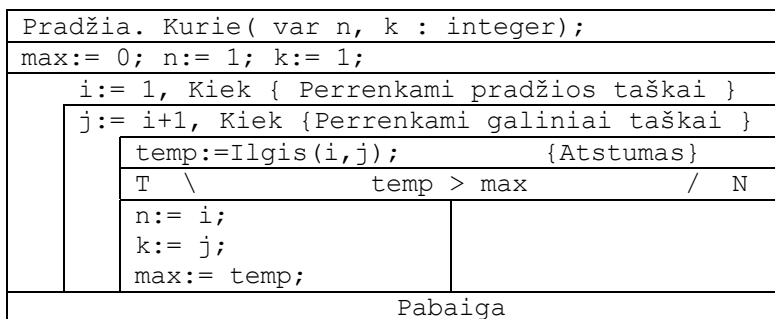
Šis uždavinys yra nepilnai apibrėžtas, nes nenurodyta, iš kur programa gaus duomenis. Tarkime, kad iš pradžių norime išsiaiškinti didžiausio atstumo paieškos algoritmą, o duomenų įvedimo problemos sprendimą atidedame vėlesniam laikui. Tokiu atveju, nagrinėjamų taškų koordinatės patogų aprašyti dvimačio masyvo tipo konstanta, kurioje kiekviena eilutė aprašo dvi taško koordinatės:

```
const Kiek = 5;  
type  
    Taskai = array [ 1..Kiek, 1..2 ] of real;
```

const

Duom: Taskai = ((1, 2), (2, 3), (3, -4), (-4, 5), (-5, 6));

Taškų koordinatų masyvo analizę siūloma organizuoti visų atstumų peržiūros būdu. Iš pradžių randamas toliausiai nuo pirmojo taško nutolęs taškas. Po to analizuojami atstumai nuo antrojo ir kitų taškų, ieškant, ar nėra dar daugiau nutolusių taškų porų. Tokiai peržiūrai skirtos procedūros *Kurie* organizavimo principus iliustruoja 7.1 struktūrograma. Joje atstumui tarp dviejų indeksais nurodomų taškų skaičiuoti vartojama pagalbinė funkcija *Ilgis*.



7.1 pav. Daugiausiai nutolusios poros paieška masyve

Funkcijos *Ilgis* ir viso uždavinio sprendimo aprašymas pateikti programoje *Pratimas1*.

```
program Pratimas1;
const Kiek = 5;
type
  Taskai = array[ 1..Kiek, 1..2 ] of real;
const
  Duom: Taskai = ((1,2), (2,3), (3,-4),
                  (-4,5), (-5,6));
var i, n, k: integer;
{-----}
{ Atstunas tarp indeksų nurodomų masyvo Duom taškų }
function Ilgis( n, k : integer):real;
begin
  Ilgis := Sqrt(Sqr(Duom[n, 1] - Duom[k, 1])+
               Sqr( Duom[n, 2] - Duom[k, 2]));
end;
{-----}
procedure Kurie(var n, k : integer);
```

```

var
  i, j : integer;
  max, temp: real;
begin
  max:= 0;
  n:= 1; k:= 1;
  for i := 1 to Kiek do
    for j:=i+1 to Kiek do begin
      temp:= Ilgis (i, j);
      if temp > max then
        begin n:= i; k:= j; max:= temp end;
      end;
    end;
  end;
  {-----}
begin
  Writeln('Taškų masyvas');
  for i:= 1 to Kiek do
    WriteLn( Duom [i, 1]: 8:2, Duom [i, 2]: 8:2);
  Kurie (n, k);      {Daugiausiai nutolę taškai}
  WriteLn ('Didžiausias atstumas yra tarp ', n ,
           ' ir ', k, ' taškų');
  Writeln ('Atstumo dydis: ', Ilgis (n, k): 8:2);
end.

```

- Pertvarkykite pratimo programą taip, kad tiriamų taškų koordinatės būtų galima įvesti klaviatūra.
- Pertvarkykite pratimo programą taip, kad tiriamų taškų koordinatės būtų skaitomos iš tekstinio failo, kuriame kiekvieno taško koordinatėms skiriama atskira eilutė.

3 pratimas. Tarkime, kad reikia sudaryti skaičių analizės programą, kuri tirtų klaviatūra įvedamų skaičių srautą. Iš srauto reikia atskirti tuos skaičius, kurie yra dalūs iš 5.

Programa bus įdomesnė, jeigu ryšiams su vartotoju skirtame ekrane sudarysime du langus. Kairįjį langą skirkime visų įvedamų duomenų kontrolei, o dešinįjį - atrenkamiems skaičiams. Programa bus dar įdomesnė, jei atrenkami skaičiai bus ne tik parodomi dešiniajame lange, bet ir ištrinami kairiajame.

Sudarant tokią programą, reikia atkreipti dėmesį į tai, kad modulio Crt procedūra Windows leidžia ekrane sudaryti tik vieną aktyvų langą. Norint

dirbti ekrane su dviem langais, reikia saugoti šių langų parametrų rinkinius ir, prieš nukreipiant duomenis į langą, procedūra Window aktyvizuoti tą langą, kuriam duomenys yra skirti.

Išsamiam tekstinio ekrano lango aprašymui reikia žinoti jo kampų koordinates, fono ir simbolių spalvas ir rašymui parengtą vietą rodančio kursoriaus koordinates. Visi šie parametrai, išskyrus kursoriaus koordinates paprastai būna nekeičiami, todėl juos patogiu saugoti pagal lango indeksą parenkamomis masyvo tipo konstantomis.

Manipuliavimui tekstiniais ekrano langais siūloma panaudoti tokias duomenų struktūras ir konstantas:

type

```
                {Ekrano lango kampų koordinatės ir fono spalva}  
koord = array[ 1..5 ] of byte;  
                {Dviejų langų koordinačių rinkinys}  
param = array[ 1..2 ] of koord;  
                {Kursoriaus koordinatės lange}  
vieta = array[ 1..2 ] of byte;  
                {Masyvas su dviejų langų kursorių koordinatėmis}  
kuris = array[ 1..2 ] of vieta;
```

const

```
                {Pradinės ekrano langų kursorių pozicijos}  
K : kuris = ((1, 1), (1, 1));  
                { Langų vieta ir spalvos }  
Lan : param = ((5, 4, 35, 24, Blue), (45, 4, 75, 24, Magenta));
```

Keičiant programoje aktyvų ekrano langą, į kurį nukreipiami duomenys, patogiu turėti aktyvaus lango keitimo procedūrą, kuri išsaugotų tampančio pasyviu lango kursoriaus koordinates, aktyvizuotų naują langą ir atstatytų jame buvusią kursoriaus poziciją. Tokius veiksmus programoje *Pratimas3* atlieka procedūra *Langs*.

Skaičiaus perkėlimą į kitą langą taip pat galima padaryti įdomesni. Pavyzdžiui, galima pasiekti, kad jis atsisveikintų pranešimu “Bye – bye”, kuris būtų rodomas tiksliai trumpą laiką, ir būtų ištrinamas pirmajame lange. Tokius veiksmus pratimo programoje aprašo procedūra *Skusk*.

```
program Pavyzdys_7_3;  
uses Crt;
```

```

type
    koord = array [ 1..5 ] of byte;
    param = array [ 1..2 ] of koord;
    vieta = array [ 1..2 ] of byte;
    kuris = array [ 1..2 ] of vieta;
const
    K    : kuris=((1, 1), 1, 1));
           {Langu aprašymai}
    Lan  : param=((5, 4, 35, 24, Blue),
                  (45, 4, 75, 24, Magenta));
var  x  : integer;    {Tiriamas skaičius}
{-----}
procedure Langas (n: byte);
    {n - parengiamo išvedimui lango indeksas}
var  t:   byte;
begin
    {Buvusio aktyviu lango indeksas}
    t:= n mod 2 +1;
    {Kursoriaus koordinačių išiminimas}
    k[t,1] := WhereX; k[t,2] := WhereY;
    {Lango su indeksu n aktyvizavimas }
    TextBackground( lan[n,5]);
    Window( Lan[n,1], Lan[n,2], Lan[n,3], Lan[n,4]);
    {Kursoriaus atstatymas senoje vietoje}
    GoToXY( K[n,1], K[n,2]);
end;
{-----}
procedure Skusk(var x: integer);
begin
    { Perkeliama skaičiaus trinimas }
    GoToXY(1, WhereY-1);
    DelLine;
    { Atsisveikinimas ir jo pranešimo trinimas }
    Write('Bye-bye'^G); Delay(2000);
    GoToXY(1, WhereY); ClrEol;
    { Skaičiaus rašymas antrame lange }
    Langas(2); WriteLn(x);
    { Pirmo lango aktyvizavimas }
    Langas(1);
end;
{-----}
begin
    {Pradinio ekrano vaizdo formavimas}

```

```

TextBackground(Black);      ClrScr;
GoToXY(15, 2);
Write('Skaičių analizė. Pabaiga - 0');
GoToXY(10, 3);
Write('Dalūs iš 5 skaičiai perkeliami
                                           į dešinį langą');

Langas(2);      ClrScr;
Langas(1);      ClrScr;
                {Įvedimo ir skaičių analizės ciklai}
repeat
    ReadLn(x);
    if x mod 5= 0 then Skusk(x);
until x = 0;
end.

```

7.3. Įrašo tipas

Kompiuteriais analizuojami duomenys atspindi realius objektus bei reiškinius, kurie aprašomi kiekybinių ir kokybinių charakteristikų rinkiniais. Kokybinės charakteristikos nurodomos pavadinimais, o kiekybinės – skaitinėmis reikšmėmis.

Pavyzdžiui, sandėlyje saugomą prekę apibūdina jos pavadinimas, kaina, kiekis ir kiekio matavimo vienetai. Programoje visiems šiems prekės aprašymo elementams galima skirti atskirus kintamuosius, tačiau patogiau juos saugoti kartu vienoje duomenų struktūroje. Masyvo tipas tokiam tikslui netinka, nes masyvuose galima saugoti tiksliai vienodo tipo duomenų rinkinius. Skirtingo tipo duomenų grupavimui vienoje duomenų struktūroje yra skirti įrašai. Įrašo tipo deklaravimo sintaksinė struktūra yra tokia:

```

<Vardas> = record
    <Laukų sąrašas>
end;

```

<Laukų sąrašas> sudaromas pagal tas pačias taisykles, kaip kintamųjų aprašymai. Pavyzdžiui, prekių aprašymui galima naudoti tokią struktūrą:

```

type preke = record
    vardas : string[15];
    matvien: string[2];
    kaina, kiekis : real

```

```
end;  
var Pr : preke;
```

Įvedus tokį tipą, duomenims apie prekę saugoti pakanka vieno kintamojo. Kreipiantis į įrašo laukus, vartojami sudėtiniai vardai, kurių sintaksė yra tokia:

<Įrašo vardas>.<Lauko vardas>

Laukų vardų pavyzdžiai:

Pr.vardas Pr.kaina Pr.kiekis

Jeigu to paties įrašo laukams taikoma keletas veiksmų, įrašo vardą papildytą jungtuku *with* galima iškelti prieš veiksmus aprašančius sakinius, kaip matematinėse išraiškose bendrus daugiklius prieš skliaustus. Jungtuko *with* galiojimo sritis nurodoma operatoriniais skliaustais *begin end*.

Junguko *with* taikymo pavyzdys:

```
with Pr do begin  
  ReadLn( vardas, kaina, kiekis);  
  WriteLn( vardas, kaina:8:2, kiekis:8:2 )  
end;
```

Nenaudojant junguko *with*, tektų rašyti sudėtinius vardus:

```
ReadLn( Pr.vardas, Pr.kaina, Pr.kiekis );  
WriteLn( Pr.vardas, Pr.kaina: 8:2, Pr.kiekis:8:2 );
```

Įrašo tipo kintamiesiems galima taikyti priskyrimo operaciją, jeigu abu operacijos argumentai yra to paties tipo. Iš įrašų taip pat galima sudaryti masyvus.

4 pratimas. Tekstiniame faile saugomas numatomų pirkti prekių sąrašas. Duomenims apie kiekvieną prekę skiriama atskira failo eilutė. Prekės vardas eilutėje užima 15 pirmųjų simbolių toliau yra du skaičiai: kaina ir kiekis. Sudaryti programą, kuri suskaičiuotų, kiek kainuos visos prekės ir parodytų ekrane visą sąrašą ir suskaičiuotą sumą.

Tekstiniai failai, kuriuose duomenys yra sugrupuoti pagal prasmę arba kitus požymius, vadinami struktūrizuotais. Šiame pratime pradinį duomenų failo struktūrizavimas reikalingas tam, kad būtų galima atskirti ir teisingai interpretuoti ten surašytus duomenis, kad būtų lengviau parengti tokį failą.

Jeigu iš tekstinio failo skaitomi duomenys perduodami į įrašus, reikia atsiminti, kad eilutės tipo kintamajam perduodamų simbolių skaičių, jeigu skaitomos eilutės ilgis yra pakankamai didelis, apsprendžia skaitymui skirtos eilutės tipo lauko ilgis. Tarpai eilučių tipo kintamiesiems yra tokie patys reikšminiai simboliai, kaip ir visi kiti, todėl jie tekstinių laukų pabaigos žymėti negali. Tuo tarpu, skaičiams tarpai yra skirtukai. Todėl, rašant įrašų reikšmes į failus, rekomenduojama pavadinimams skirtus simbolių laukus rašyti eilučių pradžioje ir skirti kiekvienam laukui griežtai apibrėžtą simbolių skaičių. Po to rašomos skaitinių laukų reikšmės, kurias galima atskirti bet kokiomis tarpų grupėmis, tačiau bus lengviau failą parengti ir išvengti klaidų, jeigu skaitines reikšmes rašysime stulpeliais, išlygindami dešiniąjį jų kraštą.

Pagal pratimo užduoties reikalavimus sudaryto struktūrizuoto pradinio duomenų failo pavyzdys:

Manija	1.05	102
Bananai	3.66	12.5
Mandarinai	5.5	3.4
Duona	2.9	0.5

```
type zodis = string[15];
preke= record vardas: zodis;
      kaina : real;
      kiekis: real;
end;
```

Pradinio duomenų failo skaitymui, skaičiavimams ir rezultatų parodymui rekomenduojama sudaryti atskiras procedūras. Programoje *Pratimas4* tokios procedūros atitinkamai yra *Skaito*, *Rasti* ir *Spausdina*.

Šioje programoje sudaryta universali išvedimo procedūra *Spausdina*, kurią galima panaudoti įvairiems tikslams: išvedimui į ekraną, rašymui į diską arba spausdinimui. Duomenų paskirties vieta keičiama kreipinyje į procedūrą nurodant sisteminį atitinkamo įrenginio arba failo vardą. Pavyzdžiui, siunčiant duomenis į personalinio kompiuterio ekraną, vartojamas sisteminis vardas *output*, o siunčiant spausdinimui, *prn*.

Derinant programą, galima apsiriboti duomenų parodymu ekrane. Ištaisius visas klaidas, paprastai programa papildoma įvairesnėmis rezultatų išsaugojimo ir atvaizdavimo priemonėmis.

Programos tekstas:

```
program Pratimas4;
const
    daug = 100;
                                { Duomenų failo vardas }
    duomenys = 'preke.dat';
type
    zodis = string[15];
    preke= record
        vardas: zodis;
        kaina : real;
        kiekis: real;
    end;
    sar = array [1..daug] of preke;
var
    F: text;
    P : sar;
    n : integer; suma : real;
{-----}
procedure Skaito( var failas : text;
                  var A: sar; var n : integer);
begin
    n := 0;
    while not EoF( failas ) and ( n < daug ) do
        begin
            n := n+1;
            with A[n] do
                ReadLn(failas, vardas, kaina, kiekis);
            end;
        end;
end;
{-----}
procedure Spausdina(var failas: text;
                   var A : sar; n : integer; s : real);
var    i : integer;
begin
    WriteLn(failas,
            '*****');
    WriteLn( failas,
            '*      Prekė      * Kaina * Kiekis *');
end;
```

```

WriteLn( failas,
        '*****');
for i := 1 to n do
  with A[i] do
    WriteLn( failas, '* ', vardas, ' * ',
             kaina:6:2, ' * ', kiekis:6:2, ' * ');
  WriteLn( failas,
          '*****');
  WriteLn( failas, ' Suminė kaina: ', s:6:2);
end;
{-----}
procedure Rasti( var A: sar; n : integer; var s: real);
var
  i : integer;
begin
  s := 0;
  for i := 1 to n do
    s := s + A[i].kaina * A[i].kiekis;
  end;
{-----}
begin
  { Duomenų įvedimas }
  Assign( F, duomenys); Reset( F );
  Skaito( F, P, n ); Close( F );
  { Suminės kainos skaičiavimas }
  Rasti( P, n, suma);
  { Rezultatų išvedimas į ekraną }
  Spausdina( Output, P, n, suma );
  ReadLn;
end.

```

- Papildykite programą taip, kad rezultatai būtų ne tik parodomi ekrane, bet ir surašomi į tekstinį rezultatų failą.
- Papildykite programą brangiausios prekės suradimo procedūra ir naujo rezultato išvedimo priemonėmis.

5 pratimas. Sudaryti programą, kuri užpildytą ekraną atsitiktiniais simboliais, ištrintų neraidinius simbolius, surastų ekrane visas pozicijas, kur rodoma vartotojo nurodyta raidė, ir pakeistų jos spalvas.

Sprendžiant šį uždavinį, ekrano duomenų analizei ir tvarkymui patogu naudoti įrašus ir dvimatį masyvą. Tai darant, reikia žinoti, kad kompiuterio atmintyje kiekvieną tekstinio ekrano elementą aprašo dviejų baitų grupė

(žodis). Pirmasis baitas saugo ekrane matomo simbolio kodą, o antrasis - jo spalvų aprašymą. Tokio duomenų rinkinio saugojimui galima vartoti struktūras:

```
type  simbolis = record
                        lo : char;
                        hi : byte
                    end;
Eilute = array[ 1..80 ] of simbolis;
Screan = array[ 1..25 ] of Eilute;
```

Struktūra *Screan* galės atstovauti ekranui tiktai tada, kai jai bus paskirta kompiuterio atmintyje ta pati vieta, kuri vartojama ekrano duomenims saugoti. Tai galima padaryti specialiu absoliutaus adreso paskyrimo sakiniu, kurio sintaksė:

<Struktūros vardas>: <Duomenų tipas> absolute <Absoliutus adresas>

Ekraninės atminties absoliutus adresas yra \$B800:0000.

```
program  Pratinimas5;
uses     Crt;
type
    simbolis  = record
                        lo   : char;
                        hi   : byte
                    end;
    Eilute     = array[ 1..80 ] of simbolis;
    Screan     = array[ 1..25 ] of Eilute;
    simboliai = set of char;
var
    { Ekraninės atminties adreso suteikimas masyvui }
    ekranas: Screan absolute $B800:0000;
    geri: simboliai;
    c: char;
{-----}
procedure Fill;      { Ekrano užpildymas simboliais }
var
    x, i, j: integer;
begin
    Randomize;
    for i:= 1 to 24 do begin
        for j:= 1 to 40 do begin
            { Atsitiktinis raidės kodas }
```

```

        x:= Ord('0') + Random(60);
        Write( Chr(x):2);
    end
end
end;
{-----}
procedure Stay (g: simboliai);
    { Simbolių, kurių nėra aibėje g trinimas }
var i, j: integer;
begin
    for i:= 1 to 25 do
        for j:= 1 to 80 do
            if not (ekranas[i, j].lo in g) then begin
                ekranas[i, j].lo:= '  '; Delay(20);
            end
        end
    end;
{-----}
procedure Mark( c: char; spalva: byte);
    { Raidžių paieška ir žymėjimas ekrane }
var i, j : integer;
begin
    for i := 1 to 25 do
        for j := 1 to 80 do
            if ekranas[i, j].lo = c then begin
                ekranas[i, j].hi:= spalva;
                Delay(500); Write(^G);
            end;
        end
    end;
{-----}
begin
    geri:= ['A'..'Z', 'a'..'z']; Fill;
    Stay(geri);
    repeat
        GoToXY(1, 25); ClrEol;
        Write(' Kokios raidės ieškoti? Pabaiga - "q": ');
        c:= ReadKey;
        Mark(c, Red+Blue*16 );
    until (c= 'q') or (c='Q') ;
end.

```

6 pratimas. Sudaryti programą, kuri iš ekrane matomo teksto “nuleistų” vartotojo nurodytas raides į ekrano apačią taip, kad “krentanti”

raidė ekrane paliktą spalvotą savo pėdsaką. ir ištrintų kelyje pasitaikančius simbolius.

Šio pratimo programą galima nesunkiai padaryti iš 5-o pratimo programos, pakeitus simbolių žymėjimo procedūrą *Mark* nauja procedūra *Krenta*. Ekranu duomenų modeliavimo priemonės gali būti tos pačios:

type

```
    simbolis = record
                                lo : char;
                                hi : byte
                                end;
```

```
    Eilute = array[ 1..80 ] of simbolis;
```

```
    Screan = array[ 1..25 ] of Eilute;
```

```
    simboliai = set of char;
```

var

```
    { Ekraninės atminties adreso suteikimas masyvui }
```

```
    ekranas: Screan absolute $B800:0000;
```

Papildomai siūloma įsivesti konstantas “krentančio” simbolio ir jo tako spalvoms bei kritimo greičiui aprašyti:

const

```
    Simbolis = Black; Takas = Red; Greitis = 100;
```

Rekomenduojami procedūros *Krenta* ir jos pagalbinės procedūros *Meta* tekstai:

```
procedure Meta( simb : char; x, y : byte );
```

```
    { Simbolio simb, kurio koordinatės (x, y), “kritimas” }
```

var

```
    i : integer;
```

begin

```
    for i:= y to 24 do begin
```

```
        { Simbolio trinimas }
```

```
        Ekranas [ i, x ].hi := Takas*16+Simbolis;
```

```
        Ekranas [ i, x ].lo := ‘ ‘;
```

```
        { Simbolio rašymas žemesnėje eilutėje }
```

```
        Ekranas [ i+1, x ].b := Takas*16+Simbolis;
```

```
        Ekranas [ i+1, x ].lo := simb;
```

```
        delay ( Greitis )
```

```
        { Pauzė }
```

```
    end
```

end;

```
procedure Krenta ( simb: char )  
    { Visų simbolių simb nuleidimas į ekrano apačią }  
var  
    x, y : byte;  
begin  
    for y := 1 to 24 do  
        for x := 1 to 80 do  
            if ekranas[x, y].lo = simb then  
                Meta ( simb, x, y );  
        end;  
    end;
```

Išsamus 6-o pratimo programos tekstas yra prie knygos pridedamame diskelyje.

7.4. Tipizuoti failai

Failais vadinami vienodos struktūros duomenų rinkiniai kompiuterio išoriniuose įrenginiuose. Programose failus atstovauja failų kintamieji, kurių tipai aprašomi struktūra:

type <Vardas> = *file of* <Elemento tipo vardas>

Taip aprašytuose failuose yra griežtai apibrėžtas struktūrinių elementų tipas, todėl jie vadinami tipizuotais.

Failo kintamasis skiriamas programos ryšiui su aptarnaujančios operacinės sistemos failais palaikyti. Kintamojo atstovaujamas operacinės sistemos failas nurodomas tokia procedūra:

```
procedure Assign( var <Failo kintamasis>,  
                  <Failo vardas>: string );
```

Failų kintamieji pagalbinėse procedūrose gali būti tik parametrai-kintamieji. Failo vardas nurodomas simbolių eilute, kurios reikšmė gali būti tiek paprastas, tiek jungtinis failo vardas, pavyzdžiui:

```
Assign( F, 'Rasa.dat' );  
Assign( F, 'C:\Petras\Darbai\Rasa.dat');
```

Programoms prieinami tik tie paprastu vardu nurodyti failai, kurie yra tame pačiame kataloge su programa arba yra operacinės sistemos paieškos keliuose aprašytuose kataloguose. Failams taikomos parengimo, paieškos, skaitymo/rašymo, uždarymo ir kitos tvarkymo operacijos. Parengimo operacijomis, kurios aprašytos lentelėje 7.1, failas paruošiamas skaitymui, rašymui arba papildymui. Baigus darbą su failais, reikia juos uždaryti procedūra *Close*:

procedure *Close*(*var* <Failo kintamasis>);

7.1 lentelė. Pagrindinės failų parengimo ir analizės priemonės

Prototipas	Paskirtis
procedure <i>Rewrite</i> (<i>var</i> <Failo kintamasis>);	Failo ištrinimas ir parengimas rašymui.
procedure <i>Reset</i> (<i>var</i> <Failo kintamasis>);	Failo parengimas skaitymui arba papildymui gale.
procedure <i>Seek</i> (<i>var</i> <Failo kintamasis>, <i>var</i> <k>: <i>longint</i>);	Failo rodyklės perkėlimas į poziciją <i>k</i> .
procedure <i>Truncate</i> (<i>var</i> <Failo kintamasis>);	Failo elementų nuo rodyklės iki galo trinimas.
function <i>EoF</i> (<Failo kintamasis>): <i>boolean</i> ;	Loginė failo pabaigos kontrolės funkcija.
function <i>FilePos</i> (<i>var</i> <Failo kintamasis>): <i>longint</i> ;	Grąžina failo rodyklės pozicijos numerį.
function <i>FileSize</i> (<i>var</i> <Failinis kintamasis>): <i>longint</i> ;	Grąžina failo dydį.

Tipizuotų failų skaitymas ir rašymas skiriasi nuo tekstiniams failams taikomų analogiškų operacijų tuo, kad čia galima rašyti ir skaityti tik failo tipu duomenis. Be to, leidžiamos tik šios skaitymo/rašymo operacijos:

procedure *Write* (*var*<Failo kintamasis>, <Reikšmė>);

procedure *Read*(*var*<Failo kintamasis>; <Kintamasis>);

Aprašant failų tvarkymo veiksmus, patogiau vartoti failo rodyklės sąvoką. Tai yra sąlyginis tvarkymui parengto failo elemento numerio pavadinimas. Naujai atidaryto failo rodyklė rodo jo pirmąjį elementą. Po kiekvienos rašymo/skaitymo operacijos rodyklė pastumiama per tiek elementų, kiek buvo įrašyta arba perskaityta reikšmių.

Papildant failą nauju elementu gale, failo rodyklė perkeliama į galą (už paskutinio elemento) procedūra *Seek(F, FileSize(F))*, kur **F** – failo kintamojo vardas. Apie failo dydį informuoja funkcija *FileSize(F)*. Failų elementai numeruojami pradedant nulių, todėl rodyklė pozicijoje *FileSize(F)* rodo į laisvą vietą už paskutinio failo elemento. Nustačius rodyklę, failas papildomas nauju elementu procedūros *Write* pagalba.

Dažniausiai vartojami tipizuoti failai su įrašo tipo elementais. Į tokius failus galima rašyti ir iš jų skaityti sudėtingos loginės struktūros duomenų grupes. Aprašant uždavinius jų poreikiams pritaikytomis struktūromis, tampa aiškesnė programų logika. Tokias programas lengviau prižiūrėti ir pertvarkyti. Tačiau visa tai susiję su papildomomis darbo sąnaudomis, nes, vartojant specialios struktūros failus, tenka projektuoti ir visas jų tvarkymo operacijas.

7 pratimas. Tarkime, kad reikia suprojektuoti moksleivių pažangumo analizės informacinę sistemą. Sprendžiant pažangumo analizės uždavinius, patogu vartoti įrašus, kuriose būtų tokie elementai: pavardė, klasė ir egzaminų pažymiai – atkarpos [1 .. 10] sveikieji skaičiai.

Tokių struktūrų projekto pavyzdys:

const **N** = 10; **L** = 500;

type

zodis = *string* [15];

{ Nulinio indekso elementas aprašo faktinį masyvo užpildymą }

pazymiai = *array* [0 .. N] *of byte*;

mok = *record*

pav, k : zodis;

p : pazymiai;

end;

failas = *file of* mok;

Turint bent apytikriai suprojektuotas duomenų struktūras, reikia numatyti pagrindinius jų apdorojimo principus ir veiksmus, kurie tikėtų pageidaujamos informacinės paslaugoms aprašyti. Pavyzdžiui, visus informacinius uždavinius galima spręsti, skaitant po vieną ir analizuojant atskirus duomenų failo įrašus arba perrašant duomenis į masyvą ir analizuojant masyvą. Jeigu failai nėra labai dideli, pastarasis būdas yra gerokai efektyvesnis. Taip pat reikia numatyti, kaip bus parengiami įrašų failo duomenys, tvarkomų duomenų kontrolės priemonės. Įrašų failo

parengimui iš klaviatūra įvedamų duomenų galima sudaryti specialią procedūrą, tačiau galima pasinaudoti ir bendros paskirties redaktoriais parengiamais tekstiniais failais.

Dažniausiai informacinių paslaugų programos kuriamos palaipsniui. Iš pradžių sukuriamas minimalias paslaugas teikiantis tokios sistemos branduolys, o po to jis papildomas. Siūloma sudaryti tokias 7-o pratimo užduoties sprendimo priemones:

- dialoginę įrašų failo su duomenimis apie moksleivių pažangumą formavimo procedūrą;
- failo duomenų perdavimo į pagalbinį įrašų masyvą ir duomenų kontrolės ekrane procedūrą;
- vartotojo pasirinkimo meniu formavimo procedūrą;

Kadangi pagalbinis įrašų masyvas gali būti naudojamas daugelyje kitų čia neaprašomų procedūrų, šį masyvą naudinga paskelbti globaliu. Globalus yra ir faktinį masyvo užpildymą aprašantis kintamasis **Kiek**. Taip sumažintas procedūrų ryšius aprašančių parametrų skaičius.

Moksleivių laikomų egzaminų skaičius gali būti skirtingas. Jis nustatomas kiekvienam moksleiviui įvedimo metu, nurodant pažymių sekos pabaigą neigiamu skaičiumi. Faktinis pažymių skaičius saugomas pažymių masyvo elemente su nuliniu indeksu.

Pradinių duomenų rinkinio pavyzdys:

Jonaitis 11a 8 9 4 5 6
Gerulaitis 11b 9 9 9 9 9
Slunkaitis 10a 4 5 5 7 0 0
Razumas 11b 10 10 10 10 10

Procedūroje *Reset* nurodžius neegzistuojantį failą, operacinė sistema nutraukia programos darbą. Šito galima išvengti kompiliatoriaus direktyva **{SI-}** išjungus įvedimo/išvedimo operacijų kontrolę. Ryšio su išore operacijos klaidą grąžina Turbo Paskalio sisteminės bibliotekos kintamasis **IOResult**. Jo reikšmė lygi nuliui (0), kai skaitymo/rašymo veiksmas baigiasi sėkmingai, ir lygi klaidos kodui, kai veiksmas nepavyko. Šios priemonės pratimo programoje naudojamos sudarant universalią įrašų failo papildymo ir formavimo procedūrą *Rasyk*. Jei duomenų failo aktyviame kataloge nėra, ši procedūra kuria naują tokį failą. Jei failas yra, jis parengiamas papildymui gale.

```

Program Pavyzdys7;
uses Crt;
const
  N = 10;           { Dižiausias pažymių skaičius }
  L = 500;          { Didžiausias moksleivių skaičius }
  fvardas = 'duom.dat';
  a: char = 'A';
type
  zodis  = string [15];
  pazymiai = array [0 .. N] of integer;
  mok = record      { Įrašas apie moksleivį }
    pav, k : zodis;
    p : pazymiai;
  end;
  failas = file of mok; { Duomenų failas }
var
  Mas: array[1..L] of mok; { Pagalbinis masyvas }
  Kiek: word;             { Faktinis moksleivių skaičius }
  F: failas;

procedure Rasyk (var F : failas);
var i: byte;
begin
  Assign(F, fvardas);
  {SI-}           { Sisteminės kontrolės išjungimas }
  if IOResult<>0 then Rewrite(F)
  else
  begin Reset(F); Seek(F, FileSize(F)) end;
  {SI+}           { Sisteminės kontrolės atstatymas }
  Kiek:= 0;
  WriteLn (' ': 10, ' Moksleivių sąrašas'); Writeln;
  Write      (Kiek+1:3, ' '          Pavarde:      ');
  Readln(Mas[Kiek+1].pav);
  while Ucase(Mas[Kiek+1].pav[1])<> 'Q' do begin
    Kiek:= Kiek +1;
    with Mas[Kiek] do begin
      Write ('Klase: '); Readln(k);
      i:= 0;
      repeat
        i:=i+1;
        Write (i:3, ' Pazymis: '); Readln(p[i]);
      until p[i]<0;
    end;
  end;
end;

```

```

        p[0]:= i-1;
    end;
    Write(F, Mas[Kiek]);
    Write (Kiek+1:3, ' Pavarde: ');
    Readln(Mas[Kiek+1].pav);
end;
Close(F);
end;

procedure Rodyk (var F: failas);
var i: integer;
begin
    Assign(F, fvardas);
    {SI-}          { Sisteminės kontrolės išjungimas }
    if IOResult<>0 then begin
        WriteLn (' Klaida, failas nerastas'); Exit
    end;
    {SI+}          { Sisteminės kontrolės atstatymas }
    Reset(F);
    Kiek:= 0;
    WriteLn (' ': 10, ' Moksleivių sąrašas'); Writeln;
    WriteLn ('Nr.': 5, 'Klasė':10, 'Pavardė': 15,
                                                    'Pazymiai':10);
    while not Eof (F) do begin
        Kiek:= Kiek+1;
        Read(F, Mas[Kiek]);
        with Mas[Kiek] do begin
            Write ( Kiek:3, k:12, pav:15);
            for i:= 1 to p[0] do Write (p[i]: 3);
        end;
        Writeln;
    end;
    WriteLn ( ' _____ ');
    Close (F);
end;

procedure Meniu (var a: char);
begin
    Textbackground (blue); Clrscr;
    GotoXY(10,25);
    Writeln ('Moksleivių sąrašo tvarkymas'); Writeln;
    Writeln ('Failo parengimas - R': 45);
    Writeln ('Failo peržiūra - P': 45);

```

```

Writeln ('Pabaiga - Q': 45);
a:= Uppcase (ReadKey);
Textbackground (black); Clrscr;
end;
(***** Pagrindinė dalis *****)
begin
  repeat
    Menu (a);
    case a of
      'R':   Rasyk (F);
      'P':   begin Rodyk (F); Readln end;
    end;
  until a= 'Q';
end.

