СЕМАКИН И.Г., ШЕСТАКОВ А.П.

АЛГОРИТМДЕУ ЖӘНЕ ПРОГРАММАЛАУ НЕГІЗДЕРІ

ОҚУЛЫҚ

«Білім беруді дамытудың федералдық институты»
Федералды мемлекеттік автономды мекемесі
«Компьютерлік жүйелер мен кешендер», «Ақпараттық жүйелер [салалар бойынша]» мамандықтары бойынша орта кәсіптік білім беру бағдарламаларын жүзеге асыратын білім беру мекемелерінің оқу үрдісінде қолдануға арналған оқулық, «Алгоритмдеу негіздері және программалау» оқу пәні ретінде ҰСЫНЫЛҒАН

Рецензияның тіркеу нөмірі 308,
25 маусым 2012 ж. «БДФИ» ФММ

3-басылым, стереотипт

ACADEMA

Мәскеу
«Академия» баспа орталығы,
2016

ӘОЖ 681.3.06(075.32)
КБЖ 22.18я723
С30

Бұл кітап Қазақстан Республикасының Білім және ғылым министрлігі және «Кәсіпқор» холдингі» КЕАҚ арасында жасалған шартқа сәйкес «ТжКБ жүйесі үшін шетел әдебиетін сатып алуды және аударуды ұйымдастыру жөніндегі қызметтер» мемлекеттік тапсырмасын орындау аясында қазақ тіліне аударылды. Аталған кітаптың орыс тіліндегі нұсқасы Ресей Федерациясының білім беру үдерісіне қойылатын талаптардың ескерілуімен жасалды.

Қазақстан Республикасының техникалық және кәсіптік білім беру жүйесіндегі білім беру ұйымдарының осы жағдайды ескеруі және оқу үдерісінде мазмұнды бөлімді (технология, материалдар және қажетті ақпарат) қолдануы қажет.

Аударманы «Delta Consulting Group» ЖШС жүзеге асырды, заңды мекенжайы: Астана қ., Иманов көш., 19, «Алма-Ата» БО, 809С , телефоны: 8 (7172) 78 79 29, эл. поштасы: info@dcg.kz

П і к і р б е р у ш і л е р :
физ.-мат. ғыл. канд. мұғалім ГАОУ СПО «*№ 11* Кәсіпкерлік колледжі» Ночка Е.И.; аға ғылыми қызметкер ГНУ ГОСНИТИ Россельхозакадемиясы *Соломашкин А.А.*

Семакин И.Г.
Алгоритмдеу және программалау негіздері: оқулық орта профильді білім беру мекемелері студенттеріне арналған / Семакин И.Г., Шестаков А. П. — 3-ші басп., стер. — М. : «Академия» баспа орталығы, 2016. — 304 п.

С30 ISBN 978-601-333-310-6 (каз.)
ISBN 978-5-4468-3155-5 (рус.)

Оқулықортакәсібибілімніңфедералдықмемлекеттікбілімберу стандарттарының талаптарына сәйкес «Компьютерлік жүйелердегі программалау» мамандықтары бойыншашығарылған;«Компьютерлік жүйелер және кешендер» және «Ақпараттық жүйелер (салалар бойынша)»; ОП «Алгоритмдеу және программалау негіздері».

Паскаль тіліне негізделген алгоритмдеу және программалау принциптері қарастырылған(ТурбоПаскаль7.0нұсқасы).Объектіге-бағытталған программалаудың негізгі ұғымдары берілген және оны Турбо Паскальда жүзеге асыру әрекеттері көрсетілген. Delphi біріктірілген программалау ортасы және программалардың графикалық интерфейсін жасаудың визуалды технологиясы сипатталған. Осы ортада программалық модульдерді жасау көрсетілген.

Орта кәсіптік білім беру мекемелерінің студенттеріне арналған.

ӘОЖ 681.3.06(075.32)
КБЖ 22.18я723

ISBN 978-601-333-310-6 (каз.)
ISBN 978-5-4468-3155-5 (рус.)

Құрметті оқырман!

Бұл оқулық «Компьютерлік жүйелерде программалау», «Компьютерлік жүйелер мен кешендер» және «Ақпараттық жүйелер (салаларына байланысты)» мамандықтары бойынша оқу-әдістемелік жинақтың бір бөлігі болып табылады, сонымен қатар, «Алгоритмдеу және программалау негіздері» жалпы кәсіби пәнін үйретуге арналған.

Оқулық «Алгоритмдеу және программалау негіздері» жалпы кәсіби пәнін үйретуге арналған.

Жаңа буынның оқу-әдістемелік жинағы жалпы және кәсіби пәндер мен кәсіби модульдерді зерттеуді қамтамасыз ететін дәстүрлі және инновациялық білім беру материалдарын қамтиды. Әрбір жинақта жұмыс берушінің талаптарын ескере отырып, жалпы және кәсіби құзыреттілікті меңгеруге қажетті оқулықтар мен оқу құралдары, оқыту және мониторинг құралдары бар.

Оқу басылымдары электронды білім беру ресурстарымен толықтырылып отырылады. Электрондық ресурстарда интерактивті жаттығулар мен тренажерлар, мультимедиялық нысандар, Интернеттегі қосымша материалдар мен ресурстарға сілтемелері көрсетілген теориялық және практикалық модульдер қамтылған. Оған терминологиялық сөздік және оқу үрдісінің негізгі параметрлері белгіленетін электронды журнал кіреді: жұмыс уақыты, бақылау жұмыстары мен практикалық тапсырмалардың орындалу нәтижесі. Электронды ресурстар оқу үдерісіне оңай енеді және әртүрлі оқу бағдарламаларына бейімделуі мүмкін.

Кіріспе

Программалау күннен-күнге тек кәсіпқой мамандардың ісі болып бара жатыр. 1980 жылдардың ортасында жарияланған «Программалау - екінші сауаттылық» ұраны өткен жылдардың құзырында қалды. Бүгінгі күні «компьютерлік сауаттылық» ұғымы ең алдымен, көптеген ақпараттық технология құралдарын пайдалану дағдысын қамтиды. Осы немесе басқа да ақпараттық есептерді шешу кезінде, алдымен тиісті құралдарды (электрондық кестелерді, деректер қорын басқару жүйелерін, математикалық пакеттерді және т.б.) табуға тырысу керек және бұл құралдар берілген есептерді шешуге мүмкіндік бермеген жағдайда ғана әмбебап программалау тілдерін қолдану керек.

Программалауда екі санаттағы программалаушылар бар: жүйелік және қолданбалы. Жүйелік программалаушылар - жоғарғы деңгейдегі кәсіби мамандар болып табылатын ЭЕМ-нің базалық программалық құралдарын (операциялық жүйелер, трансляторлар, сервистік құралдар және т.б.) қамтамасыз етуді әзірлеушілер. Қолданбалы программалаушылар қызметтің белгілі бір салаларындағы (ғылым, технология, өндіріс, қызмет көрсету, оқыту және т.б.) есептерді шешуге арналған ЭЕМ-нің программалық қамтамасыз ету құралдарын әзірлейді. Қолданбалы программалардың сапасына қойылатын талаптар да жүйелік программалардың сапасы сияқты жоғары болып табылады. Кез-келген программа берілген есепті дұрыс шешіп қана қоймай, қазіргі заманғы интерфейсті, жоғарғы сенімді, ыңғайлы және т.б. қасиеттерге ие болуы тиіс. Тек осындай программалар ғана программалық өнімдер нарығында бәсекеге қабілетті бола алады.

Компьютерлік техникалардың дамуымен қатар программалаудың әдістемесі және технологиясы әзірленді. Алғашында командалық-операторлық программалау пайда болды, 1960 жылдары құрылымдық программалау, логикалық және функционалдық программалау желілері жылдам дамып, қазіргі кезде нысанға бағытталған және визуалды программалау кеңінен таралуда.

Программалауды алғаш оқудағы негізгі мақсат – бұл оның құрылымдық әдістерінің негіздерін игеру болып табылады. Қазіргі заманға дейін құрылымдық әдістеме программалаушы мәдениетінің негізі болып қалады. Оны меңгермей программалауды оқуды қолға алған адам кәсіби маман болуы мүмкін емес.

1969 жылы швейцариялық профессор Никлаус Вирт студенттерді программалаудың құрылымдық әдістемесін оқытуға арналған Паскаль программалау тілін жасап шығарды. Тіл алғашқы механикалық құрылғыны ойлап тапқан Блез Паскаль есімінің құрметіне орай аталды. Кейінірек Borland International, Inc (АҚШ) оқыту мақсатында қолдану шекарасынан асып, ғылыми және өндірістік мақсаттарда пайдаланыла бастаған дербес компьютерлер үшін Турбо Паскаль программалау жүйесін жасап шығарды.

Турбо Паскальға Н. Вирт сипаттаған Паскальдың негізгі стандарттарына елеулі толықтырулар енгізілді. Уақыт өте келе тіл дамыды. Тілдің 5.5 нұсқасынан бастап Турбо Паскальға объектіге-бағытталған программалауды қолдайтын құралдары қосылды. Кейінірек бұл объектіге-бағытталған программалау мүмкіндігі бар Object Pascal тілінің құрылуына әкеп соқты. 1990 жылдардың басында Object Pascal-дағы объектіге-бағытталған программалау мен визуалды программалау технологиясы элементтерінің бірігуі - Delphi программалау жүйесінің туындауына әкеп соқты.

Осы оқулықтың *1-тарауында* кез-келген жоғарғы деңгейлі процедуралық программалау тіліне қатысты негізгі ұғымдар қарастырылған. Алгоритмдерді құрудың құрылымдық әдістемесіне басты назар аударылады: негізгі алгоритмдік құрылымдарды қолдану, шығарылатын есептердің ішкі есептерін анықтау және көмекші алгоритмдерді жасау. Осы негізде құрылымдық әдістемені қолдайтын кез келген процедуралық тілді оқып үйренуге болады.

2-тарауда Паскаль тілі Турбо Паскаль нұсқасында сипатталады (7.0 нұсқасы). Object Pascal және Delphi тілдерінде Турбо Паскальдың үздіксіздігі сақталады.

3-тараудың мазмұны Паскаль мысалдарын негізге ала отырып жоғары деңгейлі программалау тілдерінің негізгі ұғымдарын терең меңгеруге бағытталған. Мұндай дайындық болашақта өзге программалау тілдерін үйренуді жеңілдетуге көмектеседі.

4-тарауда Object Pascal-да іске асыруда келтірілген мысалында объектіге-бағытталған программалау негіздері сипатталады. Мұнда Паскаль тілінің объектіге-бағытталған кеңейтілуі болып табылатын, визуалды программалау технологиясын жүзеге асырушы Delphi программалау тілі қарастырылған.

Осы курсты оқып-үйренбес бұрын, алдымен оқырмандар

мектептегі информатика курсы шеңберіндегі алгоритмдеу негіздерін меңгеріп алғандары жөн. Әдетте мектепте алгоритмдеу негіздері оқытудың орындаушыларын қолдану арқылы меңгеріледі, бұл құрылымдық әдістеме негіздерін жақсы меңгеруге мүмкіндік береді:

- негізгі құрылымдардан алгоритм құрастыру;
- тізбектелген бөлшек әдісін қолдану.

Сондай-ақ, ЭЕМ-нің машиналық командалар деңгейіндегі архитектурасы туралы білімнің болғаны дұрыс (мектеп информатикасында оқытылған компьютердің модельдік мысалдарын білу жеткілікті, яғни нақты командалар немесе ассемблер тілдерін білу қажет емес). Бұл программалаудың негізгі түсініктерін (айнымалы, меншіктеу); «Транслятор жағдайына ену» және қателіктер жібермей, тіпті тілдің синтаксисінің ешбір дерегін ұмытпастан, программаны іске асыру барысында «кідіріп қалуға» болатын «тұзақтар» туралы болжам жасауды меңгеруге мүмкіндік береді. Шындығында, осы қасиеттер кәсіби программалаушыны әуесқой маманнан ажыратады.

Кәсіби маманның тағы бір қасиеті - программаның әдемілігін қабылдау мүмкіндігі, яғни жақсы жазылған программадан эстетикалық рахат алу. Бұл сезім қате программаны дұрыс жазылған программадан интуитивті түрде ажыратуға жиі көмектеседі. Дегенмен, программаны бағалаудың негізгі критерийі, әрине, адамның түйсігі емес, сауатты түрде ұйымдастырылған тестілеу болуы қажет.

Оқу үдерісі және программалауды практикалық жағынан меңгеру үш бөлікке бөлінеді:

- алгоритмдерді құру әдістерін үйрену;
- программалау тілін үйрену;
- белгілі бір программалау жүйелерін оқу және практикалық тұрғыда меңгеру.

Бірінші бөлім бойынша есептерді шығаруға оқулықтың 1-ші және 3-ші тараулары арналған. Шамалармен жұмыс жасау алгоритмдерінің негізгі мағлұматтары, олардың құрылуы мен принциптері 1-тарауда берілген. 3-тарау алгоритмдердің толық құрылысы үшін белгілі әдістерді сипаттайды, тестілеу программаларының мәселелерін зерттейді және алгоритмдердің күрделілігін бағалайды. 3-тарауда программаларды тестілеу мәселелері қарастырылған және алгоритмдерді толық құрудың кейбір белгілі әдістемелері баяндалады.

Турбо Паскаль және Delphi программалау тілдері оқулықтың сәйкесінше 2 және 4-тарауларында көрсетілген. Дегенмен, біз бұл оқулық бірінші кезекте Паскаль және Delphi тілдері туралы емес,

программалау бойынша оқу құралы екенін атап көрсетеміз, сондықтан сіз осы тілдердің толық сипаттамасын бұл оқулықтан таба алмайсыз, олар программалаудың бастапқы курсына қажетті көлемде берілген. Осы тілдердің егжей-тегжейлі сипаттамасын әдебиеттер тізімдері көрсетілген оқулықтардан табуға болады.

Берілген оқулықта тілдерге байланысты нақты программалау жүйелерімен жұмыс істеу бойынша нұсқаулықтар жоқ, олармен студенттер арнайы әдебиеттерді қолдана отырып ЭЕМ-де тәжірибелік жұмыстарды орындау барысында танысулары қажет.

1-тарау

АЛГОРИТМДЕУ ЖӘНЕ ПРОГРАММАЛАУДЫҢ НЕГІЗГІ ПРИНЦИПТЕРІ

Осы тарауда білесіңдер:

- программалау құралдарының көмегімен компьютерде есептерді шешу кезеңдерінің реттілігін;
- компьютерде есепті шешу алгоритмі дегеніміз не екенін;
- компьютердің қандай берілгендермен жұмыс жасайтынын;
- қандай командалар жиынтығының көмегімен (командалар жүйесі) компьютерде кез келген есепті шешу алгоритмін құруға болатынын;
- алгоритмдерді блок-сызбалар мен алгоритмдік тілде сипаттау ережелерін;
- үш негізгі алгоритмдік құрылымдарды - тізбектелген, тармақталу, цикл;
- алгоритмнің трассалануын;
- программалаудың логикалық негіздері; «логикалық шама», «логикалық өрнек», «логикалық амалдар» ұғымдарының мәнін;
- логикалық амалдар және логикалық өрнектерді есептеу ережелерінің орындалуын;
- көмекші алгоритм дегеніміз не және ол алгоритмдік тілдегі процедура түрінде қалай сипатталатынын;
- негізгі алгоритмде процедураны қалай шақыра аталатынын;
- құрылымдық программалау принциптерін;
- программалау технологиясы мен тілдердің даму тарихын;
- программалау тілдерінің классификациясын;

- трансляция дегеніміз не және трансляцияның қандай тәсілдері бар екенін;
- жоғарғы деңгейлі процедуралық программалау тілдерінің құрылымы мен құрамын;
- программалау тілдері синтаксисінің сипатталу тәсілдерін;

Сендер үйренесіңдер:

- алгоритмдік тіл мен блок-сызба түріндегі әр түрлі құрылымды алгоритмдерді (сызықтық, тармақталған, циклдік) сипаттауды;
- тармақталу мен циклдерге логикалық өрнектерді шарт түрінде жазуды;
- алгоритмдердің қадамдап орындалуын немесе трассалануын;
- процедураларды алгоритмдік тілде сипаттауды және оларды негізгі алгоритмге шақыруды.

1.1. АЛГОРИТМДЕР ЖӘНЕ ШАМАЛАР

Есептерді компьютерде шешу кезеңдері. Компьютерде кез-келген есепті шешу бойынша жұмыс алты қадамды қамтиды:

1. Есептің қойылымы.
2. Есепті формальдау.
3. Алгоритм құру.
4. Программалау тілінде программа құрастыру.
5. Программаны түзету және тексеру.
6. Есептеулерді жүргізу және алынған нәтижелерді талдау.

Бұл реттілік *ЭЕМ-де есептерді шешудің технологиялық тізбегі* деп аталады (бұл тізімнің тікелей программалауға 3...5 пункттері жатады).

Есептерді қойылымы кезеңінде тапсырмада *не берілген*, *нені табу керек екендігін* нақты анықтау керек. Есепті шешуге қажетті бастапқы деректердің толық жиынтығын сипаттау өте маңызды.

Формальдау кезеңінде есептер әдетте математикалық

формулалар, теңдеулер мен қатынастар тіліне аударылады. Есептің шешімі қандай да бір нақты объектінің, құбылыстың немесе процестің математикалық сипаттамасын қажет етсе, онда оны формалдау тиісті математикалық модельді алумен тең нәтижеге ие болады.

Үшінші кезең - бұл алгоритм құру. Тәжірибелі программалаушылар әдетте алгоритмдерді сипаттайтын арнайы құралдарға (блок-сызба, псевдокод) жүгінбей белгілі бір тілде бірден программа жазады, бірақ оқу мақсатында алынған алгоритмді программалау тіліне аудармас бұрын, алдымен бұл құралдарды пайдаланған абзал.

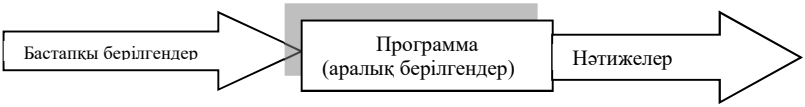
Алғашқы үш кезең компьютерсіз жұмыс істейді. Келесі екі кезең — белгілі бір программалау жүйесіндегі нақты тілде программалау болып табылады. Соңғы — алтыншы кезеңде әзірленген программа практикалық мақсаттарда қолданылады.

Осылайша, программалаушы алгоритмдерді құруды, программалау тілдерін білуді, сәйкес программалау жүйесінде жұмыс істей алуы тиіс.

Программалаушының негізгі кәсіби сауаттылығы ол - дамыған алгоритмдік ойлау қабілеті болып табылады.

Алгоритм ұғымы. Информатикадағы негізгі ұғымдардың бірі - алгоритм түсінігі болып табылады. Математикаға тәуелді «алгоритм» термині *лат.* Algorithmi, ортағасырдағы әйгілі шығыс математигі Мұхамед аль-Хорезмидің (787 - 850) есімін жазумен байланысты пайда болған. XII ғ. оның математикалық трактатының латын тіліндегі аудармасы жүзеге асырылды, соның нәтижесінде еуропалықтар ондық позициялық санау жүйесі және көпмәнді сандардың арифметикалық ережелері туралы біле бастады. Осы ережелерді сол кезде алгоритмдер деп аталды. Көпмәнді сандарды қосу, азайту, «баған» арқылы көбейту, «бұрыштап» бөлу, бұлар - математикадағы алғашқы алгоритмдер. Математикалық алгоритмдерге алгебралық түрендірулер мен теңдеулердің түбірін есептеу ережелерін жатқызуға болады.

Қазіргі уақытта алгоритм ұғымы кеңінен қарастырылады. *Алгоритм* — белгілі бір орындаушы басқаратын командалар тізбегі. Алгоритм ұғымы және алгоритмдерді құрастыру әдісімен оқушылар мектеп курсында келесідей орындаушылар мысалдары арқылы танысады: Робот, Тасбақа, Сызғыш және т.б. Бұл орындаушылар ешқандай есептеулер жүргізбейді. Олар экранда сызбалар жасап, лабиринттермен қозғалады, нысандарды бір орыннан екінші орынға тасымалдап апарады. Мұндай орындаушылар *әдетте ортада*



Сур.1.1. Программаға қатысты берілгендер деңгейлері

жұмыс істейтін орындаушылар деп аталады.

Информатиканың «Программалау» бөлімінде ЭЕМ жұмысын программалық басқару әдістері оқытылады, яғни орындаушы рөлін компьютер атқарады. Компьютер шамалармен жұмыс істейді — әр түрлі ақпараттық объектілермен: сандармен, символдармен, кодтармен және т.б., сондықтан компьютерді басқаруға арналған алгоритмдер *шамалармен жұмыс жасайтын алгоритмдер* деп аталады.

Берілгендер және шамалар. Компьютер жұмыс істейтін шамалар жиынтығы *берілгендер* деп аталады. Программаға қатысты есептеу барысында алынатын *бастапқы*, соңғы (нәтижелер) және *аралық* берілгендер болады (сур. 1.1).

Мысалы, $ax^2 + bx + c = 0$ квадрат теңдеуін шешу барысында бастапқы берілгендер — *a, b, c,* нәтижелер — теңдеу түбірлері болып табылатын x_1, x_2, және аралық берілгендер — $D = b^2 - 4ac$ теңдеу дискриминанты.

Программалауды сәтті меңгеру үшін келесі ережені білу керек: *әрбір шаманың ЭЕМ жадысында белгілі бір өз орны болады*, кейде ол жады ұяшығы деп аталады. Қазіргі ЭЕМ архитектурасы үшін «ұяшық» термині біраз ескірді, бірақ білім беру мақсатында оны пайдалану ыңғайлы.

Кез-келген шама үш басты қасиетке ие: *атауы, мәні* және *типі.* Шама процессордың командалық деңгейінде сақталатын жады ұяшығының мекен-жайы арқылы анықталады. Алгоритмдер мен программалау тілдерінде шамалар тұрақты және айнымалы болып бөлінеді. *Тұрақтылар* — мәндері өзгермейтін, алгоритмде өзіндік мәні бар шамалар болып табылады, мысалы: 15, 34.7, k, True және т.б. *Айнымалы шамалар* программаны орындау кезінде өздерінің мәндерін өзгерте алады және символдық атауларымен – идентификаторлармен ұсынылады, мысалы: X, S2, cod15 және т.б. Кез-келген тұрақты және айнымалы мәндер жадыда орнын алады және осы шамалардың мәндері осы ұяшықтағы екілік кодпен анықталады.

Енді шамалар типтері — *берілгендер типтері* — информатика курсындағы мәліметтер қорлары мен электрондық кестелерді оқу

барысында кездескен ұғымдар жөнінде баяндаймыз. Бұл ұғым программалаудағы негізгі ұғым болып саналады.

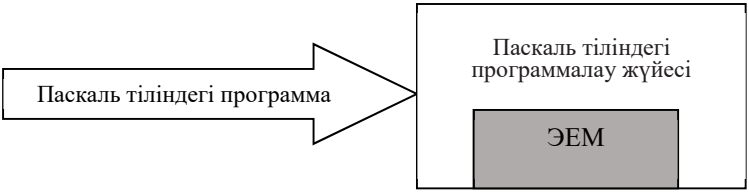
Әрбір программалау тілінің өз тұжырымдамасы және өзіндік берілгендер типтер жүйесі бар. Дегенмен, кез келген тіл кем дегенде негізгі берілгендер типтер жиынтығын қамтиды: *бүтін, нақты, логикалық* және *символдық.* Шамалар типтері рұқсат етілген мәндер жиынтығымен, рұқсат етілген операциялар жиынтығымен, ішкі ұсыныс формасымен сипатталады (1.1-кесте).

Тұрақты типтері мәтін мәнімен анықталады (яғни, мәтіндегі жазылу формасы бойынша) және айнымалы типтері айнымалылардың сипаттамаларында белгіленеді.

Құрылымы бойынша берілгендер *қарапайым* және *құрылымдық* болып бөлінеді. Сонымен қатар, скаляр деп аталатын қарапайым шамалар үшін мынадай тұжырымдама сәйкес келеді: *бір шама — бір мән,* ал құрылымдық шамалар үшін: *бір шама — бірнеше мәндердің жиынтығы.*

1.1- кесте. Негізгі берілгендер типтері.

Тип	Мағынасы	Амалдар	Ішкі көрініс
Бүтін	Кейбір диапазон бойынша оң бүтін және теріс сандар, мысалы: 23, -12, 387	Бүтін сандармен орындалатын арифметикалық амалдар: қосу, азайту, көбейту, бүтін бөлу және қалдықпен бөлу. Қатынас амалдары (<, >, = және т.б.)	Тұрақты нүктелі формат
Нақты	Кез-келген (бүтін және бөлшек) кейбір диапазондағы сандар, мысалы: 2,5, - 0,01, 45,0, 3,6	Арифметикалық амалдар. Қатынас амалдары	Жылжымалы нүктелі формат
Логикалық	True (ақиқат) False (жалған)	Логикалық амалдар: ЖӘНЕ (and), НЕМЕСЕ (or), ЖОҚ (not). Қатынас амалдары	1 — True; 0 — False
Символдық	Компьютер әліпбиінің кез-келген символдары, мысалы: a,5, +, $	Қатынас амалдары	Символдық кодтау кестесі, мысалы: ASCII — бір символ — 1 байт; Unicode — бір символ — 2 байт



Сур. 1.2. Паскаль тілінде программаны орындаушы рөліндегі компьютер.

Құрылымдық шамаларға жиымдар, жолдар, жиындар және т.б. жатады.

ЭЕМ — алгоритмдерді орындаушы. Әрбір алгоритм (программа) белгілі бір орындаушы, яғни оның командалық жүйесі шеңберінде жасалады. «Компьютерде программалау» тақырыбын оқу барысында қандай орындаушы болуы мүмкін? Жауабы айқын: мұнда орындаушы компьютер, немесе керісінше, компьютерлік кешен + программалау жүйесі (ПЖ). Программалаушы ПЖ бағытталған тілде программа құрастырады. 1.2-суретте орындаушының *кіріс тілі* Паскаль программалау тілі болып табылатындығы көрсетілген.

Программа қай тілде жазылатына қарамастан, ЭЕМ-де кез-келген есепті шешу алгоритмі келесі командалардан тұрады: меншіктеу, енгізу, шығару, көмекші алгоритмді шақыру, цикл және тармақталу.

Ары қарай алгоритмдерді сипаттау үшін мектеп информатика курсында өткен блок-сызбалар және алгоритмдік тіл (АТ) қолданылады.

1.2. СЫЗЫҚТЫҚ ЕСЕПТЕУ АЛГОРИТМДЕРІ

Есептеу алгоритмдеріндегі негізгі қарапайым әрекет бұл *айнымалы шамаларға мәндерді меншіктеу.*

Егер тұрақты мәні оның жазба түріне байланысты анықталса, онда айнымалы шама екі тәсілмен жүзеге асырылатын тек меншіктеу нәтижесінен кейін ғана нақты мәнге ие болады: меншіктеу және енгізу командаларының көмегі арқылы. Бір мысал қарастырайық.

Мектептегі математика оқулығында екі қарапайым бөлшекті бөлу ережесі келесідей сипатталады:

1) бірінші бөлшектің алымын екінші бөлшектің бөліміне көбейту;
2) бірінші бөлшектің бөлімін екінші бөлшектің алымына көбейту;
3) бөлімнің орындалу нәтижесі — п. 1, ал алымның орындалу нәтижесі — п. 2. болатын бөлшекті жазып алу

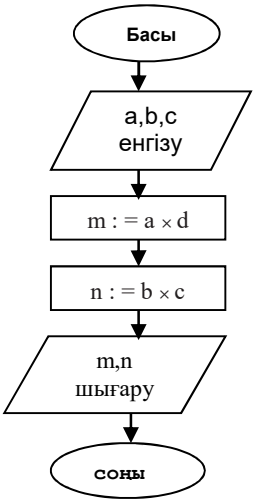
Алгебралық түрде бұл мысалды былай жазып көрсетеміз:

$$\frac{a}{b}:\frac{c}{d}=\frac{d\cdot d}{d\cdot c}=\frac{m}{n}$$

Енді алгебралық өрнекте қолданылған айнымалылардың белгіленуін сақтай отырып, ЭЕМ үшін бөлшектерді бөлу алгоритмін құрайық.

Мұнда бастапқы берілгендер бүтін санды айнымалылар — *a, b, c, d,* ал нәтиже — *m* және *n* бүтін шамалары. Бөлшектерді бөлу алгоритмінің блок-сызбасы 1.3-сур. көрсетілген. Берілген алгоритмнің алгоритмдік тілде жазылуы төмендегідей болады:

алг бөлшектерді бөлу
басы
бүт a, b, c, d, m, n
енг a, b, c, d
m : = a × d
n : = b × c
шығ m, n
соңы



1.3-сур. Бөлшектерді бөлу алгоритмінің блок-сызбасы

Меншіктеу командасы мына түрде көрсетіледі:

`айнымалы:= өрнек`

«:=» белгісі «меншіктеу» деп оқылады. Меншіктеу командасы компьютермен орындағанда келесідей мағына береді:

1) *өрнек* есептеледі;
2) алынған нәтиже *айнымалыға* меншіктеледі.

Берілген алгоритмде меншіктеудің екі командасы көрсетілген. Меншіктеу командалары блок-сызбаларда тіктөртбұрыш арқылы сипатталады. Мұндай блок *есептеу* деп аталады. Алгоритмді сипаттау барысында өрнектерді жазудың қатаң ережесін сақтау міндетті емес, бұл әлі қатаң синтаксисті программалау болмағандықтан, оларды қарапайым математикалық түрде жазуға болады.

Қарастырылып отырған алгоритмдегі енгізу командасы:

```
a, b, c, d енгізу
```

Блок-сызбаларда енгізу командалары параллелограммның ішіне — енгізу-шығару блогында жазылады. Бұл команданы орындау барысында процессор жұмысты тоқтатып, орындаушының әрекетін күтеді. Орындаушы енгізу құрылғысы (пернетақта) арқылы айнымалылардың мәнін жазып, <Enter> пернесін басу керек. Мәндерді енгізу тізіміндегі орналасу реті бойынша жазу керек. Әдетте енгізу командасының көмегімен бастапқы берілгендер мәні меншіктеледі, ал меншіктеу командасы аралық және соңғы шамаларды алуда қолданылады.

Есепті шешудегі компьютермен алынған нәтижелер қолданушыға хабарлануы керек, сол себепті шығару командасы тағайындалған:

```
шығару  m, n
```

Бұл команданың көмегімен нәтижелер экранда немесе басып шығару құрылғысы арқылы қағазда көрсетіледі.

Меншіктеу операторы есептеу алгоритмдеріндегі ең маңызды амал болып табылғандықтан, оны толығырақ қарастырайық.

Төрт меншіктеу командаларының рет-ретімен орындалуын қарастырамыз, мұнда екі айнымалы шама - a, b қатысады. Әрбір меншіктеу командасы үшін 1.2-кестеде бұл команда орындалғаннан кейін тағайындалатын айнымалы мәндері көрсетілген.

Бұл мысал үш негізгі меншіктеу қасиетін көрсетеді:

- айнымалыға мән меншіктелмей, ол анықталмаған күйде қалады;
- айнымалыға меншіктелген мән осы айнымалы келесі тағайындалғанға дейін сақталады;
- айнымалыға меншіктелген жаңа мән оның алдыңғы мәнін ауыстырады.

1.2-кесте. Меншіктеу командасының орындалу барысындағы айнымалы мәндерінің өзгеруі

Команда	*a*	*b*
$a := 1$	1	—
$b := 2 \times a$	1	2
$a := b$	2	2
$b := a + b$	2	4

1.3-кесте. Айнымалылар арасындағы мәндердің алмасуы

Команда	*X*	*Y*	*Z*
X, *Y енгізу*	1	2	—
$Z := X$	1	2	1
$X := Y$	2	2	1
$Y := Z$	2	1	1

Программалауда жиі қолданылатын алгоритмді қарастырайық. Екі шама берілген: *X* және Y. Олардың арасында мәндерді алмастыруды жүзеге асыру қажет. Мысалы, егер бастапқыда $X = 1$, $Y = 2$ болса, алмастырудан кейін $X = 2$, $Y = 1$ болу керек.

Бұл есеп келесі жағдайға ұқсас. Екі стақан берілген: біреуіне - сүт, екіншісіне – су құйылған. Олардың өзара мәндерін алмастыру қажет. Бұл жағдайда үшінші стақан қажет. Алмасу бойынша әрекеттер тізбегі келесідей болады:

1) 1-ші стақаннан сүтті 3-шіге құю;
2) 2-ші стақаннан 1-ші стақанға суды құю;
3) сүтті 3-ші стақаннан 2-шіге құю.

Осылайша, екі айнымалының мәндерін алмастыру үшін үшінші қосымша айнымалы қажет. Біз оны Z деп белгілейміз. Содан кейін мәндерді айырбастау есебін үш тапсырма командаларын дәйекті орындау арқылы шешуге болады (1.3-кесте).

Бұл мысал айнымалылардың арасындағы алмастыруды нақты сипаттай алмайды, өйткені сұйықтық құю кезінде стақанның біреуі бос болады. Меншіктеу командаларын орындау нәтижесінде (X: = Y) оң жақта (Y) айнымалы өз мәнін сақтап қалады.

Бөлшектерді бөлу алгоритмі сызықты құрылымға ие, яғни онда барлық командалар қатаң анықталған ретпен әрқайсысы бір рет орындалады. Сызықты алгоритм меншіктеуден, енгізуден, шығудан және қосалқы алгоритмдерді (одан әрі талқыланатын) шақырудан тұрады.

Блок-сызбаларды қолдана отырып алгоритмдерді сипаттау барысында, берілгендер типтері әдетте көрсетілмейді (бірақ көзделеді). Оқу алгоритмдерінде (АТ үшін) барлық айнымалы мәндер үшін берілгендер типтері айқын көрсетіледі және олардың сипаттамасы алгоритм тақырыбынан кейін жазылады. Бұл жағдайда

келесі белгілеулер пайдаланылады: **бүт** - бүтін, **нақ** - нақты, **лит** - символдық (литерлік), **лог** - логикалық. Бөлшектерді бөлу алгоритмінде барлық айнымалылар бүтін типті болады.

1.3. ЕСЕПТЕУ АЛГОРИТМЕРІНДЕГІ ТАРМАҚТАЛУ МЕН ЦИКЛДЕР

Квадрат теңдеуді шешу алгоритмін құрастырайық

$$ax^2 + bx + c = 0.$$

Бұл есеп математика пәнінен жақсы таныс. Мұндағы бастапқы берілгендер *a, b, c* коэффициенттері, ал жалпы жағдайдағы шешімдер — (x_1 және x_2) екі түбірлер, олар келесі формула бойынша есептеледі:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Бұл программадағы қолданылатын барлық шамалар нақты типті болып келеді. Квадрат теңдеуді шешу алгоритмі келесі түрде сипатталады:

```
алг  квадрат теңдеудің түбірлері
нақ  a, b, c, d, x1, x2
басы енгізу a, b, c
     d : = b^2 - 4ac
     x 1  : =  ( -b  +  √d ) / ( 2 a )
     x 2  : =  ( - b  -  √d ) / ( 2 a )
     ш ы ғ а р у   x1, x2
соңы
```

Алгоритмнің толық еместілігі бірден көзге түседі. Ол сапалы алгоритмдер үшін қажетті ең маңызды бастапқы берілгендерге қатысты әмбебаптылық қасиетке ие емес. *Бастапқы берілгендердің мәндері қандай болмасын, алгоритм белгілі бір нәтижеге келіп, соңына дейін орындалуы қажет*. Нәтиже сандық жауап болуы мүмкін, немесе мұндай берілгендер үшін шешім жоқ екендігі туралы хабарлама болуы мүмкін. Қандайда бір амалдың орындалмауы нәтижесінде алгоритмнің ортасында тоқтауға жол берілмейді.

Программалау әдебиетінде мұндай қасиет *алгоритмнің нәтижелілігі* деп аталады. (кез-келген жағдайдағы нәтиженің болуы).

Әмбебап алгоритм құру үшін алдымен есептің математикалық мазмұнын мұқият талдау қажет.

Квадрат теңдеудің шешімі *a, b, c* коэффициенттерінің мәндеріне тәуелді.

Осы есепті талдай отырып, тек нақты түбірлерді табумен шектелейік:

егер $a = 0$, $b = 0$, $c = 0$ болса, онда x-тің кез-келген мәні — теңдеудің шешімі болады;

егер $a = 0$, $b = 0$, $c = 0$ болса, онда теңдеудің шешімі болмайды;

егер $a = 0$, $b \neq 0$ болса, онда бұл бір шешімі болатын сызықтық теңдеу $x = -c/b$;

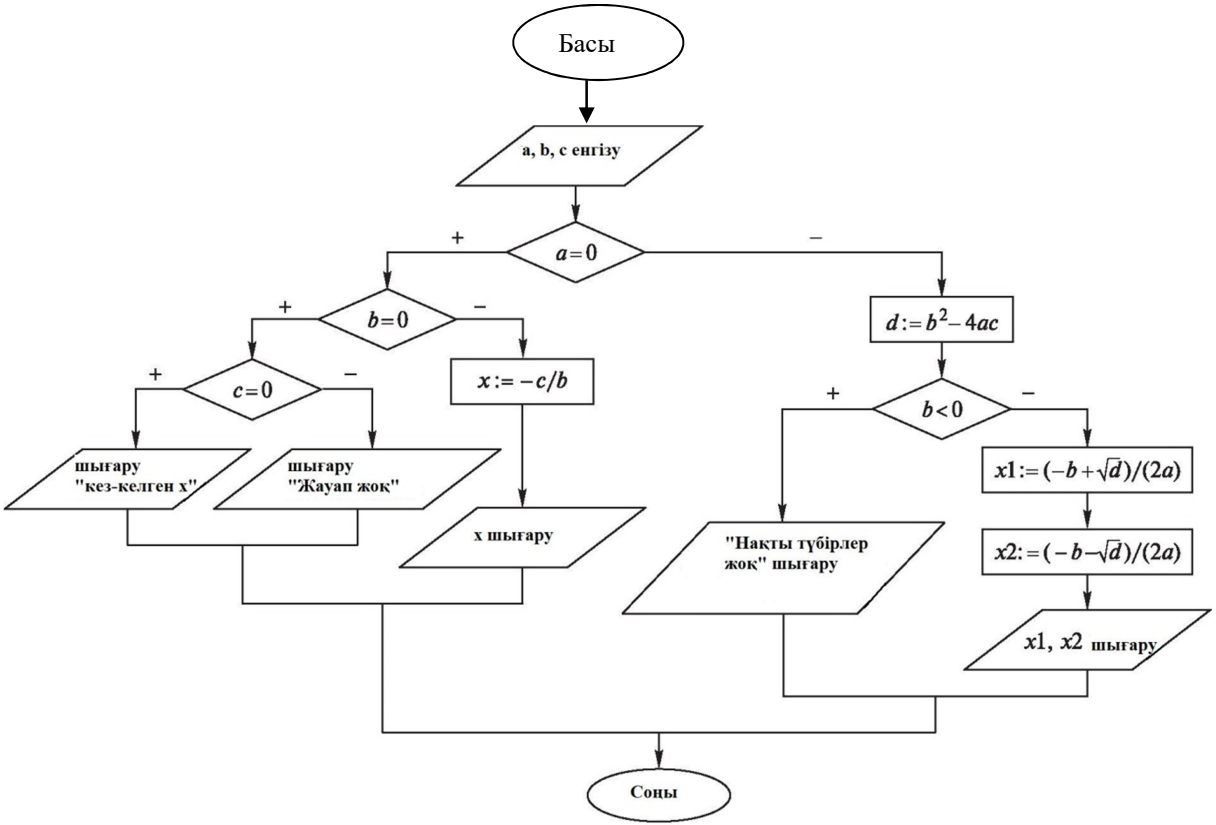
егер $a \neq 0$ және $d = b^2 - 4ac \geq 0$ болса, теңдеудің x_1, x_2 екі нақты түбірі болады;

егер $a \neq 0$ және $d < 0$ болса, онда теңдеудің нақты түбірлері болмайды.