```

- Papildykite 7 pratimo programą nauja procedūra taip, kad ji galėtų parodyti moksleivių pažangumo vidurkius ir išvalyti duomenų failą. Failo išvalymui siūloma vartoti procedūrą *Rewrite*.
- Pertvarkykite 7 pratimo programą taip, kad valdymo meniu būtų viename ekrano lange, o programos darbo rezultatai – kitame.

7.5. Užduotys

1 užduotis. Sudaryti tekstiniam failui surašytų skaičių analizės programą, kuri skaitomus iš failo skaičius paskirstytų į keturis ekrano langus pagal tokius požymius: lyginiai, nelyginiai, teigiami, neigiami. Jeigu skaičius atitinka kelis požymius, jis turi būti rašomas visuose tiems požymiams skirtuose languose.

2 užduotis. Sudaryti mokinių matematikos dalyko per trimestrą gautų pažymių analizės programą. Kiekvienas mokinys gauna keturis pažymius: už namų darbą, kontrolinį darbą, uždavinio sprendimo programą kompiuteriui ir už savarankiškai įsisavintos pamokos pristatymą (pranešimą klasėje).

Programa turi būti valdoma meniu pagalba, kuris leistų vartotojui pasirinkti tokias procedūras:

- įrašų failo užpildymo,
- įrašų failo peržiūros ekrane,

- vidurkių peržiūros ekrane,
- filtravimo pagal vidurkį, kai ekrane parodomi tik tie įrašai, kurių pažymių vidurkiai yra didesni už vartotojo nurodytą.

3 užduotis. Pertvarkykite 2 užduoties programą taip, kad ji galėtų skaityti duomenis iš struktūrizuoto tekstinio failo.

4 užduotis. Sudaryti tekstinio ekrano atminties analizės programą, kuri vartotojo nurodytas raides ‘nustumtų’ į kairįjį ekrano kraštą netrindama kelyje pasitaikančių simbolių (‘prastūmus’ tvarkomą raidę, raidės kelyje esantys simboliai turi būti atstatomi).

7. PAGALBINĖS BIBLIOTEKOS (MODULIAI)

Kiekvienas bent kiek didesnę patyrimą turintis programuotojas turi parengęs dažniausiai jo vartojamų naujų programų projektavimui struktūrų rinkinius. Tokie rinkiniai vadinami pagalbinėmis bibliotekomis arba moduliais. Bibliotekos dažniausiai specializuojamos giminingos paskirties uždavinių klasėms arba duomenų struktūroms. Jose saugomi konstantų, duomenų tipų, kintamųjų ir įvairių procedūrų bei funkcijų aprašai.

Pagalbinės priemonės turi būti parengtos taip, kad jas būtų galima įterpti į naujai rengiamas programas su minimaliomis darbo sąnaudomis. Siekiant išspręsti šią problemą, kompiliatorių darbas skaidomas į du etapus. Pirmajame etape iš pirminio programos teksto parengiamas tarpinis ruošinys, kuris būna pritaikytas papildymui naujais elementais arba komponavimui į kitas programas. Jis vadinamas objektiniu kodu. Antrajame kompiliavimo žingsnyje, kurį valdo sisteminių ryšių redaktoriumi vadinama programa, parengiamas programos mašininis kodas. Kompiuterio atmintyje patogiausia saugoti pagalbinių bibliotekų objektinius kodus. Objektinių kodų saugojimui skirta struktūra Turbo Paskalio aplinkoje vadinama moduliu (*unit*). Moduliais taip pat vadinamos ir pagalbinių bibliotekų pirminių tekstų sudarymui skirtos Turbo Paskalio kalbos struktūros.

Naudojami dviejų tipų moduliai: vartotojo ir vidiniai. Vartotojo modulius programuotojai ruošia patys, savo asmeniniams poreikiams. Tuo tarpu, vidiniuose moduliuose yra Paskalio kalbos praplėtimai, kuriuos firma Borland siūlo visiems programuotojams. Čia taip pat pateikiamos specifinės kompiuterio įrenginių valdymo priemonės, pritaikytos konkrečiai operacinei sistemai arba konkrečiam kompiuterio įrenginių tipui. Pagrindinių Turbo Paskalio versijos 7.0 vidinių bibliotekų paskirtys aprašytos 7.1 lentelėje. Norint naudotis pagalbinių bibliotekų paslaugomis, jas reikia iš pradžių parengti darbui, aktyvizuoti. Tai daroma sakiniu *uses*, kuris programoje turi būti pirmas, rašomas iš karto už antraštės. Sakinio *uses* sintaksė:

uses <Aktyvizuojamų bibliotekų sąrašas>

Pagalbinių bibliotekų failai vartotojams pateikiami kartu su programavimo aplinka. Jie saugomi atskirame pagalbinių modulių kataloge, kurio vieta nurodoma integruotos aplinkos komandos *Options.Directories* užklausoje lauke *EXE&TPU directory*. Sisteminė biblioteka *System* visuomet yra parengta darbui ir programose jos aktyvizuoti nereikia.

7.1 lentelė. Turbo Paskalio versijos 7.0 bibliotekos

Biblioteka	Paskirtis
<i>Crt</i>	Tekstinio ekrano tvarkymo biblioteka, kurioje yra specialios ekrano vaizdo ir klaviatūros valdymo priemonės.
<i>Dos</i>	Ryšio su operacine sistema priemonės.
<i>Graph</i>	Grafinio ekrano valdymo biblioteka su turtingu grafinių primityvų rinkiniu.
<i>Graph3</i>	Grafinio ekrano valdymo biblioteka su “vėžlinės grafikos” priemonėmis.
<i>Overlay</i>	Didelių programų segmentavimo priemonės.
<i>Printer</i>	Palengvinto kreipimosi į sisteminį spausdintuvą priemonės.
<i>System</i>	Paskalio kalbos standartinės procedūros bei funkcijos, jų papildymai Turbo Paskalyje, dalis ryšio su OS priemonių.
<i>String</i>	Nulinio kodo simboliu užbaigiamų ASCIIZ eilučių tvarkymo biblioteka.
<i>Turbo3</i>	Ryšio su Turbo Paskalio versija 3.0 priemonės.
<i>Turbo</i>	Daugialangio tekstinio vartotojo interfeiso projektavimui skirta objektinė biblioteka.
<i>Vision</i>	Programavimo Windows aplinkoje priemonės.
<i>WinAPI</i>	

Pagrindinės bibliotekos, su kuriomis rekomenduojama susipažinti pradedančiam programuotojui, yra *System*, *Dos*, *Crt* ir *Graph*. Bibliotekos *Graph3* ir *Turbo3* yra reikalingos tik tai, kai siekiama naujai rašomose programose panaudoti senai Turbo Paskalio versijai 3.0 sukurtas procedūras arba programų fragmentus. Nebevartojama ir 1990 m. sukurta biblioteka *Turbo Vision*, kuri siūlo ištisą programavimo technologiją. To priežastis yra jos orientacija į tekstinį ekraną. Dabar daugumoje personalinių kompiuterių vartojami grafiniai interfeisai, todėl programuotojams svarbiau įvaldyti programavimui tokiose aplinkose skirtas priemones. Šios priemonės pateikiamos galingoje bibliotekoje *WinAPI*, kurią sudaro ištisa siauriau specializuotų modulių sistema, skirta Turbo Paskalio 7.0 versijai Borlan Pascal. Biblioteka *WinAPI* čia nenagrinėjama, nes tai atskiros knygos reikalaujanti tema.

Jeigu kompiuterį aptarnauja OS Windows, norint naudotis MS DOS skirtomis bibliotekomis (*Crt*, *Dos*, *Graph*), reikia, kad Turbo Paskalio

aplinka būtų paleista iš šiai OS skirto ekrano lango. Taip pat reikia pabrėžti, kad šių bibliotekų teikiamų paslaugų analizė padeda pasiruošti programavimui Windows aplinkoje, nes WinAPI bibliotekoje yra daugumos MS DOS skirtų bibliotekų analogai.

7.1. Diskų analizės priemonės

Šios priemonės naudojamos tada, kai pageidaujama patikrinti, ar yra diske ieškomi failai, kiek ten laisvos vietos duomenų saugojimui ir sprendžiant kitus panašaus pobūdžio uždavinius. Jos saugomos modulyje *Dos*. Labai panašus į modulį *Dos* yra bibliotekos *WinAPI* modulis *WinDos*, kuris skirtas palaikyti ryšiams su MS DOS pagal aplinkos Windows standartą, tačiau dažnai vartojamas ir savarankiškai.

Dažniausiai vartojamos analizės procedūros, kurios leidžia gauti išsamius duomenis apie kompiuterio diskus ir juose saugomus failus, atlikti failų paiešką. Užklausoms apie diske saugomų failų savybes skirta procedūra:

```
procedure FindFirst (<Kelias>: string; <Savybės>: word;  
var <Rezultatas>: SearchRec);
```

Procedūros parametras *Kelias* nurodo paieškos kelią ir ieškomų vardų šabloną, kuriame galima vartoti metasimbolius * ir ?. Parametro *Kelias* nurodymo pavyzdys:

```
‘\*.?’.
```

Parametras *Savybės* leidžia aprašyti OS vartojamais kodais ieškomų failų savybes. Kodus oatogu nurodyti modulio *Dos* konstantomis:

```
const    Readonly = $01;          { Tik skaitomi failai}  
          Directory = $10;        { Katalogai }  
          Hidden = $02;           { Paslėpti failai}  
          Archive = $20;          { Dar nearchyvuoti failai}  
          Sysfiles = $04;         { Sisteminiai failai}  
          Anyfile = $3F;          { Visi failai }  
          VolumeID = $08;         { Duomenų tomo žymė }
```

Aprašant savybių grupes, konstantos sumuojamos, pavyzdžiui, *Hidden+Sysfiles*. Duomenys apie paieškos rezultatus pateikiami modulyje *Dos* apibrėžto tipo *SearchRec* kintamojo *Rezultatai* laukuose. Įrašo *SearchRec* struktūra:

```
type  
SearchRec= record
```

```

file: array [1..21] of byte;    { Specialūs OS duomenys }
attr: byte;                     { Požymių kodų suma }
time: longint;                  { Sukūrimo data ir laikas }
size: longint;                  { Failo dydis baitais }
name: string[12];               { Failo vardas su išplėtimu }
end;

```

Kartojant paiešką su procedūrai *FindFirst* perduotais parametrais, vartojama procedūra *FindNext*:

```

procedure FindNext ( var <Rezultatai>: Searchrec );

```

Ar paieška baigėsi sėkmingai, galima patikrinti *integer* tipo kintamojo *DosError* pagalba. Šio kintamojo reikšmės informuoja apie modulio *Dos* procedūrų sėkmę ir nurodo klaidų kodus. Sėkmingos paieškos požymis yra *DosError* reikšmė 0.

Failų paieškos procedūrų panaudojimo principus iliustruoja programa *Pratimas_7_1*.

7.1.1 pratimas. Sudaryti programą, kuri parodo ekrane aktyvaus diskinės atminties katalogo turinį.

Programos tekstas:

```

program Pratimas_7_1; { Aktyvaus katalogo turinio parodymas }
uses Dos;
var   info: SearchRec; { Informacijai apie failą skirtas įrašas }
      i: integer;      { Pagalbinis kintamasis }
      e: string;       { Įrašas keliui ir vardų šablonui }
begin
  e:= '\*. *';         { Pagrindinis aktyvaus disko katalogas }
  FindFirst (e, anyfile, info); { Pirmo failo paieška }
  WriteLn ('Directory: ', 30, e);
  i:= 0;                { Surasto failo numeris }
  while Doserror=0 do begin { Failų paieškos ciklai }
    i:= i+1;
    if i=24 then begin
      Write ('Kita lentelės dalis - paspaudus klavišą Enter');
      Readln;          { Pauzė }
      i:= 0;           { Skaitiklio gesinimas }
    end;
    WriteLn (info.name:12, info.attr:3, info.size:8);
  end;

```

```

        Findnext (info);                { Kito failo paieška }
    end
end.

```

Norint sužinoti failo saugojimo vietą, galima vartoti funkciją *FExpand*, kuri failo vardą išplečia iki jungtinio vardo:

```
function FExpand (<Vardas>: PathStr): PathStr;
```

Primename jungtinio failo vardo sintaksę:

```
<Kelias į failo katalogą>\<Vardas>.<Vardo plėtinys>
```

Jungtinį failo vardą į atskiras dedamąsias skaido procedūra:

```

procedure FSplit (<Jungtinis failo vardas>: PathStr;
    var <Kelias>: DirStr; var <Vardas>: NameStr;
    var <Plėtinys>: ExtStr);

```

{ Procedūra parametro nurodomą jungtinį failo vardą suskaido į dedamąsias, kurias grąžina per parametrus *Kelias*, *Vardas* ir *Plėtinys*. }

Failų savybes galima sužinoti ir pakeisti tokiomis procedūromis:

```

procedure SetFAttr (var<Failo kintamasis>; <Savybės>: word);
{ Failui suteikiamos word tipo parametro nurodomos savybės, kurios
koduojamos taip pat, kaip procedūroje FindNext. Failas negali būti
atidarytas.}

```

```

procedure GetFAttr ( var <Failo kintamasis>;
                    var <Savybės>: word);
{ Failo savybių kodai perduodami word tipo parametrai Savybės. Failas
negali būti atidarytas.}

```

Failo paieška apibrėžtame katalogų rinkinyje aprašoma funkcija *FSearch*, kuri grąžina jungtinį failo vardą arba, nesėkmės atveju, tuščią eilutę:

```

function FSearch (<Failo vardas>: PathStr;
    <Katalogų sąrašas>: string): PathStr;

```

{ Katalogų sąrašo elementai atskiriami kableliais. }

Failų tvarkymo procedūrose ir funkcijose įvairių tipų vardai nurodomi modulyje *Dos* apibrėžtomis eilučių modifikacijomis:

```

type
    PathStr= string[79];    { Jungtiniams failų vardams.}
    DirStr= string[67];    { Katalogų vardams.}
    NameStr= string[8];    { Paprastiems failų vardams.}

```

ExtStr= string[4]; { Failų vardų išplėtimams. }

Informaciją apie diskų talpą ir užpildymą galima gauti funkcijomis:

```
function DiskFree (diskas: word): longint;  
{ Laisvų baitų skaičius diske }
```

```
function DiskSize (diskas: word): longint;  
{ Disko informacinė talpa baitais }
```

Diskai modulio *Dos* funkcijose yra nurodomi skaitmeniniais kodais: 0 – aktyvus diskas, 1– A, 2 – B, 3 – C ir t. t.

7.1.2 pratimas. Tarkime, kad norime sudaryti programą, kuri galėtų pati pranešti apie savo savo saugojimo vietą. Tokią informaciją galima gauti išskleidžiant jungtinį programos vardą, kurį galima sužinoti funkcijos *ParamStr* pagalba. Funkcijos prototipas:

```
function ParamStr (<Indeksas>: word): string;
```

Nulinė indekso reikšmė kreipinyje į funkciją, nurodo programos vardą, o kitos indekso reikšmės - programai perduodamus parametrus. Programos tekstas:

```
program Pratimas_7_2;                    { Informacija apie programą }  
  uses Dos;  
  var     e: string; k: DirStr;  
         v: NameStr; t: ExtStr;  
  
  begin  
    e:= ParamStr(0);                    { Jungtinis programos vardas }  
    Fsplit(e, k, v, t);                { Vardo skaidymas }  
    WriteLn('Darbo katalogas yra', k);  
    WriteLn('Programos failas: ', v, ' , o jo tipas: ', t);  
    ReadLn  
  end.
```

7.2. Kompiuterio laikrodžio panaudojimas

Vartotojams naudingos ryšio su sisteminiu laikrodžiu funkcijos. Toks laikrodis, kuris maitinamas iš autonominio energijos šaltinio, yra kiekviename kompiuteryje. Jis naudojamas tiek sisteminiams tikslams, tiek suteikiant papildomas paslaugas vartotojams. Operacinė sistema naudoja laikrodį vartotojų darbo apskaitai, jos vykdomų procesų trukmės ribojimui,

sinchronizavimui ir kitiems panašiams tikslams. Vartotojams gali būti naudinga įrašyti dokumente jo spausdinimo datą, laiką, sužinoti konkretaus mėnesio darbo dienų skaičių, sužinoti, kokią savaitės dieną bus juos dominanti mėnesio diena. Sisteminio kompiuterio laikrodžio apklausai skirtos tokios procedūros:

```
procedure GetDate (var <Metai>, <Mėnuo>,  
                    <Mėnesio diena>, <Savaitės diena>: word);
```

```
procedure GetTime (var <Valandos>, <Minutės>,  
                    <Sekundės>, <Šimtosios sekundės dalys>: word);
```

Parametrų aprašai procedūrų prototipuose išsamiai nurodo jų prasmę. Sisteminio laikrodžio parodymai keičiami procedūromis:

```
procedure SetDate (<Metai>, <Mėnuo>,  
                    <Mėnesio diena>: word);
```

```
procedure SetTime (<Valandos>, <Minutės>, <Sekundės>,  
                    <Šimtosios sekundės dalys>: word);
```

Leistinos parametrų reikšmių atkarpos:

Metai – [1980..2099], Mėnuo – [1..12], Mėnesio diena – [1..31], Savaitės diena – [0..6], Valandos – [0..23], Minutės – [0..59], Sekundės – [0..59], Šimtosios sekundės dalys – [0..99],

Parametro *Savaitės diena* reikšmė 0 atitinka sekmadienį. Norint sužinoti, kokią savaitės dieną atitinka konkreti data, reikia atlikti tokius veiksmus:

- Išsaugoti laikrodžio duomenis apie esamą datą (*GetDate*);
- Pakeisti laikrodžio rodomą datą (*SetDate*);
- Perskaityti laikrodžio parodymą (*GetDate*);
 - Atstatyti buvusią laikrodžio datą (*SetDate*).

Kompiuterio viduje data ir laikas koduojami vienu *longint* tipo skaičiumi. Tokį kodą taip pat grąžina procedūros *FindFirst* ir *FindNext*. Todėl aktuali yra laiko dekodavimo problema. Šiam tikslui skirta procedūra:

```
procedure UnpackTime (<Laiko kodas>: longint;  
                      var <Data ir laikas>: DateTime);
```

Atvirkščias veiksmas valdomas procedūra:

```
procedure PackTime (var <Data ir laikas>: DateTime;  
                    var <Laiko kodas>: longint);
```

Laiko kodavimo ir dekodavimo procedūrose data ir laikas perduodami modulio *Dos* specialaus įrašo tipo *DateTime* laukuose:

```
type DateTime = record
    year, month, day, hour, min, sec: word;
end;
```

Atidaryto failo sukūrimo laiką tikrina ir keičia procedūros:

```
procedure GetFTime (var <Failo kintamasis>;
                    var <Laiko kodas>: longint);
    { Grąžina laiko kodą.}

procedure SetFTime (var <Failo kintamasis>;
                    <Laiko kodas>: longint);
    { Pakeičia laiko kodą.}
```

Failų sukūrimo laiko kontrolės ir laiko kodo išskleidimo techniką iliustruoja funkcija *Naujas*. Jos reikšmė *True* praneša, kad tikrinamas failas buvo sukurtas tą patį mėnesį, kurį vyksta tikrinimas.

```
function Naujas (var f: text): boolean;    { Vartoja modulį Dos }
var
    metai, menuo, diena, savd: word;
    laikas : DateTime;  t : longint;
begin
    GetDate (metai, menuo, diena, savd);
    GetFTime (f, t);
    UnpackTime (t, laikas);
    if menuo = laikas.month then Naujas:= True
    else Naujas := False;
end;
```

7.2.1 pratimas. Tarkime, kad reikia sudaryti mėnesio kalendoriui spausdinti skirtą procedūrą. Tai padaryti nesunku, jeigu žinome, kokią savaitės dieną prasideda mėnuo. Šią dieną praneš procedūra *GetDate*, jeigu į sisteminį laikrodį įrašysime mėnesio pradžios datą.

Papildomai dar reikės sudaryti funkciją, kuri leistų sužinoti dienų skaičių mėnesyje. Kaip tai daroma, parodyta programoje *Pratimas_7_3*.

```
program Pratimas_7_3;
uses Dos;
```

```

const          { Savaitės dienų žymės ir mėnesių vardai }
dienos: array [1..7] of char = 'patkpšs';
              { Masyvas mėnesio dienų grupavimui }
men: array [1..12] of string = ('sausis', 'vasaris', 'kovas', 'balandis',
'gegužis', 'birželis', 'liepa', 'rugpjūtis', 'rugsėjis', 'spalis',
'lapkritis', 'gruodis');
type mas= array [1..7, 1..6] of integer;
var
  md: mas;
  dsk, metai, mn, i, j: word;

  { Dienų skaičiaus mėnesyje skaičiavimo funkcija }
function Mend (metai, menuo: word): integer;
begin
  case menuo of
    4, 6, 9, 11 :      Mend := 30;
    1, 3, 5, 7, 8, 10, 12 :  Mend := 31;
    2 : if (metai mod 400 = 0) or (metai mod 100 <> 0) and
        (metai mod 4 = 0)
        then Mend:= 29
        else Mend:= 28;

  end;
end;

{ Mėnesio dienų paskirstymas savaitėms ir savaitėių dienoms }
procedure Masyvas (var md: mas; metai, mn: word);
var
  k, i, j: integer;
  dsk, sm, smen, sd, ssavd, savd, d: word;
begin
              { Sena laikrodžio reikšmė }
  GetDate (sm, smen, sd, ssavd);
  SetDate (metai, mn, 1);      { Mėnesio 1-os d. savaitės diena }
  GetDate (metai, mn, d, savd);
  if savd=0 then  savd:= 7;    { Sekmadienio perkodavimas }
  SetDate (sm, smen, sd);     { Datos atstatymas }
  for i:= 1 to 7 do           { Masyvo valymas }
    for j := 1 to 6 do
      md[i,j] := 0;
    dsk:= Mend(metai, mn);
  i:= savd; j:= 1;           { Pirmos dienos vieta }

```

```

for k:= 1 to dsk do begin           { Masyvo užpildymas }
    md[i,j]:= k;
    if i = 7 then
        begin i := 1; j := j+1 end
    else i := i+1;
    end;
end;

{ PAGRINDINĖ PROGRAMOS DALIS }
begin
    WriteLn('Kalendoriaus sudarymas mėnesiui');
    WriteLn('Nurodykite metus (1980-2099)'); ReadLn(metai);
    WriteLn('Nurodykite mėnesį (1-12)');   ReadLn(mn);
    WriteLn (metai, ' metų ', mn[mn] : 10);
    WriteLn ('    kalendorius');
    Masyvas(md, metai, mn);
    for i := 1 to 7 do begin           { Kalendoriaus išvedimas }
        Write (dienos[i]);
        for j := 1 to 6 do
            if md[i,j] = 0 then Write (' ' : 3)
            else Write (md[i,j] : 3);
        WriteLn;
    end;
end.

```

7.3. Procesų valdymas

Procesu vadinama aktyvi programos būseną – jos vykdymas OS aplinkoje. OS procesai valdomi įvairiais būdais: kompiuterio klavišais (*Ctrl+Break*), taikomojoje programoje įrašytomis instrukcijomis (*Halt*) ir kompiuterio įrenginių formuojamais signalais, kuriuos tikrina operacinė sistema. Standartinėse aukšto lygio kalbose OS procesų valdymui skiriamos labai kuklios priemonės, nes jos priklauso tiek nuo kompiuterį aptarnaujančios operacinės sistemos, tiek nuo kompiuterio architektūros savybių. Todėl šios priemonės paprastai pateikiamos atskirose bibliotekose ir yra aprašomos kalbų realizacijų techninėje dokumentacijoje. Pradedantiems programuotojams ir net daugumai profesionalų pakanka mokėti naudotis keliomis paprastomis plačiau vartojamomis OS darbo kontrolės procedūromis, kurios yra *Dos* bibliotekoje. Šių procedūrų prototipai:

```

procedure GetCBreak ( var <OS reakcija>: boolean);
{ Gražina OS reakciją į klavišus Ctrl+Break aprašančią reikšmę.}

```

```

procedure SetCBreak ( <OS reakcija>: boolean);
{ Nurodo OS reakcijos į klavišus Ctrl+Break būdą.}

```

```
procedure GetVerify ( var <Diskų kontrolė>: boolean);
{ Pranešama, ar įjulta diskinių operacijų kontrolė. }
```

```

procedure SetVerify ( var <Diskų kontrolė>: boolean);
{ Nurodo, ar pageidaujama įjungti diskinių operacijų kontrolę.}

```

Standartinė MS DOS reakcija į klavišų porą *Ctrl+Break* yra programos darbo nutraukimas visų programos kreipinių į OS metu. Jei procedūrai *SetCBreak* perduodama parametro reikšmė *False*, taikomąją programą bus galima nutraukti tik tada, kai ji kreipiasi į konsolę, spausdintuvą ar ryšio jungtis.

Projektuojant programų sistemas, reikalingos procedūros, kurias leidžia paleisti kompiuterio išorinėje atmintyje saugomas programos ir patikrinti, kaip tai pavyko. Prieš kreipiantis į kitą programą, turi būti vykdoma parametrų neturinti procedūra *SwapVectors*, kuri priverčia programą įsiminti OS būvį (sukeisti pertraukimo vektorius). Grįžtant atgal į pirmąją programą, ši būvį reikės vėl atstatyti. Išsaugojus OS būvį, vykdoma procedūra *Exec*, kurios prototipas:

```

procedure Exec (<Programos failo vardas>,
                <Argumentai>: string );

```

Iškviečiama programa nurodoma jungtiniu vardu, o antrasis procedūros parametras aprašo programai perduodamus argumentus. Pavyzdžiui, jeigu norime programos darbo metu pamatyti aktyvaus katalogo turinį, reikia iškviešti MS DOS komandų interpretatorių vardu *command.com* ir nurodyti jam argumentą – OS komandą *dir*:

```
Exec ('\\command.com', '/c dir');
```

Argumentų eilutėje požymis **/c** rašomas tiktai programai *command.com* perduodamo parametrų sąrašo pradžioje. Kreipiantis į kitas programas, šio požymio nereikia.

Įvykdžius procedūrą *Exec*, reikia atstatyti išsaugotą OS būvį (*SwapVectors*). Apie procedūros *Exec* iškviesto proceso pabaigos priežastis praneša funkcija *DosExitCode*:

```
function DosExitCode: word;
```

Reikšmė nulis informuoja apie sėkmingai pasibaigusį procesą. Operacinės sistemos procesus aktyvizuojančių programų kompiliavimą reikia

organizuoti taip, kad jos užimtų galimai mažiau operatyviosios atminties ir paliktų ten pakankamai vietos kitiems procesams organizuoti. Atminties ribojimai kompiliatoriui nurodomi aplinkos komanda *Options.Memory sizes*. Jos užklauskos rėmelyje yra trys laukai: *Stack size* (steko dydis), *High heap limit* (viršutinė dinaminės atminties riba) ir *Low heap limit* (apatinė dinaminės atminties riba).

Stekas skiriamas lokaliems kintamiesiems ir procedūrų-parametrų reikšmėms. Jo dydžio geriau nekeisti. Eksperimentiniu būdu mažinama tik viršutinė dinaminės atminties riba. Jei ši riba nesumažinama, dinaminei programos atminčiai skiriama visa laisva operatyvioji atmintis ir kitiems procesams gali nelikti vietos. Tada procedūros *Exec* iškvietimas visuomet būna nesėkmingas. Apie tai informuoja kintamojo *DosError* reikšmė 8 (trūksta atminties). Jeigu pageidaujama programą palikti operatyvioje atmintyje pastoviai, padaryti ją rezidentine, programos darbą reikia nutraukti procedūra *Keep*:

procedure **Keep** (<Kodas>: word);

Procedūros parametras nurodo, kokį kodą reikia perduoti OS, grąžinant jai valdymą. Čia vartojami tie patys kodai, kaip ir procedūroje *Halt*. Turi prasmę tik pertraukimais valdomos rezidentinės programos. Pertraukimais vadinamos specialios OS procedūros, kurios gali pertraukti kitų procesų darbą. Ryšys su jomis palaikomas per kompiuterio registrus. Dažniausiai taip sudaromos įvairių kompiuterio įrenginių, į kurių signalus kompiuteris turi reaguoti operatyviai, valdymo programos.

Modulyje *Dos* yra tiek kompiuterio registrų, tiek OS pertraukimų valdymo priemonės. Šioje knygoje tokios priemonės nenagrinėjamos, nes jas galima sąmoningai vartoti tikrai turint išsamių žinių apie operacines sistemas, kompiuterio architektūrą, procesų organizavimą ir programų derinimo priemones. Neturint tokių žinių, vienintelis rezultatas, kurio galima pasiekti tiesiogiai kreipiantis į registrus ir pertraukimų procedūras – padaryti kompiuterį nevaldomą.

Kreipiantis iš vienos programos į kitą, gali būti pageidaujama perduoti jos darbui reikalingus parametrus, pavyzdžiui, apdorojamo duomenų failo vardą. Tada vartojama tokia kreipinio į programą struktūra:

<Programos vardas> <Faktinių programos parametrų sąrašas>

Faktiniai parametrai gali būti tik eilutės. Jie vienas nuo kito ir nuo programos vardo atskiriami tarpais arba tabuliacijos simboliais. Norint sužinoti, kiek programai buvo perduota parametrų, vartojama parametrų neturinti funkcija:

function ParamCount: word;

Į konkrečius parametrus kreipiamasi funkcija *ParamStr*, kuri jau buvo naudota skyrelyje 7.1:

function ParamStr (<Indeksas>: word): string;

Nulinė indekso reikšmė nurodo pačios programos vardą, o likusių parametrų paskirtis numato programos autorius.

7.3.1 pratimas. Tarkime, kad reikia sudaryti programą, kuri parodytų ekrane aktyvaus katalogo turinį.

Sudaromoje programoje rekomenduojama kreiptis į OS komandą *Dir*, kuri atlieka pageidaujamą darbą. Programoje dinaminė atmintis nenaudojama, todėl su aplinkos komanda *Options.Memory sizes* galima parinkti net ir nulinę jos viršutinę ribą.

program Pratimas__7_4;

uses Dos;

begin

SwapVectors; { Programos būvio išsaugojimas }

Exec('command.com', '/c dir');

SwapVectors; { Programos būvio atstatymas }

if DosError<>0 then { Ar procesas baigėsi sėkmingai? }

WriteLn('OS klaida: ', DosError)

else WriteLn('Sėkmingo grįžimo kodas: ', DosExitCode);

ReadLn

end.