Квадрат теңдеуді шешу алгоритмінің блок-сызбасы 1.4-сур. көрсетілген.

Бұл алгоритмнің алгоритмдік тілде жазылуы мына түрде болады:

```
алг квадрат теңдеудің түбірлері
нақ a, b, c, d, x1, x2
басы енгізу a, b, c
    егер a = 0
    онда   егер b = 0
           онда    егер c = 0
                   онда шығару «Кез-келген  x — шешім»
                   әйтпесе шығару «Шешімі жоқ»
                   тс
           әйтпесе x := -c/b
                   шығару x
           тс
    әйтпесе  d := b^2-4ac
             егер d < 0
             онда шығару «Нақты түбірлер жоқ»
    әйтпесе  x 1  : = ( -b  +  √d ) / ( 2 a ) ;
             x 2  : = ( -b  - √d) / ( 2 a )
             ш ы ғ а р у   «x1 =», x1, «x2 =», x2
    тс
тс
соңы
```



1.4-сур. Квадрат теңдеуді шешу алгоритмінің блок-сызбасы

Бұл алгоритмде *тармақталудың құрылымдық командасы* бірнеше рет пайдаланылған және блок-сызба түріндегі жалпы көрінісі 1.5-суретте көрсетілген Алгоритмдік тілде тармақталу командасы келесі түрде жазылуы мүмкін:

егер шарт
онда серия 1
әйтпесе серия 2
тс

Тармақталу командасының жоғарыда көрсетілген блок-сызбасына сәйкес, бірінші шарт тексеріледі (логикалық өрнегі есептеледі). Егер шарт ақиқат болса, онда «Иә» (оң тармақ) - «1-серия» бағыты көрсетілген командалар тізбегі орындалады. Әйтпесе, «2-серия» командаларының қарама-қарсы тармағы орындалады.

АТ-де шарт «егер» қызметші сөзінен кейін жазылады, оң жақ тармақ «онда» сөзінен кейін, қарама-қарсы тармақ «әйтпесе» қызметші сөзінен кейін жазылады. «тс» әріптері тармақталу соңы деген мағына береді.

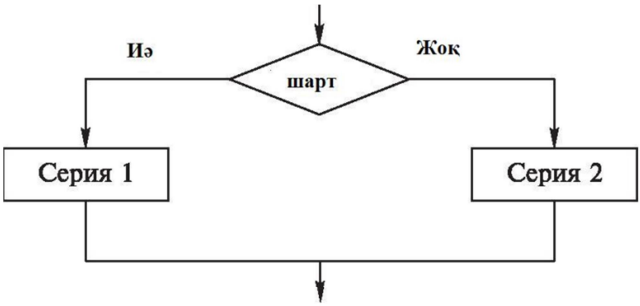
Егер бір тармақталудың тармағында басқа тармақтар болса, онда алгоритмде қабаттасқан тармақталу құрылымы бар болады. Сондай-ақ, «иә» және «жоқ» сөздерінің орнына қысқаша «+» және «-» белгілері қолданылатын квадрат теңдеуді шешу алгоритмі дәл осындай құрылымды қамтиды (1.4-суретті қараңыз).

Келесі есепті қарастырайық: n оң бүтін саны берілген. n! есептеу керек (n-факториал). Факториал анықтамасын еске түсірейік:

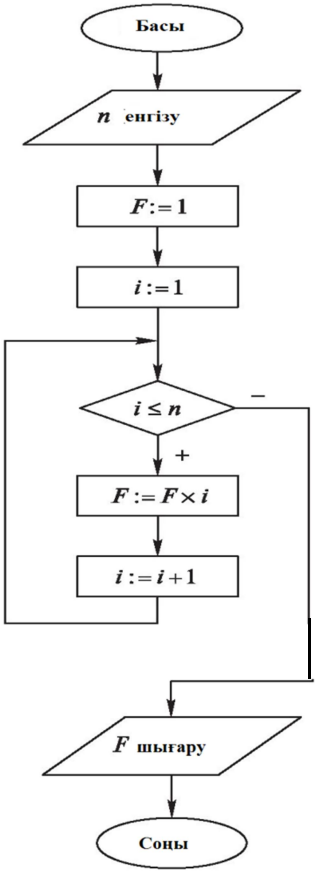
$$\text{n!}=\begin{cases}1, & \text{егер } n=0;\\ 1\times 2\times \ldots \times n, & \text{егер } n\geq 1.\end{cases}$$

Нақты типті үш айнымалы n! есептеу алгоритмінің блок-сызбасы: n — аргумент, i — аралық айнымалы, F — нәтиже. 1.6-сурет. Берілген алгоритмнің дұрыстығын тексеру үшін трассалану 1.4-кестесін құрайық. Мұнда алгоритмге кіретін нақты бастапқы берілгендер мәндері қадам бойынша қадағаланып отырады. Кесте n = 3 болған жағдайға құрастырылған.

Жоғарыда келтірілген алгоритмнің трассалануы оның дұрыстығын дәлелдейді. Енді алгоритмді алгоритмдік тілде жазып алайық:



1.5-сур. Тармақталу командасының блок-сызбасы



1.6-сур. n! алгоритмін есептеудің блок-сызбасы

1.4-кесте. n!-ды есептеу алгоритмінің трассалануы

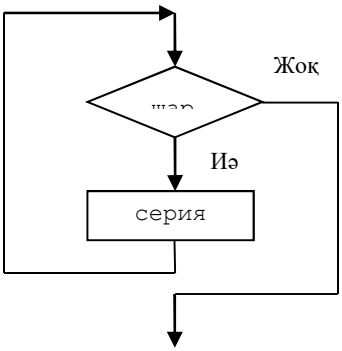
Қадам	*n*	*F*		Шарт
1	3			
2		1		
3			1	
4				1 < 3, иә
5		1		
6			2	
7				2 < 3, иә
8		2		
9			3	
10				3 < 3, иә
11		6		
12			4	
13				4 < 3, жоқ
14		Шығару		

```
алг  факториал
бұт  n, i, F
басы  n енгізу
      F : = 1; i : = 1;
      әзір i ≤ n, қайталау
      цб   F : = F × i
           i : = i + 1
      цс
      F шығару
соңы
```

Берілген алгоритмде қайталау құрылымы қамтылған. Мұнда циклдердің құрылымдық командалары «Әзірше», немесе алғышартты циклдер қолданылған. «Әзірше» циклінің блок-сызбасы 1.7-суретте көрсетілген. АТ-де төмендегідей жазылады:

```
әзір        шарт, қайталау
цб
      серия
цс
```

Рис. 1.7. Блок-схема команды цикла «Пока»



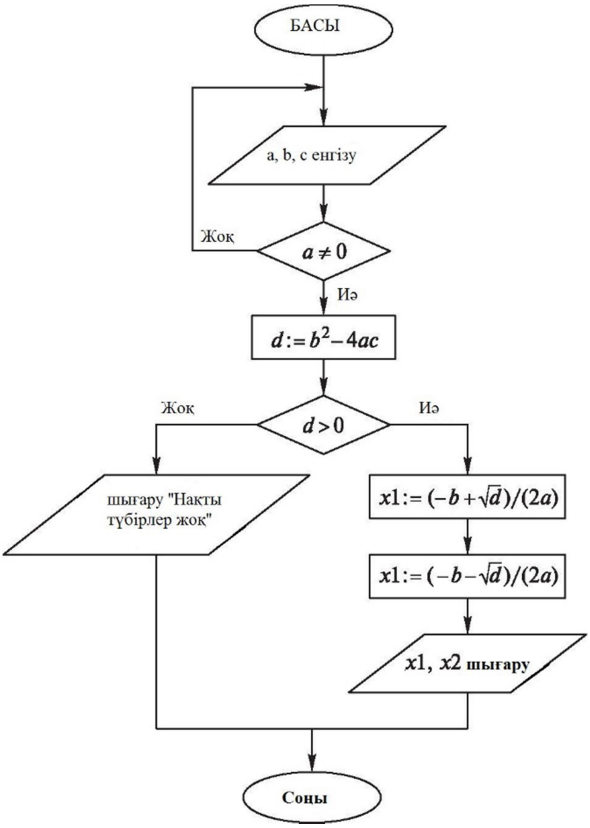
Командалар сериясының орындалуы (цикл денесі) цикл шарты ақиқат болғанша қайталанады. Шарт жалған болған жағдайда циклдің орындалуы тоқтатылады. «цб» және «цс» қызметтік сөздері сәйкесінше циклдің басы мен соңын білдіреді.

Циклдің алғышарты — бұл циклдік алгоритмді ұйымдастырудағы негізгі форма бірақ, одан өзге циклдің ілесушарты да бар.

Квадрат теңдеуді шешу алгоритміне оралайық, оны келесі позицияда қарастырамыз: егер $a = 0$ болса — бұл квадрат теңдеу болмайды және оны шешпей-ақ қоюға болады. Бұл жағдайда қолданушы берілгендерді енгізу барысында қателесті деп есептеп, енгізу әрекетін қайталаймыз (1.8-сур.). Басқаша айтқанда, алгоритмде қолданушыға қателерді жөндеу мүмкіндігін ұсына отырып, бастапқы берілгендердің сенімділігін бақылау алдын-ала қарастырылады. Мұндай бақылаудың болуы — сапалы программаның тағы бір жақсы белгісі.

АТ-де берілгендерді енгізуді бақылай отырып, квадрат теңдеуді шешу алгоритмі келесі түрде болады:

```
алг квадрат теңдеу
нақ a, b, c, d, x1, x2
басы
    қайталау
            a, b, c енгізу
        дейін   a ≠0
            d : = b^2 - 4ac
        егер d ≥ 0
        онда x 1 : = ( -b + √d ) / ( 2 a )
             x 2  :=  ( - b - √d )  / ( 2 a )
            шығару x1, x2
        әйтпесе
              «Нақты түбірлер жоқ» шығару
        кв
соңы
```



1.8-сур. Берілгендерді енгізуді бақылай отырып, квадрат теңдеуді шешу алгоритмінің блок-сызбасы.

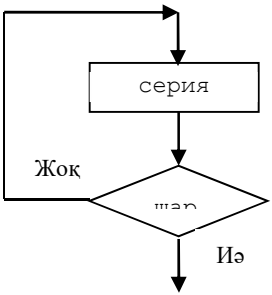
«Дейін» ілесу шарты циклдерінің құрылымдық командаларының блок-сызбасы 1.9 сур. көрсетілген. АТ бойынша берілген команданы келесі түрде жазуға болады:

қайталау
серия
дейін шарт

Мұнда циклдің аяқталу шарты қолданылады, яғни ол ақиқат болған жағдайда цикл жұмысын тоқтатады.

Келесі есепті шешу алгоритмін құрайық: *M* және N екі натурал сандары берілген. Олардың ең үлкен ортақ бөлгішін есептеу керек — ЕҮОБ (M, N).

1.9.-сур. «Дейін» цикл командасының блок-сызбасы.



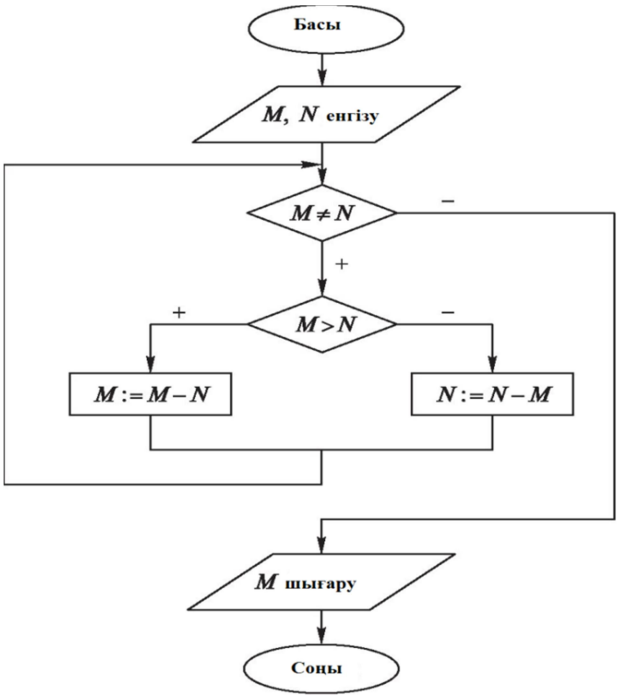
Бұл есеп белгілі Евклид алгоритмі әдістерінің көмегімен шығарылады.

Бұл әдістің негізгі идеясы мынадай: егер *M* > *N,* онда ЕҮОБ (M, *N)* = ЕҮОБ *(M - N,* N). Бұны өз беттеріңше дәлелдеп көріңдер. Берілген алгоритмді шешудегі негізгі өзге тұжырым:

ЕҮОБ (M,M*)* = M. Жазбаша бұл алгоритмді келесі нұсқау бойынша сипаттауға болады:

1) егер сандар тең болса, олардың ортақ мәнін нәтиже ретінде алу, әйтпесе алгоритмнің орындалуын жалғастыру;

2) сандардың үлкенін анықтау;



1.10-сур. Евклид алгоритмінің блок-сызбасы

3) үлкен санды үлкен және кіші мәннің айырымымен алмастыру;
4) 1.п. оралу

Евклид алгоритмінің блок-сызбасы 1.10-сур. көрсетілген. АТ-де келесідей көрсетіледі:

```
алг Евклид
бұт M, N
басы M, N енгіз
    әзірше M ≠ N, қайталау
    цб егер M > N
       онда M : = M - N
       әйтпесе N : = N - M
    тс
    цс
соңы
```

Евклид алгоритмі қабаттасқан тармақталу цикл құрылымын қамтиды. Өз беттеріңмен M = 18, N = 12 болғандағы алгоритмнің трассалануын орындаңдар. Нәтижесінде ЕҮОБ = 6 болу керек.

1.4. АЛГОРИТМДЕУДІҢ ЛОГИКАЛЫҚ НЕГІЗДЕРІ

Программалауға негізі алгебра логикасы және пікірлерді есептеу болып табылатын математикалық логика пәнінің тікелей қатысы бар. Пікір айтылған уақытта оны ақиқат немесе жалған екенін бірден айтуға болады. Мысалы, «Ай - Жердің серігі» деген ұғым ақиқат, «5>3» ақиқат, «Мәскеу - Қытайдың астанасы» - жалған, «1 = 0» - жалған. АҚИҚАТ және ЖАЛҒАН логикалық мәндер болып табылады. Берілген тұжырымдамалардың логикалық мәндері бірегейлі анықталған. Басқаша айтқанда, олардың мәні *логикалық тұрақтылар* болып табылады.

X <0 теңсіздігінде x - айнымалы болып табылады, яғни x мәніне байланысты ол ақиқат немесе жалған болуы мүмкін. Осылайша логикалық айнымалы түсінігі енгізілді. Алгоритмдердің сипаттауда, сондай-ақ әр түрлі программалау тілдеріндегі программаларда логикалық айнымалылар берілгендердің логикалық типіне сәйкес келетін символдық атаулармен белгіленуі мүмкін. Басқаша айтқанда, A, B, X, Y және т.б. — логикалық типтегі айнымалы екені белгілі болса, олар тек АҚИҚАТ немесе ЖАЛҒАН мәндерді қабылдай алады дегенді білдіреді.

XIX ғ. ортасында ағылшын математигі Джордж Буль математикалық логиканың формалды аппаратының негіздерін ойлап тапты. Оның құрметіне пікірлерді есептеу бульдық алгебра, ал логикалық шамалар бульдық шамалар деп аталады.

Логикалық өрнек (логикалық формула) — бұл қарапайым немесе күрделі пікір болып табылады. Күрделі пікір қарапайым пікірлерден құралған логикалық амалдардың (байланыстардың) көмегімен жасалады.

Логикалық амалдардың негізгі үш түрі бар: терістеу, конъюнкция (логикалық көбейту) және дизъюнкция (логикалық қосу).

Терістеу математикалық логикада «¬» таңбасымен белгіленеді және ЕМЕС деп оқылады. Бұл бірорынды амал. Мысалы, ¬ (x = y) пікірі келесі түрде оқылады: (x тең y) ЕМЕС, яғни *x* тең болмаса *y*-ке, пікірдің мәні ақиқат болады да, *x* тең болса *y-ке*, онда пікірдің мәні жалған болады. Терістеу логикалық шаманың мәнін қарама-қарсы мәнге өзгертеді.

Конъюнкция & таңбасымен белгіленеді де, ЖӘНЕ деп оқылады. Бұл екіорынды амал. Мысалы, (x > 0) & (x < 1) жазбасы егер *x* ϵ (0, 1) болса, берілген логикалық формуланың мәні АҚИҚАТ болады, әйтпесе ЖАЛҒАН болады. Екі операнд та ақиқат болса, сәйкесінше конъюнкция нәтижесі де АҚИҚАТ мәнін қабылдайды.

Дизъюнкция ∪ таңбасымен белгіленіп, НЕМЕСЕ деп оқылады. Мысалы, (x = 0) ∪ (x = 1) жазбасы егер *x* — екілік сан (0 немесе 1) болса, онда формула ақиқат мәнді қабылдайды. Егер кем дегенде бір операнд АҚИҚАТ болса, сәйкесінше дизъюнкция нәтижесі АҚИҚАТ болады.

Қарастырылған логикалық амалдардың орындалу ережесі *ақиқат кестесі* деп аталатын 1.5-кестеде көрсетілген (мұндағы A және *B* — логикалық шамалар, ал «А» және «Ж» әріптерімен сәйкесінше АҚИҚАТ және ЖАЛҒАН мәндері белгіленген).

1.5-кесте. Ақиқат кестесі

№	A	B	ЕМЕС A	A ЖӘНЕ *B*	A НЕМЕСЕ *B*
1	А	А	Ж	А	А
2	А	Ж	Ж	Ж	А
3	Ж	А	А	Ж	А
4	Ж	Ж	А	Ж	Ж

Логикалық өрнектерде амалдардың орындалу реті келесі кезекпен жүзеге асырылады: терістеу, конъюнкция, дизъюнкция. Сонымен қатар, логикалық формулаларда амалдардың орындалу ретін жақшалардың көмегімен алмастыруға болады:

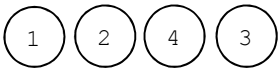
(А ЖӘНЕ *В)НЕМЕСЕ* (ЕМЕС А ЖӘНЕ В) НЕМЕСЕ (ЕМЕС А ЖӘНЕ ЕМЕС В).

Логикалық формуланың мәнін есептеуге мысал қарастырайық,

ЕМЕС *X* ЖӘНЕ *Y* НЕМЕСЕ Х ЖӘНЕ *Z*

мұндағы логикалық айнымалылардың мәні: *X* = ЖАЛҒАН, *Y* = АҚИҚАТ, *Z* = АҚИҚАТ.

Берілген өрнектегі амалдардың орындалу ретін сандармен белгілейік:



ЕМЕС *X* ЖӘНЕ *Y* НЕМЕСЕ *X* ЖӘНЕ *Z*.

Ақиқат кестесін пайдаланып есептеуді қадамдап орындаймыз:

1) ЕМЕС ЖАЛҒАН = АҚИҚАТ;
2) АҚИҚАТ ЖӘНЕ АҚИҚАТ = АҚИҚАТ;
3) ЖАЛҒАН ЖӘНЕ АҚИҚАТ = ЖАЛҒАН;
4) АҚИҚАТ НЕМЕСЕ ЖАЛҒАН = АҚИҚАТ.

Қарастырылған логикалық формуланың мәні — АҚИҚАТ.

Келесі тұжырымды қарастырайық: «*X*-тің мәні 0 мен 1 интервалында орналасқан». Бұл тұжырымды теңсіздіктер жүйесі түрінде жазуға болады

$$0 \le X \le 1.$$

Сәйкес логикалық өрнек келесі түрде болады

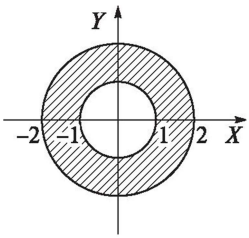
$$(X \ge 0) \text{ ЖӘНЕ } (X \le 1).$$

Енді тапсырманы қиындатамыз, логикалық өрнек түрінде келесі тұжырымды жазамыз: Сақинада орналасқан координаттар басы центрі болып келетін, ішкі радиусы 1-ге, сыртқы радиусы 2-ге тең координаттары (X, *Y*) болатын жазықтықтағы нүкте берілген (1.11-сур.). Бұл өрнек мына түрде сипатталады:

$$(X^2 + Y^2 \ge 1) \text{ ЖӘНЕ } (X^2 + Y^2 \le 4).$$

Конъюнкция амалы келесі түрде орындалады: $(X^2 + Y^2 \geq 1)$ бірінші тұрған теңсіздіктің шарттарын қанағаттандыратын нүктелер жиыны таңдалады және пайда болған жиыннан $(X^2 + Y^2 \leq 4)$ екінші теңсіздіктің шарттарын қанағаттандыратын ішкі жиындар ерекшеленеді де олар соңғы нәтиже болады.

Басқаша айтқанда, ізделіп отырған жиын екі жиынның қиылысы, олардың біреуі бірінші теңсіздікті қанағаттандырса, екіншісі соңғы теңсіздікті қанағаттандырады. Конъюнкция — логикалық көбейту, яғни осындай қиылысуларды есептейтін амал болып табылады.



1.11-сур. Сақиналық аймақ тапсырмасы

Келесі тұжырымды логикалық өрнек түрінде жазайық: Сақинаның сыртында орналасқан координаттар басы центрі болып келетін, ішкі радиусы 1-ге, сыртқы радиусы 2-ге тең координаттары (*X*, *Y*) болатын жазықтықтағы нүкте (1.11-сур.). Бұл өрнек мына түрде жазылады:

$$(X^2 + Y^2 < 1) \text{ НЕМЕСЕ } (X^2 + Y^2 > 4).$$

Дизъюнкция амалының орындалуы келесі түрде жүзеге асырылады: $(X^2 + Y^2 < 1)$ бірінші теңсіздікті қанағаттандыратын нүктелер жиыны таңдалады және оған $(X^2 + Y^2 > 4)$ екінші теңсіздікті қанағаттандыратын нүктелер жиыны қосылады. Осылайша, дизъюнкция амалының басқа атауының мағынасы анықталады, дизъюнкция — логикалық қосу, яғни екі жиынның қосындысы.

Соңғы есепті басқаша шығаруға болады. Сәйкес логикалық өрнекті бірінші өрнекке терістеу амалы ретінде жазуға болады, себебі сақинаның сыртында орналасқан нүктелер сақинаның ішінде ЕМЕС.

$$\text{ЕМЕС } ((X^2 + Y^2 \geq 1) \text{ ЖӘНЕ } (X^2 + Y^2 \leq 4)).$$

Сәйкес екі логикалық өрнектердің теңдіктері заңды:

$$\text{ЕМЕС } ((X^2 + Y^2 \geq 1) \text{ ЖӘНЕ } (X^2 + Y^2 \leq 4)) = (X^2 + Y^2 < 1) \text{ НЕМЕСЕ}$$

$$(X^2 + Y^2 > 4).$$

Бұл мысалдан келесі түрлендіру ережесі түсінікті болады: Конъюнкциямен байланысқан екі теңсіздікке қолданылған терістеу амалы осы теңсіздіктердің таңбаларын қарама-қарсыға, ал

конъюнкцияны — дизъюнкцияға (немесе дизъюнкцияны конъюнкцияға) өзгертеді.

Күрделі логикалық өрнек қолданылған алгоритм мысалын қарастырайық. Үшбұрыштың қабырғаларының ұзындығы арқылы оның ауданын есептейміз.

Бұл есепті шығару үшін Герон формуласын қолданамыз:

$$\sqrt{p(p-a)(p-b)(p-c)}$$

мұндағы $p = (a + b + c)/2$ — үшбұрыш периметрінің жартысы.

Бастапқы берілгендер үшбұрыштың қабырғаларының негізгі қатынастарын қанағаттандыру керек: әрбір қабырғаның ұзындығы қалған екі қабырғалардың ұзындықтарының қосындысынан кем болуы қажет.

Үшбұрыштың ауданын есептеу алгоритмі тармақталған құрылым болып табылады. Тармақталудың құрылымдық командасына шартты дұрыс емес бастапқы берілгендердің барлық нұсқаларын «сүзуге» мүмкіндік беретін күрделі логикалық өрнек түрінде жазайық:

```
алг Герон
нақ A, B, C, P, S
басы енгізу A, B, C
егер (A > 0) ЖӘНЕ (B > 0) ЖӘНЕ (C > 0)ЖӘНЕ (A + B > C)
      ЖӘНЕ (B + C > A) ЖӘНЕ (A + C > B)
      онда
            P := (A + B + C ) / 2
            S : = түбір (P *(P - A) * (P - B) * (P - C))
             шығару «Аудан =», S
      әйтпесе шығару «Бастапқы берілгендер қате»
      тс
соңы
```

1.5. КӨМЕКШІ АЛГОРИТМДЕР МЕН ПРОЦЕДУРАЛАР

Алгоритмдер теориясында *көмекші алгоритмдер* ұғымы қолданылады, яғни негізгі шығарылатын есептің кейбір ішкі есептерін шешу алгоритмдері. Мұндағы бастапқы есепті шешу

алгоритмі *негізгі* деп аталады.

Мысалға келесі есепті қарастырайық. Бүтін көрсеткішті дәрежелі функцияны есептеу алгоритмін құру керек: $y = x^k$, мұндағы $к$ — бүтін сан; $x \neq 0$.

Алгебрада мұндай функция келесі түрде анықталған:

$$x^n = \begin{cases} 1 \text{ при } n = 0 \\ \dfrac{1}{x^{-n}} \text{ при } n < 0 \\ x^n \text{ при } n > 0 \end{cases}$$

Берілген есепте ішкі есеп ретінде санды бүтін оң дәрежеге алуды қарастыруға болады.

$1/x^{-n} = (1/x)^{-n}$ есепке ала отырып, есепті шешудің негізгі алгоритмін жазайық:

```
алг дәрежелі функция
бүт n; нақ x, y
басы енгізу x, n
      егер n = 0 онда
      y:=1
      әйтпесе егер n >0
            онда ДӘРЕЖЕ (x, n, y) әйтпесе
            ДӘРЕЖЕ (1/x, -n, y)
            тс
      тс
      шығару y
соңы
```

Негізгі алгоритмде көмекші алгоритмді шақыру командасы ДӘРЕЖЕ деген атаумен екі рет қайталанып тұр. Көмекші алгоритмді шақыру командасындағы жақшада тұрған шамалар *нақты параметрлер* деп аталады.

Алгоритмдік тілде көмекші алгоритмдер процедура түрінде жазылады. АТ-де ДӘРЕЖЕ процедурасын сипаттап көрейік:

```
процедура ДӘРЕЖЕ (нақ a, бүт k, нақ z)
бүт i
басы z := 1; i := 1
     әзірше i ≤ k, қайталау
     цб z := z × a
        i := i + 1
     цс
соңы
```

Көмекші алгоритмнің тақырыбы «процедура» сөзімен басталып, одан кейін осы процедураның атауы және жақшаның ішінде — формалды параметрлер тізімі жазылады. Бұл тізімге типтері көрсетілген айнымалы аргументтер және айнымалы нәтижелер кіреді. Берілген мысалда *a, к* — формалды параметр-аргументтер, z — параметр-нәтиже. Сонымен, ДӘРЕЖЕ процедурасы $z = a^k$ формуласы бойынша анықталады.

Дәрежелік функцияны есептеу алгоритмінде негізінен процедураны шақыру - оның атауын көрсету және жақшаның ішінде нақты параметрлер тізімдерінің жазылуы арқылы жүзеге асырылады.

Процедураның формалды және нақты параметрлерінің арасында сәйкестену ережесі орындалу керек:

- сандары бойынша (формалды қанша болса, нақты параметрлер саны сонша болады);
- реті бойынша (бірінші формалды параметрге бірінші нақты параметр, екіншіге – екінші және т.с.с.);
- типтері бойынша (формалды және нақты параметрлер типтері сәйкес болуы қажет).

Нақты параметр-аргументтер сәйкес типін өрнегі болуы мүмкін. Процедураны шақыру келесі әрекеттерді орындайды:

1) параметр-аргументтердің мәні сәйкес формалды параметрлерге меншіктеледі;

2) процедура денесі орындалады (процедура ішіндегі командалар);

3) нәтиженің мәні сәйкес нақты параметрге беріледі, және негізгі алгоритмнің келесі командасын орындауға көшу жүзеге асырылады.

ДӘРЕЖЕ процедурасында бастапқы берілгендерді енгізу және нәтижелерді шығару командасы жоқ. Мұнда (a, n) аргументтерге бастапқы мәндерді меншіктеу параметр-аргументтерге берілу арқылы орындалады, ал (y) айнымалысының нәтижесін меншіктеу (z) параметр-нәтижесіне беру арқылы орындалады.

Осылайша, процедуралардың параметр мәндерінің берілуі меншіктеудің үшінші тәсілі болып табылады (меншіктеу және енгізу командаларымен қатар).

Процедураларды қолдану реттелген талдау әдісімен күрделі алгоритмдер құруға мүмкіндік береді.

1.6. ҚҰРЫЛЫМДЫҚ ПРОГРАММАЛАУ НЕГІЗДЕРІ

Алғашқы ЭЕМ пайда болғаннан бері жарты ғасыр уақыт өтті. Осы күнге дейін есептеу техникасы қарқынды дамып келе жатыр. ЭЕМ-нің элементтік базасы өзгерді, олардың жылдамдықтары мен жады көлемдері ұлғайды, адамның машинамен интерфейс құралдары өзгерді. Бұл өзгерістер программалаушының жұмысына да өз ықпалын тигізгені сөзсіз. Белгілі – бір көпшілік мақұлдаған нәрсені өңдеп шығару (бұл жағдайда - программа) тәсілін технология деп атайды, сондықтан ары қарай *программалау технологиясы* туралы айтатын боламыз.

Жадысы «аз» және кішкене жылдамдыққа ие алғашқы компьютерлер үшін программа сапасының негізгі көрсеткіші оның жадыдағы алатын орны мен уақыт есептеудің үнемділігі болды. Программа неғұрлым қысқа жасалса, программалаушының дәрежесі соғұрлым жоғары болып саналды. Программаның мұндай қысқартылуы көп күш жұмсауды талап етті. Кейде программа автордың өзі «ауытқып кетуі» мүмкін болатын «қулыққа» ие болатындай құрастырылатын: біраз уақыттан кейін өз программасына оралып, өзгертулер енгізгісі келсе, программалаушы шатасып кетіп, «өзінің тамаша идеясын» ұмытып қалуы мүмкін еді.

Күрделі техникалық құралдардың сәтсіздікке ұшырауы сияқты күрделі алгоритм әрдайым программаның қате жасалу ықтималдығын арттыруы мүмкін.

Программалаушының программа жасақтама өнімін дайындау кезеңдері төмендегідей:

1) жобалау;
2) кодтау;
3) дұрыстау.

Жобалау жасалатын программа алгоритмін алдын-ала өңдеу, мысалы блок-сызбалар. *Кодтау* — программаның программалау тіліндегі мәтінін құрастыру. *Дұрыстау* тестілеудің көмегімен жүзеге асырылады, яғни қандайда бір алдын-ала анықталған нәтижесі белгілі бастапқы деректер жиынтығымен программаның орындалуы жүзеге асырылады. Сонымен қатар, программа неғұрлым күрделі болса, оның толыққанды тестіленуі үшін соғұрлым көп сынақтар қажет. Өте «қу» программаны толықтай тестілеу қиын, өйткені әрқашан «су астындағы тасты» байқамау мүмкін емес.

Жады мен ЭЕМ-нің шапшаңдылығының ұлғаюы, программалау тілдері мен трансляторлардың дамуы программа көлемінің аз

болуына кепілдік бере алмайды.

Программалардың негізгі сапалы мінездемелері олардың қарапайымдылығы, көрнекілігі және сенімділігі болып табылады. Үшінші кезең машиналарының пайда болуымен бұл қасиеттер басты орында.

ХХғ. 60-шы жылдардың соңында және 70-жылдардың басында құрылымдық программалау деп аталатын пән анықталды. Оның пайда болуы мен дамуы Дейкстр Э.В, Милс, Кнут Д.Е және т.б. есімдерімен тығыз байланысты. Қазіргі уақытқа дейін құрылымдық программалау программалаудың негізгі технологиясы болып қала береді. Оның принциптерін сақтау арқылы программалаушы анық, қатесіз, сенімді программаларды жасауды шапшаң үйрену мүмкіндігіне ие болады.

Құрылымдық программалаудың негізі программалау теориясында дәлелденген теорема болып табылады: Кез-келген логикалық есепті шешу алгоритмін негізгі алгоритмдік құрылымдар деп аталатын *Тізбектелу, Тармақталу, Цикл* құрылымдарымен ғана жасауға болады.

Осыған дейін бұл құрылымдарды қарастырған болатынбыз. Барлық программа мысалдарында құрылымдық программалау принциптері қолданылды.

Тізбектелу — әрекеттердің сызықты реті (1.12-сурет).

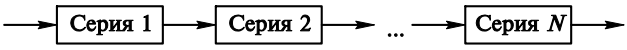
Мұндай тізбектердегі әрбір серия жай және күрделі командаларды қамтуы мүмкін, бірақ оның міндетті түрде бір енгізу және бір шығару операторы болады.

Тармақталу — бұл келесі түрде жазуға болатын алгоритмдік балама

```
егер      шарт
    онда 1-серия
    әйтпесе 2-серия
тс
```

Бұл құрылымда басқару шарттың ақиқаттылығы мен жалғандығына байланысты екі тармақтың біреуіне беріледі (командалар серияларына). Содан кейін нәтиже шығарылады (1.5-сур.).

Тармақталудың толымсыз формасы «әйтпесе» тармағы болмаған жағдайда орын алады:



1.12-сур. *Тізбектелудің* сызықтық құрылымдық командасы

```
егер      шарт
    онда серия
тс
```

Цикл — шарт бойынша кейбір әрекеттердің қайталануы. Циклдің екі түрі бар. Циклдік құрылымның бірінші типі – алғышартты цикл немесе шарты алдын ала берілген цикл (1.7-сур.):

```
әзірше    шарт, қайталау
цб
    серия
цс
```

Мұнда шарт әзірше ақиқат болғанша, цикл денесі болып табылатын серия орындала береді.

Циклдік құрылымның екінші типі – ілесушартты цикл немесе шарты соңынан берілген цикл (1.9-сур.):

```
қайталау
      серия
дейін шарт
```

Мұнда цикл денесі цикл шартынан бұрын орналасады және егер шарт жалған болса, өзінің орындалуын қайталайды. Шарт ақиқат болған жағдайда ғана қайталау тоқтатылады.

Кез-келген циклдік алгоритмді құру үшін шарты алдын-ала берілген циклдер қажет және жеткілікті, яғни олар шарты соңынан берілген циклдерге қарағандағы жалпы нұсқа болып табылады. Негізінде «Дейін» цикл денесі кем дегенде міндетті түрде бір рет орындалады, себебі шартты тексеру оның орындалғанынан кейін жүзеге асады, ал «Әзірше» циклі үшін цикл денесі мүлдем орындалмайды деген нұсқа болуы мүмкін. Осыған орай, кез-келген программалау тілдерінде «Әзірше» циклін қолданумен ғана шектелуге болар еді, алайда «Дейін» циклі анағұрлым ыңғайлы болғандықтан оны қолдану тоқтатылмайды.

Кейде әдебиетте құрылымдық программалауды GOTO-сыз программалау деп атайды (шартсыз көшу операторы жоқ). Шындығында, программалауға осы көзқараспен қарасақ шартсыз көшу операторына орын жоқ екені сөзсіз. GOTO операторын программада негізсіз пайдалану олардың құрылымдық қасиетін яғни, алгоритмнің барлық оң қасиеттерін: мөлдірлігі мен сенімділігін жояды. Бұл оператор барлық процедуралық программалау тілдерінде болғанымен, құрылымдық программалауды қамтамасыз ету үшін оны пайдалануды болдырмау

керек.

Күрделі алгоритмдер өзара байланысқан негізгі құрылымдардан тұрады. Бұл құрылымдардың байланысы екі жолмен орындалуы мүмкін: тізбектелген және қабаттасқан. Бұл кез-келген күрделі электр тізбегін тізбектелген түрде және параллельді байланыстырылған аймақтарға ыдырата алатын электротехникадағы жағдайға ұқсас.

Дегенмен, қабаттасқан алгоритмдік құрылымдар параллель қосылған электрөткізгіштердің баламасы емес. Мұнда бір-бірінің ішіне орналастырылған матрешкалармен ұқсастық барынша орынды. Егер цикл денесін құрайтын блоктың өзі циклдік құрылым болып табылса, бұл қабаттасқан циклдердің бар екенін білдіреді. Өз кезегінде, ішкі циклдің ішінде тағы цикл болуы мүмкін және т.с.с. Осыған байланысты *циклдердің тереңдігі* туралы түсінік енгізілді. Сол сияқты, тармақталулар да бір-біріне қабаттасқан болуы мүмкін.

Құрылымдық блок-сызба алгоритмдері олардың стандартты бейнесін талап етеді. Әрбір базалық құрылымда бір енгізу және бір шығару әрекеті болуы тиіс. Алгоритмнің стандартқа сай емес салынған блок-сызбасының оқылуы нашар болады және өзінің көрнекілігін жоғалтады.

Құрылымдық алгоритмдердің блок-сызбасы 1.13-суретте келтірілген. Мұндай блок-сызбалардың оқылуы оңай және көзге жақсы көрінеді, осыған байланысты әрбір құрылымға атау беруге болады.