7.3.2 pratimas. Sudaryti programą, kuri dirba tikrai tada, kai kriptinyje į ją nurodomas teisingas slaptažodis - eilutė '1'. Jei slaptažodis įvedamas teisingai, apie tai informuojama ekrane ir parodomas duomenų failo turinys. Jei slaptažodis įvedamas klaidingai, programa ištrina duomenų failą.

program Pratimas_7_5;

var

f: text; e: string[12]; i: byte; c: char;

begin

{SI-}

Assign(f, 'duomenys.txt'); Reset(f);

{SI+}

```

if IOResult<>0 then begin
    WriteLn ('Duomenø nebëra'); Halt(0);
end;
e:= Paramstr(1);
for i:= 1 to 3 do
    if e<>'1' then begin
        Write('Slaptaþodis: '); ReadLn(e)
    end;
    if e='1' then begin          { Slaptaþodþio tikrinimas }
        WriteLn ('Paþinau. Esi savas. '); Reset(f);
        while not Eof(f) do begin
            Read(c); Write(c)
        end;
        WriteLn; Close(f)
    end
    else begin                  { Programos failo trynimas }
        WriteLn('Esi svetimas. Sunaikinu duomenis!');
        Erase(f);
    end;
end.

```

7.5. Vartotojo moduliai

Varotojo moduliuose sudaromi ir saugomi asmeniniam poreikiams pritaikyti pagalbiniø priemoniø aprašymø rinkiniai, asmeninės bibliotekos. Moduliø aprašø sintaksë:

```

unit <Modulio vardas>;
    interface
        <Ryšio dalis>
    implementation
        <Instrumentinė dalis>
    [begin
        <Inicializavimo dalis>]
    end.

```

Ryšio dalyje aprašomi tie modulio objektai, kuriuos numatoma vartoti modulio išorëje, kituose moduliuose ir programose. Tai gali būti duomenø tipø, konstantø, kintamøjø, struktūrø aprašai, procedūrø ir funkcijø

antraštės (prototipai). Jeigu modulio viduje vartojami kitų modulių elementai, šie moduliai taip pat turi būti aprašomi ryšio dalyje sakiniu *uses*. Instrumentinėje dalyje pateikiami detalūs tekstai visų procedūrų ir funkcijų, kurių antraštės yra ryšio dalyje. Kadangi išsamūs paprogramių antraščių aprašymai pateikiami ryšio dalyje, instrumentinės dalies paprogramių aprašymų antraštėse parametrų sąrašus galima praleisti. Instrumentinėje dalyje taip pat gali būti ir lokalių, skirtų naudojimui tik modulio viduje, duomenų ir paprogramių aprašai.

Inicializavimo dalyje surašomi operatoriai, kuriuos būtina atlikti prieš panaudojant modulį kitose programose. Jos turinys priklauso nuo modulio pobūdžio. Dažniausiai čia aprašomos ryšio dalies kintamųjų pradinės reikšmės. Jeigu paruošiamųjų veiksmų atlikti nereikia, inicializavimo dalies gali nebūti.

Pagalbiniai moduliai kompiliuojami aplinkos komanda *Compile.Compile*, nurodant, kad rezultatų paskirties vieta yra diskas. Paskirties vietą keičia komanda *Compile.Destination*, kuri leidžia pasirinkti diską arba operatyviąją atmintį. Kompiliavimo rezultatas yra *tpu* tipo failas, kurį sakiniu *uses* galima įtraukti į bet kurias kitas programas.

7.4.1 pratimas

Pratime pateiktas modulis *Langai*, kuriame saugomos procedūros *Langas* ir *Lapas*. Pirmoji procedūra formuoja ekrane parametrais nurodomo dydžio ir spalvos langą, o antroji – rašo ekrane tekstinio failo duomenis pageidaujamo dydžio puslapiais. Ryšio dalyje taip pat yra deklaruotas naujas tipas parametrai, skirtas procedūrai *Langas* perduodamiems parametrams registruoti. Procedūros *Langas* parametrai yra dviejų jo priešingų viršūnių koordinatės ir fono spalva.

unit Langai;

interface { Modulio ryšio dalis }

uses Crt;

{ Lango parametrai }

type parametrai= array [1..5] of byte;

procedure Langas(k: parametrai);

procedure Lapas(n: integer; var f: text);

{ Siunčia failo **f** tekstą į ekraną **n** eilučių dydžio puslapiais }

implementation

{ Instrumentinė dalis }

procedure Langas;

```

begin
    Window(k[1], k[2], k[3], k[4]);    { Aktyvus langas dalis }
    TextBackground(k[5]);              { Lango dažymas }
    ClrScr;
end;

procedure Lapas;
var
i: integer;                            { Eilučių skaitiklis }
e: string;                             { Skaitoma eilutė }
begin
    Reset(f);                          { Paruošimas skaitymui }
    repeat                              { Failo skaitymo ciklas }
        i:= 0;                         { Skaitiklio paruošimas }
        repeat                          { Puslapio skaitymo ciklas }
            ReadLn(f, e); WriteLn(e);
            i:= i+1;                    { Eilučių skaičiavimas }
            until EoF(f) or (i=n);      { Puslapio pabaiga }
            ReadLn;                     { Pausė }
            until EoF(f);               { Failo pabaiga }
        Close (f);                     { Failo uždarymas }
    end;

begin                                  { Inicializavimo dalis }
    TextBackground(black); ClrScr;     { Ekrano išvalymas }
end.

```

Modulio *Langai* galimybes iliustruoja programa *Pratimas_7_6*, kurioje parodyta, kaip ekrane galima sudaryti du skirtingos funkcinės paskirties langus: vieną – dialogui, antrą – rezultatams. Programa skaičiuoja dialogo metu nurodytame tekstiniame faile esančių eilučių kiekį ir didžiausią eilutės ilgį. Vartotojui pageidaujant, atskirame lange parodomi pradinio failo duomenys.

```

program Pratimas_7_6;
uses Langai, Crt;
const
                                { Langų parametų rinkiniai }
m1: parametrai=(1, 1, 37, 20, blue);
m2: parametrai=(40, 1, 80, 20, green);

```

```

var
  f: text;
  ek: byte; { Ekranos spalvos }
  m, max: integer; { Pagalbiniai kintamieji }
  e: string; { Analizės procedūra }
procedure Fsav (var f:text; var m, max: integer);
{ m – eilučių skaičius faile, max – didžiausias eilutės ilgis }
var fv: string[20]; { Tiriama failo vardas }
begin
  WriteLn('Tiriama failo vardas :'); ReadLn(fv);
  Assign(f, fv); Reset(f);
  max:= 0; m:= 0;
  while not Eof(f) do begin
    ReadLn(f, e); m:= m+1; { Eilutės skaitymas }
    if Length(e)>max then { Eilutės analizė }
      max:= Length(e);
  end;
end;

{ PAGRINDINĖ PROGRAMA }
begin
  ek:= TextAttr;
  Langas(m1); { Dialogo langas }
  Fsav(f, m, max); { Failo analizė }
  WriteLn('Faile buvo ', m, ' eilučių. '); { Analizės rezultatai }
  WriteLn('Didžiausias eilutės ilgis yra: ', max);
  Write('Ar žiūrėsite failo tekstą (t/n)? ');
  if Upcase(ReadKey)='T' then begin
    Write('Kitas puslapis – klavišas Enter. ');
    Langas(m2); { Darbo lango keitimas }
    Lapas(19, f); { Skaitymas puslapiais }
  end;
  Window (1, 1, 80, 25); { Spalvų atstatymas }
  TextAttr:= ek;
  ClrScr;
end.

```

7.5. Manipuliavimas ekrano langais

Kompiuterio ekranas yra pagrindinė ryšių su personalinių kompiuterių taikomosiomis programomis (išorinio interfeiso) priemonė. Nuo išorinio interfeiso vaizdumo ir patrauklumo priklauso parengtos programos sėkmė, todėl programuotojams svarbu žinoti, kokiomis priemonėmis jį projektuoti. Pagrindinis ekraninio vaizdo konstrukcinis elementas yra langas. Ekranu vaizdas būna gražesnis, jei jame sudaromi langai įrėminami. Tam galima vartoti pseudografikos simbolius, kuriuos reglamentuoja firmos IBM standartas.

Pavyzdžiui, dvigubi rėmeliai sudaromi iš simbolių “ $\text{F} = \text{q} \parallel \text{L} \text{J}$ “, kurių kodai yra atitinkamai: 201, 205, 187, 186, 200, 188. Viengubų rėmelių simbolių “ $\text{r} - \text{r} \mid \text{L} \text{J}$ “ kodai yra: 218, 196, 191, 179, 192, 217. Šiuos simbolius į ekraną galima perduoti struktūra #<kodas>, funkcija *chr*(<kodas>) ir eilutėmis. Pastarasis būdas yra vaizdžiausias, tačiau, jį vartojant, reikia mokėti tekstiniais redaktoriais suformuoti grafinius simbolius. Tai daroma renkant simbolio kodą, esant nuspaustam klavišui *Alt*. Taip galima gauti tik nevartojamų redaktoriaus valdymui kodų grafinius vaizdus.

Manipuliuojant ekrano langais, patogiau vartoti modulio *Crt word* tipo kintamuosius *WindMin* ir *WindMax*, kurių reikšmės saugo atitinkamai kairiojo viršutiniojo aktyvaus lango kampo (mažąsias) koordinatas ir dešiniojo apatiniojo kampo (didžiąsias) koordinatas. Atskiros kampo koordinatės (ekrano eilutės ir kolonėlės numeriai) yra saugomos atitinkamai vyresniajame ir jaunesniajame šių kintamųjų baite.

Atskiriant *word* tipo kintamųjų baitus, vartojamos sisteminės Turbo Paskalio bibliotekos funkcijos:

<i>function Hi</i> (<argumentas>): <i>byte</i> ;	{ Vyresnysis baitas }
<i>function Lo</i> (<argumentas>): <i>byte</i> ;	{ Jaunesnysis baitas }

Pavyzdžiui, aktyvaus (vėliausiai sukurto) lango mažąsias koordinatas galima sužinoti operatoriais:

y := Hi(WindMax); x := Lo(WindMin);

Vartojant kintamuosius *WindMin* ir *WindMax*, reikia atsiminti, kad jais aprašomo ekrano kairiojo viršutiniojo kampo koordinatės yra (0, 0), o to paties ekrano taško koordinatės procedūrose *Window* ir *GotoXY* yra (1, 1). Langų ir rėmelių kūrimui, manipuliavimui jais ir specialioms rašymo į langus procedūroms patartina susidaryti atskirą pagalbinę biblioteką. Tokios bibliotekos pavyzdys yra pateiktas modulyje **LanI**:

<i>unit LanI</i> ;	{ Langai ir rėmeliai }
--------------------	------------------------

```

interface                                     { Ryšio dalis }
  uses Crt;
  type
    sct= array [1..2000] of word;      { Masyvas ekrano modeliui }
    byterec= record lo,hi: byte end;    { Dviejų baitų junginys }
    titlet = string[60];                { Lango antraštė }
    framet = string[6];                 { Rėmelių simbolių rinkiniai }
    state = record                      { Lango parametrų rinkinys }
      min, max: word;                   { Lango koordinatės }
      x, y: byte;                       { Kursoriaus vieta }
      attr: byte;                       { Simbolių ir fono spalvos }
    end;
  const
    single = '┐┌';                     { Paprastas rėmelis }
    double = '══';                     { Dvigubas rėmelis }
    noframe= ' ';                       { Rėmelio nėra }
  var
    screen: sct absolute $B800:$0000; { Ekrano modelis }

    { Rašymo koordinatėmis nurodomoje lango vietoje procedūros }
    procedure WriteChar(x, y: byte; count: integer; c: char; attr: byte);
    procedure ChangeAttr(x, y: byte; count: integer; attr: byte);
    procedure WriteStr(x, y: byte; s: string; attr: byte);

    { Langų valdymo procedūros }
    function WinSize: integer;         { Simbolių kiekis lange }
    procedure FrameWin(title: titlet; frame: framet;
      titleattr, frameattr: byte); { Rėmelis su antrašte }
    procedure UnFrameWin;              { Rėmelio naikinimas }

  implementation                           { Realizavimo dalis }

  function WinSize;
  begin
    winsize:=(lo(WindMax)-lo(WindMin)+1)
      *(hi(WindMax)-hi(WindMin)+1);
  end;
  procedure WriteChar;                  { Rašymas nestumiant kursoriaus }
  var
    w,                                     { Aktyvaus lango plotis }
    temp: word;                           { Rašymo vieta ekrane }

```

```

begin
  w := lo(WindMax) - lo(WindMin) + 1;
  while count>0 do      begin
    while x>w do
      begin y:=y+1; x:=x-w end;
      if y>((hi(WindMax)-hi(WindMin)+1)) then Exit;
      temp:= (hi(WindMin)+y-1)*80+lo(WindMin)+x;
                                     { Simbolio rašymas }
      byterec(screen[temp]).lo:= ord(c);
                                     { Spalvų nurodymas }
      byterec(screen[temp]).hi:= attr;
      x:= x+1;                      { Kito simbolio vieta }
      count:= count-1;              { Simbolių skaičiavimas }
    end;
  end;

  procedure ChangeAttr;           { Spalvų keitimas }
  var w, temp: word;
  begin
    w := lo(WindMax) - lo(WindMin) + 1;
    while count>0 do      begin
      while x>w do
        begin y:=y+1; x:=x-w end;
        if y>((hi(WindMax)-hi(WindMin)+1)) then Exit;
        temp:= (hi(WindMin)+y-1)*80+lo(WindMin)+x;
        byterec(screen[temp]).hi:= attr;      { Spalvų keitimas }
      x:= x+1;      count:= count-1;
    end;
  end;

  { Eilutės rašymas nekeičiant kursoriaus vietos }

  procedure WriteStr;
  var i: byte;                      { Eilutės indeksas }
  begin
    if Length(s)= 0 then Exit;
    for i:=1 to Length(s) do
      WriteChar(x+i-1, y, 1, s[i], attr)
    end;

  procedure FrameWin;
  var w, h, y: word;

```

begin

{ Lango parametrai }

w := lo(WindMax) - lo(WindMin) + 1;

h := hi(WindMax) - hi(WindMin) + 1;

{ Rėmelio viršus }

WriteChar(1, 1, 1, frame[1], frameattr);

WriteChar(2, 1, w - 2, frame[2], frameattr);

WriteChar(w, 1, 1, frame[3], frameattr);

{ Antraštės ilgio ribojimas }

if Length(title) > w - 2 then title[0] := chr(w - 2);

{ Antraštės įterpimas }

Writestr((w - length(title)) div 2 + 1, 1, title, titleattr);

for y := 2 to h - 1 do *begin* { Rėmelio šonai }

WriteChar(1, y, 1, frame[4], frameattr);

WriteChar(w, y, 1, frame[4], frameattr);

end;

WriteChar(1, h, 1, frame[5], frameattr);

WriteChar(2, h, w - 2, frame[2], frameattr);

WriteChar(w, h, 1, frame[6], frameattr);

{ Aktyvi ekrano dalis }

WindMin:= WindMin + \$0101;

WindMax:= WindMax - \$0101;

GotoXY(1,1);

end;

procedure UnframeWin;

{ Nuimti rėmelį }

var w, h, x, y: word;

begin

WindMin:= WindMin - \$0101;

{ Aktyvi ekrano dalis }

WindMax:= WindMax + \$0101;

{ Rėmelio trynimasis }

FrameWin(“”, noframe, textattr, textattr);

end;

end.

Naudojant bibliotekas, vartotojui dažniausiai tenka pačiam pagal procedūrų prototipus ryšio dalyje ir jų komentarus išsiaiškinti procedūrų ir jų parametrų paskirtis. Todėl čia trumpai aptarsime tik dalį modulio procedūrų savybių.

Pagrindinė modulio procedūra, kuri vartojama visose kitose procedūrose, yra *WriteChar*. Ji rašo ekrane, nekeisdama kursoriaus vietos, simbolį ‘c’ tiek kartų, kiek nurodo parametras **count**.

Apribojant procedūroje *FrameWin* rėmelio antraštės ilgį, yra keičiamas antraštę saugančios eilutės nulinio baito, kuriame nurodomas eilutės ilgis, turinys. Tą patį veiksmą galima aprašyti ir standartine procedūra *Delete*.

Keičiant kintamųjų *WindMin* ir *WindMax* abiejų baitų reikšmes iš karto, yra vartojamos simboliais \$ žymimos šeioliktainės konstantos, nes tokiomis konstantomis labai paprastai aprašomi atskiri žodžio (word) baitai (lo ir hi).

Modulio *Lan1* galimybes iliustruoja programa *Pratimas_7_7*, kurioje parodyta, kaip galima formuoti dialogui su vartotoju skirtą ekrano vaizdą ir valdyti grupę jame esančių langų. Programa leidžia rinkti tekstą aktyviame lange ir operatyviai reaguoja į du klavišus: *F9* ir *F10*. Modulį *Lan1* taip pat labai patogų vartoti meniu tipo valdymo sistemoms projektuoti.

program **Pratimas_7_7**;

uses Lan1, Crt;

type

langai= array [1..3] of state; { Langų parametrų masyvas }

frames= array [0..2] of framet; { Rėmelių simboliai }

const

{ Langų parametrai }

ld: langai=

((min: \$0508; max: \$1025; x: 1; y: 1; attr: white+blue*16),

(min: \$0528; max: \$1045; x: 1; y: 1; attr: white+blue*16),

(min: \$121B; max: \$1733; x: 1; y: 1; attr: white+green*16));

fr: frames= (single, double, noframe);

fri: byte= 0; { Pradinis rėmelio indeksas }

{ Spalvų rinkiniai }

colblack: byte= black+lightgray*16;

colred: byte= red+lightgray*16;

var

i: byte; { Pagalbinio lango indeksas }

c: char; { Įvedamas simbolis }

out: boolean; { Darbo pabaigos žymė }

procedure **InitScr**; { Pradinis ekrano vaizdas }

begin

Window(1, 1, 80, 25); ClrScr;

WriteChar(1, 1, 80, ‘_’, colblack);

```

    Writestr(35, 1, ' LANGAI ', colred);
    WriteChar(1,25,80,' ', colblack);
    Writestr(3,25, 'F9-          F10-', colred);
    Writestr(8,25,'Rėmelio tipo keitimas', colblack);
    Writestr(36,25,'Pabaiga', colblack);
    Window(1, 2, 80, 24);
    FrameWin(' ', fr[fri], textattr, yellow+blue*16);
end;

procedure SaveCurs(i: byte);
    { Kursoriaus koordinacių išsaugojimas }
begin
    with ld[i] do
        begin x:= Wherex; y:= Wherey end;
end;

procedure ChangeWin(i:byte; fr:frame);
    { Aktyvaus lango keitimas }
var e:title;
begin
    with ld[i] do begin
        Str(i, e);
        Window(lo(min), hi(min), lo(max), hi(max));
        FrameWin(' '+e+' ', fr, white+magenta*16, attr);
        GotoXY(x,y);
    end;
end;

    { Pradiniai langų vaizdai }

procedure InitWin (fr:frame);
begin
    for i:=1 to 3 do begin
        Changewin(i, fr);
        Write('Pasyvus langas'); SaveCurs(i);
    end;
end;

    { Valdančių simbolių interpretavimas }

procedure Control(i: byte; var flag: boolean);
begin
    case ReadKey of
        { Klavišų apklausa }

```

```

#67:  begin                { Reakcija į klavišą F9 }
      fri:= (fri+1) mod 3;
      Changewin(i, fr[fri]);
      end;
#68:  flag:= true;        { Reakcija į klavišą F10 }
end;
end;

{          PAGRINDINĖ PROGRAMOS    DALIS          }
begin
  InitScr;
  WriteLn('Paspauskite klavišą Enter !');  ReadLn;
  UnframeWin;
  WriteLn('Apatinė ekrano eilutė aprašo programos valdymo klavišus');
  InitWin(fr[fri]);
  ClrScr; Write('Aktyvus langas. Rinkite tekstą');
  Write(' šiam lange arba spauskite valdymo klavišus ');
  SaveCurs(i);                                { Kursoriaus vieta }
  ChangeAttr(1, 1, 16, white+blue*16);      { Spalvų pakeitimas }
  c:= ' '; out:= false;                       { Pradinės reikšmės }
  while not out do  begin                     { Rašymas lange }
    c:= ReadKey;
    if c = #0 then Control(3, out)           { Valdymo klavišas }
    else Write(c);
    if c = #13 then WriteLn;
    SaveCurs(i);
  end;
end.

```

7.6. Pratimai

7.6.1 pratimas. Sudaryti programą, kuri parodytų ekrane jos iškviatimo komandoje nurodyto tekstinio failo turinį. Programos iškviatimo komandos pavyzdys: *skaityk duomenys.txt*.

7.6.2 pratimas. Pertvarkykite 7.6.1 užduoties programą taip, kad ji, neradusi iškviatimo komandoje nurodyto failo, praneštų apie tai vartotojui ir paprašytų patikslinti failo vardą.

7.6.3 pratimas. Sudaryti programą, kuri pateiktų ekrane informaciją apie kompiuterio kaupiklius (diskus): jų informacinę talpą ir laisvų baitų skaičių.

7.6.4 pratimas. Sudaryti programą, kuri parodytų ekrane vartotojo nurodytų metų kalendorių.

7.6.5 pratimas. Sudaryti programą, kuri ekrane suformuotų vartotojo pageidaujamo dydžio ir spalvos langą. Programoje turi būti pateikiama užklausa lango spalvai ir kairiojo viršutinio kampo koordinatėms. Po to sudaromas pradinis 2x2 langas, kurio dydis gali būti keičiamas klavišais ↑ ir →. Paspaudus klavišą →, langas turi pailgėti vienu stulpeliu, o paspaudus klavišą ↑, paplatėti viena eilute. Paspaudus klavišą *Enter*, lango formavimas turi būti baigiamas.

7.6.6 pratimas. Pertvarkykite užduotyje 7.6.5 sudarytą programą taip, kad suformuotą langą būtų galima stumdyti ekrane kursoriaus valdymo klavišais. Programos darbas turi būti nutraukiamas paspaudus klavišų porą *Ctrl+Q*.

7.6.7 pratimas. Sudarykite tekstinių failų analizės programai dviejų ekrano langų interfeisą. Vieną langą skirkite dialogui su vartotoju, o antrą - analizės rezultatams. Programa turi parodyti antrajame lange visus tiriamo failo žodžius, kurie prasideda vartotojo nurodyta raide.

7.6.8 pratimas. Papildykite 7.6.7 užduoties programą taip, kad jos darbą būtų galima nutraukti bet kuriuo dialogo ciklo metu paspaudus klavišą *F10*.

8. Duomenų tvarkymas

Šiame skyriuje:

- tvarkingas ir netvarkingas duomenų sujungimas;
- duomenų peržiūra ekrane;
- duomenų paieška netvarkingame ir tvarkingame sąrašė;
- duomenų rikiavimas išrinkimu (Mini-maksi būdas);

8.1. Duomenų paieška

Duomenų panaudojimo efektyvumą lemia gebėjimas greitai surasti reikalingus duomenis, juos atrinkti ir grupuoti. Paieškos algoritmų yra daug. Juos galima skirstyti į dvi grupes: paieška netvarkingame ir tvarkingame duomenų sąrašė. Kiekvienas duomenų paieškos algoritmas turi pranešti, ar paieška buvo sėkminga ar ne. Jeigu ieškomi duomenys buvo surasti, tuomet nurodoma jų vieta sąrašė arba (ir) atrenkami duomenys.

Paieška netvarkingame sąrašė. Paieška pradedama nuo sąrašo pradžios. Paieška nutraukiama suradus reikalingus duomenis. Jeigu duomenų su ieškoma savybe yra daug, tai būtina paiešką tęsti iki sąrašo galo. Nerandant ieškomų duomenų, visą sąrašą tenka peržiūrėti iki galo.

Paieškos algoritmo struktūrograma netvarkingame skaičių masyve:

Pradžia RastiNetvarkingame(A : mas; n : integer; b : integer; var k : integer);	
Kintamieji i : integer;	
k := 0;	i := 1
kol (i <= n) ir (k = 0)	
t \	A[i] = b
/ n	

	k := i	i := i + 1
Pabaiga RastiNetvarkingame		

Paieška tvarkingame saraše. Ankstesnis algoritmas papildomas veiksmu, kuriuo nustatoma, ar tolesnė paieška skaičių masyve tikslinga. Skaičiai masyve išdėstyti mažėjančia tvarka. Paieškos algoritmo struktūrograma tvarkingame skaičių masyve:

Pradžia RastiTvarkingame(A : mas; n : integer; b : integer; var k : integer);		
Kintamieji i : integer; yra : boolean		
k := 0; i := 1; yra := True		
kol (i<=n) ir yra		
t \		A[i] <= b
/ n		
t \		
/ n		
k := i;		
yra := False		i := i+1
Pabaiga RastiTvarkingame		

Tvarkingame saraše labai efektyvus dvejetainės paieškos algoritmas. Ieškomas objektas lyginamas su duomenų sekos viduryje esančia reikšme. Rezultate nustatome, kurioje pusėje jis gali būti: prieš ar po viduriniojo. Atmetame nereikalingą pusę. Paieškos intervalas dvigubai sumažėja. Čia vėl taikome tą pačią taisyklę: lyginame su viduriniaja intervalo reikšme. Vėl paieška trumpėja dvigubai. Procesas tęsiamas tol, kol intervale lieka tik viena reikšmė: jeigu ji lygi ieškomai, tai radome, kitaip objekto su mus dominančia savybe saraše nebuvo. Dvejetainės paieškos algoritmo struktūrograma tvarkingame skaičių masyve:

Pradžia DvejetainėPaieška(var A : mas; n : integer ; b : integer ; var k : integer);		
Kintamieji i, pr, gal : integer ;		
pr := 1; gal := n		
kol pr< gal		
i := (pr+gal) div 2		

i := (pr+gal) div 2	
t \	A[i]>b
/ n	
pr := i + 1	gal := i
t \	A[i]=b
/ n	
k := i	k := 0
Pabaiga DvejjetainėPaieška	

Čia mas = **array**[1 .. 100] **of integer**;

A(n) – sveikų skaičių masyvas.

b – ieškoma reikšmė masyve;

k = 0 ieškoma reikšmė nerasta ir k<>0 rodo surasto skaičiaus vietą masyve A(n).

8.1 pratimas. Tekstiniame faile duotas mokyklos mokytojų sąrašas: mokytojo pavardė, vardas ir gimimo data (metai, mėnuo, diena). Sudaryti nurodyto mėnesio mokytojų gimtadienių sąrašą.

Duomenų failo pavyzdys:

Astrauskas	Antanas	1959	1	30
Brazdeikis	Juozas	1965	10	15
Zukauskas	Liudas	1950	6	12
Petrauskas	Petras	1966	10	18
Juozaitis	Algis	1969	7	28
Giedraitis	Dalius	1950	1	17

Duomenis nuskaitysime į įrašų masyvą. Pageidaujamą mėnesį įvesime klaviatūra.

```

program Pratimas1;
uses Crt;
const L = 100;
        duomenys = 'Prat1';           { Failo pavadinimas }
type zodis = string[15];
        Mokytojas = record
                p, v : zodis;           { mokytojas }
                mt,           { metai }
                mn,           { mėnuo }
                dn : integer;          { diena }
        end;
        Sar = array[1..L] of Mokytojas;
```

```

{----- Įrašų masyvo skaitymas iš failo -- -----}
procedure Skaito(var A : Sar; var n : integer);
var F : text;
begin
    Assign(F, duomenys+'.dat'); Reset(F); n := 0;
    while not Eof(F) and (n<L) do begin n := n+1;
        with A[n] do ReadLn(F, p, v, mt, mn, dn);
    end;
    Close(F);
end;

{----- Įrašų masyvo spausdinimas į rezultatų failą ---}
procedure Raso(var A : Sar; var n : integer;
    var F : text);
var i : integer;
begin
    for i := 1 to n do with A[i] do
        WriteLn(F, p, v, mt:4, mn:4, dn:4);
    WriteLn(F);
end;

{----- Naujo sąrašo sudarymas -----}
procedure NaujasSar(var A, B : Sar; n : integer;
    men : integer; var m : integer);
var i : integer;
begin
    m := 0;
    for i := 1 to n do
        if A[i].mn=men then begin
            m := m+1; B[m] := A[i];
        end;
end;

{----- Pagrindinės programos kintamieji -----}
var A : Sar; n : integer; { Pradinis sąrašas A(n) }
    B : Sar; m : integer; { Naujas sąrašas B(m) }
    menesis : integer;
    F : text;

{----- Pagrindinė programa -----}
begin
    ClrScr;
    WriteLn('Programa pradeda darbą...');
    WriteLn('Paieška įrašų masyve');
    Assign(F, duomenys+'.rez'); Rewrite(F);
    Skaito(A, n);
    WriteLn(F, 'Pradinis mokytojų sąrašas'); Raso(A, n, F);

```

```

Write('Įveskite pageidaujamą mėnesį:');
ReadLn(menesis);
NaujasSar(A, B, n, menesis, m);
if m <> 0 then begin
    WriteLn(F, ' Suformuotas mokytojų sąrašas');
    Raso(B, m, F);
end
else
    WriteLn(F, menesis, 'mėnesį gimusių mokytojų nėra');
Close(F);
WriteLn('Rezultatų ieškokite faile:', Duomenys, '.rez');
WriteLn('Programa baigė darbą.');
```

ReadLn;

end.

Užduotys savarankiškam darbui

- ☒ Tekstiniame faile duotas netvarkingas laimingų loterijos bilieta sąrašas: bilieto numeris ir miestas. Suraskite miestą, kuriame buvo nupirktas bilietas su numeriu L.
- ☒ Tekstiniame faile duotas tvarkingas telefono abonentų sąrašas: telefono numeris ir abonto pavardė. Suraskite, kuris abonentas turi numerį N.

8.2. Duomenų rikiavimas

Duomenų rikiavimas (sudėliojimas) pageidaujama tvarka yra viena iš pagrindinių darbo su duomenimis operacijų. Netvarkinguose duomenų rinkiniuose surasti mus dominančią informaciją yra nepatogu ir daug pastangų reikalaujantis darbas, ypač, kai duomenų labai daug. Pavyzdžiui, telefonų abonentų knyga yra netvarkinga adreso požiūriu. Žinant tik mus dominančio žmogaus adresą (telefonas registruotas ne jo pavarde), surasti telefono numerį mažai kas bandys. Tačiau, surikiavus toki sąrašą pagal adresą, paieška tampa greita ir patogi.

Rikiavimo algoritmų yra daug. Kiekvienas jų turi savo trūkumus ir kiekvienas jų savaip geras.

Vienas paprasčiausių ir populiariausių algoritmų yra rikiavimas išrinkimu. Tokio algoritmo atstovas yra dažnai vadinamas „Mini-Max“ vardu. Jo modifikacijų yra keletas. Štai viena iš jų, rikiuojanti masyvą $A(n)$ mažėjimo tvarka:

Pradžia RikMinMax(var A : mas; n : integer)	
Kintamieji C : real ; i, j, maxi : integer	
i := 1, n-1	
maxi := i	
j := i, n	
t \	A[j] > A[maxi]
/ n	
maxi := j	
C := A[i]; A[i] := A[maxi]; A[maxi] := C;	
Pabaiga RikMinMax	


Šiame algoritme tinkamas duomenų tipas gali būti:

```
type mas = array [1..100] of real;
```

Žodinis „Mini-Max“ algoritmo aprašymas:

- Masyve A tarp elementų $a_1, a_2, a_3, a_4, \dots, a_n$ randamas didžiausias ir jis sukeičiamas vietomis su pirmuoju a_1 .
- Toliau, masyve A tarp elementų $a_2, a_3, a_4, \dots, a_n$ randamas didžiausias ir jis sukeičiamas vietomis su antruoju a_2 (pirmuoju paieškos intervale).
- Toliau, masyve A tarp elementų a_3, a_4, \dots, a_n randamas didžiausias ir jis sukeičiamas vietomis su trečiuoju a_3 (pirmuoju paieškos intervale).
- Veiksmai kartojami tol, kol intervale liks du elementai. Intervalo pradžią nusako pirmojo ciklo parametras i.

Jeigu didžiausios reikšmės paieškos veiksmė pakeisime ženklą $>$ (daugiau) į $<$ (mažiau), gausime mažiausios reikšmės paiešką. Tuo pačiu duomenų masyvas bus rikiuojamas didėjimo tvarka. Loginis reiškiny (A[j]>A[maxi]) vadinamas rikiavimo raktu. Nuo jo priklauso duomenų sutvarkymo pobūdis. Sudėtingose duomenų struktūrose šis reiškiny gali būti sudėtingas.

 **8.2 pratimas.** Tekstiniame faile turime mokinių sąrašą. Duomenys apie vieną mokinį surašyti vienoje eilutėje struktūriškai: pavardė, vardas, klasė, pažymiai. Reikia suformuoti mokinių sąrašą mokymosi vidurkio mažėjimo tvarka.