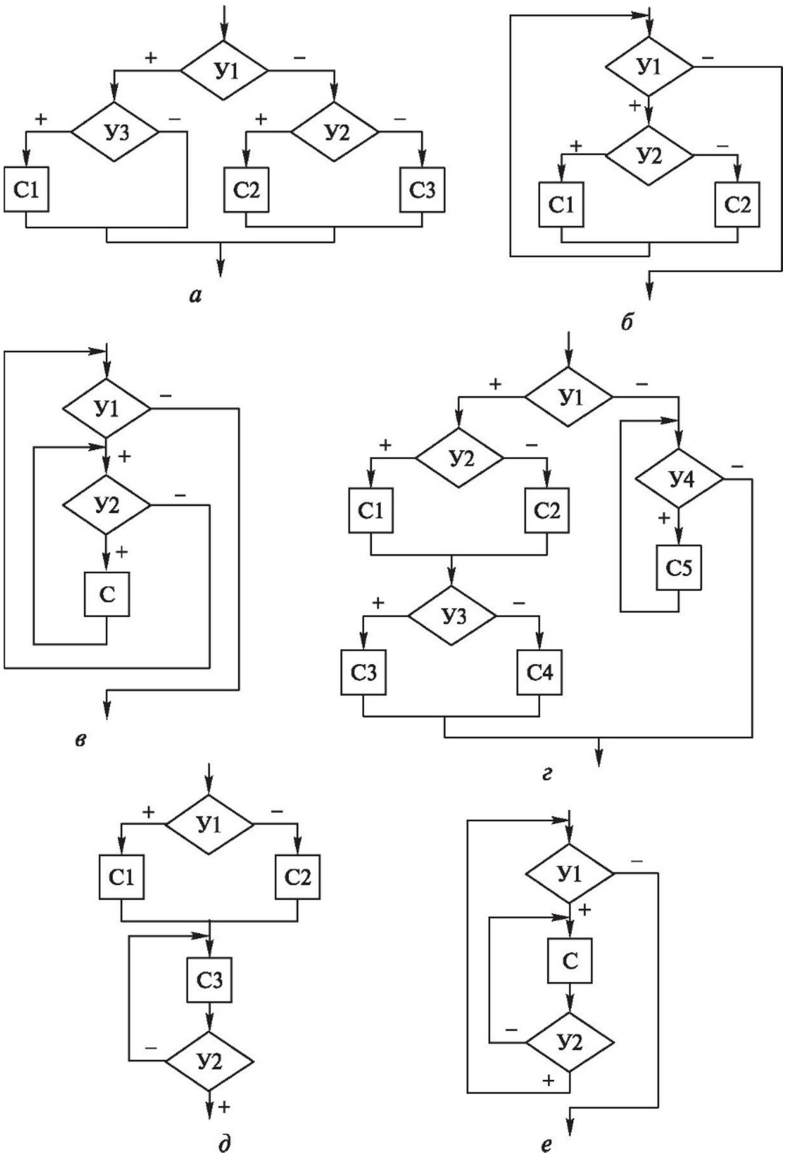
Құрылымдық алгоритмдерді сипаттауға арналған алгоритмдік тілде шартсыз көшу командасы болмайды. Бұл тілді қолдану программалаушы үшін оны оқу барысында «құрылымдық тәрбиеге» баулиды.

Алгоритм құрылымының АТ-гі көрнекілігін оның құрылуы мен мәтіні береді. Ол үшін қолданылатын негізгі тәсіл келесі ережелерге бағынуы тиіс — жолдарды қозғалту болып табылады:

- бір деңгейдегі қабаттардың конструкциялары жоғарыдан төмен қарай бір деңгейде орналасулары қажет (яғни, жолдағы бір позициядан басталады);
- қабаттасқан конструкциялар бір жолда оң жақтағы конструкцияға қатысты сыртқы бірнеше позицияларда орналасады

Алгоритмдердің АТ-дегі мәтіндерінің құрылымы 1.13-суретте, ал блок-сызбалары 1.6-кестеде көрсетілген.

Алгоритмдеудің құрылымдық әдістемесі — бұл алгоритмді сипаттау формасы ғана емес, *программалаушының ойлау қабілеті*



1.13-сур. Құрылымдық алгоритмдер блок-сызбаларының мысалдары: *а* — күрделілігі 1-ге тең болатын күрделі тармақталулар; *б* — күрделі тармақталуы бар цикл; *в* — күрделілігі 1-ге тең болатын «әзірше» күрделі тармақталулар; *г* — оң тармақтары тізбектелген және теріс тармақтары «Әзірше» болатын күрделі тармақталулар; *д* — тармақталу мен «дейін» циклінің тізбегі; *е* — «Әзірше» сыртқы күрделі циклі және ішкі «дейін» циклі. У — шарт; С — серия

1.6-кесте. Сызбаларға сәйкес 1.13-суретте көрсетілген АТ-гі алгоритмдер

Сызба	Алгоритм	Сызба	Алгоритм
а	**егер** <У1> **онда егер** <У3> **онда** <С1> **тс** **әйтпесе егер** <У2> **онда** <С2> **әйтпесе** <С3> **тс** **тс**	*б*	**әзірше** < У1>, **қайталау цб** **егер** <У2> **онда** <С1> **әйтпесе** <С2> **тс** **цс**
в	**әзірше** <У1>, **қайталау** **цб** **әзірше** <*У2*>, **қайталау** **цб** <С> **цс** **цс**	*г*	**егер** <У1> **онда егер** <У> **онда** <С1> **әйтпесе** <С2> **тс** **егер** <У3> **онда** <С3> **әйтпесе** <С4> **тс** **әйтпесе әзірше** <У4>, **қайталау** <С5> **цс** **тс**
д	**егер**<У1> **онда** <С1> **әйтпесе** <С2> **тс** **қайталау** С3 **дейін** <У2>	*е*	**әзірше** <У1>, **қайталау** **қайталамау** <С> **дейін** <У2> **цс**

болып табылады. Стандартты құрылымды алгоритмдерді құрастырған жөн. Алгоритм құрудың құрылымдық әдістемесі стандартты секцияларды құруға ұқсас.

Құрылымдық программалаудың тағы бір маңызды технологиялық қасиеті шығарылатын есептің *ішкі есептерге декомпозициясы*, яғни программалауға қажет бастапқы есептің әлдеқайда қарапайым бөліктері. Осындай ішкі есептерді шешу алгоритмдері *көмекші* деп аталады. Алгоритмді екі түрлі тәсілмен құруға болады:

- жоғарыдан төмен қарай — алдымен негізгі алгоритм, одан кейін — көмекші алгоритм құрылады;
- төменнен жоғары қарай — алдымен көмекші алгоритм, одан кейін — негізгі алгоритм құрылады.

Алгоритмді бірінші жолмен құруды тізбектелген детализация әдісі, ал екінші жолын құрама әдіс деп те атайды.

Тізбектелген детализация әдісінде алдымен негізгі алгоритм құрылады да, оған бірінші деңгейлі көмекші алгоритмдер шақырылады. Одан кейін бірінші деңгейлі көмекші алгоритмдер құрылады да оған екінші деңгейлі көмекші алгоритмдер шақырылады және т.с.с. Ең төменгі деңгейлі көмекші алгоритмдер тек қарапайым командалардан тұрады.

Кез-келген күрделі объектілерді құру үшін тізбектелген детализация әдісі қолданылады. Бұл құрастырушының табиғи логикалық ойлау тізбегі: біртіндеп құралдарды тереңірек ойлауы болып табылады. Программалауда да құрастыру жөнінде сөз қозғалады, бірақ бұл техникалық құрылғыларды құрастыру емес, алгоритмдерді құрастыру. Басқа тәсілмен күрделі алгоритмдер құрастыру мүмкін емес.

Тізбектелген детализация әдісі программалаушы ұжымының жұмысын күрделі жобаларын ұйымдастыруға мүмкіндік береді. Мысалы, топ басшысы негізгі алгоритмді құрады, ал көмекші алгоритмдерді өңдеуді және сәйкес ішпрограммаларды жазуды өзінің әріптестеріне тапсырады. Бұл жағдайда жұмыс топтары өздерінің программалық модульдерінің арасында интерфейс жөнінде келісіп (яғни өзара келісім) алулары қажет, ал программаның ішкі ұйымдастырылуы ол программалаушының жеке жұмысы.

Құрастыру әдісі ішкі программа, процедура және функция түрінде берілген программалау тілдерінде жүзеге асырылған көмекші алгоритмдердің жиналуы мен кітапханаларды қолдануға мүмкіндік береді.

1.7. ТІЛДЕР МЕН ПРОГРАММАЛАУ ТЕХНОЛОГИЯЛАРЫНЫҢ ДАМУЫ

Программалау тілі - компьютерге «түсінікті» формада жазылған ЭЕМ-де әртүрлі есептерді шешуге арналған программаларды жазу тәсілі. Компьютердің процессоры машиналық командалар тілін (МКТ) қабылдау үшін тікелей қызмет етеді.

Дегенмен, МКТ-ғы программалар тек бірінші буын ЭЕМ-дері үшін жасалған. МКТ-ғы программалау - бұл оңай шаруа емес. Программалаушы барлық машиналық командалардың сандық кодтарын біліп, программа командалары мен деректерге жадыны өзі бөлу керек.

1950 жылдары программалауды автоматтандыруға арналған алғашқы құралдар - автокод тілдері пайда болды. Кейінірек осы деңгейдегі тілдер ассемблер деп атала бастады. Автокод және ассемблер сияқты тілдер программалаушылардың жұмысын жеңілдетті. Айнымалы шамалар символдық атауларды бейнелей бастады. Сандық кодтау амалдары есте сақталуы оңай мнемоникалық (ауызша) белгілерімен ауыстырылды. Осыдан кейін, программалау тілі машиналық командалар тілінен жойылып, адам үшін түсінікті бола бастады. Компьютер автокодта программаларды орындау үшін арнайы аудармашы - транслятор қажет болды, яғни автокодтағы программа мәтінін МКТ-дағы баламалы программа мәтініне аударатын жүйелік программа.

Автокодтағы трансляторомен жабдықталған компьютер автокодты түсінеді. Бұл жағдайда сіз тілі автокод болып табылатын псевдоЭЕМ (жабдықтама + автокодтағы транслятор) жайлы айта аласыз. Автокод және ассемблер типті тілдер машинаға бағытталған тілдер болып табылады, яғни олар нақты компьютердің машиналық командалар құрылымына сәйкес бапталған. Түрлі процессорлары бар әр түрлі компьютерлердің ассемблерлері де әр түрлі. Жоғары деңгейлі программалау тілдері (ЖДПТ) машинадан тәуелсіз, яғни сәйкес трансляторомен жабдықталған сол тілдегі программа әр түрлі ЭЕМ-дерде орындала береді. ЖДПТ-дегі жазылған программалар формасы автокодпен салыстырғанда дәстүрлі математикалық формаға және табиғи тілге жақын. Мысалы, Паскаль тілінде жазылған программа мектептегі алгоритмдік тілдегідей. Жоғары деңгейлі программалау тілдері оқуға оңай және құрылымдық программалау әдістемесін жақсы қолдайды.

1950 жылдары пайда болған алғашқы жоғары деңгейдегі тілдер Фортран, Кобол (АҚШ-та) және Алгол (Еуропада) болып табылады.

Бұл ЭЕМ-нің *бірінші буын тілдері*. Фортран және Алгол математикалық сипаттағы ғылыми-техникалық есептеулерге бағытталған. Кобол - бұл математикалық құралдары әлсіз дамыған экономикалық есептерді программалау тілі, бірақ мәтінді өңдеу құралдары жақсы дамыған және қажетті құжат түрінде деректерді шығару жақсы ұйымдастырылған. Бірінші ЖДПТ пәндік бағдар сипатына ие болды.

1960 және 1970 жылдары ЭЕМ-нің *екінші буыны* - программалау тілдерінің көптеген саны пайда болды, (ЭЕМ тарихында олардың мыңнан астамы құрылды), дегенмен, таралуы мен уақыт сынағынан кейбіреулері ғана өтті.

1965 жылы Дартмут университетінде Бейсик тілі жасалды. Авторлардың ойлары бойынша, бұл қарапайым, үйренуге оңай және күрделі емес есептерді программалауға арналған тіл болды. Бейсиктің ең көп таралуы микроЭЕМдер мен дербес компьютерлерде (ДК) болды. Мектеп компьютерлерінің алғашқы үлгілерінде Бейсик тілінде ғана программалау мүмкін болды, бірақ бұл құрылымдық емес тіл, сондықтан ол сапалы программалауға жарамайды. Дербес компьютерлерге арналған Бейсиктің кейінгі нұсқалары, мысалы QBasic, құрылымдық сипатқа ие болды және олар өздерінің көрнекі мүмкіндіктерінің арқасында Паскаль сияқты тілдерге жақындады.

IBM әзірлеген PL/1 (ProgramLanguageOne) *үшінші буын* компьютерлер дәуірінде кең таралды. Бұл әмбебаптылығымен көзге түскен, яғни кез-келген есепті шешу мүмкіндігіне ие: есептеу, мәтінді өңдеу, жинақтау және ақпараттық іздеуді талап ететін алғашқы тіл болды. Алайда PL/1 тым күрделі тіл болып шықты. IBM360/370 сияқты машиналар үшін PL/1 тілінің трансляторы жеткілікті оңтайлы емес және көптеген анықталмаған қателерді қамтыды. Шағын және микроЭЕМ үшін PL/1 тілі мүлдем таралған жоқ. Дегенмен, программалау тілдерін әмбебаптаудың желісі қолдауға ие болды: Алгол-68 және Фортран-77 ескі тілдерінен әмбебап нұсқалары әзірленді.

Программалау тілдерінің тарихындағы маңызды оқиға - 1971 жылы Паскальды компьютерлік *төртінші буын* тілі - құрылымдық программалаудың оқыту тілі ретінде құру болды.

Паскаль ДК-де кеңінен таратылды. BorlandInternational, Inc. (АҚШ) фирмасы оларға Турбо Паскаль программалау жүйесін жасап шығарды. Турбо Паскаль - бұл тек тіл мен транслятор ғана емес, сонымен қатар пайдаланушыны Паскальда жұмыс істеуге ыңғайлы ететін *біріктірілген программалау ортасы* болды. Турбо Паскаль оқу мақсаты шеңберінен асып, әмбебап мүмкіндіктерге ие кәсіби программалау тіліне айналды. Турбо Паскаль трансляторы

оның жасаған программаларының оңтайлығы бойынша сапа жағынан Фортранның көшбасшы трансляторымен тұспа-тұс келді. Паскаль өзінің мүмкіндіктерінің арқасында көптеген заманауи программалау тілдерінің бастамасы болды, мысалы Ада, Модула-2 және т.б.

Си программалау тілі (ағылшынша атауы - C) операциялық жүйелерді, трансляторларды, деректер базаларын және басқа да жүйелік және қолданбалы программаларды әзірлейтін аспаптық тіл ретінде құрылды. Паскаль сияқты, Си тілі де құрылымдық программалау тілі болып табылады, бірақ Паскальға қарағанда, ол белгілі бір машиналық командаларға және компьютер жадысының кейбір бөліктеріне тікелей қол жеткізе алу мүмкіндігі бар.

Модуль-2 — Н. Вирт ұсынған бұл Паскаль тілінің дамыған және үлкен программаларды құруға арналған құралдары бар тағы бір тіл.

Болашақтың ЭЕМ-дері — *бесінші буын* — бұл жасанды интеллект машиналары. Бірақ осы машиналар үшін тілдердің прототипі олардың физикалық көріністерінен әлдеқайда бұрын жасалған. Бұл ЛИСП және Пролог тілдері.

ЛИСП тілі туралы алғашқы рет 1958 жылы айтыла басталды. Ол рекурсивті түрде анықталған функциялардың тұжырымдамасы негізінде құрылды. Кез-келген алгоритм рекурсивті функциялардың кейбір жиынтығы арқылы сипатталатыны дәлелденгендіктен, ЛИСП әмбебап тіл болып саналады. Оның көмегімен ЭЕМ-де күрделі процестерді, атап айтқанда, адамдардың интеллектуалдық қызметін модельдеуге болады. ЛИСП-те жүзеге асырылған программалау парадигмасы *функционалды программалау* деп аталады.

Пролог тілі 1972 жылы Францияда жасанды интеллект мәселелерін шешу үшін әзірленген. Бұл тілдің ойлау логикасын, түрлі пікірлерді формалды түрде сипаттауға және ЭЕМ-ді қойылған сұраққа жауап беруге мәжбүр ететін мүмкіндігі бар. Пролог тілінде *программалаудың логикалық парадигмасы* жүзеге асырылады.

XX ғасырдың аяғында ЭЕМ-нің программалық қамтамасыз етуді дамытудағы маңызды жаңа бағыты объектіге-бағытталған тәсілге айналды. Нысандар (объектілер) — бұл деректер мен программаларды біріктіруге арналған құрылымдар. Объектіге-бағытталған операциялық жүйелер (мысалы, Windows), қолданбалы программалар және *объектіге-бағытталған программалау* (ОБП) жүйелері танымал бола бастады.

Объектіге-бағытталған программалау технологиясының элементтері бар алғашқы тіл Симула-67 тілі болды. Си тілінің дамуы оның C ++ деп аталатын объектіге-бағытталған нұсқасының пайда болуына әкелді. Турбо Паскальда, 5.5 нұсқасынан бастап,

ОБП құралдары пайда болды.

Соңғы кездері программалау технологиясында жаңа *визуалды* үрдіс пайда болды. Windows-та жұмыс істейтін *визуалды* программалау жүйелері пайда болды, олар күрделі графикалық интерфейсті оңай және жылдам программалауға мүмкіндік береді. Танымал программалау тілдерінің визуалды нұсқалары пайда болды: Visual Basic, Delphi (Турбо Паскальдың дамуы), Borland C ++ Builder және басқалар.

Трансляция әдістері. Белгілі бір программалау тілін ЭЕМ-де жүзеге асыру — бұл берілген ЭЕМ-ге осы тілден транслятор жасау. Трансляцияның екі әдісі бар — *компиляция* және *интерпретация.* Осы әдістердің айырмашылығын түсіндіру үшін келесі балама ұсынылады: дәріскер аудитория алдында таныс емес тілде сөйлеу керек. Бұл жағдайда:

- толық алдын-ала аударма болу керек, яғни дәріскер сөз мәтінін аудармашыға алдын ала береді, ол аударманы жазады, оны көбейтеді және оны тыңдаушыларға таратады (осыдан кейін дәріскер дәрісін оқымаса да болады);
- синхронды аудару, яғни дәріскер баяндаманы оқиды, ал аудармашы бірден оның сөздерін аударады.

Компиляция толық алдын ала аударма баламасы болып табылады, ал интерпретация - синхронды аударма баламасы. Компиляция принципімен жұмыс істейтін транслятор *компилятор* деп аталады, ал интерпретация принципімен жұмыс істейтін транслятор *интерпретатор* болып табылады.

Компиляция кезінде ЭЕМ жадысына – компилятор программасы жүктеледі. Ол программа мәтінін ЖДПТ-не мәтіндік файлдан жүктелген *бастапқы модуль* деп аталатын бастапқы ақпарат ретінде қабылдайды. Компилятормен программа өңдегеннен кейін, ЖПДТ-де *объектілі модуль* алады. Программаны өңдеудің келесі кезеңі *байланыстарды түзету* деп аталады, ол жұмысты арнайы программа — байланыс редакторы атқарады.

Бұл редактор бастапқы программаға өзіне қажетті программалық модульдерді қосады: процедуралар, функциялар және т.б., нәтижесінде орындалуға дайын машиналық командалар тілінде орындалатын *жүктеу модулі программасын* алады. Осыдан кейін, жедел жадыда тек МКТ-дегі программалар ғана қалады, оның орындалуы қажетті нәтиже береді.

Интерпретатор программаның барлық жұмыс жасау уақытында

ЖДПТ-гі программамен бірге ішкі жадыда — жедел есте сақтау құрылғысында орналасады (ЖЕСҚ). Интерпретатор алгоритмнің орындалу реті бойынша кезекті программаның операторын санайды, оны командаларға аударады және осы командаларды дереу орындайды, содан кейін келесі операторды аударуға және орындауға кіріседі. Жадтағы бұрынғы аударымдардың нәтижелері сақталмайды, яғни сол команда қайталанған кезде қайтадан аударылады.

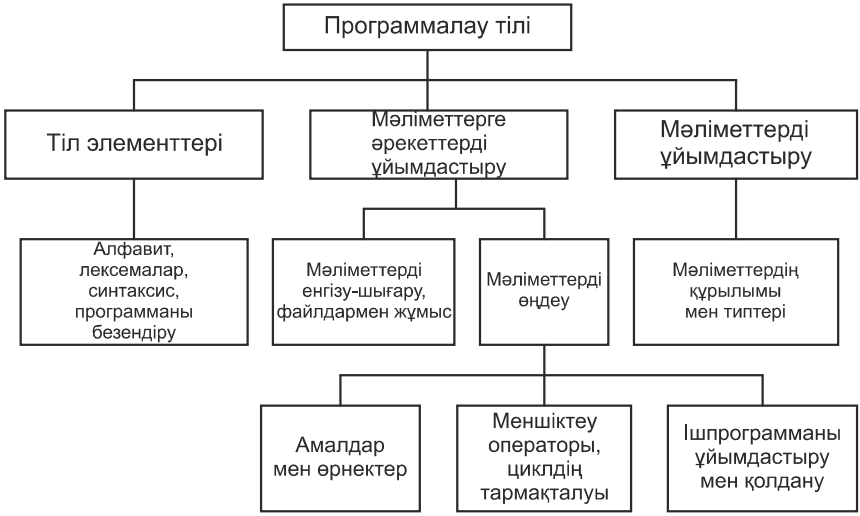
Компиляция кезінде программаны орындау үш кезеңнен тұрады: компиляция, байланысу және орындау. Интерпретация кезінде трансляция және орындау біріктірілгендіктен, программаны ЭЕМ-де өңдеу бір кезеңнен өтеді. Дегенмен, компиляцияланған программа интерпретацияланған программаға қарағанда жылдамырақ жұмыс істейді, сондықтан компиляторларды жылдам есептеуді қажет ететін ірі программаларда пайдалану ыңғайлы болып табылады. Паскаль, Си, Фортран тілдеріндегі программалар үнемі компиляцияланады. Бейсик көбінесе интерпретатор арқылы жүзеге асырылады. Кейбір тілдер компилятомен бірге интерпретатор арқылы жүзеге асырылады. Интерпретация режимінде программаны жөндеу ыңғайлы, ал есептеулерді компиляция режимінде орындаған жөн.

1.8. ЖОҒАРҒЫ ДЕҢГЕЙЛІ ПРОГРАММАЛАУ ТІЛДЕРІНІҢ ҚҰРЫЛЫМЫ МЕН ОЛАРДЫ СИПАТТАУ ТӘСІЛДЕРІ

Барлық программалау тілдерінде берілгендерді ұйымдастыру және оларды өңдеу тәсілдері анықталған. Сонымен қатар, көптеген символдар (алфавит), белгілер және басқа да көрнекі программалау құралдарын қамтитын тіл элементтері бар. Программалау тілдерінің әр түрлілігіне қарамастан, оларды үйрену бағыты шамамен бірдей болып келеді. Бұл жоғары деңгейлі тілдердің жалпы құрылымына байланысты (1.14 сурет).

Берілген оқулықта Паскаль мен Delphi-дің сипаттамасы осы бағытқа сәйкес орындалады.

Табиғи тілдерді және программалау тілдерін үйренудегі олардың ұқсастығын атап өту керек. Алдымен, шет тілінде оқу және жазу үшін осы тілдің *алфавитін* білу қажет.



1.14-сур. Жоғарғы деңгейлі программалау тілдерінің құрылымы

Екіншіден, сіз сөздерді және сөйлемдерді жазу ережелерін, яғни *тіл синтаксисін* білуіңіз керек. Үшіншіден, оларды дұрыс қабылдау үшін сөздер мен сөз тіркестерінің мағынасын түсіну қажет. Мысалы, ұшақтың кабинасында «Fastenbelts!» сөзі экранда жазылды (Белбеу тақ!). Ағылшын грамматикасын біле отырып, сіз бұл сөйлемді дұрыс оқып біле аласыз, бірақ оның мағынасын түсінбейсіз және сондықтан тиісті әрекеттерді орындамайсыз. Сауатты сөздерден мүлдем мағынасыз сөз тіркестерін жасауға болады. Тіл құрылысының мағыналық мазмұны *семантика* деп аталады.

Кез келген программалау тілі үш негізгі компоненттен тұрады: алфавит, синтаксис және семантика.

Программалау тіліндегі ережелерді сақтау ауызекі тілге қарағанда қатаң болуы керек. Адамның сөзі көптеген ақпараттарды қамтиды, яғни сөзді толық естімей, оның мағынасын түсінуге болады. Тыңдаушы немесе оқушы адам қабылданған мәтіндегі қателерді түзетуіне, толықтыруына және ойластыруына болады.

Компьютер - бұл бәрін «байыппен» қабылдайтын автомат. Программа мәтіндерінде артық ақпарат болмайды және компьютер көрініп тұрған (адамның көзқарасымен) қатені өзі түзете алмайды. Ол тек «түсінбеген» орынды көрсете алады және қатенің болжамды сипаты туралы түсініктеме бере алады. Ал қатені программалаушы өзі түзету керек.

Программалау тілінің синтаксисін сипаттау үшін белгілі бір тіл қажет, яғни басқа тілдерді сипаттауға арналған *метатіл.* Программалау әдебиетіндегі ең көп таралған метатілдер – *Бэкус-Наур металингвистикалық формулалары* (БНФ тілі) және *синтаксистік диаграммалар*. Ары қарай біз негізінен синтаксистік диаграммалардың тілін пайдаланамыз, өйткені олар әлдеқайда түсінікті әрі көрнекті. Алайда, ыңғайлы болған уақытта БНФ тілінің кейбір элементтерін қолданамыз.

БНФ-те әрбір синтаксистік ұғым «анықтама бойынша бар» сөйлемімен мағыналас болып келетін «:: =» белгісімен байланысқан оң және сол жақ бөліктерден тұратын формуланың формасына ие. Формуланың сол жақ бөлігінде бұрыштық жақшалардағы <> анықталатын ұғымдардың атауы (мета-айнымалы) және оң жақ бөлігінде – мета-айнымалы қабылдай алатын барлық мәндер жиынын анықтайтын формула немесе диаграмма орналасқан.

Тілдің синтаксисі ұғымдардың күрделенуі нәтижесінде жасалады, яғни алдымен ең қарапайым (негізгі) ұғымдар, одан кейін құрамдас бөлігі ретінде алдыңғы ұғымдары қамтылған күрделі ұғымдар анықталады.

Мұндай тізбекте соңғы айқын ұғым «программа» болу керек.

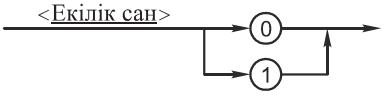
Метаформулаларды жазу барысында белгілі бір келісімдер қабылданған. Мысалы, «екілік сан» ұғымын анықтайтын БНФ келесі түрде жазылады:

<екілік сан>::=0|1

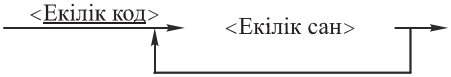
«|» таңбасы мұнда «немесе» сөзіне баламалы. Екілік санның синтаксистік диаграммасы 1.15-суретте көрсетілген.

Бұл диаграммаларда бағыт сызықтары синтаксистік құрылымның элементтерінің ретімен орналасуын көрсетеді және шеңберлер ішінде осы құрылымда кездесетін символдар көрсетілген.

БНФ-те екілік сандардың бос емес тізбегі ретінде «екілік код» ұғымы келесідей сипатталады:



1.15-сур. Екілік санның синтаксистік диаграммасы



1.16-сур.Екілік кодтың синтаксистік диаграммасы

```
<екілік код> ::= <екілік сан> | <екілік код
><екілік сан>
```

Белгілі бір ұғым өзі анықтаған анықтама *рекурсивті* деп аталады. Рекурсивті анықтамалар БНФ-ке тән.

Екілік кодтың синтаксистік диаграммасы 1.16-суретте келтірілген. Мұнда, қайтарылған бағыт көрсеткіші бірнеше қайталау амалын білдіреді.

Әлбетте, синтаксистік диаграмма БНФ-ке қарағанда айқынырақ.

Синтаксистік диаграммаларды Н. Вирт енгізді және олар өзі жасаған Паскаль тілін сипаттауда қолданылды.

ЖАТТЫҒУЛАР

1. ЕМЕС *(X > Z)* ЖӘНЕ ЕМЕС (X = *Y)* логикалық өрнегінің мәнін анықтаңдар, айнымалылардың келесі мәндері берілген:
   а) X = 3, *Y* = 5, *Z* = 2;
   ә) X = 0, *Y* = 1, *Z* = 19;
   б) X = 5, *Y* = 0, *Z* = -8;
   в) X = 9, *Y* = -9, *Z* = 9.
2. Келесі шарттарда ақиқат болатын логикалық өрнектерді (формулаларды) жазыңдар.
   а) (X, Y) координаталарындағы нүкте центрі координаталар басы болатын шеңбердің бірінші ширегінде орналасқан.
   б) (X, Y) координаталарындағы нүкте центрі координаталар басы болатын бірлік шеңбердің сыртында орналасқан және радиусы 2-ге тең центрі координаталар басында жатқан шеңберде орналасқан (график түрінде сызыңдар).
3. Жазықтықтағы үшбұрыштың үш төбелерінің декарттық координаталары берілген. Үшбұрыштың ауданын табу алгоритмін құрыңдар.
4. Жер атмосферасы шегінен тыс кеткен зымыранның жылдамдығы берілген. Қозғалтқышты өшіргеннен кейінгі зымыранның қозғалысын анықтаудың алгоритмін құрыңдар. (Үш ғарыштық жылдамдықтың мәндері: 7,5 км/сағ, 11,2 км/сағ, 16,4 км/ сағ.)
5. Үш оң сан берілген. Бұл сандар үшбұрыштың қабырғаларының ұзындығы бола алатынын анықтайтын алгоритм құрыңдар.
6. Компьютер тек екі арифметикалық амалды орындай алады делік: қосу және азайту. Келесі алгоритмдерді құрыңдар:
   а) екі бүтін сандарды көбейту;
   ә) екі санның бүтінмәнді бөлінуі;
   б) екі санның бүтін бөлінуінің қалдықтары.
7. Квадрат теңдеуді шешу алгоритмін көмекші ретінде пайдаланып, биквадрат теңдеуді шешу алгоритмін құрыңдар.
8. Екі натурал санның ЕКОБ-ін табу алгоритмін көмекші алгоритм ретінде пайдаланып, үш натурал санның ЕКОБ-ін табу алгоритмін құрастырыңдар.

БАҚЫЛАУ СҰРАҚТАРЫ

1. Программалауды қолданып компьютерде есепті шығару кезеңдері қандай? Оларды сипаттаңдар. Формалдау мен қою кезеңдерін келесі есеп мысалында ашып көрсетіңдер: екі пункт арасындағы моторлы қайықтың өзендегі қозғалыс уақытын есептеу.
2. Компьютерде программалауда қандай негізгі берілгендер типтері қолданылады?
3. Сызықтық алгоритм қандай командалардан құрастырылады? Сызықтық алгоритмі көмегімен шешілетін есеп мысалын көрсетіңдер.
4. Тармақталудың алгоритмдік құрылымы дегеніміз не? Толық және толық емес тармақталулардың айырмашылығы қандай? Тармақталу алгоритмі көмегімен шешілетін есеп мысалын көрсетіңдер.
5. 1-ші пунктте берілген есепті шешудің тармақталу алгоритмін құрыңдар.
6. Циклдің алгоритмді құрылымы дегеніміз не? Шарты алдын ала берілген циклдің («Әзірше» циклі) шарты соңынан берілген циклден («Дейін циклі») қандай айырмашылығы бар? Циклдік алгоритмдер көмегімен шешілетін есеп мысалын көрсетіңдер Бұл есепті шешудің екі нұсқасын көрсетіңдер: «Әзірше» және «Дейін» циклдерімен.
7. Көмекші алгоритм (процедура) дегеніміз не? Процедура алгоритмдік тілде қалай сипатталады? Негізгі алгоритмге процедура қалай шақырылады? Ішкі есеп пен процедураны қолданып келесі есепті шығарыңдар: сыртқы және ішкі радиустың берілген мәндері бойынша шеңбердің ауданын табыңдар.
8. Логикалық амалдардан тұратын логикалық өрнектерді есептеудің қандай ережелері бар? Бірнеше логикалық амалдарды қамтитын өрнектерді есептеуге мысал келтіріңдер.
9. Құрылымдық программалау дегеніміз не? Алгоритмдерді құрудың құрылымдық әдістемесінің негізгі принциптері қандай?
10. Ассемблерді неліктен машинаға-бағытталған тіл деп, ал жоғарғы деңгейлі тілдерді машинаға тәуелсіз программалау тілдері деп атайды?
11. Жоғарғы деңгейдегі программа трансляциясы дегеніміз не? Компиляция мен интерпретацияның бір-бірінен қандай айырмашылықтары бар?

2-ТАРАУ

ПАСКАЛЬ ТІЛІНДЕ ПРОГРАММАЛАУ

Осы тарауда білесіңдер :

- Паскаль тілінің даму тарихын;
- Турбо Паскальда программа құрылымы қандай болатынын;
- Турбо Паскальдағы берілгендер типтері концепциясын;
- тіл элементтерін — алфавит, амалдар, өрнектердің жазылу ережесі;
- берілгендерді енгізу-шығаруды ұйымдастыруды;
- меншіктеу операторының орындалу ережесін;
- Паскальда тармақталу құрылымды алгоритмдерінің программалау тәсілдерін;
- Паскальда циклдерді программалау тәсілдерін;
- ішкі программалар түрлерін (функциялар мен процедуралар) және оларды қолдану мен сипаттау тәсілдерін;
- рекурсивті-анықталған программаның не екенін (функция, процедура);
- Турбо Паскальда графиканы программалау құралдарын;
- Турбо Паскальдің құрылымдық берілгендер типтерін (жиымдар, жолдар, жиындар, жазбалар, файлдар) және оларды ұсыну мен өңдеу тәсілдерін;
- динамикалық шамалар дегеніміз не және берілгендердің динамикалық құрылымы қалай жасалатынын және өңделетінін — байланысқан тізімдер туралы;
- модуль дегеніміз не және ішкі программалар кітапханасын қалай ұйымдастыруға болатынын.

Сендер үйренесіңдер:

- сызықтық құрылымды алгоритмдерді программалауды және пернетақтадан енгізу мен экранға шығаруды ұйымдастыруды;
- программада барлық циклдік операторлардың түрлерін қолдануды — «Әзірше» циклі, «Дейін» циклі, параметрлі цикл;
- қайталау саны берілген циклдерді программалауды және итерациялық циклдерді;
- циклдік құрылымды алгоритмдері бар типтік есептерді шешуді программалау, соның ішінде функцияны кестелеу, шекті сандар тізбегін өңдеу және шексіз сандар тізбегін өңдеуді;
- бүтінсанды арифметиканың типтік есептерін программалауды;
- программада ішпрограмма – функция және ішпрограмма – процедураларды қолдануды;
- бір және екіөлшемді жиымдарды өңдеудегі типтік есептерді шешуді программалауды;
- символдық жолдарды өңдеудегі типтік есептерді шешуді программалауды;
- программада «жиым» типін қолдануды;
- жазбаларды өңдеудегі типтік есептерді шешуді программалауды;
- файлдар мен берілгендерді алмасуды, файлдарда берілгендерді өңдеуді программалауды;
- байланысқан тізімдердегі қиын емес есептерді өңдеуді программалауды;
- қарапайым графикалық бейнелердің экранда шығарылуын программалау— суреттер, сызбалар, функция графиктері; шағын программалық модульдер құруды (ішкі кітапханалар).

2.1. ПАСКАЛЬ ТІЛІМЕН ТАНЫСУ

Паскаль тіліндегі программа құрылымы. Паскальдің стандартты тілінің анықтамасы бойынша программа өзінің *тақырыбынан* және *денесінен (блок)* тұрады. Программа аяғында оның соңын білдіретін *нүкте* қойылады. Ал блок *сипаттау бөлімі* мен *операторлар бөлімінен* тұрады:

```
Program <программа атауы>;
Label <белгілер бөлімі>;
Const <тұрақтылар бөлімі>;
Type <типтер бөлімі>;
Var <айнымалылар бөлімі>;
Procedure (Function) <ішкі программалар бөлімі>;
Begin
<операторлар бөлімі>
End.
```

Операторлар бөлімі барлық программаларда болады және ол негізгі бөлім болып табылады. Сипаттау бөлімдері кейбір программаларда ғана болады.

Турбо Паскальда басқа стандартты тілдерден қарағанда келесі жағдайлар болуы мүмкін:

- программа тақырыбының болмауы;
- сипаттау бөлімінде **Const**, **Type**, **Var**, **Label** бөлімдерінің бірінен соң бірі кез-келген ретте орналасуы.

Программа мысалдары. Паскаль тілін Н.Вирт оқу мақсатында жасап шығарғандығы туралы айтып өткенбіз. Оның негізгі принципі — оның программалаудың құрылымдық әдістемесін қамтуы. АТ мен Паскаль тілінің айырмашылығы мынада: АТ— қазақ тілінде, ал Паскаль — ағылшын тілінде жазылады; Паскаль тілінің синтаксисі қатаң және бірегейлі анықталған, ал АТ синтаксисі еркін тіл болып табылады.

Паскаль тілінде жазылған программа алгоритмдердің АТ-дегі ағылшынша аудармасына ұқсайды. АТ-де жазылған жай бөлшектерді бөлу алгоритмін Паскальдағы сәйкес программамен салыстырыңдар:

```
алг бөлшектерді бөлу
бұт a, b, c, d, m, n
басы енгізу a,b,c,d

        m:=a*d
        n:=b*c

соңы
```

```
Program Division;
Var a,b,c,d,m,n : Integer;

Begin readln (a,b,c,d)
m : = a * d; n : = b * c;
WriteLn(m, n)
End.
```

Бұл жерде келесі математикалық теңдік қарастырылған:

$$\frac{a}{b}:\frac{c}{d}=\frac{a\cdot b}{c\cdot d}$$

Бұл программаны ағылшынша білген адам Паскаль тілінің оқулығына қарамай–ақ түсініп алуына болады.

Программа тақырыбы **Program** (программа) сөзінен басталады, және одан кейін оның атауы жазылады. Ол атауды программалаушы өзі ойлап табады. (division — бөлу). Айнымалыларды сипаттау бөлімі **Var** (variables — айнымалы) сөзінен басталады, және бұл сөзден кейін айнымалылар тізімі жазылады, ал олардың типтері қос нүктеден кейін **Integer** (бүтін) сөзімен сипатталады. Программаның операторлар бөлімінің басы мен соңы **Begin** (басы) және **End** (соңы) сөздерімен белгіленеді. Программа соңында *міндетті түрде нүкте қойылады.*

Бастапқы берілгендерді пернетақтадан енгізу ReadLn (read line — жолды оқу) процедурасының көмегімен жүзеге асады. Пернетақтадан арасынан бос орын қалдырылып төрт сан теріледі, олар экран дисплейінде бір жолдың бойында көрінеді, соңынан енгізу пернесі басылады.

Паскальда меншіктеу операторлары АТ-гі сияқты жазылады, ал көбейту таңбасы жұлдызшамен (*) белгіленеді.

Экран дисплейіне нәтижелерді шығару WriteLn (write line — жолға жазу) процедурасының көмегімен жүзеге асады және ол *m* және *n* екі бүтін санды бір жолға шығарады. Осыдан кейін экрандағы меңзер келесі бос жолдың басына орналасады және программа жұмысы тоқтатылады.

Жұмыс барысында дұрыс жазу ережесі — программа синтаксисі қатаң сақталады. Паскальда тыныс белгілерінің бірегейлі анықталған нұсқаулары бар:

- нүктелі үтір (;) программа тақырыбынан кейін, айнымалыларды сипаттау бөлімінен кейін және әрбір оператордан кейін қойылады. **End** сөзінің алдында бұл белгіні қоймаса да болады**;**
- үтір (,) барлық тізімдердегі элементтерді ажыратушы белгі. (сипаттау бөліміндегі айнымалылар тізімінде, енгізілетін және шығарылатын шамалар тізімінде қолданылады).