Duomenų failo pavyzdys:

Petrauskas	Petras	11	5	6	7	10	9	10
------------	--------	----	---	---	---	----	---	----

Katinas	Rudas	10	8	9	10	
Petrauskas	Algis	10	10	10	3	4 9
Lydeka	Gailius	11	6	7	8	
Katinas	Rainas	9	8	8	8	8
Katinas	Baltas	10	9	9		

Duomenis surašysime į masyvą, kur vietoje pažymių bus saugomas jų vidurkis.

```

program Pratimas2;
uses Crt;
const L = 100;
        duomenys = 'Prat2';           { Failo pavadinimas }
type zodis = string[15];
        Mokiny = record
            p, v : zodis;              { Mokiny          }
            kl : integer;              { Klasė           }
            vid : real;                { Vidurkis        }
        end;
        Sar = array[1..L] of Mokiny;

{----- Įrašų masyvo skaitymas iš failo -- -----}
procedure Skaity(var A : Sar; var n : integer);
var F : text;
        k, p : integer; { Pažymių kiekis ir darbinis kint. }
begin
    Assign(F, duomenys + '.dat');
    Reset(F);
    n := 0;
    while not Eof(F) and (n < L) do begin
        n := n + 1;
        Read(F, A[n].p, A[n].v, A[n].kl);
        k := 0; A[n].vid := 0;
        while not Eoln(F) do begin
            k := k + 1; Read(F, p);
            A[n].vid := A[n].vid + p;
        end;
        ReadLn(F);
        if k > 0 then A[n].vid := A[n].vid / k;
    end;
    Close(F);
end;

{----- Įrašų masyvo spausdinimas į rezultatų failą ---}
procedure Raso(var A : Sar; var n : integer;
               var F : text);

```

```

var k : integer;
begin
  for k := 1 to n do with A[k] do
    WriteLn(F, p, v, kl:4, vid:7:2);
  WriteLn (F);
end;

{----- Rikiavimo raktas: vidurkis -----}
function Vidurkis(A, B : Mokinys): boolean;
begin
  { Didesnis turi būti pirmesnis }
  Vidurkis := A.vid < B.vid;
end;

{----- Įrašų masyvo rikiavimas -----}
procedure Rikiuoti(var A : Sar; n : integer);
var i, j, d : integer; B : Mokinys;
begin
  for i := 1 to n-1 do begin
    d := i;
    for j := i+1 to n do
      if Vidurkis(A[d], A[j]) then d := j;
    B := A[d]; A[d] := A[i]; A[i] := B;
  end;
end;

{----- Pagrindinės programos kintamieji -----}
var A : Sar; n : integer; F : text;
{----- Pagrindinė programa -----}
begin
  ClrScr;
  WriteLn('Programa pradeda darbą...');
  WriteLn('Įrašų masyvo rikiavimas');
  Assign(F, duomenys + '.rez'); Rewrite(F);
  Skaito(A, n);
  WriteLn(F, ' Pradiniai duomenys');
  Raso(A, n, F);
  Rikiuoti(A, n);
  WriteLn(F, ' Surikiuoti duomenys');
  Raso(A, n, F);
  Close(F);
  WriteLn('Programa baigė darbą.');
```

```

  ReadLn;
end.
```

- ☒ Spausdinimo procedūra Raso neužbaigta. Papildykite procedūrą sakiniiais, kuriais rezultatų faile masyvo duomenys

būtų išvesti lentelė (lentelės antraštė, pabaiga). Lentelės eilutės turėtų būti sunumeruotos.

- ☑ Pakeiskite rikiavimo raktą taip, kad sąrašas būtų rikiuojamas vidurkio didėjimo tvarka.
- ☑ Pakeiskite rikiavimo raktą taip, kad sąrašas būtų rikiuojamas pagal abėcėlę.

Dažnai tuos pačius duomenis tenka rikiuoti pagal skirtingus raktus, t.y. turėti skirtingai surikiuotus tų pačių duomenų sąrašus. Šiuos rikiavimo rezultatus galima gauti dviem Jums žinomais būdais: kiekvienu atveju taisant programą, t.y. keičiant rakto formulę (funkcijoje *Vidurkis*) arba parašius keletą skirtingų rikiavimo procedūrų su savais raktais. Tačiau šie būdai yra nepatogūs. Paprasčiausia būtų turėti keletą rikiavimo raktų ir rikiavimo procedūrai nurodyti, kurį jų turi panaudoti. Tai galima padaryti naudojant selektorių **case** arba operatorių **if**. Bet rikiavimo procedūra tampa sudėtinga. Paprasčiau rikiavimo procedūrai per parametrus perduoti patį raktą.

Čia susipažinsite su parametrais funkcijomis. Parametrai procedūros yra analogiški ir Jūs patys sugebėsite juos savarankiškai išsiaiškinti.

Žinome, kad parametrų sąrašą sudaro vardai ir jiems nurodomi tipai. Reikia mokėti aprašyti funkcijos tipą. Jo sukūrimo taisyklė tokia:

```
<Funkcijos tipo vardas> = function(<parametrų sąrašas>)  
                                : <rezultato tipas>;
```

Pavyzdžiui:

```
Veiksmas = function(a, b : real): real;
```

Čia funkcija nėra įvardijama.

Procedūros ar funkcijos antraštėje kintamasis gali būti šio tipo, pavyzdžiui:

```
procedure Kalkuliatorius(a : integer;  
                        K : Veiksmas);
```

Tokios procedūros viduje vartojame funkciją *K* su tipo *Veiksmas* parametrais *a* ir *b*.

Pastaba: Programai reikia kelių funkcijų, kurių parametrų sąrašai turi būti tapatūs: jų turi būti tiek pat, kiek jų yra funkcijos tipo parametrų sąraše ir eilės tvarka jų tipai turi sutapti. Funkcijų rezultatų tipai taip pat turi būti vienodi.

Pavyzdžiui:

```
function Suma(x, y : real) : real;
function Sandauga(x, y : real) : real;
function Skirtumas(x, y : real) : real;
function Dalmuo(x, y : real) : real;
```

Kreipinyje į procedūrą, atitinkamo parametro vietoje, rašome reikiamos funkcijos vardą. Pavyzdžiui, galimi tokie kreipiniai:

```
Kalkuliatorius(v, Suma);
Kalkuliatorius(w, Sandauga);
```

8.3 Pratimas. 8.2 pratimo užduotis ir duomenys. Papildomai reikia surikiuoti duomenis pagal vidurkį, pagal pavardes ir pagal vidurkį ir pavardes.

```
program Pratimas3;
uses Crt;
const L = 100;
      duomenys = 'Prat2';
type  zodis = string[15];
      Skaiciai = array[1..10] of integer;
      Mokinys = record
          p, v : zodis;           { Mokinys      }
          kl   : integer;         { Klasė     }
          paz  : Skaiciai;        { Pažymiai  }
          vid  : real;            { Vidurkis  }
      end;
      Sar     = array[1..L] of Mokinys;
      Raktas = function(A, B : Mokinys) : boolean;

{----- Pažymių vidurkio skaičiavimas -----}
function Vid(A : Skaiciai) : real;
    var S, k, i : integer;
begin
    S := 0; k := 0;
    for i := 1 to 10 do begin
        S := S+i*A[i]; k := k+A[i];
    end;
    if k>0 then Vid := S/k else Vid := 0;
end;

{----- Įrašų masyvo skaitymas iš failo -- -----}
procedure Skaity(var A : Sar; var n : integer);
var F : text; k : integer;
begin
    Assign(F, duomenys+'.dat'); Reset(F);
    n := 0;
    while not Eof(F) and (n<L) do begin
```

```

    n := n+1;
    for k := 1 to 10 do A[n].paz[k] := 0;
    Read(F, A[n].p, A[n].v, A[n].kl);
    while not EoLn(F) do begin
        Read(F, k);
        A[n].paz[k] := A[n].paz[k] + 1;
    end;
    A[n].vid := Vid(A[n].paz);
    ReadLn(F);
end;
Close(F);
end;

{----- Įrašų masyvo spausdinimas į rezultatų failą ---}
procedure Raso(var A : Sar; var n : integer;
               var F : text);
    var k : integer;
begin
    for k := 1 to n do with A[k] do
        WriteLn(F, p, v, kl: 4, vid:7:2);
    WriteLn (F);
end;

{----- Raktas: pagal vidurkį -----}
function Vidurkis(A, B : Mokiny): boolean; far;
begin
    Vidurkis := A.vid < B.vid;
end;

{----- Raktas: pagal pavarde ir vardą -----}
function Vardas(A, B : Mokiny): boolean; far;
begin
    Vardas := (A.p > B.p) or
               ((A.p = B.p) and (A.v > B.v));
end;

{----- Raktas: pagal vidurkį, pavarde ir vardą -----}
function VidVard(A, B : Mokiny): boolean; far;
begin
    VidVard := (A.vid < B.vid) or
               ((A.vid = B.vid) and (A.p > B.p)) or
               ((A.vid = B.vid) and (A.p = B.p) and
                (A.v > B.v));
end;

{----- Įrašų masyvo rikiavimas -----}
procedure Rikiuoti(var A : Sar; n : integer;
                   Tikrina : Raktas);
    var i, j, d : integer; B : Mokiny;
begin
    for i := 1 to n-1 do begin
        d := i;

```

```

        for j := i+1 to n do
            if Tikrina(A[d], A[j]) then d := j;
            B := A[d]; A[d] := A[i]; A[i] := B;
        end;
    end;
end;

{----- Pagrindinės programos kintamieji -----}
var A,
    B : Sar;                      { Pagalbinis masyvas }
    n : integer;                  { Masyvo ilgis }
    F : text;

{----- Pagrindinė programa -----}
begin
    ClrScr;
    WriteLn('Programa pradeda darbą...');
    WriteLn('Irašų masyvo rikiavimas');
    Assign(F, duomenys+'.rez'); Rewrite(F);
    Skaito(A, n);
    WriteLn(F, 'Pradiniai duomenys');
    Raso(A, n, F);

    B := A;
    Rikiuoti(B, n, Vidurkis);
    WriteLn(F, ' Duomenys surikiuoti pagal vidurkį');
    Raso(B, n, F);

    B := A;
    Rikiuoti(B, n, Vardas);
    Write (F, ' Duomenys surikiuoti pagal pavardes');
    WriteLn(F, 'ir vardus');
    Raso(B, n, F);

    B := A;
    Rikiuoti(B, n, VidVard);
    Write (F, ' Duomenys surikiuoti pagal vidurkį');
    WriteLn(F, 'ir pavardes');
    Raso(B, n, F);

    Close(F);
    WriteLn ('Programa baigė darbą.');
```

ReadLn;

end.

Užduotys savarankiškam darbui

- ☒ Sudarykite stačiakampio centro koordinatų skaičiavimo programą. Argumentai – priešingų stačiakampio kampų koordinatės. Programa turi įvesti duomenis. Naudokite spalvas ir garsus.
- ☒ Išsiaiškinkite mokinio pažymių saugojimo formą.

- ☑ Sutvarkykite duomenų rašymo į rezultatų failą formą. Panaudokite tą pačią procedūrą duomenų spausdinimui ekrane.
- ☑ Kokia turi būti duomenų lentelės spausdinimo procedūra, jeigu reikia atspausdinti mokinio turimus pažymius. Kokie galimi variantai?
- ☑ Papildykite programą nauja rikiavimo rakto funkcija.
- ☑ Suraskite literatūroje kitokių duomenų rikiavimo algoritmą ir juo pakeiskite šioje programoje naudojamą (pvz *Burbuliuko*, *Dalinimo pusiau* ar pan.).
- ☑ Tekstiniam faile yra turimų prekių sąrašas. Duomenys apie prekę surašyti vienoje eilutėje: *pavadinimas, kiekis, kaina*. Reikia sudaryti tokius sąrašus: pradinių duomenų; surikiuotus pagal abėcėlę, arba pagal kainą ir abėcėlę, arba pagal turimų prekių sumą (*kiekis *kaina*). Pageidaujamam sąrašui gauti reikalingas dialogas su Jūsų programos vartotoju.

8.3 Duomenų sujungimas

Duomenų sujungimo veiksmu vadinsime duomenų, esančių skirtinguose rinkiniuose, surašymą viename duomenų rinkinyje. Duomenis sujungti galima atrenkant tik tenkinančius nurodytą sąlygą, t.y ne visus duomenis, o tik reikalingus. Sujungimo rezultatu gali būti netvarkingas arba tvarkingas sąrašas. Tai priklauso nuo naudoto algoritmo ir nuo duomenų šaltinio tvarkos. Sujungus duomenis senieji sąrašai gali būti saugomi arba naikinami. Tai priklauso nuo konkrečių užduoties sąlygų.

Atvirkščias veiksmas duomenų sujungimui – duomenų paskirstymas keliems rinkiniams pagal nurodytas sąlygas. Duomenims sujungti naudojami algoritmai panašūs į naujų sąrašų sudarymo algoritmus. Tik ten duomenys kopijuojami į naujus sąrašus, o čia jie perkeliama (jų nelieta pradiniame sąrašė).

Dviejų netvarkingų sąrašų sujungimas. Tarkime, kad turime du sąrašus $A(n)$ ir $B(m)$. Juos reikia sujungti sąrašė $C(k)$. Aišku, kad iš viso bus $k=n+m$ elementų. Kadangi tai bus netvarkingas sąrašas, todėl nesvarbu kokia seka pradinių sąrašų elementai bus surašyti į C . Galimas algoritmas *SujungtiNetvarkingai*. Pirmojo

algoritmo ciklo galima atsisakyti, pakeičiant jį tokiais veiksmais: $C := A; \quad k := n;$

Dviejų netvarkingų sąrašų sujungimo algoritmas:

Pradžia SujungtiNetvarkingai
$k:=0$
$i :=1,n$
$k := k + 1; \quad C[k] :=A[i]$
$i :=1,m$
$k := k + 1; \quad C[k] :=B[i]$
Pabaiga SujungtiNetvarkingai

Tvarkingų sąrašų sujungimas į naują tvarkingą sąrašą.
 Paprasčiausias atvejis yra, kai sąrašai $A(n)$ ir $B(m)$ sutvarkyti pagal tą patį raktą. Aišku, galime sujungti nekreipdami dėmesio į rezultatų tvarką, o paskui sutvarkyti. Tačiau kam dirbti dvigubą darbą, jeigu galime sujungti duomenis taip, kad tvarka išliktų. Darbo principą išsiaiškinsime tokiu pavyzdžiu. Masyvuose $A(n)$ ir $B(m)$ skaičiai surašyti mažėjimo tvarka. Reikia visus skaičius surašyti į masyvą C taip, kad tvarka nepakistų. Darbo schema tokia: eilės tvarka imame iš A ir B po vieną elementą; didesnįjį iš jų rašome į masyvą C ; kai vienas iš masyvų jau perrašytas, kito masyvo elementus perrašome eilės tvarka.

Dviejų tvarkingų sąrašų sujungimo algoritmas:

Pradžia SujungtiTvarkingai
$k :=0; \quad i :=1; \quad j :=1;$
kol $i \leq n \quad \text{ir} \quad j \leq m$
$t \setminus \hspace{15em} A[i] \geq B[j]$ $/ \hspace{15em} n$
$k := k+1; \quad C[k] := A[i]$
$k := k+1; \quad C[k] := B[j]$
$i := i+1$
$j := j+1$
kol $i \leq n$
$k := k+1; \quad C[k] := A[i]; \quad i := i+1$
kol $j \leq m$
$k := k+1; \quad C[k] := B[j]; \quad j := j+1$
Pabaiga SujungtiTvarkingai

Tvarkingo ir netvarkingo sarašų sujungimas. Šis atvejis aktualus, kai turime tvarkingą didelės apimties sarašą ir jį reikia papildyti keliais naujais netvarkingai surašytais įrašais. Pavyzdžiui, telefonų abonentų sarašą papildome naujais abonentais, kurių yra nedaug. Galimi keli būdai. Pirmasis labai neefektyvus: sujungiame abu, kaip netvarkingus, o po to surikiuojame. Antrasis, kai netvarkingą surikiuojame ir po to sujungiame, kaip du tvarkingus. Tai taip pat nėra labai efektyvus variantas. Racionaliausias yra trečiasis. Į naują sarašą kopijuojamas tvarkingas. Po to iš netvarkingojo sarašo iš eilės imami elementai ir įterpiami į tvarkingą taip, kad sarašo tvarka išliktų. Čia reikalingi du veiksmai: surasti įterpimo vietą pagal duotą tvarkos rakta ir po to įterpti. Šie du veiksmai buvo nagrinėti atskirai. Siūlome patiems juos pritaikyti.

8.4 pratimas. Tekstiniame duomenų faile yra krepšinio komandos sarašas, sutvarkytas amžiaus ir ūgio mažėjimo tvarka. Kitame tekstiniame faile yra komandos kandidatų sarašas tokia pat tvarka. Kandidatų yra daugiau negu komandos narių. Duomenys apie žaidėjus surašyti tokia tvarka: pavardė ir vardas užima eilutėje pirmas 30 pozicijų (1-15 ir 16-30), toliau gimimo data, kurią sudaro trys atskiri sveiko tipo skaičiai, ir žaidėjo ūgis centimetrais.

Iš komandos į kitas komandas išsina k pirmųjų žaidėjų. Tiek pat žaidėjų reikia perkelti į komandą iš kandidatų sarašo pagal esamo sarašo eilę. Parašykite programą, kuri formuotų naują komandą, koreguojant kandidatų sarašą. Sarašų tvarka turi išlikti.

Duomenų pavyzdys. Failas Prat4_1.dat:

Petrauskas	Petras	1975	12	13	201.5
Kazlauskas	Mindaugas	1975	12	31	200
Taiklioji	Akis	1980	05	05	197.8
Berzinis	Petras	1983	10	15	187
Katinas	Batuotas	1983	10	15	182

Failas Prat4_2.dat:

Petraitis	Gintautas	1985	12	13	215
Ungurys	Rapolas	1986	12	12	216
Juodoji	Pantera	1987	05	04	204
Smilga	Mindaugas	1988	11	11	199.5

```

program Pratimas4;
uses Crt;
const Komanda      ='Prat4_1.dat';
        Kandidatai ='Prat4_2.dat';
        Rezultatai ='Prat4.rez';
        L = 20;
type      zodis  = string[15];
        Zmogus = record
                pav, vard      : zodis;
                met, men, die  : integer;
                ugis           : real;
        end;
        Sarasas = array[1..L] of Zmogus;
{----- Įrašų masyvo skaitymas iš failo -- -----}
procedure Skaityto(var A : Sarasas; var n : integer;
                   failas : string);
        var F : text;
begin
        Assign(F, failas); Reset(F); n := 0;
        while not Eof(F) and (n<L) do begin
                n := n +1;
                with A[n] do
                        ReadLn(F, pav, vard, met, men, die, ugis);
                end;
        Close(F);
end;
{----- Brėžia liniją -----}
procedure Linija(var F : text; sim : char; n : integer);
        var i : integer;
begin
        for i := 1 to n do Write(F, sim);
        WriteLn(F);
end;
{----- Braižo ir užpildo lentelę -----}
procedure Raso(var A : Sarasas; var n : integer;
               tekstas : string; var F : text );
        var i : integer; data, s1, s2 : string[10];
begin
        WriteLn(F, '      '+ tekstas +'sąrašas:');
        Linija(F, '-', 58);
        WriteLn(F, '|      Krepšininkas                                |',
                'Gimimo data| Ūgis      |');

```

```

Linija(F, '-', 58);
for i := 1 to n do
    with A[i] do begin
        Str(met:4, data);
        Str(men:2, s1);
        Str(die:2, s2);
        data := data+'.'+s1+'.'+s2;
        WriteLn(F, '|', pav, '|', vard, '|', data, '|',
            ugis:5:1, '|');
    end;
Linija(F, '-', 58); WriteLn(F);
end;

{----- Iš sąrašo pradžios pašalina k žaidėjų -----}
procedure Mesti(var A : Sarasas; var n : integer;
    k : integer);
    var i : integer;
begin
    for i := k+1 to n do A[i-k] := A[i];
    n := n-k;
end;

{----- Rikiavimo raktas: pagal datą ir ūgį -----}
function Raktas(A, B : zmogus) : boolean;
begin
    Raktas := (A.met < B.met) or
        ((A.met = B.met) and (A.men > B.men)) or
        ((A.met = B.met) and (A.men = B.men) and
            (A.die > B.die)) or
        ((A.met = B.met) and (A.men = B.men) and
            (A.die = B.die) and (A.ugis > B.ugis));
end;

{----- Sąrašo A papildymas iš sąrašo B -----}
procedure Papildyti(var A,B : Sarasas; var na : integer;
    k : integer);
    var i, j : integer; C : Sarasas; nc : integer;
begin
    nc := 0; i := 1; j := 1;
    while (i<=na) and (j<=k) do
        if Raktas(A[i], B[j]) then begin
            nc := nc + 1; C[nc] := A[i]; i := i+1;
        end
        else begin
            nc := nc + 1; C[nc] := B[j]; j := j+1;
        end;
    while (i<=na) do begin

```

```

        nc := nc+1; C[nc] := A[i]; i := i+1;
    end;
    while (j <= k) do begin
        nc := nc+1; C[nc] := B[j]; j := j+1;
    end;
    A := C; na := nc;
end;
{----- Pagrindinės programos kintamieji -----}
var F : text;
    A : sarasas;      { Komandos žaidėjų sąrašas }
    na : integer;
    B : sarasas;      { Komandos kandidatų sąrašas }
    nb : integer;
    k : integer;      { Tiek žaidėjų palieka komanda }
{----- Pagrindinė programa -----}
begin
    ClrScr;
    WriteLn('Programa pradeda darbą...');
    WriteLn('Krepšinio komandos formavimas');
    Assign(F, Rezultatai); Rewrite(F);
    Skaity(A, na, Komanda);
    Raso(A, na, 'Pagrindinis komandos', F);
    Skaity(B, nb, Kandidatai);
    Raso(B, nb, 'Komandos kandidatų', F);
    WriteLn('Kiek žaidėjų paliko komandą?');
    Writeln(' (Viso komandoje yra:', na:2,') ');
    ReadLn(k);
    Mesti(A, na, k);
    Raso(A, na, 'Likusios komandos', F);
    Papildyti(A, B, na, k);
    Mesti(B, nb, k);
    Raso(A, na, 'Naujos komandos', F);
    Raso(B, nb, 'Likusios kandidatų komandos', F);
    Close(F);
    WriteLn('Programa baigė darbą. ');
    ReadLn;
end.

```

Užduotys savarankiškam darbui

- ☑ Išsiaiškinkite programą ir papildykite ją komentarais. Naujas komandos sąrašas spausdinamas lentelė, tačiau nėra surašomas atgal į buvusio sąrašo vietą faile Pr3_1.dat. Papildykite programą tokiu veiksmu.

- ☑ Programos vartotojas nori matyti ekrane kurią nors lentelę. Sudarykite tokią paslaugą atliekančios programos dalį (pageidavimų užklausa ir įvykdymas).
- ☑ Tekstiniame faile yra Jūsų fonotekos sąrašas: muzikinis kūrinys, autorius (atlikėjas, ansamblis), kūrinio ilgis (minutėmis). Sudarykite programą duomenims iš failo rašyti į įrašų masyvą ir atlikti norimo kūrinio paiešką. Tam paklauskite vartotojo, pagal kurią iš kūrinio aprašo parametą (pavadinimą ar autorių) ieškoti informacijos. Paskui sutvarkykite sąrašą pagal reikiamą parametą ir atlikite paiešką. Programa turi dirbti tol, kol vartotojas norės naudotis jos paslaugomis.

9. Ekraninė grafika

9.1. Grafinio ekrano tvarkyklė

Kompiuterio ekrane visi vaizduojami elementai sudaromi iš taškų. Tekstiniame ekrano darbo režime visos išvedimo ekraną priemonės dirba fiksuoto dydžio simboliais, kurių forma yra saugoma Turbo Paskalio kalbos failuose. Šiuo atveju ekrane galima formuoti pranešimus, naudojant tik simbolius, esančius konkrečiame simbolių rinkinyje. Turbo Paskalio bibliotekoje **Graph** yra priemonių rinkinys darbui su kompiuterio ekranu grafiniame režime. Programuotojas gali pats formuoti vaizdus iš taškų. Norint dirbti grafiniame režime, reikia kompiuterio ekraną paruošti. Tam naudojamos specialios priemonės. Ekrano grafinio darbo režimą nusakantys parametrai yra saugomi specialiuose *.bgi failuose. Grafinių priemonių biblioteka saugoma **Graph.tpu** faile.

```
procedure InitGraph( var GraphDriver : integer;  
                     var GraphMode   : integer;  
                     DriverPath     : string );
```

Ši procedūra paruošia kompiuterio ekraną darbui grafiniu režimu.

GraphDriver kintamasis, kurio reikšmė paprogramei nurodo, koks bus ekrano grafinis draiverio numeris. Standartinio draiverio numeris saugomas konstanta **Detect** = 0.

GraphMode konstanta, kuri nusako ekrano formatą: taškų skaičių, spalvas ir jų kiekį, grafinių puslapių kiekį. Nenurodžius reikšmės, parenkama standartinė pagal nutylėjimą.

DriverPath kelias į katalogą, kur saugomas draiverio failas.

GrOK standartinė konstanta, kurios reikšmė atitinka teisingai aktyvizuoto grafinio ekrano darbo režimo požymiai.

function GraphResult : integer; funkcija, kurios rezultatas yra reikšmė, rodanti kaip sėkmingai buvo aktyvizuotas grafinis ekrano darbo režimas. Iš reikšmės (sveikas skaičius) galima nustatyti nesėkmės atveju galimą priežastį.

procedure CloseGraph; uždaro grafinį ekrano darbo režimą. Grįžtama į tekstinį darbo režimą.

```
function GetMaxX : integer;
```

```
function GetMaxY : integer;
```

Šios funkcijos pasako aktyvizuoto ekrano taškų kiekį horizontalia ir vertikalio kryptimis. Tai galimos didžiausios X ir Y koordinatų reikšmės. Atskaitos taškas yra kairysis viršutinis ekrano kampas, kurio koordinatės yra (0,0).

Ekrano aktyvi spalva nurodoma procedūra

```
procedure SetBkColor( spalva : word );
```

Ekraną valo nurodyta spalva procedūra **ClearDevice**; Pagal nutylėjimą standartinė fono spalva bus juoda. Išbandykite tai programoje.

Visose programose, dirbančiose grafiniame ekrane, reikalinga grafinio ekrano inicializacija. Tikslinga visus tuos veiksmus apiforminti procedūra ir patalpinti vartotojo bibliotekoje. Šio skyriaus programose naudojama procedūra bus bibliotekoje Grafika.tpu (programos tekstas faile Grafika.pas).

```
unit Grafika;
interface
uses Graph;
procedure Ekranas ( var dx , dy : integer;
                    kelias : string );
{ dx, dy - grafinio ekrano taškų skaičius horizontalia ir
  vertikalio kryptimis; kelias - draiverių katalogas. }
implementation
  procedure Ekranas;
    var gd, gm : integer;
begin
  gd := Detect;
  InitGraph ( gd ,gm , kelias );
  if GraphResult <> grOk then begin
    WriteLn ( ' Aparaturine klaida !!! '); Halt ( 1 );
    end;
  dx := GetMaxX;      dy := GetMaxY;
end;
begin
end.
```

9.2. Ekrano grafikos elementai.

Taškai.

Ekrane piešiamas taškas nurodomas jo koordinatėmis (x,y) ir spalva. Tam skirta procedūra:

```
procedure PutPixel( x, y, spalva : word );
```

1 pratimas. Išbandysime visų spalvų taškus. Juos galima atsitiktinai dėti ekrane ir gauti malonų akių vaizdą, tačiau šioje programoje ekraną daliname į kvadratėlius, kuriuose dėliojami tik tam tikros spalvos taškai. Gausime taškuotus kvadratėlius iš skirtingų spalvų. Kuo daugiau taškų bus padėta, tuo sodresnė bus kvadratėlio spalva. Programoje atsitiktinių skaičių generatoriumi gaunamos taško koordinatės. Pagal jas yra nustatomas kvadratėlio numeris, kuris ir yra taško spalva. Ekrane matysime tris eilutes po penkis kvadratėlius. Juodai spalvai kvadratėlis neskirtas, nes tai ekrano fono spalva.

```

program Pratimas1;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Langeliai (  mx, my, dx, dy : integer );
    { mx, my - langelio kraštinių ilgiai. }
    { dx, dy - didžiausios leistinos taško koordinatės. }
    var sp, x, y : integer;
    begin
        x := Random( dx ); { Taško koordinatės }
        y := Random( dy );
                                { Langelį atitinkanti spalva }
        sp := ( x div mx )+1 + ( y div my )*5;
        PutPixel( x , y, sp );
    end;
{-----}
var mx, my, dx, dy : integer;
begin
    Randomize; Ekranas( dx, dy, Vieta );
    mx := dx div 5; { Langelio išmatavimai }
    my := dy div 3;
    repeat
        Langeliai( mx, my, dx, dy )
    until KeyPressed;
ReadLn;
CloseGraph;
end.

```

✓ Pertvarkykite programą taip, kad vietoje spalvotų kvadratėlių būtų spalvotos juostos (horizontalios arba vertikalios).

✓ Kvadratėlius pakeiskite kitokia figūra, pavyzdžiui skrituliais.

2 pratimas. Iš taškų sudaroma figūra labai paprastai. Tam generuojamos taško koordinatės (ir spalva, jeigu reikia) figūrą apibrėžiančios kreivės viduje. Galimas kitoks būdas: generuojamos taško

koordinatės nurodytose ribose, po to tikrinama taško padėtis: jeigu figūroje, tai taškas piešiamas, kitaip - ne. Visais atvejais reikia turėti matematinės formules, aprašančias figūrą. Programoje tai demonstruojama su skrituliais ir kvadratais. Taškai generuojami figūrą aprašančio stačiakampio viduje. Toliau patikrinama, ar taškas priklauso figūrai. Kadangi figūros simetriškos, tai taškas padedamas keturiose vietose: vertikalios ir horizontalios simetrijos ašių atžvilgiu.

```

program Pratimas2;
uses Crt, Graph, Grafika;
const Greitis = 1; { Pauze piešiant taškus }
      Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Skritulys( x, y, r, spalva : integer );
      { x, y - skritulio centro koordinatės; }
      { r - skritulio spindulys; spalva- taškų spalva.}
var xt, yt : longint; c : char;
begin
  Randomize;
  repeat
    xt := Random( r ); yt := Random( r );
    { Tikrinama, ar taškas apskritimo viduje }
    if Sqrt( xt * xt + yt * yt) <= r then begin
      PutPixel( x - xt, y - yt, spalva );
      PutPixel( x - xt, y + yt, spalva );
      PutPixel( x + xt, y - yt, spalva );
      PutPixel( x + xt, y + yt, spalva );
      Delay( Greitis );
    end;
  until KeyPressed; { Ar paspaustas klavišas? }
  c := ReadKey; { Paspausto klavišo skaitymas }
end;
{-----}
procedure Status( a, b, x, y, spalva : integer );
      { a, b - stačiakampio kraštinės; }
      { x, y - stačiakampio centro koordinatės ekrane }
var xt, yt : integer; c : char;
begin
  Randomize;
  repeat
    xt := Random( a div 2 ); yt := Random( b div 2 );
    PutPixel( x - xt, y - yt, spalva );
    PutPixel( x - xt, y + yt, spalva+1 );
    PutPixel( x + xt, y - yt, spalva+2 );
    PutPixel( x + xt, y + yt, spalva+3 );
    Delay( Greitis );
  
```

```

until KeyPressed;
c := ReadKey;
end;
{-----}
var dx, dy : integer;
begin
  Ekranas ( dx, dy, Vieta );
  Skritulys ( 100, 120, 100, Red );
  Skritulys ( 300, 100, 80, Green );
  Skritulys ( 410, 210, 70, Blue );
  Skritulys ( 285, 330, 100, Magenta );
  Status ( 150, 150, 550, 100, Brown );
  Status ( 150, 150, 540, 330, Red );
  Status ( 150, 150, 100, 300, Green );
  ReadLn;
  CloseGraph;
end.

```

✓ Pakeiskite skritulio piešimo programą taip, kad figūros ketvirtadaliai būtų skirtingos spalvos.

✓ Panaudodami skritulio piešimo procedūrą, nupieškite didelį skritulį. Jo viduje mažesnį. To mažesniojo viduje dar mažesnį. Kiek galima mažinti skritulį? Gal galite tai aprašyti ciklu?

Linijos

Linija ekrane suprantama kaip tiesės atkarpa. Ją nusako galų taškų koordinatės (x,y). Atkarpą nusako tokios charakteristikos: spalva, tipas ir storis. Šios charakteristikos yra nurodomos atskirai ir galioja visoms brėžiamoms atkarpoms tol, kol jų nepakeisime naujomis.

```
procedure SetColor ( spalva : word );
```

Nustatoma linijų spalva (konstantos vardas arba kodas: 0..15).

```
procedure SetLineStyle( tipas, kodas, storis : word );
```

Nurodomos linijų charakteristikos. Galimos tipo ir storio reikšmės surašytos 9.1 lentelėje. Kodas kreipinyje rašomas 0 (nulis). Jis naudojamas tik tuo atveju, kai nurodomas linijos tipas 4. Tuomet kodas yra rašomas tas, kurį sukūrė vartotojas. Tai šešiolyktainis skaičius.

9.1 lentelė. Linijų piešimo tipai ir storis.

Tipas	Storis
0 ištisinė linija	1 normalus (vieno taško)

1	punktyrinė linija	3	storas (trijų taškų)
2	ašinė linija		
3	taškinė linija		
4	vartotojo sukurta		

Kodas. Naudojamas, kai nurodomas tipas 4. Tai linijos tipo aprašas. Jį sudaro atkarpos fragmentas, kuris telpa žodyje iš 16 bitų. Jeigu bito reikšmė **1**, tuomet naudojama taško spalva, o jeigu **0**, tuomet lieka fono spalva (tokia, kokia buvo). Pavyzdžiui:

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	\$FFFF
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	\$AAAA
1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1	\$F249

procedure Line (x1, y1, x2, y2 : integer);

Nubrėžia atkarpą tarp nurodytų dviejų taškų su anksčiau nurodytomis charakteristikomis.

procedure MoveTo (x, y : integer); Žymeklis perkeliamas į ekrano vietą, kurią nusako taškas (x, y). Žymeklis grafiniame ekrane nematomas.

procedure LineTo (x1, y1 : integer); Iš taško, kur yra žymeklis, brėžama linija į tašką (x1, y1).

procedure MoveRel (dx, dy : integer); Analogiška **MoveTo**, tik dx, dy nurodo per kiek pasikeičia pradinio taško (x, y) koordinatės.

procedure LineRel (dx, dy : integer); Analogiška **LineTo**, tik dx, dy nurodo per kiek pasikeičia pradinio taško (x, y) koordinatės.

3 pratimas. Demonstruojamas linijų piešimas, keičiant atributus.

```

program Pratimas3;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta);
  SetColor( Red );   SetLineStyle( 0, 0, 3 );
{1}  Line ( 100, 100, 300, 100 );

```

```

{2}   Line    ( 100, 300, 300, 300 );
{3}   Line    ( 100, 100, 100, 300 );
{5}   Line    ( 300, 100, 300, 300 );
      SetColor( Green ); SetLineStyle( 3, 0, 3 );
{4}   Line    ( 200, 200, 200, 300 );
{6}   Line    ( 100, 100, 200, 200 );
{7}   Line    ( 200, 200, 300, 100 );
      SetColor( Blue );
      SetLineStyle( 4, $F555, 3 );      { Linijos tipas $F555 }
{8}   Line( 100, 350, 400, 111 ); { Sukurto tipo linija }
repeat until KeyPressed;
CloseGraph;
end.

```

✓ Sugalvokite savo paveikslėlį iš linijų ir jį nubraižykite.

✓ Sukurkite savo linijų tipus ir išbandykite. Pasiruoškite labiausiai jums patikusią kodų rinkinį.

4 pratimas. Šiame pratime yra nurodytoje ekrano vietoje nubrėžiama sinusoidė. Sinusoidės reikšmės vaizduojamos taškais. Nuo jų tankio priklauso gaunamas vaizdas. Programa brėžia tik 3 pusperiodžius. Sinuso paskaičiuota reikšmė yra didinama 60 kartų ir ji tampa dedamo taško Y reikšme, kuri sumuojama su ašinės linijos Y reikšme ekrane. Ekrane yra padedamas taškas ir jis sujungiamas atkarpomis su kiekvienu sinusoidės tašku. Gaunamas “Vėduoklės” paveikslas. Jo konfigūracija priklauso nuo taško padėties sinusoidės atžvilgiu. Kreipinyje “Vėduoklės” spalvą (Yellow) paketę į ekrano fono spalvą (Black), stebėsime tik sinusoidę.

```

program Pratimas4;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Raso(  xp, xg,      { Sinusoidės pradžia ir galas }
                 ya,          { Sinusoidės ašis }
                 xc, yc : integer; { Taško vieta }
                 spt, spl : byte; { Sinusoidės ir linijų spalvos }
                 n : byte); { Taškai dedami kas n laipsnių }
{ Trys sinusoidės pusperiodžiai 180*3 atkarpoje [xp, xg]. }
var x, y, h : integer;      t, th : real;
begin
  SetColor( spl ); SetLineStyle( 1, 0, 1 );
  h := Round(540 / (xg-xp+1)); { Laipsniai atkarpos vienetui }
  h := n div h;                { Taško koordinatės x kitimo žingsnis }
  x := xp; t := 0 ;
  while t <= 540 do begin { Sin reikšme didinama 60 kartų }

```

```

y := Trunc( Sin( t*Pi/180 )*60 );
PutPixel( x, y+ya, spt );           { Sinusoidės taškas }
{ Atkarpa tarp sinusoidės taško ir "Vėduoklės" centro }
Line ( x, y+ya, xc, yc );
t := t+n;    x := x+h;
end;                                end;
{-----}
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta );
  Raso( 0, 600, 200, 150, 350, Red, Yellow, 3 );
  SetColor( Green);
  Line( 0, 200, dx, 200); { Asinė linija }
  repeat until KeyPressed;
  CloseGraph;
end.

```

✓ Stebint “Vėduoklę” ašinė linija nereikalinga. Pašalinkite iš paveikslo.

✓ Pasirinkdami “Vėduoklės” centro koordinates (**xc, yc**) gauname paveikslą, kurio vaizdas labai priklauso nuo pasirinkto centro taško vietos ekrane. Išbandykite.

✓ Keisdami centro koordinates pasirinkite jums įdomią “Vėduoklės” formą. Nupieškite keletą vėduoklių ekrane.

✓ Jeigu nupieštą “Vėduoklę” perpiešime kitoje vietoje (“senoje vietoje naikindami”), tuomet galime gauti įvaizdį, jog ji “mosuoja”. Žinoma, tai priklauso nuo pasirinktų koordinačių, spalvų, perpiešimo greičio. Tai padaryti galima taip:

```

for i := 1 to 20 do begin { "Vėduoklė" mosuoja 20 kartų}
  Raso( 10, 550, 200, 450, 250, Black, Black );
{1}
  Raso( 10, 550, 100, 450, 250, Red, Green );
{2}
  Delay( 250 );
  Raso( 10, 550, 100, 450, 250, Black, Black );
{2}
  Raso( 10, 550, 200, 450, 250, Green, Red );
{1}
  Delay( 100 );    end;

```

Čia komentaruose skaičiais pažymėtos “Vėduoklės”: pirma ir antra. Piešiant juodai ekrane “Vėduoklė” valoma.

■ 5 pratimas. Padarius sulanuotos figūros piešimui procedūrą, galima kurti ekrane paveikslus, dėlioiant figūras simetriškai, susikertančias ar

persidengiančias. Jeigu figūros gabaritai valdomi per parametrus, tuomet galima piešti skirtingo dydžio figūras ir iš jų komponuoti norimus vaizdus. Panaudodami pauzės procedūrą `Delay`, galime gauti dinamiškumo įvaizdį. Rašant tokias procedūras labai svarbu nepamiršti suderinti skaitmenines reikšmes su ekrane leistinomis.

```

program Pratimas5;
  uses Crt, Graph, Grafika;
  const Vieta = 'C:\programs\tp7\bgi';
        Greitis = 800;
{-----}
procedure Remas( x1, y1, x2, y2 : integer ;
                 spalva, storis : byte );
begin
  SetColor( spalva ); SetLineStyle( 0, 0, storis );
  MoveTo( x1, y1 );
  LineTo( x2, y1 ); Delay( Greitis );
  LineTo( x2, y2 ); Delay( Greitis );
  LineTo( x1, y2 ); Delay( Greitis );
  LineTo( x1, y1 ); Delay( Greitis );
end;
{-----}
}
procedure Figura (x1, y1, a, b : integer;
                  { x1, y1 - vieta ekrane; a - plotis; b - aukštis }
                  spr, sp1, sp2 : byte; { Linių spalvos:
                  spr - rėmo; sp1 - horizontaliu ir sp2 - vertikalinių }
                  z : integer; { Linių tankis }
                  lr, ls : byte; { Storis rėmo ir kitų linių }
                  taip : boolean);
  { Taip = True, jeigu norime nubrezti rema, kitaip - False }
var x, y, xc, yc : integer;
begin
  xc := x1 + a div 2;          { Figūros centro koordinatės }
  yc := y1 + b div 2;
  SetColor( sp1 ); SetLineStyle( 0, 0, ls );
  x := x1;                     { Horizontali kryptis }
  while x <= x1+a do begin
    Line( xc, yc, x, y1 ); Delay( Greitis );
    Line( xc, yc, x, y1+b );
    x := x+z; end;
  SetColor( sp2 ); SetLineStyle( 0, 0, ls );
  y := y1;                     { Vertikali kryptis }
  while y < y1+b do begin
    Line( xc, yc, x1, y ); Delay( Greitis );
    Line( xc, yc, x1+a, y );

```

```

    y := y+z;                end;
    if taip then Remas( x1, y1, x1+a, y1+b, spr, lr );
end;
{-----}
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta );
  Figura( 10, 10, 100, 200, Blue, Green, Red,
          10, 3, 1, true );
  Figura( 120, 120, 300, 200, Blue, Green, Red,
          10, 3, 1, true );
  Figura( 440, 10, 100, 200, Blue, Green, Red,
          10, 3, 1, true );
  Figura( 230, 150, 70, 90, Blue, Yellow, Yellow,
          10, 3, 1, false
);
  Figura( 170, 10, 200, 90, Blue, Yellow, Yellow,
          10, 3, 1, false
);
  repeat until KeyPressed;
  CloseGraph;
end.

```

✓ Sugalvokite figūrą, kurios visus išmatavimus pažymėkite kintamaisiais, priklausomais nuo kelių, vadinamų baziniais. Pavyzdyje baziniais priimti aukštis, plotis ir vieta ekrane. Figūros piešimo procedūra parašoma remiantis tik tų kintamųjų reikšmėmis. Kreipinyje keisdami bazinių kintamųjų reikšmes, ekrane gauname figūras su skirtingais išmatavimais. Jeigu linijų tipą ir storius pažymime kintamaisiais, tuomet galime piešti įvairiaspalves figūras. Paveikslo grožis priklausys nuo figūrų išdėstymo, dydžio ir spalvų. Padarykite procedūrą savo figūros piešimui.

✓ Modifikuokite programą kitokio vaizdo piešimui. Pabandykite truputį pakeisti procedūros Figūra piešinį. Panaudokite savo figūrą.

✓ Naudodami turimas procedūras sukomponuokite ekrane jums gražų paveikslą.

✓ Figūros centro koordinatės perstumdami į kitą vietą gausime nesimetrišką vaizdą. Jis gali sudaryti erdvės įvaizdį. Pabandykite procedūroje **Figura** **xc** ir **yc** padidinti, pavyzdžiui, 35 taškais.

Geometrinės figūros. Figūrų užpildymo raštai

Geometrinių figūrų piešimui yra keletas procedūrų: stačiakampiui, apskritimui, apskritimo lankui, elipsės lankui. Linijų, kurios apibrėžia

figūrą, spalva ir tipas nurodomi anksčiau paaiškintomis procedūromis. Gauta uždaro kontūro figūra gali būti užpildyta nurodomu raštu, kurio spalva pasakoma atskirai.

```
procedure Rectangle ( x1, y1, x2, y2 : integer );
```

Nubrėžia stačiakampį, kurio viršutinio kairio kampo koordinatės (x1, y1), o dešinio apatinio kampo koordinatės (x2, y2).

```
procedure Circle ( x, y : integer; r : word );
```

Nubrėžia apskritimą, kurio centras yra taške (x, y), o spindulys r.

```
procedure Arc( x, y : integer; k1, k2, r : word );
```

Nubrėžia apskritmo lanką, kurio centras yra taške (x, y), spindulys r, k1 - lanko pradžios kampas, k2 - lanko pabaigos kampas. Kampai nurodomi laipsniais pradedant skaičiuoti nuo horizontalios linijos prieš laikrodžio kryptį.

```
Procedure Ellipse( x, y : integer;
```

```
    k1, k2, rx, ry : word );
```

Nubrėžia elipsės lanką, kurio pradžios kampas yra k1, o pabaigos- k2. Figūros centras yra taške (x, y). Spindulys horizontalia kryptimi yra rx, o vertikalio- ry.

☞ Figūros nubrėžiamos naudojant aktyvias linijų spalvas ir tipus. Nubrėžtą linijomis uždara kontūrą galima užpildyti tam tikros spalvos raštu.

```
procedure SetFillStyle ( raštas, spalva : word );
```

Nurodome kokį raštą ir kokią spalvą naudosime. Raštas nurodomas vienas iš esančių dvylikos užrašant jo numerį: 0-11. Jeigu norime naudoti savo sukurta raštą, tuomet nurodome tipą 12.

```
procedure FloodFill ( x, y : integer;
```

```
    KontūroSpalva : word );
```

Nurodome bet kokį tašką figūros viduje (x, y) ir figūros kontūro spalvą. Bus užpildomas visas plotas nuo nurodyto taško iki nurodytos spalvos linijos (kontūro).

☝ Jeigu figūra bus atvira, tuomet bus užpildytas plotas ir išorėje. Jeigu taškas bus uždaro figūros išorėje, tuomet raštu bus padengta išorinė

figūros dalis neliečiant vidaus.

6 pratimas. Demonstruojamos visos figūros.

```
program Pratimas6;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta );
  SetBkColor( LightGray );    ClearDevice;
  SetColor( red );    SetLineStyle( 0, 0, 1 );
  Rectangle( 100, 100, 150, 150 );    { Stačiakampis }
  Circle( 300, 100, 50 );    { Apskritimas }
  Arc( 100, 250, 50, 340, 50 );    { Lankas }
  Ellipse( 300, 250, 50, 340, 50, 30 ); { Elipsė }
  Ellipse( 450, 250, 0, 360, 50, 30 ); { Elipsės lankas }
  Circle( 450, 100, 20 ); { Apskritimas, vidus raudonas }
  SetFillStyle( 1, Red );    FloodFill( 450, 100, Red );
  repeat until KeyPressed;
  CloseGraph;
end.
```

✓ Keiskite spalvas ir figūrų parametrus. Sukurkite paveikslėlį, pavyzdžiui Senį Besmegenį.

7 pratimas. Pasinaudodami geometrijos žiniomis nupiešiame ratą, kurį sudarys du skrituliai (vidinis ir išorinis) ir stipinai tarp jų. Ratui piešti padaroma procedūra, kuriai išeities duomenys yra skritulių spinduliai, rato centro koordinatės ekrane, spalvos. Keičian tuos dydžius, ekrane galima nupiešti skirtingo dydžio ratus skirtingose vietose.

```
program Pratimas7;
uses Crt, Graph, Grafika;
const Greitis = 2000;
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Ratelis( xc, yc,    { Rato centro koordinatės }
                  { Vidinio ir išorinio apskritimų spinduliai }
                  r1, r2 : integer;
                  sp1, { Vidinio ir išorinio apskritimų spalvos }
                  sp2, { Rato stipinų spalva }
                  spv : byte ); { Rato centro spalva }
var x1, y1, x2, y2, i : integer; t1, t2 : real;
begin
  SetColor( sp1 );    SetLineStyle( 0, 0, 3 );
  Circle( xc, yc, r1 );    Circle( xc, yc, r2 );
```

```

SetFillStyle( 1, spv ); FloodFill( xc, yc, spl );
SetFillStyle( 1, Blue );
FloodFill( xc+r1+1, yc+r1+1, spl );
SetColor( sp2 ); SetLineStyle( 0, 0, 3 );
i:= 0; { Stipinai piešiami kas i - laipsnių. }
while i <= 90 do begin
  { t1, t2 - apskritimo taško koordinatės, kai r = 1. }
  t1 := Sin( i* Pi/180 ); t2 := Cos( i* Pi/180 );
  { Randamos taškų koordinatės ant apskritimų r1 ir r2 }
  x1 := Trunc( r1 * t2 ); x2 := Trunc( r2 * t2 );
  y1 := Trunc( r1 * t1 ); y2 := Trunc( r2 * t1 );
  { Nupiešiami keturi simetriški stipinai }
  Line( xc-x1, yc-y1, xc-x2, yc-y2 );
  Line( xc+x1, yc+y1, xc+x2, yc+y2 );
  Line( xc-x1, yc+y1, xc-x2, yc+y2 );
  Line( xc+x1, yc-y1, xc+x2, yc-y2 );
  i := i+20;
end;
{ Storinamos apskritimų linijos: stipinų galų taškai išnyksta }
SetColor( spl ); SetLineStyle( 0, 0, 3 );
Circle( xc, yc, r1-1 ); Circle( xc, yc, r2-1 );
Circle( xc, yc, r1+2 ); Circle( xc, yc, r2+2 );
end; {-----}
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta );
  SetBkColor( Blue ); ClearDevice;
  Ratelis( 150, 300, 30, 100, Red, Green, Yellow );
  Delay( Greitis );
  Ratelis( 150, 100, 20, 60, Green, Red, Yellow );
  Delay( Greitis );
  Ratelis( 325, 200, 10, 30, Green, Red, Yellow );
  Delay( Greitis );
  Ratelis( 500, 300, 30, 100, Red, Green, Yellow );
  Delay( Greitis );
  Ratelis( 500, 100, 20, 60, Green, Red, Yellow );
  Delay( Greitis );
  Line( 0, 0, dx, 0 ); Line( 0, dy, dx, dy );
  Delay( Greitis );
  Line( 0, 0, 0, dy ); Line( dx, 0, dx, dy );
  Delay( Greitis );
  Line( 3, 3, dx-3, 3 ); Line( 3, dy-3, dx-3, dy-3 );
  Delay( Greitis );
  Line( 3, 3, 3, dy-3 ); Line( dx-3, 3, dx-3, dy-3 );
  Delay( Greitis );
repeat until KeyPressed; CloseGraph;

```

end.

✓ Pasinaudodami ratelio piešimo procedūra, ekrane nupieškite tris eilutes ratelių. Kiekvienoje eilutėje po keturis ratelius skirtingos spalvos ir raštų.

✓ Nupieškite stilizuotus akinius.

8 pratimas. Programoje **Pratimas8** procedūra **Masina** piešia ekrano nurodytoje vietoje stilizuotą mašinėlę. Jos dydis nurodomas parametrais. Mašinos sudedamosioms dalims piešti padarytos atskiros paprogramės. Mašiną aprašantis stačiakampis sudalinamas į 20 langelių, kurių koordinatės surašomos į masyvus **x** ir **y** (paprogramė **Koordinates**). Tame stačiakampyje yra piešiama mašina. Kitos paprogramės piešia mašinos atitinkamą dalį pagal paskaičiuotas koordinatas. Pagrindinėje programoje parodyta, kaip ekrane gauname keletą skirtingo dydžio mašinėlių.

```
program Pratimas8;
uses Crt, Graph, Grafika;
const n = 21;
    Vieta = 'C:\programs\tp7\bgi';
type mas = array [ 1..n ] of integer;
{-----}
procedure Koordinates(
    var dx, dy : integer ; { Koordinačių pokyčiai }
    var x, y : mas;        { Koordinačių masyvai }
    xp, yp, a, b : integer ); { Vieta ekrane ir gabaritai }
var i : integer;
begin
    dx := a div 20;      dy := b div 20;
    x[ 1 ] := xp;        y[ 1 ] := yp;
    for i := 2 to n do begin
        x[i] := x[ i-1 ] + dx;    y[i] := y[ i-1 ] + dy;
    end;
{-----}
procedure Remas( x, y : mas; dx, dy : integer );
var i : integer;        { Piešiamas mašinos rėmas }
begin
    SetColor( red );    SetLineStyle( 0, 0, 1 );
    Line( x[ 1 ], y[ 1 ], x[ 1 ], y[ 14 ] );
    Line( x[ 1 ], y[ 1 ], x[ 17 ], y[ 1 ] );
    Line( x[ 17 ], y[ 1 ], x[ 17 ], y[ 14 ] );
    Line( x[ 1 ], y[ 14 ], x[ 2 ], y[ 14 ] );
    Line( x[ 6 ], y[ 14 ], x[ 12 ], y[ 14 ] );
    Line( x[ 16 ], y[ 14 ], x[ 17 ], y[ 14 ] );
```

```

Rectangle( x[ 17], y[ 8], x[ 21], y[ 14] );
Arc ( x[ 4], y[ 14], 0, 180, 2*dx);
Arc ( x[ 14], y[ 14], 0, 180, 2*dx);
for   i := 17   to   21   do
    Line( x[ 1], y[ i], x[ 21], y[ i] );
end;
{-----}
procedure Ratas ( x, y, dx, dy : integer );
begin
    { Rato piešimas }
    SetColor( Red );    SetLineStyle( 0, 0, 1 );
    Circle( x, y, dx ); Circle( x, y, dx+2 );
    SetColor( Green );    SetLineStyle( 0, 0, 1 );
    Circle( x, y, dx div 2 ); Circle( x, y, dx div 3 );
end;
{-----}
procedure Langai( x, y : mas; dx, dy : integer );
begin
    { Langu piešimas }
    SetColor( Red );    SetLineStyle( 0, 0, 3 );
    Rectangle( x[ 2], y[ 3], x[ 4], y[ 6] );
    Rectangle( x[ 5], y[ 3], x[ 7], y[ 6] );
    Rectangle( x[ 8], y[ 3], x[ 12], y[ 14] );
    Rectangle( x[ 14], y[ 3], x[ 16], y[ 6] );
    Rectangle( x[ 9], y[ 4], x[ 11], y[ 9] );
end;
{-----}
procedure Priekis ( x, y : mas; dx, dy : integer );
begin
    { Mašinos variklio dalis }
    SetColor( red );    SetLineStyle( 0, 0, 3 );
    Line ( x[ 18], y[ 9], x[ 18], y[ 13] );
    Line ( x[ 19], y[ 9], x[ 19], y[ 13] );
    Line ( x[ 20], y[ 9], x[ 20], y[ 13] );
    SetColor( red ); SetLineStyle( 0, 0, 1 );
    Ellipse( x[ 21], y[ 10], 0, 360, dx div 4, dy );
end;
{-----}
procedure Masina( xp, yp, a, p : integer;
    spalva, kuris : word );
{ xp, yp -mašina aprašančio stačiakampio kairio viršutinio
    kampo vieta ekrane.}
{ a, b - mašiną aprašančio stačiakampio aukštis ir plotis. }
{ spalva - mašinos spalva. }
{ kuris - langas (1..3), kuriame bus "šviesa". }
var   x, y : mas;    ax, ay : integer;
begin
    Koordinates( ax, ay, x, y, xp, yp, a, p );
    Remas ( x, y, ax, ay );

```

```

Ratas ( x[4], y[14], ax, ay );
Ratas( x[14], y[14], ax, ay );
Langai( x, y, ax, ay );
Priekis( x, y, ax, ay );
SetFillStyle( 1, spalva ); FloodFill( xp+1, yp+1, red );
SetFillStyle( 1, yellow );
case kuris of { "Šviesa" lange }
1: FloodFill( x[15], y[4], red );
2: FloodFill( x[ 6], y[4], red );
3: FloodFill( x[ 3], y[4], red );
end; end;
{-----}
var dx, dy : integer;
begin
  Ekranas( dx, dy, Vieta);
  SetBkColor( LightGray ); ClearDevice;
  Masina( 100, 100, 200, 100, Green, 2 );
  Masina( 350, 100, 200, 100, Blue, 1 );
  Masina( 100, 250, 100, 50, Magenta, 3 );
  Masina( 350, 250, 100, 50, LightGray, 0 );
  Masina( 300, 300, 50, 20, LightGray, 0 );
  Masina( 50, 300, 200, 100, Blue, 1 );
  repeat until KeyPressed; CloseGraph;
end.

```

✓ Papildykite procedūrą Masina trimis naujais sakiniais (pabaigoje). Tai bus “lempos”. Papildykite paveikslą kitais atributais, pavyzdžiui, švyturėliu.

```

SetColor ( Yellow );
PieSlice ( x[21]+3, y[10], 0, 60, 50 );
PieSlice ( x[21]+3, y[10], 340, 360, 50 );

```

✓ Pagrindinę programą pakeiskite nurodytais sakiniais. Gausime “judėjimo” efektą. Mašina slinks ekranu trūkčiodama, nes perpiešimo greitis yra nedidelis.

```

Ekranas( dx, dy); SetBkColor( LightGray ); ClearDevice;
i := 100;
repeat
  SetFillStyle( 1, LightGray );
  Bar( i-5, 100, 700, 200 ); { Valymas }
  Masina( i, 100, 200, 100, Green, 2 ); { Piešimas }
  Delay( 50 );
  i := i+5;
until ( i > 300 );
repeat until KeyPressed; CloseGraph;

```

Figūros užpildymas savo sukurtu raštu

Norint savo raštu užpildyti figūrą, reikia sukurti raštą. Tam naudojama:

```
procedure SetFillPattern
```

```
( savo_raštas : FillPatternType;    spalva : word );
```

Ši procedūra daro tą patį, ką ir **SetFillStyle**. Norint savo raštu užpildyti figūrą, pakanka tik su šia procedūra nurodyti pageidavimus. Procedūrą **SetFillStyle** galime naudoti su užpildymo šifru **12**, jeigu buvo anksčiau panaudota procedūra **SetFillPattern** ir mes vėl norime tuo savo sukurtu raštu pasinaudoti. Jeigu savo raštas nebuvo sukurtas, tuomet naudoti rašto kodą **12** negalima. Savo raštą galime sukurti taip. Naudojame duomenų tipą (bibliotekoje Graph) :

```
type FillPatternType = array[ 1..8 ] of byte;
```

Rašto paveikslą galime aprašyti kaip tipizuotą konstantą, pvz.:

```
const    savo_rastas : FillPatternType =  
        ( $AA, $55, $AA, $55, $AA, $55, $AA, $55 );
```

Tai stačiakampis 8x8 taškų plotas, kur bito reikšmė **1** rodo, kad bus piešiamas taškas, o reikšmė **0**, kad nebus piešiamas taškas. Rašte surašomos bitų reikšmės kaip baitai su šešiolyktainio skaičiaus reikšme, pavyzdžiui:

		1	2	3	4	5	
		6	7	8			
1		1	0	1	0	1	\$AA
		0	1	0			
2		0	1	0	1	0	\$55
		1	0	1			
3		1	0	1	0	1	\$AA
		0	1	0			
4		0	1	0	1	0	\$55
		1	0	1			
5		1	0	1	0	1	\$AA
		0	1	0			
6		0	1	0	1	0	\$55
		1	0	1			
7		1	0	1	0	1	\$AA
		0	1	0			
8		0	1	0	1	0	\$55
		1	0	1			
		baitas					

		1	2	3	4	5	
		6	7	8			
1		1	1	1	1	0	\$F0
		0	0	0			
2		0	1	1	1	1	\$78
		0	0	0			
3		0	0	1	1	1	\$3C
		1	0	0			
4		0	0	0	1	1	\$1E
		1	1	0			
5		0	0	0	1	1	\$1E
		1	1	0			
6		0	0	1	1	1	\$3C
		1	0	0			
7		0	1	1	1	1	\$78
		0	0	0			
8		1	1	1	1	0	\$F0
		0	0	0			
		baitas					

✧ Išbandykite sekančias dvi programas su įvairiais užpildymo raštais. Papildykite programas jums reikalingais sakiniais, kuriais jūs galėtumėte patikrinti savo mintis.

✧ Sukurkite savo keletą užpildymo raštų ir juos išbandykite.

9 pratimas. Pieðiami apskritimai su visais užpildymo raðtais ir spalvomis.

```
program Pratimas9;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Uzpildymai( SpLinija : byte); { Kontûro spalva}
    { Apskritimai su visais galimais užpildymo raštais }
var i, j, c, x1, x2, y1, y2, x, y : integer;
begin
    SetColor( SpLinija );
    y := 0; c := 5;
    x1 := 40*c;      x2 := x1 + 42 * c;
    y1 := 20*c;      y2 := y1 +150 + c;
    for j :=1 to 11 do begin { j - užpildymo raštai }
        y := y+30;
        for i:= 0 to 15 do begin { i - spalvos }
            Circle( 20+32*i, y, 15 );
            SetFillStyle( j, i );
            FloodFill( 20+32*i, y, SpLinija );
        end;
    end; end;
{----- Figûra -----}
procedure Figura( x1, y1, { Staèiakampio vieta }
    r1, f1, { Staèiakampio užpildymo raðtas ir spalva}
    r2, f2, { Apskritimo užpildymo raðtas ir spalva }
    linija : integer ); { Kontûro spalva }
var x, y : integer;
begin
    SetColor( linija ); SetFillStyle( r1, f1 );
    Rectangle( X1, Y1, X1+100, Y1+100 );
    FloodFill( x1+1, y1+1, linija );
    x := x1+50; y := y1+50;
    SetFillstyle( r2, f2 );
    Circle( x, y, 20 ); FloodFill( x, y, linija );
end;
{-----}
var dx, dy : integer;
begin
```

```

Ekranas( dx, dy, Vieta);
SetBkColor( LightGray );  ClearDevice;
Uzpildymai( 4 );
Figura( 30, 350, 1, Green , 4, Red, Blue );
Figura( 200, 350, 2, Brown , 5, Cyan, Blue );
Figura( 400, 350, 3, Magenta, 6, Yellow, Blue );
ReadLn ;  CloseGraph;
end.

```

10 pratimas. Demonstruojamas raštų panaudojimas.

```

program Pratimas10;           { Užpildymo raštų iliustracija. }
uses Crt, Graph, Grafika;
const  Vieta = 'C:\programs\tp7\bgi';
       savo : FillPatternType =      { Raštas }
       ( $AA, $33, $AA, $33, $33, $33, $AA, $33
);
{----- Figūra -----}
procedure Figura( x1, y1, { Stačiakampio vieta }
                 r1, f1, { Stačiakampio užpildymo raštas ir spalva }
                 r2, f2, { Apskritimo užpildymo raštas ir spalva }
                 linija : integer ); { Konturo spalva }
var    x, y : integer;           { Panaudojimo
pavyzdys.}
begin
    SetColor( linija );      SetFillStyle( r1, f1 );
    Rectangle( X1, Y1, X1+100, Y1+100 );
    FloodFill( x1+1, y1+1, linija );
    x := x1+50;              y := y1+50;
    SetFillstyle( r2, f2 );
    Circle( x, y, 20 );      FloodFill( x, y, linija );
end; {-----}
var dx, dy, i : integer;
begin
    Ekranas( dx, dy, Vieta);
    SetBkColor( LightGray );  ClearDevice;
    for i := 1 to 4 do begin
        Figura( 30, 10+110*(i-1), 1, Green +(i-1), 4,
                Red +(i-1), Blue );
        Figura( 200, 10+110*(i-1), 2, Brown +(i-1), 5,
                Cyan +(i-1), Blue );
        Figura( 350, 10+110*(i-1), 3, Magenta+(i-1), 6,
                Yellow +(i-1), Blue );
    end; { Vidurinės 4-ios figūros apibrėžiamos stačiakampiu }
    SetColor( Blue ); SetFillStyle( 10, Green );
    Rectangle( 180, 5, 320, 450 );      FloodFill( 181, 6,
Blue );

```

```

    { Stačiakampis, kuris užpildomas savo sukurtu raštu      }
    SetColor( Red );      SetFillPattern( savo, Blue );
    Rectangle( 500, 100, 600, 300 );
    FloodFill( 501, 101, Red );
    ReadLn ;   CloseGraph;
end.

```

Pilnavidurės figūros

Stačiakampis piešiamas dviem būdais.

procedure Bar(x1, y1, x2, y2 : integer); Stačiakampis be ribojančių linijų užpildytas aktyvia spalva.

procedure Bar3d(x1, y1, x2, y2 : integer; z : word; virsus : boolean); Piešiamas gretasienis, kurio trečia koordinatė yra z (45^0 brėžiama linija). Jeigu kintamojo **virsus** reikšmė yra **False**, tuomet gretasienio viršaus stačiakampis nebus nubrėžtas, jeigu **True**, tuomet bus nubrėžtas. Priekinė figūros dalis bus užpildyta (pilnavidurė), o kitos dalys tuščiavidurės. Linijų spalvas nurodome su **SetColor** ir formą su **SetLineStyle**.