Программалау тіліндегі қатаң синтаксис ең алдымен транслятор үшін қажет. Транслятор — формалды түрде орындалатын программа. Мысалы, егер айнымалылар тізімінде ажыратушы белгі ретінде үтір болуы керек болса, транслятор басқа кез-келген тыныс белгілерді қате есебінде қабылдайды. Егер нүктелі үтір

операторларды ажыратушы болса, онда транслятор бірінші қойылған нүктелі үтірден бастап келесі нүктелі үтірге дейін барлық программа мәтінін оператор ретінде қабылдай береді. Белгілі бір екі операторлардың арасына нүктелі үтір қойылмаса, транслятор оларды бір оператор ретінде қабылдайды деген сөз.

Синтаксистік ережелердің негізгі мақсаты — тілдік құрылымдарға бір мәнді мағынаны беру болып табылады. Егер қандай да бір құрылым екімәнді түсіндірілсе, онда ол құрылымда қате бар дегенді білдіреді. Сондықтан интуицияға сүйенбей, тілдің ережелерін жаттау керек.

Бұдан әрі, Паскальдың синтаксистік ережелерінің қатаң сипаттамасы берілетін болады, бірақ қазіргі уақытта тілдің алғашқы идеясын алу үшін қарапайым алгоритмдерді программалаудың бірнеше мысалына жүгінеміз.

N! факториал алгоритмін есептеуді Паскаль тіліне «Аударайық»:

```
алг факториал
бұт N, I, F
басы енгізу (N)
    F := 1
    I := 1
    әзірше I<=N
    цб F := F * I
        I := I + 1
    цс
    шығару F
соңы
```

```
Program Factorial;
Var N, I, F: Integer;
Begin ReadLn(N);
    F : = 1;
    I : = 1;
While I <= N Do
Begin F := F * I;
        I := I + 1
    End;
    WriteLn(F)
End.
```

Біріншіден, бұл мысалдан Паскальда шарты алдын ала берілген цикл операторы («әзірше» циклі) қалай жазылатынын көруге болады:

While <орындалу шарты> **Do** <цикл денесі>

While — әзірше, Do — орындау. Егер цикл денесінде операторлардың тізбегі бар болса, онда олардың басы мен соңында сөздерін жазу керек болатын *құрама операторы* қалыптастырылады деп айтуға болады. Begin және End қызметші сөздері бірнеше операторларды бір құрамға біріктіретін *операторлық жақшалар* деп аталады. Егер цикл денесі бір ғана оператор болса (құрама оператор емес), онда операторлық жақшаларды қою міндетті емес.

Осыдан кейін транслятор циклдің денесі ең жақын «;» белгісінде аяқталатынын түсінеді. Тағы бір мысал келтірейік:

```
алг түбірлер;
нақ a, b, c, d, x1, x2
басы
қайталау
  шығару («a , b , c  енгізу;  a ≠0»)
  енгізу a, b, c
  дейін  a ≠ 0
  d := b2 - 4ac

  егер d ≥ 0
  онда
  x1:=(-b + sqrt(d))/(2a)
  x2:=(-b -sqrt(d))/(2a)
  шығару x1, x2

  әйтпесе          шығару
  «Нақты түбірлер жоқ»
  тс
соңы
```

```
Program Roots;
Var a , b , c , d , x 1 , x 2 : Real; Begin
{берілгендерді енгізу}
  Repeat
    WriteLn('енгізу a, b, c; a < > 0 ' ) ;
    ReadLn(a, b, c)
  Until a <> 0;
{дискриминантты енгізу}
d := b*b - 4*a*c;
  If d >= 0
  Then Begin {Түбірлер бар}
  x1:=(-b + sqrt(d))/2/a);
  x2: = (-b - sqrt(d))/2/a);
WriteLn('x1=',x1,'x2=',x2);
        End
     Else WriteLn(Нақты
          түбірлер жоқ)
End.
```

Екіншіден, келтірілген мысалдан Паскальда циклдің (цб) басын және циклдің соңын (цс) белгілеудің арнайы сөздері жоқ екені анық осыған орай, барлық жағдайларда Begin және End қызметші сөздері бар.

Осы программада жоғарыда айтылғандармен салыстырғанда көптеген жаңа элементтер пайда болды. Паскальда нақты деректер типінің атауы бұл — Real.

Шарты соңынан берілген цикл («Дейін» циклі) мына операторлық жақшалардың рөлін орындайды. (Тең емес)

Шарты соңынан берілген цикл («Дейін» циклі) мына операторлық жақшалардың рөлін орындайды. (Тең емес)

```
Repeat <цикл денесі> Until <шарттың аяқталуы>
```

Repeat — қайталау, **Until** — дейін. Цикл денесі өзін бір немесе құрама оператор ретінде көрсетуі мүмкін, бірақ Begin және End сөздерін пайдалану талап етілмейді, себебі «Repeat» және «Until» сөздері операторлық жақшалардың рөлін орындайды. (Тең емес)

таңбасы «≠» Паскальда «<>» таңбасымен, ал (үлкен немесе тең) «≥» таңбасы Паскальда «>=» таңбасымен белгіленеді.

Арифметикалық өрнектерді жазу ережесін тереңірек кейін қарастырамыз. Формулаларда түбірлерді есептеу үшін қолданылатын стандартты ($\sqrt{x}$) квадрат түбір функциясы Паскаль тілінде sqrt(x) түрінде жазылады. Өрнектердегі амалдарды орындау тәртібі жақшалармен және амалдардың реттілігімен анықталады. Амалдардың орындалу тәртібі алгебрадағыдай анықталады.

Паскальдағы тармақталу шартты операторды пайдалана отырып программаланады, оның жазылу формасы мынадай:

```
If <шарт> Then <1-оператор> Else <2-оператор>
```

If — егер, **Then** — онда, **Else** — әйтпесе. 1-ші және 2-ші операторлар жай және құрама бола алады. Құрама операторларды **Begin** және **End** операторлық жақшаларында жазған абзал.

Алгоритмдік тілдегі сияқты шартты операторды толық емес формасы болуы мүмкін:

```
If <шарт> Then <оператор>
```

Бұл программаның ерекшелігі мәтінде түсініктемелерді пайдалану болып табылады. *Түсініктеме* – фигуралық жақшаларда {...} жазылған кез-келген символдар тізбегі. Сондай-ақ, түсініктеме ретінде жұлдызшалармен берілген жақшаларды да (* ... *) пайдалануға болады. Түсініктеме программаның ешқандай әрекетін анықтамайды, ол түсіндірме мәтін ретінде ғана қолданылады. Оны программаның бос орын қоюға болатын кез келген жерінде беруге болады. Программалаушы компьютер үшін емес, программа мәтінін неғұрлым анық түсіну мақсатында өзі үшін түсініктеме жазады.

Жақсы түсініктемеленген программалар өздігінен құжатталған деп аталады. Кейбір программаларда түсініктемелердің саны есептелетін операторлар санынан асып түседі.

Түсініктемелерді табысты қолдану - жақсы программалау стилінің белгісі.

Программаны ЭЕМ-де орындау үшін оны жадыға енгізіп, аударуға және орындау керек. Ол үшін компьютерде Turbo Pascal жүйесін құрайтын арнайы программалық қамтамасыз ету құралдары болуы керек.

ЖАТТЫҒУЛАР

Алгоритмдік тілден Паскаль тіліне «Аударыңдар»:

а) Евклид алгоритмін;

ә) үш мәннің үлкенін таңдау алгоритмін;

б) берілген үшбұрыштың қабырғаларының ұзындығын анықтайтын алгоритм;

в) тек қосу және азайту амалдарын пайдаланып екі сандарды көбейту алгоритмі;

г) бүтін бөлшекті бөлу және бүтін бөлудегі қалдықты есептеу алгоритмі;

2.2. ПАСКАЛЬДАҒЫ КЕЙБІР ПРОГРАММАЛАУ ЖҮЙЕЛЕРІ ТУРАЛЫ МАҒЛҰМАТ

Программалау жүйесі — нақты программалау тілінде программалар әзірлеуге арналған жүйелік құралдар жинағы. Қазіргі заманғы программалау жүйелеріне мыналар жатады:

- транслятор (компилятор немесе интерпретатор);
- программаларды өңдеудің біріктірілген ортасы;
- орнатылған мәтіндік редактор;
- стандартты программалар мен функциялар кітапханасы;
- программаны дұрыстау құралдары;
- орнатылған анықтамалық қызметтер.

1980 жылдары Borland фирмасы Turbo Pascal деп аталатын стандартты Pascal тілінің модификацияланған нұсқасында жұмыс істей бастады. Сонымен қатар, IBM PC дербес компьютерлерге арналған Turbo Pascal тілі үшін программалау жүйесі жасалды.

Кейінірек, «Турбо Паскальмен» программалау жүйесін де атай бастады. Turbo Pascal программалау жүйесі MS DOS операциялық жүйесінің ортасында жұмыс істеуге арналған.

Турбо Паскаль (тіл және программалау жүйесі ретінде) өзінің пайда болған уақытынан бері айтарлықтай өзгерді.

1980-шы жылдардың ортасында Borland фирмасы Турбо Паскальдың бірінші нұсқасын шығарды. Содан кейін 3.0, 4.0, 5.0, 5.5, 6.0, 7.0 нұсқалары ретінде жүйенің алты модификациясы жасалды. Олардың әрқайсысы алдыңғы нұсқаны жақсарту мақсатында ойластырылған болатын. Олардың барлығы IBM PC машиналары үшін жасалды және компьютерлермен бірге жетілдірілді.

3.0 нұсқасы төменгі қуатты компьютерлерге (IBM PC / XT) бағытталған. Онда әзірленген программалар ұзындығы шектеулі (64 кб-тан аспайды), өзара байланысқан программалардың жеке

компиляцияға арналған құралдары жоқ және операциялық ортасы аса жетілмеген болды. Негізгі өзгерістер 4.0 нұсқасына енгізілді. Бұл нұсқада қазіргі заманғы диалогты орта, программалық модульдерді жеке компиляциялау құралы, қуатты графикалық кітапханасы болды.

5.0 нұсқасы негізінен орнатылған дұрыстау құралы бар даму ортасының жақсаруы арқылы ерекшеленді. 5.5 нұсқасында алғаш рет ОБП - программаларды жасаудың заманауи технологиясын қолдауға арналған құралдар енгізілді.

6.0 нұсқасының негізгі айырмашылығы – енгізу құрылғысы «тышқанның» жұмысына бағытталған және көп терезелік жұмыс режимін қолдануға арналған жаңа орта; объектіге - бағытталған кітапхана Turbo Vision, сондай-ақ Ассемблер программасы командаларының мәтіндерін қосу мүмкіндігі болып табылады.

7.0 нұсқасы 6.0 нұсқасымен салыстырғанда ешқандай негізгі жаңалықтарды қамтымаған. Программалау тілінің кейбір кеңейтімдері, сондай-ақ жүйелік қабықтың қосымша қызмет мүмкіндіктері енгізілді. Турбо Паскальдың жоғарғы нұсқаларына «Object Pascal» атауы пайдаланылды.

Windows операциялық жүйесі MS DOS-та әзірленген қосымшалардың жұмысын қолдайды. Бұл Турбо Паскальға қатысты. Windows операциялық жүйесінде жұмыс істеуге арналған Паскальдың бірнеше заманауи программалау жүйелері қарастырылған. Олардың ішінде еркін тараған жүйелер бар: Free Pascal, GNU Pascal. Оңтүстік федералдық университетінде әзірленген Pascal ABC тілі және программалау жүйесі Ресейдегі оқу орындарында программалауды үйрету үшін кеңінен пайдаланылады.

Осы тараудың келесі бөлімдерінде Паскаль тілі Турбо Паскальдың жоғарғы нұсқаларына сәйкес сипатталады. Сонымен қатар, зерттелген көлемде Паскальдың қазіргі заманғы іске асырылуымен тілдің негізгі құралдарының дәйектілігі қамтамасыз етіледі.

2.3. ТУРБО ПАСКАЛЬ ТІЛІНІҢ ЭЛЕМЕНТТЕРІ

Алфавит. Турбо Паскаль тілінің алфавиті әріптерді, сандарды және арнайы символдарды қамтиды:

- А-дан *Z*-ке, *a-дан z-ке* дейінгі латын әріптері (бас әріптері мен кіші әріптері); сандар 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

- оналтылық жүйедегі сандар 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F;
- арнайы символдар + - * / = <> [ ]. , ( ) : ; { } л @ $ #.

Келесі арнайы символдардың тұтас комбинацияларын бос орынмен ажыратуға болмайды:

: = меншіктеу операторы;
>= үлкен немесе тең;
<= кіші немесе тең;
<> тең емес;
(* *) түсініктеме (немесе { }) жазу жақшалары;
(..) эквивалент [ ].

Бос орындар — (ASCII-32) бос орындар символы және ASCII (0 –ден 31-дейін) кодының барлық басқару символдары.

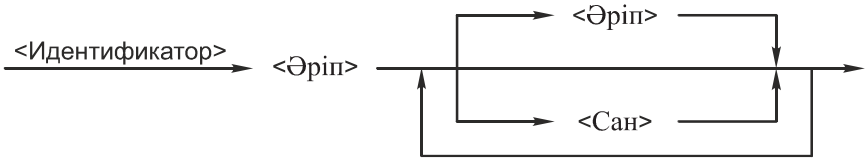
Арнайы белгілерге сондай-ақ *қызметші сөздер* кіреді, олардың мағынасы бірдей анықталған және басқа мақсаттарда оларды пайдалануға болмайды. Тіл үшін бұл бірегей символдар.

Паскаль тілінің қызметші сөздері: absolute, and, array, begin, case, const, div, do, downto, else, end, external, file, for, forward, function, goto, if, implementation, in, inline, interface, interrupt, label, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

Турбо Паскальдың ең соңғы нұсқалары объектілермен жұмыс істеуге және орнатылған ассемблерге қатысты бірнеше қосымша сөздерді қамтиды.

Идентификаторлар. Идентификатор дегеніміз - белгілі бір программа нысанының символикалық атауы: тұрақтылар, айнымалылар, мәліметтер типтері, процедуралар, функциялар, программалар. Синтаксистік диаграмманы пайдалана отырып, идентификаторды 2.1-суреттегідей көрсетуге болады.

Идентификаторға келесі түрдегідей анықтама берсек те болады: идентификатор — бұл әріптен басталатын кез-келген әріптер мен сандардың тізбегі.



2.1-сур. Идентификатордың синтаксистік диаграммасы

Турбо Паскальда астыңғы сызықша белгісі де әріптер сияқты қызмет атқарады.

Идентификаторлар мен қызметші сөздердегі кіші және үлкен әріптердің айырмашылығы жоқ. Мысалы: max, MAX, MAX, mAx – барлығы бір атау.

Турбо Паскальда идентификатордың ұзындығы ерікті болуы мүмкін, бірақ тек алғашқы 63 таңба ғана маңызды.

Түсініктемелер. Түсініктемелер келесі түрде жазылады, компилятор оларды қабылдамайды:

```
{символы жоқ кез-келген мәтін «}» }
(*символы жоқ кез-келген мәтін «*)» *)
```

Қазақ алфавиті тек осы литерлік және мәтіндік тұрақтылар мен түсініктемелерде ғана қолданылады.

«{$» немесе «(*$» символдарынан басталатын жолдар компиляторлардың директивалары болып табылады. Осы сөздерден кейін компилятор командаларының мнемоникасы орындалады.

2.4. МӘЛІМЕТТЕР ТИПТЕРІНІҢ ТҰЖЫРЫМДАМАСЫ

Мәліметтер типтерінің тұжырымдамасы кез келген программалау тілінде орталық болып табылады. Шаманың типімен үш қасиет байланысты: *ішкі көрініс формасы, қабылданған мәндердің жиынтығы және рұқсат етілген амалдар жиынтығы.* Турбо Паскаль көптеген мәліметтер типтерімен сипатталады (2.2 - сурет).

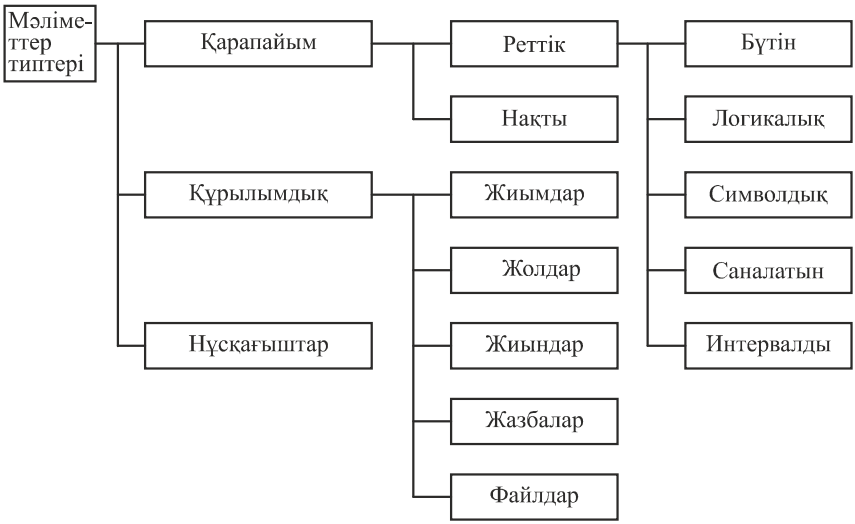
Стандартты Паскальда жолдық мәліметтер типі жоқ. Сонымен қатар, Турбо Паскальдағы бүтін және нақты типтер - бұлар топтық мәліметтер типтері. Турбо Паскальдың кейінгі нұсқаларында процедуралық мәліметтер типі және «объект» мәліметтер типі бар.

Әрбір мәліметтер типінің өз идентификаторы бар.

Турбо-Паскальда анықталған мәліметтердің қарапайым типтері туралы ақпарат 2.1-кестеде келтірілген. Нақты мәліметтер типтері үшін жақшадағы мантиссаның сақталатын саны деректер санның ондық көрінісі ретінде көрсетілген.

Стандартты Паскальда нақты мәліметтер типтерінің ішінен тек Real, ал бүтін типтерден тек Integer анықталған.

Single, Double, Extended типтері Паскаль-программаларда ДК «жылжымалы арифметикасы» бар сопроцессорлармен қамтылған жағдайда ғана қолданылады. (Intel-80486 бастап IBM PC процессорлары үшін бұл шарт әрқашан орындалады) .



2.2-сурет. Турбо Паскальдағы мәліметтер типтерінің құрылымы

Мәліметтер типі *реттік* деп аталады, егер ол саналатын мәндер жиынынан тұрса және оларды нөмірлей алатын болсақ. Бұл мәндердің жиынтықтары үшін «келесі» және «алдыңғы» түсініктері туралы айтуға болады.

Айнымалыларды сипаттау. Программада пайдаланылатын барлық айнымалы мәндер үшін олардың типтері айнымалылар бөлімінде көрсетілуі керек. Айнымалы бөлімінің құрылымы 2.3-сур. көрсетілген

Программадағы айнымалы бөлімінің мысалы:

```
Var  m, n, k : Integer;
x, y, z : Real;
Symbol : Char;
```

Тұрақтылар. Тұрақтылар типі мәтін мәнімен, яғни оның программадағы жазылу формасымен анықталады.

Бүтін ондық тұрақтылар таңбалы немесе таңбасыз қарапайым бүтін сан формасында жазылады. Мысалы: 25, -24 712, 376.

Оналтылық бүтін тұрақтылар «$» префиксімен жазылады. Олар $00000000–ден $FFFFFFFF дейінгі аралықта орналасуы керек.

Бекітілген нүктелі нақты тұрақтылар қарапайым бүтін бөлшекті ондық сандар түрінде жазылады. Бүтін және бөлшек бөлімдердің арасына нүкте қойылады. Мысалы: 56.346, 0.000055, -345678.0.