11 pratimas. Piešiama šachmatų lenta. Vieną lentos langelį aprašome įrašo tipo struktūra, kur du laukai skirti langelio koordinatėms ekrane saugoti (netikslinga kiekvieną kartą jas skaičiuoti), trečias langelio spalvai, o ketvirtas užimtumui (laisvas ar ne: jame yra šachmatų figūra, ir kokia). Procedūra **Lenta** nupiešia nurodytoje ekrano vietoje nurodyto dydžio lentą ir surašo langelių koordinates į įrašų masyvą. Langelis piešiamas kaip pilnaviduris stačiakampis. Procedūra **Saske** nupiešia nurodyto langelio centre nurodytos spalvos skritulį. Procedūra **SudedaSaskes** piešia šakių partijos pradinę padėtį: keturias eilutes su skirtingų žaidėjų spalvų skrituliais-šaškėmis. Programa demonstruoja kelias skirtingo dydžio lentas ekrane.

```

program Pratimas11;
uses  Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
type Langelis = record
    x, y,      { Lentos langelio koordinatės }
    s, u      : integer; { s - šaškės spalva, u - užimtumas }
end;
Mas          = array [ 1..8 ] of Langelis;
TLenta       = array [ 'A'..'H' ] of Mas;
{-----}
{ Lentos piesimas ir langeliu koordinacių surasymas i

```

```

masyva  A      }
procedure Lenta( var A : TLenta;
                 x1, y1,           { Lentos vieta ekrane }
                 d,                { Langelio kraštinės ilgis }
                 s1, s2 : integer ); { Langelių spalvos }
var  xx, yy, i, t : integer;
      j : char;
      sp : byte; { Aktyvaus (piešiamo) langelio spalva }
begin
  sp := s1;  t := 1;                { t - eilutės numeris }
  for j := 'A' to 'H' do begin
    if sp = s1 then sp := s2 { Langelio spalva s1 arba s2 }
    else sp := s1;
    yy := y1 + d*( t-1 );           { Eilutės koordinatė (y) }
    for i := 1 to 8 do begin { Eilutės langelių piešimas }
      xx := x1 + d*( i-1 );         { Langelio koordinatė (x) }
      SetFillStyle( 1, sp );      Bar( xx, yy, xx+d, yy+d );
      with A[ j,i ] do { Rašomi langelio duomenys į masyvą }
        begin x := xx;  y := yy;  s := sp;  u :=0; end;
      if sp = s1 then sp := s2 { Keičiama langelio spalva }
      else sp := s1;
    end;
    t := t+1;                      { Kita lentos eilutė }
  end; end;
{-----}
procedure Saske( var Vieta : Langelis; { Lentos langelis }
                 spalva : byte;        { Saškės spalva }
                 d : byte ); { Langelio kraštinės ilgis }
var  cx, cy, r : integer;
begin
  SetFillStyle( 1, spalva );      SetColor( Black );
  with Vieta do begin
    cx := x + d div 2;  cy := y + d div 2;
    r := d div 2 - 6;   s := spalva;  u := 1;
    Circle( cx, cy, r ); FloodFill( cx, cy, Black );
  end; end;
{-----}
procedure SudedaSaskes( var A : TLenta;  d : byte );
var  i : integer;                { Žaidimo pradžia }
begin
  for i := 1 to 4 do Saske( A[ 'A', 2*i ], Red, d
);
  for i := 1 to 4 do Saske( A[ 'B', 2*i-1 ], Red, d
);
  for i := 1 to 4 do Saske( A[ 'G', 2*i ], Magenta, d
);

```

```

    for i := 1 to 4 do  Sasse( A[ 'H', 2*i-1 ], Magenta, d
);
end; {-----}
var  dx,    dy  : integer;
      A : TLenta;           {  Lenta  }
      i : integer;
begin
  Ekranas( dx, dy, Vieta );
  Lenta( A, 10, 10, 40, Green, White );
                                     SudedaSasses( A, 40 );
  Lenta( A, 350, 10, 20, Green, White );
                                     SudedaSasses( A, 20 );
  Lenta( A, 400, 200, 25, Green, White );
                                     SudedaSasses( A, 25 );
  ReadLn ;    CloseGraph;
end.

```

✓ Perrašykite pagrindinę programą taip, kad ekrane būtų tik viena lenta su šaškėmis.

✓ Parašykite paprogramę vienos šakės perkėlimui į laisvą langelį.

✓ Parašykite veiksmus, skirtus žaidėjo apklausai, kurią šaškę kur perkelti.

✓ Parašykite programą, kuri organizuotų žaidimą: dviejų žaidėjų pakaitomis veiksmus. Numatykite apklausos duomenų kontrolę.

12 pratimas. Piešdami ir perpiešdami figūras galime sudaryti judėjimo iliuziją. Tai mes pabandėme su mašina. Įdomiau, kuomet galime valdyti judantį objektą. Tai daro programa **Pratimas12**. Čia pagal žmogaus norus spalvotas skritulys juda ekrane valdomas klavišais ←↑→↓. Tiksliau, jis perpiešiamas naujoje vietoje, kuri nutolusi nuo ankstesnės per 5 taškus. Senojoje vietoje skritulio paveikslas nevalomas, todėl lieka judėjimo pėdsakas. Veiksmai nutraukiami paspaudus klavišą Esc. Skaitytojui paliekama mįslė: kodėl skritulys sustoja, jeigu paspaudžiame ne valdymo klavišą (bet kokį kitą, išskyrus pabaigos).

```

program Pratimas12; { Burbuliukas: Darbo pabaiga  Ecs      }
uses    Crt,    Graph,  Grafika;
const  Vieta = 'C:\programs\tp7\bgi';
        Greitis = 600;
{-----}
procedure  Burbulas(      xc,    yc,    r :    integer );
var    i : integer;
begin
                                     { Raudonas skritulys }

```

```

SetFillStyle( 1, Red );          SetColor( Red );
PieSlice( xc, yc, 0, 30, r );
PieSlice( xc, yc, 360, 360, r );
                                { Nedidelė akis }
SetFillStyle( 1, Brown );        SetColor( Blue );
PieSlice( xc+r div 6, yc-r+ r div 3 , 0, 360, r div 6
);
                                { Pašalinamas skritulio segmentas }
SetFillStyle( 1, Black );        SetColor( Black );
PieSlice( xc, yc, 0, 30, r );
PieSlice( xc, yc, 330, 360, r );
end; {-----}
procedure Klavisas( var sx, sy, b : integer );
var s : char; { Judėjimo krypties koeficientai sx ir sy }
begin
    { Esc - pabaigos požymis (b) }
    b := 0;  sx := 0 ;    sy := 0;
    while ( b=0 ) and ( sx=0 ) and ( sy=0 ) do begin
        s := ReadKey;
        if s = #27 then b := 1;
        if s = #0 then begin
            s := ReadKey;
            if s = #72 then sy := -1;
            if s = #80 then sy := 1;
            if s = #77 then sx := 1;
            if s = #75 then sx := -1;
        end;
    end; end;
{-----}
procedure Judeti( xc, yc, r, dx, dy : integer);
    { sc, yc - pradinis taškas; r - burbulo spindulys; }
    { dx, dy - judesio ribos: [0..dx] ir [0..dy] }
var Out, x, y, zx, zy : integer;
begin
    Burbulas ( xc, yc, r );
    x := xc; y := yc; Out := 0; zx := 0; zy := 0;
    while Out = 0 do begin
        if KeyPressed then Klavisas( zx, zy, Out );
        x := x + 5*zx; y := y + 5*zy;
        Delay( Greitis ); { Jeigu burbulas ekrane, tai piešiamas, }
        if ( x >= 1 ) and ( x <= dx ) and
            ( y >= 1 ) and ( y < dy ) then Burbulas( x, y, r )
        else begin Write( #7 ); { kitaip - jis stovi vietoje. }
            if x < 1 then x := 1;
            if x > dx then x := dx;
            if y < 1 then y := 1;
            if y >= dy then y := dy;
        end;
    end; end;
end; end;
end;

```

```

{-----}
var  dx, dy      : integer;
begin
  Ekranas( dx, dy, Vieta );
  SetBkColor( Green );  ClearDevice;
  Judeti ( 100, 100, 20, dx, dy );
  Judeti ( 100, 100, 50, dx, dy );
  repeat until      KeyPressed;  CloseGraph;
end.

```

✓ Burbulas nelabai panašus į pagranduką: pakeiskite nauju piešinėliu.

✓ Papildykite programą sakiniiais, kuriais būtų valomas senasis burbulo paveikslas, t.y. judesio metu neliktų pėdsako.

Grafiniai langai

Grafiniame, kaip ir tekstiniame ekrane, galima sukurti langus.

```

procedure SetViewport( x1, y1, x2, y2 : integer;
                      riba : boolean );

```

Lango viduje visos koordinatės atskaitomos viršutinio kairio kampo, kurio reikšmės yra (0,0), atžvilgiu. Jeigu kintamojo *riba* reikšmė yra *True*, tuomet leidžiama piešti tik lango viduje. Piešinio dalys, kurios netelpa lange, nupiaunamos. Jeigu kintamojo *riba* reikšmė yra *False*, tuomet piešiama ir už lango ribų. Tokiu atveju koordinatės gali būti neigiamos (lango viršuje ir kairėje pusėje).

Atstatomas ekranas taip:

```

SetViewport( 0, 0, GetMaxX, GetMaxY, True );

```

Tekstas, šriftai

Kompiuterio grafiniame ekrane galima rašyti tekstą. Tam yra tokios procedūros.

```

procedure OutTextXY ( x, y : integer; tekstas : string);
procedure OutText   ( tekstas : string );

```

Pirmoji nurodytą simbolių eilutę **tekstas** parašo ekrane pradedant nurodyta vieta (x, y). Antroji procedūra parašo simbolių eilutę **tekstas** nuo tos vietos ekrane, kurioje yra (nematomas) žymeklis (nuo aktyvaus taško). Aktyvus atskaitos taškas rašant tekstą yra stačiakampio, kuriame bus parašytas tekstas, kairiame viršutiniame taške.

Teksto simbolių spalva nurodoma su

```
procedure SetColor (    spalva    :    word    );
```

Teksto charakteristikos nurodomos su

```
procedure SetTextStyle( šriftas, kryptis, dydis: word);
```

Šriftai:	0 pagrindinis	3 figūrinis	Kryptis:	0 horizontali
	1 pastorintas	4 gotiškas		1 vertikali
	2 plonas			(iš apačios į viršų)

Pagrindinio šrifto dydis yra 8x8 taškų kvadratas. Tie šriftai saugomi matriciniu būdu. Kitų šriftų simboliai yra vektoriniu (atkarpomis) principu saugomi; jų piešimo programos saugomos failuose *.chr. Jeigu šrifto numeris procedūroje nurodomas didesnis už nulį, tuomet ieškoma atitinkamo šrifto programa ir ji talpinama kompiuterio atmintyje. Jeigu failas nesurastas, tuomet naudojamas pagrindinis šriftas. Dydis gali būti nurodomas skaičiumi iš 0..10.

Vektorinio šrifto proporcijas galima keisti procedūra:

```
procedure SetUserCharSize ( Ax, Bx, Ay, By : word );
```

Tuomet vykdomi perskaiciavimai:

$$\text{SimbolioPlotis} = (\text{SimbolioPlotis} * Ax) / Bx;$$

$$\text{SimbolioAukštis} = (\text{SimbolioAukštis} * Ay) / By;$$

13 pratimas. Demonstruojamas teksto rašymas ekrane. Čia parodomi ne visi galimi šriftai ir jų rašymo formos. Skaitytojui paliekame išbandyti jam rūpimus variantus.

```

{ Teksto rasymo grafiniame ekrane iliustracija }
program Pratimas13;
uses    Crt,    Graph,    Grafika;
const   Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Raso( X, Y: integer; Tekstas: string;
    Spalva : word; Sriftas, Kryptis, Dydis : word );
begin
    SetColor( Spalva );
    SetTextStyle( Sriftas, Kryptis, Dydis );
    OutTextXY( X, Y , Tekstas );
end;
{-----}
var dx, dy : integer;
begin
```

```

Ekranas( dx, dy, Vieta );
SetBkColor( Green );          ClearDevice;
Raso( 10, 10, '1-T e s t a s', Red, 0, 0, 0 );
Raso( 150, 10, '2-T e s t a s', Red, 0, 0, 4 );
Raso( 10, 50, '3-T e s t a s', Red, 1, 0, 2 );
Raso( 80, 50, '4-T e s t a s', Red, 1, 0, 8 );
Raso( 10, 150, '5-T e s t a s', Red, 2, 0, 6 );
Raso( 150, 150, '6-T e s t a s', Red, 2, 0, 4 );
Raso( 10, 250, '7-T e s t a s', Red, 3, 0, 2 );
Raso( 80, 250, '8-T e s t a s', Red, 3, 0, 8 );
Raso( 10, 350, '9-T e s t a s', Red, 4, 0, 2 );
Raso( 80, 350, '10-T e s t a s', Red, 4, 0, 8 );
Raso( 250, 150, '11-T e s t a s', Red, 0, 1, 1 );
Raso( 300, 150, '12-T e s t a s', Red, 1, 1, 1 );
Raso( 350, 150, '13-T e s t a s', Red, 2, 1, 6 );
Raso( 400, 150, '14-T e s t a s', Red, 3, 1, 1 );
Raso( 450, 150, '15-T e s t a s', Red, 4, 1, 1 );
ReadLn ;   CloseGraph;
end.

```

✓ Išbandykite įvairias šrifto ir teksto rašymo galimybes.

✓ Nupieškite paprastą figūrą iš linijų ir užrašykite jos pavadinimą bei skaičius, reiškiančius linijų ilgius.

14 pratimas. Demonstruojamas grafikos priemonių panaudojimas. Išbandykite programą. Išbandykite kiekvieną atskirai demonstracinę procedūrą, tuo tikslu kitas apskliausdami komentariniais skliaustais.

```

program Graphdemo;
uses Crt, Graph, Grafika;
type
    A      = array[ 1..2, 1..4 ] of integer;
    zodis  = string[10];
const
    Vieta = 'C:\programs\tp7\bgi';
    K : A = (( 30, 120, 250, 330),
              (290, 120, 600, 330)); { Langų koordinatės }
{-----}
procedure Langas( n: integer; spalva: byte);
begin
    SetfillStyle( 1, spalva ); { Užpildo tipas ir spalva }
    Bar( K[n,1], K[n,2], K[n,3], K[n,4] ); { Lango dažymas }
                                         { Aktyvus langas }
    SetViewPort( K[n,1], K[n,2], K[n,3], K[n,4], TRUE);
end;
{-----}

```

```

procedure Points;          { Atsitiktiniai taskai lange 2 }
  var dx, dy: integer;  r : char;
begin
  Randomize;
  OuttextXY(15,50,'Tęsinys - bet kuris klavišas!');
  dx:= k[2,3]-k[2,1];      { Lango matmenys }
  dy:=k[2,4]- k[2,2];
  repeat                  { Atsitiktiniai taškai }
    PutPixel( Random( dx ), Random( dy ), Random( 16 ));
    Delay(10);              { Pauzė }
  until KeyPressed;
  r := ReadKey;
end;
{-----}
procedure Lines;
type  zodis = string[ 5 ];      { Linijos lange 2 }
var  dx, dy, y : integer;  t : zodis;
begin
  Randomize;  y:= 0;      Setlinestyle( 0, 0, 3 );
  dx := K[2,3] - K[2,1];  dy := K[2,4] - K[2,2];
  while y<= dy do begin
    Setcolor(random(16));
    Line(0,y, dx, y);      { Horizontali linija }
    Delay(100);            y:=y+1;
  end;
  Write(^G);
  end;
{-----}
procedure Lines1; { Tušuoja langą įstrižomis linijomis }
var dx, dy, y, x: integer;
begin
  Randomize;  y:= 0;
  dx := K[2,3] - K[2,1];  dy := K[2,4] - K[2,2];
  SetLineStyle( 0, 0, 3 );  x := dx;
  while y<= dy do begin
    SetColor( Random( 16 ));  Line(0, 0, dx, y);
    Delay( 100 );            y := y+1;
  end;
  while x>=0 do begin
    SetColor( Random( 16 ));  Line( 0, 0, x, dy);
    Delay( 100 );            x := x-1;
  end;
  Write( ^G );
  end;
{-----}
procedure Circles;          { Augantis skritulys }

```

```

var dx, dy, r: integer;
begin
  dx := K[2,3] - K[2,1];  dy := K[2,4] - K[2,2];
  SetColor( LightGray );
  SetLineStyle( 0, 0, 3);  r := 0;
  while r<= dy div 2 do begin
    Circle( dx div 2, dy div 2, r);
    Delay( 100 );          r := r+1;
  end;
  Write(^G);
end;
{-----}
procedure Move;          { Besisukantis skritulys }
var dx, dy, al, d, r: integer;  sim : char;
begin
  dx := K[2,3] - K[2,1];  dy := K[2,4] - K[2,2];
  SetColor( Red );        r := dy div 2;
  SetLineStyle ( 0, 0, 3);  al := 0;      d := 10;
  repeat
    SetFillStyle( 1, Blue);
    PiesLice( dx div 2, dy div 2, al, al+d, r);
    Delay(100);
    SetFillStyle( 1, LightGray);
    PiesLice( dx div 2, dy div 2, al, al+d, r);
    al := al+10;
    if al = 360 then al := 0;
  until KeyPressed;
  Write(^G);      sim := ReadKey;
end;
{-----}
var
  x, y : zodis;
  mx, my : integer;      { Ekrano skiriamoji galia }
begin
  Ekranas( mx, my, Vieta );
                                { Liniju spalva ir tipas }
  SetColor( White ); SetLineStyle( UserBitLn, $111, 3 );

  Rectangle(3, 3, mx-3, my-3);  { Rémelis }

  SetColor( Red );
  SetTextStyle( GothicFont, 0, 6 ); { Šrifto aprašymas }
  SetFillStyle( 1, Green );
  Bar( 10, 10, mx-10, my-10);{ Žalias stačiakampis }
  OutTextXY( 150, 30, 'Grafika Ekrane');

```

```

Langas(1, Blue);           { Kairysis ekrano langas }
SetColor( LightGray ); SetTextStyle( 1, 0, 1 );
OutTextXY( 15, 50, 'Ekrano parametrai:' );

Str( mx, x ); OutTextXY( 50, 80, 'maxX = '+ x );
Str( my, y ); OutTextXY( 50, 110, 'maxY = '+ y );
Str( GetGraphMode, x );
        OutTextXY( 35, 140, 'GraphMode = '+ x );

SetViewport( 0, 0, mx, my, True);
Langas(2, blue);           { Dešinysis ekrano langas }
                                { Demonstracinės procedūros }

Points;
Lines;
Lines1;
Circles;
Move;

Setviewport(0, 0, mx, my, true);{ Aktyvus visas ekranas
}
Settextstyle(3, 0, 2);
Outtextxy(200,400,'Pabaiga - kl. Enter !');
Readln
end.

```

✓ Pakeiskite `Lines1` procedūrą taip, kad linijos būtų iš dešinio viršutinio kampo brėžiamos.

✓ Pakeiskite `Circles` procedūrą taip, kad užaugusį vienos spalvos skritulį keistų augantis kitos spalvos skritulys tol, kol procesas bus nutrauktas bet koku klavišu.

✓ Pakeiskite `Move` procedūrą taip, kad skritulys ekrane suktūsi į priešingą pusę ir nesimatytų skritulio sementų kraštus žyminčių linijų.

9.3. Funkcijų grafikai.

Piešiant ekrane funkcijų grafinius vaizdus, patogiausia vartoti tiesinę jų aproksimaciją, kai ekrane pažymimi ir sujungiami tiesių atkarpomis pasirinkti funkcijos taškai.

Pavyzdžiui, tarkime, kad reikia sudaryti programą, kuri ekrane sudarytų funkcijos $y = x^2$, kai argumento x reikšmės priklauso atkarpai $[0,50]$. Sudarant programą, reikia atkreipti dėmesį į tokias funkcijų grafinio vaizdavimo ekrane savybes:

✧Norint perkelti koordinačių pradžią iš kairiojo viršutinio ekrano (lango) kampo į kitą ekrano tašką (x_0, y_0), reikia vaizduojamos funkcijos formulėje y pakeisti skirtumu $y-y_0$, o x - skirtumu $x-x_0$;

✧Ekrano arba lango kordinatės gali būti tik sveikieji skaičiai, todėl skaičiuojamas funkcijos reikšmės būtina apvalinti;

✧Siekiant vaizdumo, pageidautina įvesti x ir y reikšmių grafinio vaizdavimo mastelius atitinkamai mx ir my ;

✧Norint, kad y ašis ekrane eitų ne į apačią, o į viršų, reikia funkcijos reikšmių skaičiavimo formulę padauginti iš -1 .

Įvertinus šias savybes, vaizduojama funkcija turi būti perrašyta taip:

$$y - y_0 = -\text{Trunc}(my * \text{Sqrt}(x*mx - m_0))$$

Dviejų gretimų taškų, kurių abscisės (x) skiriasi 1, ordinatės turės būti skaičiuojamos taip:

$$y_1 := y_0 - \text{Trunc}(my * \text{Sqrt}(x_1*mx - m_0));$$

$$y_2 := y_0 - \text{Trunc}(my * \text{Sqrt}((x_1+1) * mx - m_0));$$

15 pratimas. Demonstruojamas aptartos funkcijos vaizdas ekrane.

```
program Funkcija;
uses Graph, Grafika;
const
  mx = 5;   my = 10; { Masteliai }
  xp = 170; yp = 100; { Vaizdo lango pradžia }
  k = 300; { Lango dydis }
  m = 50; { Funkcijos reikšmių skaičius }
  x0 = 0; y0 = k; { Koordinačių pradžia lange }
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure AxisX ( mx: integer); { X ašies vaizdas }
var { mx - mastelis }
  i, x : integer; s : string[5];
begin
  SetColor( Black );
  SetLineStyle( 0, 0, 3 ); { Ašies linijos tipas }
  Line (0, k, k, k); { Ašies linija }
  i:= 0;
  SetTextStyle( SmallFont, 0, 3 ); { Žymėjimų šriftas }
  while x< k do begin
```

```

        x := i * mx;                      { Mastelio įvertinimas }
        OutTextXY( x, k-6, '|');          { Ašies padalos      }
        Str(i, s);
        OutTextXY( x, k- 12, s);          { Padalų reikšmės      }
        i := i + 10;
    end;
end;
{-----}
var
    mex, mey : integer;                   { Ekranos dydis taškais }
    x, y1, y2, i: integer;                 { Pagalbiniai kintamieji }
begin
    Ekranas( mex, mey, Vieta );

    SetFillStyle( 1, White);
    Bar( 0, 0, mex, mey);                 { Ekranos dažymas }

    SetColor( Magenta ); SetTextStyle( GothicFont, 0, 4);
    OutTextXY( 250, 30, 'Grafika');        { Ekranos antraštė }

    SetFillStyle( 1, LightGreen );
    Bar( xp, yp, xp+k, yp+k);              { Lango grafikui }
    SetViewport( xp, yp, xp+k, yp+k, true);
    { Lango dydis - k*k taškų, o koordinacių ašys - lango kraštai }
    SetLineStyle( 0, 0, 3);
    SetColor( Red );                       { Grafiko linija }
    { Tiesinė funkcijos grafiko aproksimacija;
      i- tašku numeriai }

    for i:= 0 to m do begin
        x := i * mx - x0;
        y1 := y0 - Trunc( my * Sqrt( x      ) );
        y2 := y0 - Trunc( my * Sqrt( x+mx ) );
        Line ( x, y1, x+mx, y2 );
    end;
    AxisX( 2 );
    { Pranesimas apie programos nutraukimo sąlyga }
    SetViewport( 0, 0, mex, mey, True);
    SetTextStyle( 1, 0, 2 ); SetColor( Magenta );
    OutTextXY( 210, 420, 'Pabaiga - kl. Enter' );
    ReadLn;
end.

```

✓ Pakeiskite programą taip, kad grafinį vaizdą formuotų atskira procedūra **Grafikas**.

✓ Funkcijos grafinio vaizdo langą papildykite baltos spalvos koordinacių ašimis su argumentų ir funkcijos dydžių skalėmis.

✓ Papildykite programą `Funkcija` procedūromis taip, kad funkcijos reikšmių masyvas būtų sudaromas iš tekstiname faile įrašytos funkcijos reikšmių lentelės.

Polinė koordinačių sistema. Dažniausiai vartojama yra dekartinė koordinačių sistema, tačiau daugelio linijų lygtys yra paprastesnės kitose koordinačių sistemose, pavyzdžiui polinėje. Šią sistemą sudaro polius ir pradinę poliuję prasidedančio spindulio - vektoriaus kryptis. Taško padėtis polinėje koordinačių sistemoje aprašoma jo spindulio - vektoriaus moduliui ir šio vektoriaus kampui su pradine kryptimi.

Koordinačių transformavimo iš polinės sistemos į ekrano langą lygtys atrodo taip:

$$x = x_0 + r \cdot \cos(t), \quad y = y_0 - r \cdot \sin(t)$$

Taškas (x_0, y_0) aprašo poliaus vietą aktyviame lange.

Polinėje sistemoje labai paprasta apskritimo lygtis: $r = \text{const}$. Gražios linijos, kurios vadinamos daugialapėmis rožėmis aprašomos lygtimi

$$r = R \cdot (1 - \cos(N \cdot t)).$$

Čia R ir N žymi linijų formą aprašančias konstantas.

16 pratimas. Surinkite ir išbadykite polinės koordinačių sistemos savybių demonstravimo programą `Ji` piešia keturlapę rožę.

```
program Demo;
uses Graph, Crt, Grafika;
const x0 = 150; y0 = 150; { Poliaus koordinatės }
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Langas( x, y, l, d, fonas: integer);
{ Grafinio ekrano langas: (x, y) - lango vieta,
  l - plotis, d - aukstis, fonas - lango fono spalva }
var x1, y1 : integer;
begin
  SetViewport( 0, 0, Getmaxx, Getmaxy, ClipOn );
  x1 := x + l - 1; y1 := y + d - 1;
  SetFillStyle( l, fonas );
  Bar( x, y, x1, y1 );
  SetViewport( x, y, x1, y1, ClipOn);
end;
```

```

{-----}
procedure Vaizdas( xc, yc: integer);{ Funkcijos vaizdas }
var { Gretimų taškų spinduliai ir kampo kitimo žingsnis }
    r1, r2, f: real;
    x1, y1, x2, y2, i: integer;
begin
    f := 2 * pi / 200; { Kampo kitimo žingsnis }
    for i := 0 to 200 do begin
        r1 := 50 *(1 - Cos( 4*i*f)); { 1-jo spindulio ilgis }
        r2 := 50 *(1 - Cos( 4*(i+1)* f));{2-jo spindulio ilgis }
    }
    { Dekartinės taškų koordinatės }
    x1 := Round( x0 + r1 * Cos(i*f) );
    y1 := Round( y0 - r1 * Sin(i*f) );
    x2 := Round( x0 + r2 * Cos((i+1)*f) );
    y2 := Round( y0 - r2 * Sin((i+1)*f) );
    Line( x1, y1, x2, y2 ); { Funkcijos grafikas }
    if i mod 5 =0 then
        Line( xc, yc, x1, y1 );{ Papildomi spinduliai }
    end;
end;
{-----}
procedure Suk( xc, yc: integer); { Spindulių sukimas }
    { Taškas ( xc, yc ) yra spindulių sukimo centras }
var
    r1, r2, f: real;
    x1, y1, x2, y2, i: integer;
    a: char;
begin
    f := 2 * pi / 200; { Kampo kitimo žingsnis }
    i := 0;
    repeat
        r1 := 50 * (1 - Cos(4*i*f) ); { Spindulių ilgiai }
        r2 := 50 * (1 - Cos(4*(i+1)*f));
        { Dekartinės koordinatės }
        x1 := Round( x0 + r1 * Cos(i*f));
        y1 := Round( y0 - r1 * Sin(i*f));
        x2 := Round( x0 + r2 * Cos((i+1)*f));
        y2 := Round( y0 - r2 * Sin((i+1)*f));
        if i mod 5 =0 then begin
            SetColor( White );
            Line( xc, yc, x1, y1 ); { Baltas spindulys }
            Delay(1000); { Pauzė }
            SetColor( Red ); { Spalvos atstatymas }
            Line( xc, yc, x1, y1 );
        end;
    until i = 200;
end;

```

```

                                end;
    i := i+1;                                { Naujo taško indeksas }
until KeyPressed;
a := ReadKey;
end;
{-----}
procedure Info;
begin { Informacija apie polinę koordinačių sistemą }
    SetColor( Red );
    Circle( 150, 150, 3);                    { Polius }
    Circle( 50, 50, 3);                      { Taškas }

    SetFillStyle( 1, Red );                  { Taško ir poliaus dažymas }
    FloodFill ( 151, 151, Red );
    FloodFill ( 50, 51, Red );

    Line ( 150, 150, 50, 50);                { Taško vektorius }
    Line ( 150, 150, 290, 150);              { Pradinė kryptis }
    Arc ( 150, 150, 0, 135, 50); { Kampą žymintis lankas }
}

SetColor( LightGray );                      { Informaciniai užrašai }
    OutTextXY( 130, 160, 'polius');
    OutTextXY( 70, 80, 'r');
    OutTextXY( 160, 90, 't');
    OutTextXY( 30, 30, 'taskas');
    OutTextXY( 40, 205, 'r - spinduys-vektorius');
    OutTextXY( 40, 220,
        't - spindulio kampas su pradine
kryptimi');
end;
{-----}
var mx, my, i : integer;
begin
    Ekranas( mx, my, Vieta );
    SetColor( Yellow ); SetTextStyle( GothicFont, 0, 5);
    OutTextXY( 250, 30, 'Grafika');

    SetTextStyle( 2, 0, 4); SetColor( Brown );
    OutTextXY( 110, 75, 'Polines koordinates');
    OutTextXY( 450, 75, 'Funkcijos grafikas');

    Langas( 20, 100, 300, 300, Blue);

Info;                                { Informacinis langas }

```

```

Langas( 340, 100, 300, 300, blue); { Langas grafikui }

SetLineStyle( 0, 0, 2);
SetColor( Red ); { Linijų parametrai }
Vaizdas( x0, y0 ); { Funkcijos grafikas }
    { Pranešimas apie programos nutraukimo sąlygą }
SetViewport( 0, 0, Getmaxx, Getmaxy, True);
SetTextStyle( 1, 0, 1 ); SetColor( Brown);
OutTextXY( 230, 420, 'Pabaiga - kl. Enter');
ReadLn;
SetViewport( 340, 100, 639, 399, True);
Suk( x0, y0 ); Suk( 0, 0 ); { Spindulių
sukimas }
ReadLn;
end.

```

9.4. Judesio programavimas.

Jau buvo minėta, kad perpiešiant figūrą naujoje vietoje, galima sudaryti judėjimo įvaizdį. Norint gauti natūraliai atrodantį judesį, reikia pakankamai greitai dirbančio kompiuterio ir išraiškos priemonių. Turbo Paskalio priemonėmis, esančiomis **Graph** bibliotekoje, galima programuoti nesudėtingų figūrų judėjimą ekrane, kaip kad parodyta programoje **Zuikis**. Sudėtingesnes figūras ekrane kiekvieną kartą perpiešti netikslinga, nes tam gaišamas laikas. Judesys gaunamas trukčiojantis.

■ 17 pratimas. Programa Zuikis demonstruoja nurodyto dydžio ekrano lange šviesos zuikelio (spalvoto skrituliuko) skritulio judėjimą atsitiktinėmis kryptimis. Judesiui gauti zuikelis senoje vietoje valomas (piešiamas lango fono spalva), o naujoje vietoje piešiamas kita spalva.

```

program Zuikis;
uses Crt, Graph, Grafika;
const lx = 50; ly = 50; { Zuikio narvelio koordinatės }
      hx = 250; hy = 250; { grafiniame ekrane }
      r = 4; greitis = 150;
      Vieta = 'C:\programs\tp7\bgi';
{----- Zuikelis -----}
procedure Zuikelis( x, y : integer; spalva: byte );
begin
    SetFillStyle( 1, spalva ); { Valomas ekrane zuikelis }
    SetColor( spalva); FillEllipse( x, y, r, r );
end;
{-----}

```

```

procedure Narvas( var x, y : integer );
begin
    SetColor( Green );
    Line( lx-2, ly-2, hx+2, ly-2 );
    Line( lx-2, hy+2, hx+2, hy+2 );
    Line( lx-2, ly-2, lx-2, hy+2 );
    Line( hx+2, ly-2, hx+2, hy+2 );
    SetFillStyle( 1, Yellow ); { Ekrane piešiamas zuikio }
    Bar ( lx, ly, hx, hy ); { narvas. }
    x := lx + ( hx - lx ) div 2; { Zuikis tupdomas į }
    y := ly + ( hy - ly ) div 2; { narvo vidurį. }
    Zuikelis( x, y, Green )
end;
{----- Kitas -----}
procedure KitasZingsnis ( var x, y : integer );
var al : integer; { Zuikio žingsnis generuojamas}
begin
    al := Random( 360 );
    x := Trunc( Cos( al / 180 * Pi ) * 10 );
    y := Trunc( Sin( al / 180 * Pi ) * 10 );
end;
{----- Galima -----}
function Galima ( x, y, tx, ty : integer ): boolean;
var k : integer;
begin { Zuikio narvo ribų kontrolė }
    Galima := True;
    for k := x+r+1 to x+r+1+tx do
        if( GetPixel( k, y ) <> Yellow ) then Galima := False;
    for k := x-r-1 downto x-r-1+tx do
        if( GetPixel( k, y ) <> Yellow ) then Galima := False;
    for k := y+r+1 to y+r+1+ty do
        if( GetPixel( x, k ) <> Yellow ) then Galima := False;
    for k := y-r-1 downto y-r-1+ty do
        if( GetPixel( x, k ) <> Yellow ) then Galima := False;
end;
{=====}
var x, y, { Zuikio vieta ekrane }
    tx, ty, { zuikio žingsnis }
    dx, dy : integer;
begin
    ClrScr; GoToXY( 5,5 ); TextColor( Red ); Randomize;
    WriteLn('Darbui pabaigti reikia paspausti bet kuri klavisa ');
    GoToXY( 5,15 );
    TextColor ( Green );
    Write ( ' Darbui pradėti paspauskite');
    TextColor ( Magenta ); Write ( ' "E N T E R" ');

```

```

ReadLn;
  Ekranas( dx, dy, Vieta );
  Narvas( x, y );
repeat
  KitasZingsnis ( tx, ty );
  while Galima ( x, y, tx, ty ) do begin
    Delay( greitis );
    Zuikelis( x, y, Yellow); { Valoma zuikelio vieta }
    x := x+tx; y := y+ty; { Naujos vietos koordinatės }
    Zuikelis( x, y, Green );{ Zuikelis naujoje vietoje }
  end;
until KeyPressed;          { Darbui baigti paspausti }
CloseGraph;                { reikia bet kuri klavisa }
end.

```

✓ Keiskite Zuikelio spindulį ir stebėkite judesio greitį. Pakeiskite narvelio dydį.

Turint jau vieną kartą nupieštą figūrą, tikslinga ją dėti tam tikru dėsnio skirtingose ekrano vietose. Tam reikia figūros vaizdą išsaugoti atskirai kompiuterio atmintyje. Reikalingos paprogramės yra aprašytos 9.2 lentelėje. Veiksmų seka gali būti tokia. Sukuriamas figūros vaizdas ekrane ir funkcijos **ImageSize** pagalba sužinomas dydis baitais. Procedūrą **GetMem** panaudojame reikalingos atminties vietos išskyrimui dinamiėje atmintyje. Procedūra **GetImage** panaudojama figūros vaizdo išsaugojimui išskirtoje tam tikslui atmintyje. Figūros vieną žingsnį galima padaryti tokiais veiksmais : nupiešiama figūra su **XorPut**, parodomas žiūrovui vaizdas (**Delay** procedūra naudojama), atstatomas ekrano vietos vaizdas, perpiešiant figūrą su **XorPut**. Keičiant tam tikru dėsnio figūros koordinatės ekrane, yra gaunamas judesio vaizdas. Programuojant judesį yra įdomiausias figūros piešimo ekrane režimas **XorPut**, nes figūra tampa persišviečianti, po ja esantys ekrano taškai keičia spalvą, tačiau vaizdas išlieka. Antrą kartą toje pačioje vietoje piešiant figūrą tuo pačiu **XorPut** režimu, figūra išnyksta ir ekrano vaizdas atsistato.

9.2. lentelė.

<i>function</i> ImageSize (x1, y1, x2, y2 : <i>integer</i>): <i>word</i> ; Suskaičiuoja stačiakampės ekrano sritys, nurodytos kairio viršutinio kampo (x1,y1) ir dešinio apatinio kampo (x2,y2) koordinatėmis, dydį baitais.
<i>procedure</i> GetImage (x1, y1, x2, y2 : <i>integer</i> ; var Adresas : <i>pointer</i>); Nurodytos koordinatėmis stačiakampės ekrano sritys (kaip ir ImageSize) vaizdą užrašo į atminties sritį, nurodomą rodykle Adresas. Pirmi šeši baitai

skirti išsaugomos srities parametrus užrašyti: plotis ir aukštis.
<i>procedure</i> PutImage (<i>x, y : integer; var Adresas : pointer; Kaip : word</i>); Vaizdas, kuris saugomas dinaminėje atmintyje ir kurios rodyklė Adresas, perkeliamas į ekrano vietą, kurios viršutinio kairio kampo koordinatės (x,y). Parametro Kaip reikšmė nurodo perkėlimo būdą. Galimos reikšmės parodytos 9.3 lentelėje.

9.3 lentelė

Konstanta		Prasmė
CopyPut	0	Pakeičia ekrano sritį anksčiau išsaugota.
XorPut	1	Ekrano taško nauja spalva gaunama panaudojus operaciją Xor taško spalvoms iš buferio ir ekrano.
OrPut	2	Ekrano taško nauja spalva gaunama panaudojus operaciją Or taško spalvoms iš buferio ir ekrano.
AndPut	3	Ekrano taško nauja spalva gaunama panaudojus operaciją And taško spalvoms iš buferio ir ekrano.
NotPut	4	Ekrano taško nauja spalva gaunama invertuojant taško spalvą buferyje.

18 pratimas. Programoje **Saule** yra iliustruojamas atsitiktinis figūros “Saulute” judėjimas ekrane, kuriame kaip fonas yra piešiami įvairiaspalviai skrituliai.

```

program Saule;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
      r = 40; { Saulutės skritulio spindulys }
{-----}
procedure Skrituliai( n : byte; dx, dy : integer);
  var i, spalva, x, y, r : integer;
begin { Ekrane piešiami atsitiktiniu raštu ir }
  Randomize; { spalva skrituliukai (n - jų kiekis) }
  for i := 1 to n do
  begin
    SetFillStyle( Random(11), Random(15) );
    SetColor( Random(11) ); SetLineStyle( 0, 0, 1 );
    x := Random( dx - 30 ) + 20;
    y := Random( dy - 30 ) + 20;
    r := Random( 30 ) + 5;
    FillEllipse( x, y, r, r );
  end;
end;

```

```

end;
{-----}
procedure Saulute( x, y, r : integer);
var t, t1, t2, t3, t4, t5 : word;
begin
  t := r div 4; t1 := r div 5; t2 := r div 7;
  t3 := r div 8; t4 := r div 2;
  t5 := Trunc( r / Sqrt( 2));

  SetColor( Black ); SetFillStyle( 1, Yellow );
  FillEllipse( x, y, r, r ); { Geltonas skritulys }

  SetFillStyle( 1, White ); { Akys }
  FillEllipse( x-t, y-t4, t2, t1 );
  FillEllipse( x+t, y-t4, t2, t1 );
  SetFillStyle( 1, Black );
  FillEllipse( x-t, y-t4+3, t3, t3 );
  FillEllipse( x+t, y-t4+3, t3, t3 );

  Arc( x, y+t-2, 180, 360, r-t ); { Šypsena }
  Arc( x, y-t, 210, 340, r-2 );
  SetFillStyle( 1, Red ); FloodFill( x, y-t+r, Black );

  SetColor( red ); { Nosytė }
  SetLineStyle( 0, 0, 1 ); Arc( x, y+2, 30, 150, t-2 );

  SetColor( Yellow );
  SetLineStyle( 2, 0, 3 ); { Spinduliai }
  Line( x, y+r, x, y+2*r );
  Line( x, y-r, x, y-2*r );
  Line( x-r, y, x-2*r, y );
  Line( x+r, y, x+2*r, y );
  Line( x+t5, y+t5, x+2*r, y+2*t5 );
  Line( x+t5, y-t5, x+2*r, y-2*t5 );
  Line( x-t5, y-t5, x-2*r, y-2*t5 );
  Line( x-t5, y+t5, x-2*r, y+2*t5 );
end;
{-----}
procedure Eiti( var x, y : integer );
var al : integer; { Judesio naujos krypties generavimas }
begin
  repeat al := Random( 360 );
        x := Trunc( Cos( al / 180 * Pi ) * 20 );
        y := Trunc( Sin( al / 180 * Pi ) * 20 );
  until ( x <> 0 ) or ( y <> 0 );
end;

```

```

{*****}
var dx, dy, zx, zy, x, y, k : integer;   S : pointer;
begin
  Ekranas( dx, dy, Vieta );
  x := dx div 2;   y := dy div 2;   k := 2*r;
  GetMem( S,   ImageSize( 0, 0, 2*k, 2*k )); { Atmintis }
      Saulute( x, y, r );
      { Saulutės užrašymas į atmintį.}
  GetImage( x-k, y-k, x+k, y+k, S^ );
      { Saulutės perkėlimas į ekraną }
  PutImage( x-k, y-k, S^, 1 );
  Skrituliai( 15, dx, dy );
  repeat
    Eiti( zx, zy );           { Saulutės judesio žingsniai   }
    Skrituliai( 5, dx, dy );
                                { Judesys iki ekrano krašto }
  while ( x < dx-k ) and ( x > k ) and
    ( y < dy-k ) and ( y > k ) do begin
    PutImage( x-k, y-k, S^, 1 ); Delay( 820 );
    PutImage( x-k, y-k, S^, 1 );           { Saulutės valymas }
    x := x + zx;   y := y + zy;           { Nauja vieta       }
  end;
  x := x - zx;   y := y - zy; { Sugrižimas į ekraną   }
  until KeyPressed;
  CloseGraph;
end.

```

Norint sukurtą ekrane figūrą išsaugoti ilgesniam laikui, tikslinga užrašyti visą jos vaizdą į failą. Paprasčiausias būdas yra eilės seka surašyti visų figūrą sudarančių taškų spalvas. Failo pradžioje reikia užrašyti figūros parametrus, kaip pavyzdžiui, aukštį ir plotį taškų kiekiu. Pavyzdžiui, norint programoje **Saule** išsaugoti “Saulutės” paveikslą, procedūrą **Saulute** reikia truputį pildyti:

```

procedure Saulute( x, y, r : integer);
{ type failas = file of word; Programą papildome.      }
var t, i, j : word;
    F: failas; begin
{“Saulutės” formavimo sakiniai
}
Assign( F,  'Saule.pav' ); Rewrite( F );
  t := 4*r;
  Write( F, t );           { “Saulutės” diametras }
  for i := x-2*r to x+2*r do
    { Surašomos figūros taškų spalvos }
    for j := y-2*r to y+2*r do

```

```

begin t := GetPixel( i,j ); Write( F, t ); end;
Close( F );
end;

```

Šioje arba kitoje programoje galima “Saulutės” perkėlimui iš failo į norimą ekrano vietą pasinaudoti, pavyzdžiui, procedūra **Vaizdas**.

```

procedure Vaizdas( x, y : integer; FailoVardas :
string);
{ Figūros kairio viršutinio kampo koordinatės (x,y) ekrane }
var t, n, i, j : word;          { n – Figūros dydis }

F: failas;                      { Programą papildome tipu:
                                type failas = file of word; }

begin
Assign( F, FailoVardas); Reset( F );
Read( F, n );
for i := x to x+n do
    { Skaitomos figūros taškų spalvos }
    for j := y to y+n do
        begin Read( F, t); PutPixel( i, j, t ); end;
    Close( F );
end;

```

Šis vaizdo išsaugojimo būdas yra paprastas, tačiau lėtas. Greitesnis yra dinaminėje atmintyje saugomos figūros vaizdo užrašymas į failą, o po to skaitymas į dinaminę atmintį bet kurioje tam tikslui skirtoje programoje. Pavyzdžiui, “Saulutės” paveikslą galime faile išsaugoti, jeigu pagrindinėje programoje po sakinio `GetImage(x-r, y-r, x+r, y+r, S^);` kuriuo “saulutės” vaizdas užrašomas į dinaminę atmintį, užrašysime tam skirtus veiksmus:

```

Assign( F, 'Saulė.pav'); { var F : file; }
Rewrite( F, 1 );
t := ImageSize( 0, 0, 4*r, 4*r );
BlockWrite( F, S^, t );
Close( F );

```

Norint panaudoti taip išsaugotą vaizdą, reikia veiksmus atlikti atvirkščia seka, pavyzdžiui taip, kaip parodyta procedūroje **VaizdasFaile**.

```

procedure VaizdasFaile( var S : pointer;
                        FailoVardas : string );
var F : file; n : word;
begin
Assign( F, FailoVardas); Reset( F, 1 );
n := FileSize( F );

```

```

    GetMem( S, n );      BlockRead( F, S^, n );
    Close( F );
end;

```

Sudėtingesnes figūras perpiešti ekrane reikia laiko. Dažnai yra nepageidautina stebėti piešimo procesą. Žymiai patogiau yra parodyti ekrane iš karto visą paveikslą. Kompiuterio grafinė atmintis organizuota puslapiais. Jų kiekis ir dydis priklauso nuo monitoriaus ir draiverio, todėl prieš rašant tokio tipo programas reikia išsiaiškinti savo kompiuterio galimybes. Grafinėje atmintyje puslapiai numeruojami pradedant nuliu (0). Taigi, galima figūrą nupiešti puslapyje, kurio ekrane nesimato, o po to užbaigtą vaizdą parodyti ekrane. Darbui su grafiniais puslapiais yra skirtos dvi procedūros.

```

procedure SetVisualPage( Puslapis: word );

```

Padaro matomu ekrane nurodytą puslapį.

```

procedure SetActivePage( Puslapis: word );

```

Padaro aktyvų grafinį puslapį, kuriame atliekami visi veiksmai.

Jeigu puslapis sutampa su matomo puslapio numeriu, tai piešimo eigą galima stebėti ekrane. Pagal nutylėjamą aktyvus ir matomas puslapiai sutampa ir yra nuliniai.

Kompiuterio grafinį režimą, jeigu netinka esamas pagal nutylėjamą, galima nustatyti procedūros **SetGraphMode** pagalba:

```

procedure SetGraphMode( Režimas: integer);

```

Darbo režimų perjungimas galimas tik to paties grafinio draiverio, kuris buvo nurodytas **GraphInit** procedūra, ribose. Kai kurių galimų režimų ir juose naudojamų puslapių skaičius parodyti 9.4 lentelėje.

9.4 lentelė.

Draiveris	Režimas (konstanta)	Ekrano dydis (tšk.)	Spalvų kiekis	Puslapių kiekis
EGA	EGALo=0	640x200	16	4
	EGAHi=1	640x350	16	2
VGA	VGALo=0	640x200	16	4
	VGAMed=1	640x350	16	2
	VGAHi=2	640x480	16	1
IBM8514	IBM8514Lo=0	640x480	256	1
	IBM8514Lo=1	1024x768	256	1

■ 19 pratimas. Programoje demonstruojamas puslapių panaudojimas. Figūra (stačiakampis gretasienis ir jo priekinėje plokštumoje skritulys) piešiama nematomame puslapyje. Nupiešus (procedūra `Delay` skiriame laiko piešimui) vaizdas parodomas ekrane, o nauja figūra piešiama kitame puslapyje. Programa naudoja du puslapius, kuriuos pakaitomis parodo ekrane.

```

program Pustapiai;
uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Vaizdas( x, y, r : integer);
begin
    { Geltonas gretasienis }
    SetColor( Red ); SetFillStyle( 1, Yellow );
    Bar3d( x, y, x+r+r, y+r+r, r, True);

    SetFillStyle( 1, Green );           { Žalias skritulys }
    FillEllipse( x+r, y+r, r, r ); end;
{-----}
var i, p, pl, dx, dy, x, y, r : integer;
begin
    Ekranas( dx, dy, Vieta ); SetGraphMode( EGAHi );
    p := 0; pl := 1; { Grafinių puslapių aktyvumo reikšmės }
}
repeat
    { Pradines reikšmės }
    x := dx div 3; y := dy div 3; r := 10;
    for i := 1 to 50 do begin
        SetVisualPage( p );           { Matomas puslapis }
        SetActivePage( pl ); { Aktyvus (nematomas) puslapis }
        ClearDevice;                   { Aktyvaus puslapio valymas }
        Vaizdas( x, y, r );
        Delay( 700 );
        Inc( r, 2 );                   { Parametrų keitimas }
        if p=0 then begin p := 1; pl := 0; end
        else begin p := 0; pl := 1; end;
    end;
until KeyPressed;
CloseGraph;
end.

```

■ 20 pratimas. Keturiuose puslapiuose nupiešiamos keturios skirtingos sferos. Po to ciklu puslapiai rodomi ekrane.

```

program Sfera;

```

```

uses Crt, Graph, Grafika;
const Vieta = 'C:\programs\tp7\bgi';
      { Sferų spinduliai }
  R : array[ 0..3 ] of integer = ( 60, 80, 100, 200 );
{-----}
procedure Formuoti(k: integer; dx, dy, spalva : integer);
var t : integer;
begin
  t := 0;
  while t <= k div 2 do begin
    SetColor( spalva );
    Ellipse( dx div 2, dy div 2, 0, 360, k, t );
    t := t +3;
  end;
end;
{-----}
var i, dx, dy : integer;
begin
  Ekranas ( dx, dy, Vieta );      SetGraphMode( EGALo );
  dx := GetMaxX; dy := GetMaxY; { Naujo režimo reikšmės}

  for i := 0 to 3 do begin      { Sferų piešimas }
    SetActivePage( i );
    Formuoti( R[i], dx, dy, i+2 );
    end;
  repeat
    for i := 0 to 3 do begin    { Puslapiai rodomi ekrane }
      SetVisualPage( i ); Delay( 5000 );
    end;
  until KeyPressed;
  CloseGraph;
end.

```

9.5. Paletė.

Ekране turimo piešinio spalvas keisti galima neperpiešiant figūrų naujomis spalvomis. Visos spalvos (jų kodai) surašyti tam tikra tvarka lentelėje, kuri vadinama palete. Nulinėje masvyvo vietoje yra spalvos kodas, kuris atitinka fono spalvą. Kitose vietose yra spalvų kodai, naudojami figūrų piešimui. Standartinėje paletėje spalvos sudėliotos lentelėje pagal jų kodus, t.y. antroje vietoje yra 2 (Green), ketvirtoje vietoje yra 4 (Red) ir t.t. Jeigu lentelės ketvirtoje vietoje įrašysime kodą 2 (Green), tuomet užrašas Red (ketvirta lentelės spalva) reikš tą spalvą, kurios kodas bus čia parašytas, t.y. žalios spalvos.

■ 21 pratimas. Demonstruojamas spalvos Red pakeitimas į Green, nepakeičiant konstantos Red procedūrose.

```
program Pall;
uses Crt, Graph, Grafika;
const Greitis = 1;
      Vieta = 'C:\programs\tp7\bgi';
{-----}
var dx, dy : integer;  sim : char;
begin
  Ekranas ( dx, dy, Vieta );
  SetFillStyle( 1, Red );      { Red = 4                }
  Bar( 100, 100, 300, 300 ); { Raudonas stačiakampis }
  sim := readKey;
SetPalette( 4, Green ); { Ketvirta spalva keičiama Green = 2}
                        { Pirmasis stačiakampis tampa žaliu }
  SetFillStyle( 2, Red );      { Red = 2 perkoduota    }
  Bar( 350, 100, 450, 200);   { Žalias stačiakampis   }
  ReadLn;
  CloseGraph;
end.
```

Darbai su palete yra skirtos 4 procedūros.

```
procedure SetPalette( Nr : word;  sp : shortint );
procedure SetAllPalette( var P );
procedure GetPalette( var P : PaletteType );
procedure GetDefaultPalette( var P : PaletteType);
```

Čia Nr – paletės lentelės numeris;

sp – spalvos kodas;

P – paletės tipo kintamasis.

```
type PaletteType = record
  Size : byte;
  Colors : array[ 0.. MaxColors ] of shortint;
end;
```

Konstanta MaxColors = 15 nurodo didžiausią galimą spalvos kodo reikšmę. Lauke Size yra saugomas aktyvioje paletėje esančių spalvų skaičius. Masyve Colors yra aprašoma spalvų tvarka paletėje.

Procedūra SetPalette į spalvų masyvo vietą Nr įrašomas spalvos kodas sp.

Procedūra SetAllPalette pakeičia visą aktyvią paletę nauja

nurodyta kintamuoju P.

Procedūra GetPalette gauname aktyvios paletės kopiją P.

Procedure DetDefaultPalette gauname standartinės paletės kopiją P.

22 pratimas. Demonstruojamas paletės spalvų perkodavimas. Nupiešiami skirtingomis spalvomis stačiakampiai, kurių centrai yra toje pačioje ekrano vietoje: stačiakampės juostos, kurių kiekviena kitokios spalvos. Procedūra Keisti paletėje esančias spalvas žiediškai perstumia per vieną vietą atgal. Automatiškai kinta vaizdo ekrane spalvinė gama.

```
program Palette;
uses Crt, Graph, Grafika;
const Greitis = 1;
      Vieta = 'C:\programs\tp7\bgi';
{-----}
procedure Status( dx, dy : integer );
var i, a : integer;
begin
  a := 1;                                { Užpildymo rašto kodas }
  for i := 1 to 15 do begin
    SetFillStyle( a, i );                { i - spalvos kodas      }
    Bar( i*20, i*15, dx - i*20, dy - i*15);
    a := a+2;  if a >= 4 then a := 1;
  end;
end;
{-----}
procedure Keisti;
var P : PaletteType;  sp, i : integer;  sim : char;
begin
  GetPalette( P );  { Paletės kopija }
  repeat
    sp := P.Colors[2];                    { Spalvų perstūmimas }
    for i := 2 to MaxColors do
      P.Colors[i-1] := P.Colors[i];
      P.Colors[ MaxColors ] := sp;
      SetAllPalette( P );                  { Nauja paletė }
      Delay( 3000 );
    until KeyPressed;
    sim := readKey;
  end;
{-----}
var  dx, dy : integer;
begin
```

```

Ekranas    ( dx, dy, Vieta );
    Status  ( dx, dy );      { Vaizdo piešimas      }
    Keisti;                      { Paletės keitimai    }
ReadLn;
CloseGraph;
end.

```

✓ Išbandykite paletės valdymo priemones su savo sukurtais piešiniais. Pabandykite keisti tik fono spalvos kodą, kuris yra nulinėje paletės masvyvo vietoje.

Kiekvienas draiveris leidžia naudoti skirtingą spalvų skaičių. Tik IBM8514 turi galimybę dirbti su 255 spalvomis. Procedūra SetRGBPalette leidžia nustatyti naudojamų spalvų 0..255 intensyvumus.

```

procedure SetRGBPalette(
    SpNr, RedValue, GreenValue, BlueValue : integer);

```

Čia SpNr spalvos numeris, o kiti parametrai nurodo spalvos pagrindinių komponentų reikšmes. Kiekviena spalva gaunama trijų bazinių spalvų: raudonos, žalios ir mėlynos pagrindu. Šią procedūrą galima naudoti tik IBM8514 ir VGA draiveriams.

23 pratimas. Demonstruojamas spalvų perkodavimas. Vietoje raudonos ir žalios spalvų (4-as ir 2-as numeriai) įrašomas skirtingo intensyvumo pilkos spalvos.

```

program Pal2;
uses Crt, Graph, Grafika;
const Greitis = 1;
      Vieta = 'C:\programs\tp7\bgi';
{-----}
var dx, dy : integer; sim : char;
begin
    Ekranas    ( dx, dy, Vieta );  SetGraphMode( IBM8514 );
    SetFillStyle( 1, Red ); { Red = 4      }
    Bar( 100, 100, 300, 300 ); { Raudonas stačiakampis }
    SetFillStyle( 1, Green );
    Bar( 130, 130, 270, 270 ); { Žalias stačiakampis }
    sim := readKey;

                                { Stačiakampiai pilki: }
    SetRGBPalette( 4, 100,105,107 ); { tamsesnis      }
    SetRGBPalette( 2, 105,107,109 ); { šviesesnis     }

    ReadLn;
    CloseGraph;

```

end.

✓ Išbandykite programą, sukurdami 4 spalvų paletę, kurioje visos spalvos būtų žalios skirtingo intensyvumo.

9.6. Užduotys.

✧ Turime kvadratą. Į jį įbrėžiamas apskritimas. Į apskritimą įbrėžiamas kvadratas, į kurį įbrėžiamas apskritimas ir t.t., kol gausime viduje apskritimą arba kvadratą, kurio spindulys ar kraštinė bus ne mažesnė kaip 10 taškų. Kampus tarp kvadrato ir įbrėžto apskritimo užpildykite parinktu raštu.

10. Ekraninės grafikos pavyzdžiai

Šiame skyrelyje pateikiami trumpi aprašymai programų, kurios demonstruoja ekraninės grafikos elementus. Tas programas parašė studentai ir moksleiviai, kurie norėjo susipažinti ir suprasti ekraninės grafikos elementus, išsiaiškinti jų panaudojimo galimybes. Knygos autoriai parinko savo nuožiūra įdomesnius bei savo prasme skirtingus pavyzdėlius. Čia esančius aprašus Jūs galite priimti kaip savarankiško darbo užduotis. Siūlome Jums patiems jas parašyti. Diskelyje Jūs rasite tas programas ir galėsite palyginti su savomis. Pirmosios programos struktūra parodoma paprogramių aprašymu. Kitų programų pateikiamas trumpas aprašymas ir jų charakteristikos.

10.1. Stiklinė vandens ir tirpstantis varveklis.

Failas `Stikline.pas`. Ant palangės po tirpstančiu varvekliu stovi sklidina vandens stiklinė. Ant varvekliaus formuojasi vandens lašas. Jis lėtai slenka žemyn. Lašas atitrūksta ir krenta į stiklinę. Girdisi tekštelejimas. Vandens paviršiumi nubėga bangelės. Lašui vietos stiklinėje jau nėra ir jis nukrenta ant palangės. Girdisi kapsėjimas.

Programos autorius: IFT 4/1 gr. stud. Nerijus Varpučinskis, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 9 procedūros, 136 eilutės, 4,683 Kb.

procedure **Fonas**; Nupiešia varveklį ir palangę.

procedure **Stikline**; Nupiešia stiklinę vandens.

procedure **Lasas**(xx, yy : integer; c, c1 : boolean);

Nurodomos lašo koordinatės ir lašo charakteristikos.

procedure **Juda**(xx, yy, x, y : integer);

Piešiama бага stiklinėje. Nurodomos lango koordinatės ir elipsės spinduliai.

procedure **Banga**;

Bagų skleidimas vandens paviršiumi. Naudojama procedūra **Juda**.

procedure **Pum**(x : integer);

Garsiukas. Nurodomas dažnis. Skirta garsui generuoti, kai lašas krenta į stiklinę.

procedure **Kristi**; Lašas krenta iš stiklinės.

procedure **Skeveldra**;

Lašas, nukritęs ant palangės, suskyla į daug dalelių.

procedure **Valdanti**;

Varvekliaus tirpimas ir lašų kritimas, kaip vientisas reiškinys. Tai ciklinis procesas kol bus paspaustas klavišas Esc. Vieno ciklo turinys: lašas pasirodo ir slenka

varvekliu žemyn; atitrūksta nuo varvekliaus ir krenta į stiklinę (garsas Pum) ir vandens paviršiumi nubėga bangelės; lašas iškrenta iš stiklinės ant palangės, kur suskyla į daug skeveldrėlių.

Pagrindinėje programoje yra grafinio ekrano paruošimas darbui, pradinės situacijos piešimas (Fonas, Stiklinė) ir kreipinys į procedūrą Valdanti.

10.2. Judantys stačiakampiai.

Failas Lina.pas. Ekrano viduryje koncentriškai stačiakampiai artėja iš tolumos centre (taškas stačiakampio centre). Stačiakampių spalvos skirtingos ir nuolat kinta. Tokio paveikslėlio keturios mažesnės kopijos, simetriškai išdėstytos ant centrinės figūros perimetro, lėtai juda ratu prieš laikrodžio rodyklės kryptį.

Programos autorius: IFT 4/1 gr. stud. Lina Boguševičienė, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 2 procedūros, 127 eilutės, 4,431 Kb.

10.3. Kompiuterio laikrodis.

Failas Laikas.pas. Kompiuterio laikrodis vaizduojamas įprastame pavidale: ciferblatas su valandomis, valandų, minučių ir sekundės rodyklėmis. Laikrodis tiksi.

Programos autorius: Kuklus studentas, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 5 procedūros, 93 eilutės, 2,775 Kb.

10.4. Spiralės, paletė.

Failas Lab8.pas. Ekrane įvairiaspalvė daugiašakė spiralė. paspaudus klavišą Enter, spiralė juda apskritimu. Judesio efektas gaunamas keičiant paletės spalvas.

Programos autorius: Kuklus studentas, 1998 ruduo.

Programos pabaiga: klavišas Enter.

Programos apimtis: 4 procedūros, 82 eilutės, 3,302 Kb.

10.5. Kvadratinės lygties sprendimas.

Failas Kvadlygt.pas. Kvadratinės lygties $ax^2 + bx + c = 0$ koeficientų reikšmės įvedamos klaviatūra. Funkcijos grafikas piešiamas koordinatinių plokštumoje. Pažymimi charakteringi taškai (susikirtimų su

koordinacių ašimis, viršūnė). Parenkamas lygties vaizdavimo mastelis.

Programos autorius: Marijus Paškevičius, Kauno kalniečių v.m., mokytoja Lina Boguševičienė, 1998 m.

Programos pabaiga: klavišas Enter.

Programos apimtis: 128 eilutės, 3,881 Kb.

10.6. Judantys spalvoti taškai: “chaosas”.

Failas Danzv1.pas. Įvairiaspalvės žvaigždutės juda ekrane skirtingais greičiais. Jos pasirodo ekrano kairėje ir viršuje. Išnyksta pasiekusios ekrano apačią ir dešinę pusę.

Programos autorius: Marijus Paškevičius, Kauno kalniečių v.m., mokytoja Lina Boguševičienė, 1998 m.

Programos pabaiga: klavišas Esc.

Programos apimtis: 3 procedūros, 70 eilulučių, 1,766 Kb.

- Šios programos modifikacija faile Danzv2.pas, kur pradinė graži tvarka žvaigždėtame danguje tampa chaosu. Čia pradžia yra taškas ekrano centre, iš kur, kaip sprogimas, plečiasi apskritimu įvairiaspalvių žvaigždučių juosta. Pasiekusios ekrano ribas žvaigždutės atšoka.

Programos pabaiga: klavišas Esc arba Enter.

Programos apimtis: 2 procedūros, 62 eilulutės, 1,602 Kb.

- Programos modifikacija faile Danzvkk.pas. Dėsningas žvaigždžių judėjimas. Judesys valdomas Taškai juda iš centro iki ekrano krašto ir vėl sugrįžta į centrą. Piešiama daugialapė figūra.

Programos pabaiga: klavišas Esc.

Programos apimtis: 3 procedūros, 63 eilulutės, 1,599 Kb.

- Faile Danzv9.pas sukasi ir tolsta galaktika. Tai spiralinis žvaigždučių paveikslas.

Programos pabaiga: klavišas Esc.

Programos apimtis: 2 procedūros, 81 eilulutės, 2,344 Kb.

10.7. Snaigės.

Failas Snaige.pas. Ekrane atsitiktinėse vietose piešiamos įvairiaspalvės šakotos įvairaus dydžio snaigės. Snaigės dydis nusakomas


skritulio spinduliu, kuriame iš taškų sukuriamas vaizdas. Piešimo ciklas nutraukiamas bet kokio klavišo paspaudimu. Ekrane galima stebėti gautą vaizdą.

Programos autorius: IFT-9 studentė Jūratė Kanclerienė, 1999 m.

Programos pabaiga: klavišas Enter.

Programos apimtis: 2 procedūros, 59 eilutės, 2,125 Kb.

10. Ekraninės grafikos pavyzdžiai

 Stiklinė vandens ir tirpstantis varveklis. Failas `Stikline.pas`.

Ant palangės po tirpstančių varvekliu stovi sklidina vandens stiklinė. Ant varvekliaus formuojasi vandens lašas. Jis lėtai slenka žemyn. Lašas atitrūksta ir krenta į stiklinę. Girdisi tekstelėjimas. Vandens paviršiumi nubėga bangelės. Lašui vietos stiklinėje jau nėra ir jis nukrenta ant palangės. Girdisi kapsėjimas.

Programos autorius: IFT 4/1 gr. stud. Nerijus Varpučinskis, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 9 procedūros, 136 eilutės, 4,683 Kb.

procedure **Fonas**; Nupiešia varveklį ir palangę.

procedure **Stikline**; Nupiešia stiklinę vandens.

procedure **Lasas**(xx, yy : integer; c, cl : boolean); Lašo koordinatės ir lašo charakteristikos.

procedure **Juda**(xx, yy, x, y : integer); Piešiama banga stiklinėje. Lango koordinatės ir elipsės spinduliai.

procedure **Banga**; Bagų skleidimas vandens paviršiumi.

procedure **Pum**(x : integer); Garsiukas. Nurodomas dažnis.

procedure **Kristi**; Lašas krenta iš stiklinės.

procedure **Skeveldra**; Lašas, nukritęs ant palangės, suskyla.

procedure **Valdanti**; Varvekliaus tirpimas ir lašų kritimas, kaip vientisas reiškiny.

Pagrindinėje programoje yra grafinio ekrano paruošimas darbui, pradinės situacijos piešimas ir kreipinys į procedūrą Valdanti.

```
{Atlikėjas: Ift 4/1 gr. stud. Nerijus Varpučinskis -1998 ruduo }  
program Grafika;  
  uses Crt, Graph;  
  const a = 10;      { Lašo parametras      }  
  var Gd, Gm : integer; { Grafinio režimo tipas      }
```

```

kl      : integer; { Klaidos kodas }
x, y    : integer; { Judančių langų kairių viršutinių kampų
                                   koord. }
P       : pointer; { Fono saugojimo adresas }
{ ----- Fono formavimas ----- }
procedure Fonas;
begin
  SetBkColor( 0 );
  SetColor ( 15 );
  SetLineStyle( 0, 0, 1 );
  MoveTo( 250, 0 );
  LineRel( 0, 100 );
  LineRel( 5, -100 );
  SetFillStyle( 1, 6 );
  Bar( 0, 380, GetMaxX, GetMaxY );
end;
{ ----- Stiklinės formavimas ----- }
procedure Stikline;
var i, y, k, x1, y1 : integer;
begin
  Ellipse( 250, 400, 0, 360, 40, 10 );
  Line( 290, 400, 310, 270 );
  Line( 210, 400, 190, 270 );
  Ellipse( 250, 270, 0, 360, 60, 20 );
  for i := 1 to 9 do begin { Šviesos (linijų) pavaizdavimas
                                   ant stiklines }
    y := 400 - 13 * i;
    k := 360 - 17 * i;
    x1 := 40 + 2 * i;
    y1 := 10 + 1 * i;
    Ellipse( 250, y, 180, k, x1, y1 );
  end;
end;
{ ----- Lašo judėjimas ----- }
procedure Lasas( xx, yy : integer; c, c1 : boolean );
begin
  SetViewport( xx, yy, xx + a, yy + a, true );
  GetImage( 0, 0, a, a, P^ );
  SetLineStyle( 0, 0, 3 );
  if c and c1 then Line( a, 0, a, a )
    else if not c and not c1 then Line( 0, 0, 0, a )
      else Line( 0, a, a - 7, a - 7 );
end;
{ ----- Juda bangos 1 ----- }
procedure Juda( xx, yy, x, y : integer );
begin
  SetLineStyle( 0, 0, 1 );
  SetViewport( xx, yy, xx + 120, yy + 40, true );
  GetImage( 0, 0, 120, 40, P^ );
  Ellipse( 60, 20, 0, 360, 60, 20 );
  Ellipse( 60, 20, 0, 360, x, y );
end;

```

```

{ ----- Juda bangos ----- }
procedure Banga;
var x, y, xx, yy : integer;
begin
  xx := 190; yy := 250;
  x := 3; y := 1;
  GetMem( P, ImageSize( 0, 0, 120, 40 ) );
  while ( x <> 60 ) and ( y <> 20 ) do begin
    Juda( xx, yy, x, y ); Delay( 100 );
    x := x + 3; y := y + 1;
    PutImage( 0, 0, P^, 0 );
  end;
end;

{ ----- Lašo atsitrenkimo garsas ----- }
procedure Pum( x : integer );
begin
  Sound( x );
  Delay( 10 );
  NoSound;
end;

{ ----- Lašo kritimas iš stiklinės ----- }
procedure Kristi;
var b, b1 : integer;
begin
  b := 310; b1 := 270;
  while b1 <> 400 do begin
    Lasas( b, b1, false, false ); Delay( 20 );
    b1 := b1 + 1;
    PutImage( 0, 0, P^, 0 );
  end;
end;

{ ----- Lašo atsitrenkimo į pagrindą skilimas ----- }
procedure Skeveldra;
var d, d1 : integer;
begin
  d := 310; d1 := 400;
  while ( d <> 330 ) and ( d1 <> 390 ) do begin
    Lasas( d, d1, true, false ); Delay ( 20 );
    d := d + 2; d1 := d1 - 1;
    PutImage( 0, 0, P^, 0 ); end;
  d := 310; d1 := 400;
  while ( d <> 300 ) and ( d1 <> 390 ) do begin
    Lasas( d, d1, true, false ); Delay ( 20 );
    d := d - 1; d1 := d1 - 1;
    PutImage( 0, 0, P^, 0 ); end;
end;

{ ----- Valdo visos programos darbą ----- }
procedure Valdanti;
begin
  GetMem( P, ImageSize( 0, 0, a, a ) );
  Stikline;
  x := 240; y := 0; { Judančio lango kairio viršutinio kampo

```


```

                                                                    koord. }
Repeat
  Lasas( x, y, true, true );
  if y >= 100 then Delay( 10 )    { Lašo slinkimo greitis
                                     varvekliu      }
    else Delay( 100 ); { Lašo kritimo greitis nuo
                                     varvekliaus    }

  y := y + 1;
  PutImage( 0, 0, P^, 0 );
  if y = 270 then begin
    y := 0;
    Pum( 4000 );
    Banga;
    Kristi;
    Pum( 5000 );
    Skeveldra;
  end;
until KeyPressed;
end;
{ ----- Pagrindinė programos dalis ----- }
begin
  Gd := 0;                                { Grafikos įjungimas }
  InitGraph( Gd, Gm, 'C:\programs\TP7\BGI' );
  kl := graphResult; { Tikrinama ar pavyko prijungti grafiką }
  if kl <> 0 then begin
    WriteLn( ' grafikos klaida: ', grapherrormsg( kl ) );
    Halt;
  end;

  Fonas;
  Valdanti;
  CloseGraph;
end.

```

 Judantys stačiakampiai. Failas Lina.pas. Ekrano viduryje koncentriški stačiakampiai artėja iš tolumos centre (taškas stačiakampio centre). Stačiakampių spalvos skirtingos ir nuolat kinta. Tokio paveikslėlio keturios mažesnės kopijos, simetriškai išdėstytos ant centrinės figūros perimetro, lėtai juda ratu prieš laikrodžio rodyklės kryptį.

Programos autorius: IFT 4/1 gr. stud. Boguševičienė Lina, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 2 procedūros, 127 eilutės, 4,431 Kb.

```

{-----Laboratorinis darbas Nr. 8-----}
{Atliko: Mokytojų intensyviųjų studijų stud. Boguševičienė Lina
Data:   1999 01 08                                Užduoties Nr. L8 }
program Kvadratai;
  uses Graph, Crt;
  const dlsk = 10;

```

```

        spindul = 140;
        riba    = 50;
        n       = 50;
type mas = array [1..n] of real;
mas1 = array [1..n] of integer;

var a1, a2      : integer; { Grafikos elementų įjungimui }
    x, y        : mas;    { Mažųjų kvadratėlių koordinatės }
    xkr, ykr    : mas;    { Kvadratėlių vektorių koordinatės }
    spind       : mas;    { Kvadratų langų dydis }
    sp          : mas1;   { Kvadratų spalvų masyvas }
    xmax, ymax  : integer; { Maksimalių koordinatčių nustatymui }
    xc, yc      : integer; { Koordinatčių centrui nustatyti }
    xx, yy      : integer; { Kvadratų suapvalintos koordinatės }
    dyd        : integer; { Atstumas tarp langų }
    i, il       : integer; { Ciklo kintamieji }
    spindl,      : integer; { Suapvalinta spind reikšmė }
    spindulys,  : integer; { Mažųjų kvadratėlių spinduliai }
    spindulysk : integer; { Kvadratelio maksimalus dydis }
    kartot      : boolean; { Kintamasis darbo pabaigai }
{-----Procedura paskalio grafikai įjungti-----}
procedure Prij_grapf;
begin
    a1:=VGA; a2:=VGAHi;
    InitGraph(a1, a2, 'c:\programs\tp7\bgi');
    xmax := GetMaxX;      ymax := GetMaxY;
    xc := xmax div 2;     yc := ymax div 2;
end;
{-----Procedura kvadratų brėžimui-----}
procedure Brezt(var kartot : boolean); { }
begin
    for i := 1 to 4 do      begin
        x[i] := x[i] + xkr[i];
        y[i] := y[i] + ykr[i];
        if x[i] <= riba then
            if y[i] <= riba then
                begin xkr[i] := 0; ykr[i] := 1;      end
            else if y[i] >= (ymax - riba) then
                begin xkr[i] := 1; ykr[i] := 0; end;
        if x[i] >= (xmax - riba) then
            if y[i] >= (ymax - riba) then
                begin xkr[i] := 0; ykr[i] := -1;      end
            else if y[i] <= riba then
                begin xkr[i] := -1; ykr[i] := 0;      end;
        end;
    for il := 1 to 4 do
        for i := 1 to dlsk do      begin
            if spind[i] > 0 then      begin
                SetColor(sp[i]);
                spindl := Round(spind[i]);
                xx := Round(x[i1]); yy := Round(y[i1]);
                Rectangle( xc - spindl, yc + spindl, xc + spindl, yc - spindl);

```

```


        spindulys := spindl div 3;
        spindulysk := spindul div 3;
Rectangle( xx - spindulys, yy + spindulys,
           xx + spindulys, yy - spindulys);
        SetColor(0);
Rectangle( xx - spindulysk, yy + spindulysk,
           xx + spindulysk, yy - spindulysk );
        end;

spind[i] := spind[i] + 0.1 + abs(((0.9 * spind[i]) / spindul));
if int(spind[i]) = 0 then spind[i] := 1;
if spind[i] > spindul then begin
    spind[i] := 1;          sp[i] := sp[i] + 1;
    if sp[i] > 15 then sp[i] := 1;
    end;
end;

kartot := KeyPressed;
delay(0);
end;

{ P A G R I N D I N E   P R O G R A M A }
begin
    prij_grapf;
    x[1] := riba; y[1] := riba;
    x[2] := riba; y[2] := ymax - riba;
    x[3] := xmax - riba; y[3] := ymax - riba;
    x[4] := xmax - riba; y[4] := riba;
    xkr[1] := 0; ykr[1] := 1;
    xkr[2] := 1; ykr[2] := 0;
    xkr[3] := 0; ykr[3] := -1;
    xkr[4] := -1; ykr[4] := 0;
    dyd := spindul div dlsk;
    spind[1] := 1; sp[1] := 1;
    for i := 2 to dlsk do begin
        spind[i] := spind[i-1] - dyd;
        sp[i] := sp[i-1] + 1;
        if sp[i] >= 15 then sp[i] := 1; end;
    repeat
        brezt(kartot);
    until kartot;
    kartot := false;
    CloseGraph;
end.

```

 Kompiuterio laikrodis. Failas Laikas.pas. Kompiuterio laikrodis vaizduojamas įprastame pavidale: ciferblatas su valandomis, valandų, minučių ir sekundės rodyklėmis. Laikrodis tiksi.

Programos autorius: Kuklus studentas, 1998 ruduo.

Programos pabaiga: klavišas Esc.

Programos apimtis: 5 procedūros, 93 eilutės, 2,775 Kb.

```

{ Kompiuterio laikrodis. Kuklus studentas: 1998 m. ruduo.      }
program Graphdemo;
uses Graph, Crt, Dos;
var
    x, cx : integer;
    y, cy : integer;
    mx, my : integer;
    Klaida, Graphmode, Graphdriver : integer;
    v, m, s, ss, sl : word;
    i      : integer;
    sk     : string;
{-----}
procedure Init;
begin
    Graphdriver := vga;
    InitGraph ( Graphdriver, Graphmode, 'C:\PROGRAMS\TP7\BGI' );
    Klaida := GraphResult;
    if Klaida <> grOk then
        begin
            WriteLn ( 'Klaidos kodas: ', Klaida );
            WriteLn ( 'Programa nutraukta' );
            ReadLn;
            Halt (1)
        end
    end;
{-----X_Koord-----}
function X_Koord ( laikas : word; ilgis : integer ) : integer;
var
    k : real;
begin
    if laikas > 360 then    laikas := laikas - 360;
    k := laikas * Pi / 180;
    x := - Round ( ilgis * cos( k ) * 639 / 250 );
    X_Koord := x
end;
{-----Y_Koord-----}
function Y_Koord ( laikas : word; ilgis : integer ) : integer;
var
    k : real;
begin
    if laikas > 360 then    laikas := laikas - 360;
    k := laikas * Pi / 180;
    y := - Round ( ilgis * sin( k ) );
    Y_Koord := y
end;
{-----Strele-----}
procedure Strele ( cx, cy : integer; spalva : integer;
                  laikas : word; ilgis, storis : integer );
begin
    MoveTo ( cx, cy );
    SetLineStyle ( 0, 0, storis );
    SetColor ( spalva );
    LineRel ( X_Koord ( laikas, ilgis ), Y_Koord ( laikas,

```

```

ilgis ) )
end;
{-----FONAS-----}
procedure Fonas;
begin
  Bar ( 40, 10, mx - 40, my - 10 );
  for i := 1 to 12 do
    begin
      Str ( i, sk );
      SetTextStyle ( 0, 0, 2 );
      x := X_Koord ( i * 30 + 90, 70 );
      y := Y_Koord ( i * 30 + 90, 70 );
      MoveTo ( cx + x, cy + y );
      SetTextJustify ( 1, 1 );
      SetColor ( Green );
      OutText ( sk );
      SetColor ( White )
    end
  end;
{-----}
begin
  Init;
  mx := GetmaxX;      my := GetmaxY;
  cx := mx div 2;     cy := my div 2;
  s  := 0;            sl := 0;
  Fonas;
  repeat
    while s = sl do
      GetTime ( v, m, sl, ss );
      s := sl;
      if v > 360 then v := v - 360;
      SetColor ( White );
      FillEllipse ( cx, cy, Round ( 50 * 639 / 250 ), 52 );
      Strele ( cx, cy, red, s * 6 + 90, 50, 1 );
      Strele ( cx, cy, 8, m * 6 + 90, 45, 3 );
      Strele ( cx, cy, Black, v * 30 + 90 +
        Round((m * 6 + 90) / 60),25,5);

      Sound ( 100 );
      Delay ( 50 );
      NoSound
    until KeyPressed;
    CloseGraph
  end.

```

Failas Lab8.pas. Spirālēs, paletē.

```

{ Kuklaus studento programa: 1998 ruduo }
program Lab8;
uses Crt, Graph;
const PradSpalva : integer = 1;      { Spalvu intervalas }
      GalutSpalva: integer = 14;
var    Gd, Gm :integer;
{-Apskaičiuojamos atitinkamų kampų sinuso ir kosinuso reikšmės-}
procedure SinCosReiksmes(var XReiksme, YReiksme: array of real);
var    kampas : integer;

```

```

begin
  for kampas := 0 to 360 do          begin
    XReiksme[kampas] := Cos(Kampas * Pi / 180);
    YReiksme[kampas] := Sin(Kampas * Pi / 180);
                                end;
  end;
{-----procedūra nustatanti paletės atspalvius-----}
procedure PaletesNustatymas(PradSpalva, GalutSpalva: integer);
var  SpalvNr: integer;
begin
  for SpalvNr := PradSpalva to GalutSpalva div 2 do    begin
    SetRGBPalette(SpalvNr, 9 * SpalvNr, 9 * SpalvNr,
                                9 * SpalvNr);
    {Reikia pakartoti veiksmą nes susigadina viena spalva}
    SetPalette(SpalvNr, SpalvNr);
    {Tokie patys atspalviai priskiriami didžiausiai ir mažiausiai
    spalvai}    SetPalette(GalutSpalva + 1 - SpalvNr, SpalvNr);
                                end;
  end;
{-----Keičiami paletės atspalviai (banguojama)-----}
procedure PaletesKeitimas(PradSpalva, GalutSpalva: integer);
var  SpalvNr : integer;
      Palette : PaletteType;
begin
  GetPalette(Palette); {Įsimenama paletė}
  with Palette do      begin
    {kad nebūtų sugadinta pirma spalva,
    ji įsimenama numeriu kuris nenaudojamas}
    Colors[GalutSpalva + 1] := Colors[PradSpalva];
    {spalvos paslenkamos per vieną poziciją}
    for SpalvNr := PradSpalva to GalutSpalva do
      Colors[SpalvNr] := Colors[SpalvNr + 1];
    end;
    SetAllPalette(Palette); {Iš naujo nustatoma paletė}
  end;
{--Piešiamas apskritimas su skirtingais atspalviais (bangomis)-}
procedure Bangos(PradSpalva, GalutSpalva: integer);
const  PradSpind    : integer = 0;
        GalutSpind   : integer = 240;
        CentroXKoord : integer = 320;
        CentroYKoord : integer = 240;
        Pokytis      : real    = 0.22;
var     Spindulys, Kampas : integer;
        Spalva : real;
        XReiksme, YReiksme : array [ 0..360] of real;
begin
  SinCosReiksmes(XReiksme, YReiksme);
  Spalva := PradSpalva;
  for Spindulys := PradSpind to GalutSpind do
    for kampas := 0 to 360 do    begin
      {tikrinama, ar spalva nėra didesnė nei maksimali}
      if Spalva > GalutSpalva then    Spalva := PradSpalva;

```

```

    PutPixel(Trunc(XReiksme[Kampas] * Spindulys) + CentroXKoord,
             Trunc(YReiksme[Kampas] * Spindulys) + CentroYKoord,
             Trunc(Spalva));
    Spalva := Spalva + Pokytis;
                                end;
end;
{-----Pagrindine programa-----}
begin
    Gd := Vga;          Gm := VgaHi;
    InitGraph (gd, gm, 'c:\PROGRAMS\tp7\bgi');
    if GraphResult <> 0 then
    begin Write(GraphErrorMsg(GraphResult)); Halt;          end;
    {   PaletesNustatymas(PradSpalva, GalutSpalva);   }
    Bangos(PradSpalva, GalutSpalva);
    {Laukiama enter paspaudimo: galima keisti paletę (banguoti)}
    ReadLn;
    while not KeyPressed do          begin
        PaletesKeitimas(PradSpalva, GalutSpalva);          Delay(50);
                                end;
    CloseGraph;
end.

```

Failas Kvadlygt.pas. Kvadratinės lygties sprendimas.

```

program Kvadratine_lygtis;
uses Graph, Crt;
var a, b, c      : real;
    a1, a2, i1, i2 : integer;
    d, x1, x2      : real;
    x, y           : real;
    xv, yv         : real;
    xc, yc         : integer;
    xmax, ymax     : integer;
    max            : integer;
    i, sp          : integer;
    xvk, yvk       : integer;
    sxv, syv       : string;
    mast          : real;
    sx1, sx2       : string;
    text, sc       : string;
begin
    a1 := VGA; a2 := VGAHi;
    mast := 1;
    WriteLn('Iveskite a, b, c ');
    ReadLn(a, b, c);
    d := b*b-4*a*c;
    if d >= 0 then          begin
        x1 := (-b + Sqrt(d))/(2*a);
        x2 := (-b - Sqrt(d))/(2*a);
                                end;
    xv := -b/(2*a);
    yv := (a * xv * xv) + (b * xv) + c;

```

```

if ( Abs(xv)) > ( Abs(yv) ) then max := Round(xv)
                                else max := Round(yv);

while Abs(max) > 200 do          begin
    max := max div 2;
    mast := mast * 2;            end;
while Abs(max) < 120 do          begin
    max := max * 2;
    mast := mast / 2;            end;
Delay(500);
InitGraph(a1, a2, 'c:\programs\tp7\bgi');
xmax := GetMaxx;
ymax := GetMaxy;
xc := xmax div 2;
yc := ymax div 2;

SetfillStyle (7, 1);
FloodFill(xc, yc, 14);
SetlineStyle(3, 0, 1);
Line (0, yc, xmax, yc);
line (xc, 0, xc, ymax);
il := Round( xc * mast * 400);

for i := -il to il do           begin
    x := i / 400;
    y := ((a * x * x) + (b * x) + c) / mast;
    x := x / mast;
    sp := White;
    if (y > -yc) and (y < yc) then
        PutPixel (xc + Round(x), yc - Round(y), sp);
    end;

SetColor (Red);
SetLineStyle (1, 0, 1);
xvk := Round(xv / mast);
yvk := Round(yv / mast);
Circle (xc + xvk, yc - yvk, 2);
Line ( xc + xvk, yc - yvk, xc, yc - yvk);
Line ( xc + xvk, yc - yvk, xc + xvk, yc);
    if d >= 0 then                begin
        SetColor( Yellow );
        Circle( xc + Round(x1 / mast), yc, 2);
        Circle( xc + Round(x2 / mast), yc, 2);
    end;

SetColor(Red);
SetTextStyle (0, 0, 0);
Str ( xv : 6 : 2, sxv);
Str ( yv : 6 : 2, syv);
Str ( x1 : 6 : 2, sx1);
Str ( x2 : 6 : 2, sx2);
Str ( c : 6 : 2, sc);
OutTextXY ( xc + xvk, yc, sxv);
OutTextXY ( xc, yc - yvk, syv);

```

```

SetColor(Red);
if d > 0 then      begin
    OutTextXY ( 5, 5, ' Si funkcija kerta Ox asi, kai');
    text := 'x=' + sx1 + ';  x=' + sx2;
    OutTextXY ( 5, 15, text );
    end;
if d = 0 then      begin
    OutTextXY ( 5, 5, ' Si funkcija kerta Ox asi, kai');
    text := 'x=' + sx1;
    OutTextXY ( 5, 15, text);
    end;
if d < 0 then
    OutTextXY ( 5, 15, ' Si funkcija nekerta Ox asies');
text := ' Virsunes koordinates (' + sxv + ';' + syv + ')';
OutTextXY ( 5, 25, text);
if a > 0 then      begin
    text := 'nuo minus begalybes iki ' + sxv + ' ,';
    OutTextXY ( 5, 35, ' Si funkcija mazeja, kai x kinta');
    OutTextXY ( 5, 45, text);
    text := 'nuo' + sxv + ' iki plus begalybes';
    OutTextXY ( 5, 55, 'funkcija dideja, kai x kinta');
    OutTextXY ( 5, 65, text);
    end;
if a < 0 then      begin
    text := 'nuo minus begalybes iki ' + sxv + ' ,';
    OutTextXY ( 5, 35, ' Si funkcija dideja, kai x kinta');
    OutTextXY ( 5, 45, text);
    text := 'nuo' + sxv + ' iki plus begalybes';
    OutTextXY ( 5, 55, 'funkcija mazeja, kai x kinta');
    OutTextXY ( 5, 65, text);
    end;
text := ' Funkcija kerta Oy asi taske ( 0,' + sc + ')';
OutTextXY ( 5, 75, text);
Delay(500);
ReadLn;
end.

```

 Failas Danzv1.pas. Judantys spalvoti taškai: “chaosas”.

```

program Dangaus_zvaigzdes1;
uses Graph, Crt;
const n = 500;
type mas = array [1..n] of real;
      mas1 = array [1..n] of integer;
var  a1, a2 : integer;

procedure PradKoord( var xmx, ymx : integer;
                    var x, y, xkr, ykr : mas;
                    var sp : mas1);
var  a, i, zs : integer;
begin
    xmx := GetMaxX;    ymx := GetMaxY;    zs := n div 2;

```

```

    for i := 1 to zs do begin
        x[i] := Random(xmx);          ykr[i] := Random;
        sp[i] := Random(14)+1;        y[i] := 0;
        xkr[i] := 0;                  zs := n-i+1;
    end;
    for i := zs to n do begin
        y[i] := Random(ymx);          xkr[i] := Random;
        sp[i] := Random(14)+1;        x[i] := 0;
        ykr[i] := 0;                  end;
end;

procedure UzRibu( xmx, ymx : integer; var x, y : real );
begin
    if x>xmx then x := 0;
    if x<0 then x := xmx;
    if y>ymx then y := 0;
    if y<0 then y := ymx;
end;

procedure DZvaigl(pause : integer);
var kl : boolean;
    x, y, xkr, ykr : mas;
    sp : mas1;
    xmx, ymx, i : integer;
begin
    Randomize;
    PradKoord(xmx, ymx, x, y, xkr, ykr, sp);
    repeat
        for i := 1 to n do begin
            PutPixel(Trunc(x[i]), Trunc(y[i]), 0);
            x[i] := x[i]+xkr[i];          y[i] := y[i]+ykr[i];
            if (x[i]>xmx) or (y[i]>ymx) or
                (x[i]<0) or (y[i]<0) then UzRibu(xmx, ymx, x[i], y[i]);
            PutPixel(Trunc(x[i]), Trunc(y[i]), sp[i]);
        end;
        kl := KeyPressed;
        Delay(pause);
    until kl;
    kl := false;
end;

{ P R O G R A M A }

begin
    a1 := VGA;      a2 := VGAHi;
    InitGraph(a1, a2, 'c:\programs\tp7\bgi');
    DZvaigl(0);
    CloseGraph;
end.

```

? Failas Danzv2.pas. Tvarka žvaigždėtame danguje tampa chaosu.

```

program Dangaus_zvaigzdes2;
uses Graph, Crt;

```

```

const n = 500;
type mas = array [1..n] of real;
      mas1 = array [1..n] of integer;
var a1, a2 : integer;
procedure PradKoord( var xmx, ymx : integer;
                    var x, y, xkr, ykr : mas;
                    var sp : mas1);
var a, i : integer;
begin
  xmx := GetTmaxX;          ymx := GetTmaxY;
  for i := 1 to n do begin
    x[i] := xmx/2;          y[i] := ymx/2;
    a := Random(360);
    xkr[i] := cos(a*Pi/180); ykr[i] := sin(a*Pi/180);
    sp[i] := Random(14)+1;
  end;
end;

procedure UzRibu( xmx, ymx : integer; var x, y : real );
begin
  if x>xmx then x := 0;
  if x<0 then x := xmx;
  if y>ymx then y := 0;
  if y<0 then y := ymx;
end;

procedure DZvaig2(pause : integer);
var kl : boolean;
  x, y, xkr, ykr : mas;
  sp : mas1;
  xmx, ymx, i : integer;
begin
  Randomize;
  PradKoord(xmx, ymx, x, y, xkr, ykr, sp);
  repeat
    for i:=1 to n do begin
      PutPixel(Trunc(x[i]), Trunc(y[i]), 0);
      x[i] := x[i]+xkr[i]; y[i] := y[i]+ykr[i];
      if (x[i]>xmx) or (y[i]>ymx) or
        (x[i]<0) or (y[i]<0) then UzRibu(xmx, ymx, x[i], y[i]);
      PutPixel(Trunc(x[i]), Trunc(y[i]), sp[i]);
    end;
    kl := KeyPressed;
    Delay(pause);
  until kl;
  kl := false;
end;

{          P R O G R A M A          }

begin
  a1 := VGA;          a2 := VGAHi;
  InitGraph(a1, a2, 'c:\programs\tp7\bgi');
  DZvaig2(5);

```

```
    CloseGraph;
end.
```

Failas Danzvkk.pas. Dėsningas žvaigždžių judėjimas.

```
program Danguis_zvaigzdes2kk;
    uses Graph, Crt;
    const n = 500;
    type mas = array [1..n] of real;
        mas1 = array [1..n] of integer;
    var  a1, a2 : integer;

    procedure PradKoord( var xmx, ymx, xc, yc : integer;
                        var x, y, xkr, ykr : mas;
                        var sp : mas1);
    var  a, i : integer;
    begin
        xmx := GetMaxX;          ymx := GetMaxY;
        xc := xmx div 2;         yc := ymx div 2;
        for i := 1 to n do begin
            x[i] := xc;          y[i] := yc;
            a := Random(360);
            xkr[i] := cos(a*Pi/180);    ykr[i] := sin(a*Pi/180);
            sp[i] := Random(14)+1;
        end;
    end;

    procedure UzRibu( var xkr, ykr : real );
    begin
        xkr := -xkr;          ykr := -ykr;
    end;

    procedure DZvaig2kk(pause : integer);
    var  kl          : boolean;
        x, y, xkr, ykr : mas;
        sp           : mas1;
        xmx, ymx, xc, yc, i : integer;
    begin
        Randomize;
        PradKoord(xmx, ymx, xc, yc, x, y, xkr, ykr, sp);
        repeat
            for i := 1 to n do begin
                PutPixel(Trunc(x[i]), Trunc(y[i]), 0);
                x[i] := x[i]+xkr[i];          y[i] := y[i]+ykr[i];
                if (x[i]>xc+yc) or (y[i]>ymx) or
                    (x[i]<xc-yc) or (y[i]<0) then UzRibu(xkr[i], ykr[i]);
                PutPixel(Trunc(x[i]), Trunc(y[i]), sp[i]);
            end;
            kl := KeyPressed;
            Delay(pause);
        until kl;
```

```

kl := false;
end;

{          P R O G R A M A          }

begin
    a1 := VGA;                a2 := VGAHi;
    InitGraph(a1, a2, 'c:\programs\tp7\bgi');
    DZvaig2kk(5);
    CloseGraph;
end.

```

❓ Failas Danzv9.pas. Sukasi ir tolsta galaktika.

```

program Dangaus_zvaigzdes10;
uses Graph, crt;
const n = 300;
type mas = array [1..n] of real;
      mas1 = array [1..n] of integer;
var a1, a2 : integer;

procedure PradKoord( var xmx, ymx, xc, yc : integer;
                    var x, y, a, ilg : mas;
                    var sp : mas1);
var i, skl, nd2 : integer;
begin
    xmx := GetTmaxX;          ymx := GetTmaxY;
    xc := xmx div 2;          yc := ymx div 2;
    skl := 0;                 ilg[1] := 2;
    for i := 2 to n div 2 do begin
        nd2 := n-i+1;
        if Random(4)=0 then a[i] := Random(360)
            else a[i] := Random(50)+skl;
        a[nd2] := a[i]+180;
        while a[i]>360 do a[i] := a[i]-360;
        while a[nd2]>360 do a[nd2] := a[nd2]-360;
        ilg[i] := ilg[i-1]+1/3;      ilg[nd2] := ilg[i];
        x[i] := 1;                  y[i] := 1;
        x[nd2] := 1;                y[nd2] := 1;
        sp[i] := Random(4)+1;        sp[nd2] := Random(4)+4;
        skl := skl+2;
    end;
end;

procedure DZvaig10(pause : integer);
var kl, galima : boolean;
    x, y, a, ilg : mas;
    sp : mas1;
    xmx, ymx, xc,
    yc, gr, i : integer;
begin
    Randomize;
    PradKoord(xmx, ymx, xc, yc, x, y, a, ilg, sp);
    gr := 20;

```

```

repeat
  for i := 1 to n do begin
    if (x[i]<mx) and (x[i]>0) and
      (y[i]<my) and (y[i]>0) then galima := true
      else galima := false;
    if galima then PutPixel(Trunc(x[i]), Trunc(y[i]), 0);
    a[i] := a[i]-(gr/10);
    if a[i]>360 then a[i] := 1;
    if a[i]<1 then a[i] := 360;
    if ilg[i]>xc then ilg[i] := 2;
    if ilg[i]<2 then ilg[i] := xc;

    ilg[i] := ilg[i]*995/1000-1/3;

    x[i] := xc+ilg[i]*cos(a[i]*Pi/180);
    y[i] := yc+ilg[i]*sin(a[i]*Pi/180);
    if (x[i]<mx) and (x[i]>0) and
      (y[i]<my) and (y[i]>0) then galima := true
      else galima := false;
    if galima then PutPixel(Trunc(x[i]), Trunc(y[i]), sp[i]);
    end;
  kl := KeyPressed;
  Delay(pause);
until kl;
kl := false;
end;

{          P R O G R A M A          }

begin
  a1 := VGA;          a2 := VGAHi;
  InitGraph(a1, a2, 'c:\programs\tp7\bgi');
  DZvaig10(0);
  CloseGraph;
end.

```

Literatūra

1. J.Blonskis, K.Baniulis, V.Jusas, R.Marcinkevičius, J.Smolinskas. Programavimas. Vadovėlis. – Kaunas: Technologija, 1999, 378 psl.
2. A.Vidžiūnas, J.Blonskis. Turbo Paskalis 7.0. Vartotojo vadovas. Kaunas: Smaltija, 1998, 318 psl.