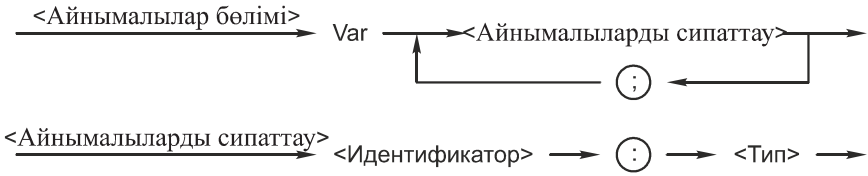
2.1-кесте. Турбо Паскальдағы мәліметтер типтері

Идентификатор	Ұзындығы, байт	Мәндер диапазоны (жиыны)
Бүтін		
Integer	2	-32768..32767
Byte	1	0..255
Word	2	0..65535
Shortint	1	-128..127
Longint	4	-2147483648..2147483647
Нақты		
Real	6	$2{,}9 \cdot 10^{-39}$...$1{,}7 \cdot 10^{38}$ (11...12)
Single	4	$1{,}5 \cdot 10^{-45}$...$3{,}4 \cdot 10^{38}$ (7.8)
Double	8	$5 \cdot 10^{-324}$...$1{,}7 \cdot 10^{308}$ (15.16)
Extended	10	$3{,}4 \cdot 10^{-4932}$…$1{,}1 \cdot 10^{4\,932}$ (19.20)
Логикалық		
Boolean	1	True, False
Символдық		
Char	1	ASCII кодының барлық символы

Жылжымалы нүктелі нақты тұрақтылар келесі түрде жазылады:

<мантисса> E <реті>

Мұндағы <мантисса> — бекітілген нүктелі бүтін немесе нақты сан, <реті> — таңбалы немесе таңбасыз бүтін сан. Мысалы: 7E-2 ($7 \cdot 10^{-2}$), 12.25E6 ($12{,}25 \cdot 10^{6}$), 1E-25 (10^{-25}) .



2.7-сур. Саналатын мәліметтер типінің сипатталуы

Символдық тұрақты — бұл апострофта жазылған алфавиттің кез-келген символы. Мысалы: 'W', '!', '9'.

Логикалық тұрақты — бұл мына сөздер: True, False.

Жолдық тұрақты — бұл апострофта жазылған символдар жолы. Мысалы: 'TurboPascal', 'Жауап: ', '35-45-79'. Жолдық тұрақтының максималды ұзындығы 255 символды құрайды.

Программаның константалар бөлімінде тұрақтыларға белгілі бір атау берілуі мүмкін. Мысалы:

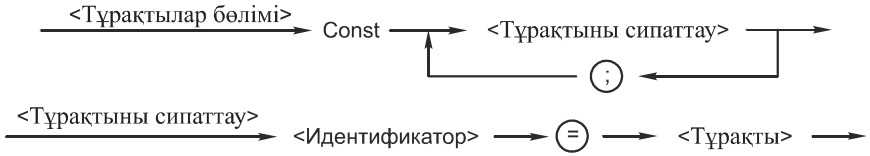
```
Const
   Max = 1000;
   G = 9.81;
   Cod = 'Қате';
```

Тұрақтылар бөлімінің құрылымы 2.4-суретте көрсетілген. Турбо Паскальда сондай-ақ *типтелген тұрақты* мәндерін пайдалануға болады. Типтелген тұрақты мән бастапқы мән берілетін айнымалыға ұқсас. Бұл компиляция кезеңінде болатын әрекет. Мысалы:

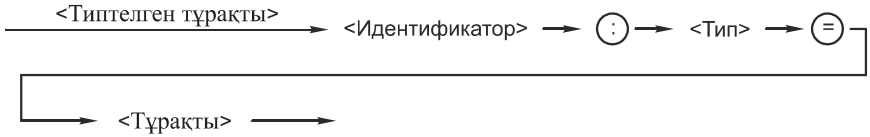
```
Const   NumberCard  :  Integer  =  1267;
        Size        :  Real     =  12.67;
        Symbol      :  Char     =  '*';
```

Типтелген тұрақтының сипатталуы 2.5-суретте көрсетілген.

Турбо Паскальда белгілі бір мәндер үшін алдын ала сақталған бірқатар атаулар бар, олар программада алдын ала анықтамасыз пайдаланылуы мүмкін (2.2-кесте)



2.4-сурет.Тұрақтылар бөлімінің құрылымы



2.5-сурет. Типтелген тұрақтының сипатталуы

2.2-кесте. Турбо Паскальдың алдын ала сақталған тұрақтылары		
Идентификатор	Тип	Мән
True	Boolean	АҚИҚАТ
False	Boolean	ЖАЛҒАН
MaxInt	Integer	32 767

Қолданушының мәліметтер типі. Паскальдағы тағы бір жақсы мүмкіндік – қолданушы өзі мәліметтер типін тағайындай алады. Ол типтер әрқашан Паскальдың стандартты мәліметтер типтеріне негізделеді.

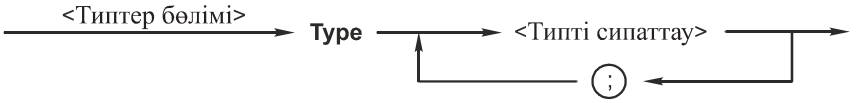
Паскальда қолданушының мәліметтер типін сипаттауы үшін арнайы типтер бөлімі қарастырылған. Ол бөлімнің құрылымы 2.6-суретте көрсетілген.

Саналатын типтердің мәліметтері олардың айнымалылары қабылдай алатын барлық мәндерді санау қабілеті бойынша тағайындалады. (2.7-сур.)

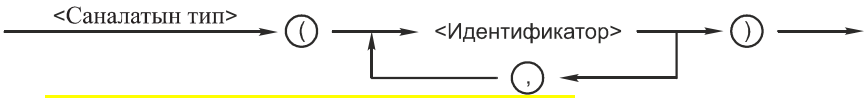
Анықталған мәліметтер типінің атауы одан кейін айнымалыларды сипаттау үшін қолданылады. Мысалы:

```
Type  Gaz = (C, O, N, F);
      Metal = (Fe, Co, Na, Cu, Zn);
Var   G1, G2, G3: Gaz;
      Metl, Met2: Metal;
      Day:  (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
```

Мұндағы Gaz және Metal — G1, G2, G3 и Met1, Met2 айнымалыларына сәйкес қойылған саналатын мәліметтер типтерінің атаулары.



2.6-сурет. Мәліметтер типі бөлімінің құрылымы



2.7-сурет. Мәліметтер типі бөлімінің құрылымы

2.8-сур. Интервалды мәліметтер типтерінің сипатталуы

Day айнымалысына атау меншіктелінбеген оған саналатын тип тағайындалған.

Саналған мәліметтер типіне жататын мәндер - тұрақты мәндер болып табылады. Оларға қатысты әрекеттер тұрақтыларға қолданылатын ережелерге бағынады. Саналатын типтегі әрбір мән жадыда 2-байт орын алады, сондықтан элементтердің саны 65 535-тен аспауы керек.

Саналатын мәліметтер типі — реттелген жиын. Оның элементтері 0-ден бастап нөмірленген.

Программада бұрын келтірілген сипаттаулары бар келесі фрагменттің болуы мүмкін:

```
If Day = Sun Then WriteLn('Алақай! Бүгін демалыс!');
```

Интервал типтегі мәліметтер (2.8-сурет) кейбір реттік санының шектелген ішкі жиыны ретінде анықталады.

Бірінші тұрақтының реттік нөмірі берілген негізгі типтегі екінші тұрақтының реттік нөмірінен үлкен болмауы керек.

Программа орындалған кезде айнымалы интервал типінің мәні автоматты түрде орнатылған аралық үшін бақыланады. Диапазоннан шығып кеткен уақытта программаны орындау тоқтатылады. Мысалы:

```
Type  Numbers = 1..31;
      Alf = 'A'..'Z';
Var   Data: Numbers;
      Bukva: Alf;
```

2.5. АРИФМЕТИКАЛЫҚ АМАЛДАР, ФУНКЦИЯЛАР, ӨРНЕКТЕР. МЕНШІКТЕУ ОПЕРАТОРЫ

Арифметикалық мәліметтер типіне нақты және бүтін типтер топтары жатады. Оларға арифметикалық амалдар мен қатынас амалдары қолданылады.

Амалдардың *унарлы* (бір операндқа қолданылады) және *бинарлы* (екі операндқа қолданылады) түрлері бар.

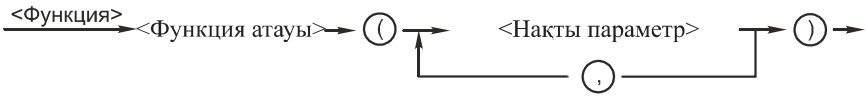
2.3-кесте. Паскальдың бинарлық амалдары

Таңба	Өрнек	Операнд типі	Нәтиже типі	Амал
+	A + *B*	R, R	R	Қосу
		I, I	I	
		I, R; R, I	R	
-	*A* - *B*	R, R	R	Азайту
		I, I	I	
		I, R; R, I	R	
*	*A* * *B*	R, R	R	Көбейту
		I, I	I	
		I, R; R, I	R	
/	*A*/*B*	R, R	R	Бөлу
		I, I	R	
		I, R; R, I	R	
div	*A* div *B*	I, I	I	Бүтін бөлу
mod	A mod	I, I	I	Бүтін
				қалдық

Унарлы арифметикалық амал біреу — бұл келесі форматты таңбасының өзгеруі:

-<шама>.

Паскаль стандартты тілінің бинарлы арифметикалық амалдары 2.3-кестеде сипатталған. Мұндағы: I — бүтін типті мәліметтер, ал R — нақты типті мәліметтер.



2.9-сур. Функцияны шақыру құрылымы

2.4-кесте. Турбо Паскальдың стандартты математикалық функциялары

Паскальда жазылуы	Аргумент типі	Нәтиже типі	Функция
Pi	—	R	Сан n = 3.1415926536E+00
abs (x)	I, R	I, R	x аргументінің модулі
arctan (x)	I, R	R	x-тің арктангенсі, рад
cos (x)	I, R	R	x-тің косинусы, рад
exp (x)	I, R	R	e^x — x-тің экспонентасы
frac (x)	I, R	R	x-тің бөлшек бөлімі
int (x)	I, R	R	x-тің бүтін бөлігі
ln (x)	I, R	R	x-тің натурал логарифмі
random		R	[0, 1]интервалындағы кездейсоқ сан
random (x)	I	I	[0, x] интервалындағы кездейсоқ сан
round (x)	R	I	x –ті жақын бүтін санға дейін дөңгелектеу
sin (x)	I, R	R	x-тің синусы, рад
sqr (x)	I, R	I, R	x-тің квадраты
sqrt (x)	I, R	R	x-тің квадрат түбірі
trunc (x)	R	I	x-тің бүтін бөлігін шығару

Арифметикалық шамаларға Паскаль тілінің стандартты функцияларын қолдануға болады. Функцияны шақыру құрылымы 2.9-суретте сипатталған.

Өрнектегі функция операнд ретінде әрекет етеді. Мысалы,

X := 2 * sin(A)/ln(3.5) + cos(C - D)

тағайындау операторында жазылуы математикадағыдай операнд рөлін үш функция: sin, ln, cos атқарып тұр. Аргументтер нақты параметрлер деп аталады, жалпы жағдайда олар арифметикалық типтің өрнектері болып табылады және жақшаның ішінде жазылады. Функцияның нәтижесі сәйкес типті шама болады.

Турбо Паскальдың стандартты математикалық функцияларының сипатталуы 2.4-кестеде келтірілген.

Арифметикалық өрнек сандық мәндердегі әрекеттердің орындалу тәртібін тағайындайды. Арифметикалық өрнектер арифметикалық амалдарды, функцияларды, операндтарды және жақшаларды қамтиды. Бір тұрақты немесе бір айнымалы арифметикалық өрнектің қарапайым түрі болып табылады.

Мысалы,

$$\frac{2a + \sqrt{0{,}5\sin(x + y)}}{0{,}2c - \ln(x - y)}$$

математикалық өрнегі, Паскаль тілінде былай жазылады:

```
(2 * A + sqrt(0.5 * sin(X + Y)))/(0.2 * C - ln(X - Y)).
```

Арифметикалық өрнектерді дұрыс жазу үшін белгілі ережелерді сақтаған жөн:

1. Барлық символдарды бір жолда жазу. «*» таңбасын қалдырмай барлық амалдар таңбаларын қою.

2. Екі амал таңбаларының қатар қойылып кетпеуін қадағалау, яғни A + -B деп жазған дұрыс емес, дұрыс жазылуы: A + (-B).

3. Басымдылығы жоғары амалдарды бірінші орындау. Басымдылықтарының кему реті келесідей болу керек:

- функцияны есептеу;
- (-) таңбасын ауыстырудың унарлы амалы;
- *, /, div, mod;
- +, -.

4. Басымдылықтары бірдей бірнеше амалдардың қатар жазылуы кезінде амалдар ретімен солдан оңға қарай орындалады.

5. Жақшаның ішіндегі амал бірінші орындалады. (Мысалы, (A + B) * (C - D) өрнегінде көбейту амалы қосу мен азайту амалдарынан кейін орындалады)

6. Математикалық мәні жоқ өрнектерді жазудың қажеті жоқ. Мысалы, нөлге бөлу, теріс санның логарифмі және т.б.

Мысал келтірейік. Берілген ереже бойынша жазылған арифметикалық өрнек (дөңгелектің ішіндегі сандар амалдардың орындалу ретін білдіреді),

① ⑦ ④ ⑤ ③ ⑥ ② ⑫ ⑪ ⑩ ⑧ ⑨

```
(1 + y) * (2 * x + sqrt(y) - (x + y))/(y + 1/(sqrt(x) - 4))
```

төмендегі математикалық формулаға сәйкес келеді:

$$(1+y)\frac{2x+\sqrt{y}-(x+y)}{y+\frac{1}{x^2-4}}$$

Паскальда кез-келген санды дәрежеге шығарудың ешқандай амалы немесе стандартты функциясы жоқ. *x*y мәнін есептеу үшін келесі әрекеттерді орындау ұсынылады:

- егер *y* — бүтін мән болса, көбейту амалын қолданған жөн, мысалы: $x^3 \rightarrow x * x * x$. Үлкен дәрежелер үшін циклмен көбейтуді қолданған дұрыс;
- егер *y* — нақты мән болса, келесі математикалық формула қолданылады: $xy = e^{y \ln(x)}$. Бұл өрнек Паскальда мына түрде жазылады:

```
exp(Y * ln(x))
```

Бұл жағдайда, *y* нақты типте болса, *x*-тің мәні теріс және 0-ге тең емес болу керектігі көрініп тұр. Ал *y* бүтін типті болса, ешқандай шектеулер қойылмайды.

Мысалы, $\sqrt[3]{a+1} = (a+1)^3$ формуласы, Паскальда мына түрде жазылады:

```
exp(1/3 * ln(A + 1))
```

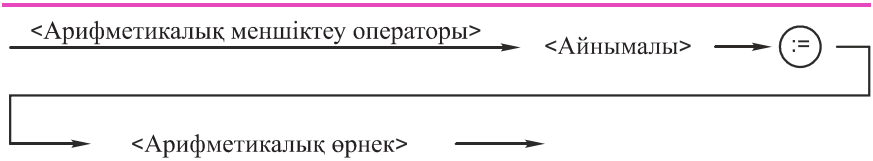
Есептеу нәтижесінде бүтін типті шама алынса, өрнек бүтін типті болады. Ал егер есептеу нәтижесінде нақты шама алынса, өрнек нақты типті болады.

Арифметикалық меншіктеу операторының құрылымы 2.10-суретте көрсетілген.

Мысалы,

```
Y := (F * N -4.5)/cos(X).
```

Меншіктеу операторының орындалу тәртібі алдын ала қарастырылады. Тек келесі ережеге назар аудару қажет:



2.10-сурет. Арифметикалық меншіктеу операторының құрылымы

айнымалы типі мен өрнек типі бірдей болу керек. Ерекше жағдай: Өрнек бүтін типті, ал айнымалы - нақты типті болуы мүмкін.

ЖАТТЫҒУЛАР

1. Мына формулаларды Паскаль тілінде жазыңдар:

а) $a + bx + cyz;$ ә) $[(ax - b)x + c]x - d;$ б) $\frac{a+b}{c} + \frac{c}{ab};$

в) $\frac{x+y}{a_1}\frac{a_1}{x-y}$ г) $10^4\alpha - 3\frac{1}{5}\beta$ ғ) $\left(1 + \frac{x}{2!} + \frac{y}{3!}\right)/(1 + \frac{2}{3+xy})$

2. Паскальдағы өрнектерді математикалық формула түрінде жазыңдар:

а) `(p + q)/(r + s) - p * q/(r * s);`
ә) `1E3 + beta/(x - gamma * delta);`
б) `a/b * (c + d) - (a - b)/b/c + 1E - 8.`

3. Неліктен Паскальда функция аргументі әрқашан жақшаның ішіне жазылады? Мысалы, ln5 емес ln(5) түрде жазылады

4. Арифметикалық амалдарды Паскаль тілінде жазыңдар:

а) $(1 + x)^2;$ ә) $\sqrt{1 + x^2};$ б) $\cos^2 x^2;$ в) $\log_2\frac{x}{5};$

г) $\arcsin x$ ғ) $\frac{e^x + e^{-x}}{2};$ д) $x^{\sqrt{2}};$ е) $\sqrt[3]{1 + x};$

ж) $\sqrt{x^3 + 8^x};$ з) $\frac{xyz - 3{,}3|x + \sqrt[4]{y}|}{10^7 + ln4!};$ и) $\frac{\beta + sin^2\pi^4}{cos2 + |ctg\gamma|}.$

5. Вычислить значения следующих выражений

а) `trunc(6.9);`
ә) `trunc(6.2);`
б) `20 div6;`
в) `2 div 5;`
г) `round(6.9);`
ғ) `round(6.2);`
д) `20 mod 6;`
е) `2 mod 5;`
ж) `3 * 7 div 2 mod 7/3 - trunc (sin(1)).`

6. Определить типы следующих выражений:

а) `1+0.0;`
ә) `20/4;`
б) `sqr (4);`
в) `sqrt (16);`
г) `sin (0);`
ғ) `trunc (-3.14).`

7. Меншіктеу операторларының қайсысы дұрыс, егер *y* — нақты айнымалы, ал *n* — бүтін айнымалы болса:

а) `y := n + 1;`
ә) `n : = y - 1;`
б) `n : = 4.0;`
в) `y : = trunc(y);`
г) `y : = n div 2;`
ғ) `y : = y div 2;`
д) `n := n/2;`
е) `n := sqr(sqrt(n)).`

8. *x* және y бүтін айнымалылардың мәнін қосымша айнымалы пайдаланбай орындарын ауыстырыңдар. Құрылған алгоритмнің кемшілігін үшінші айнымалыны қолданып мәндерді алмастыру әдісімен салыстырыңдар. Осы алгоритмді нақты сандарға қолдануға бола ма?

9. *h* бүтін айнымалысына *к* оң бүтін санының жазбасындағы жүздік разрядта тұрған санының мәнін меншіктеуді орындаңдар. (мысалы, егер *к* = 28 796, онда *h* = 7).

10. *S* бүтін айнымалысына *к* бүтін үшорынды цифрларының қосынды мәнін меншіктеудің орындаңдар.

11. Келесі программа қандай есепті орындап жатқанын анықтаңдар:

```
Program Test;
Type    Natur = 1..MaxInt;
Var     N : Natur;
        X : Real;
Begin ReadLn(N);
      X := 0;
    While N > 0 Do
    Begin
      X : = X + 1.0;
      N := N - 1
    End;
    WriteLn(X)
End.
```

Алынған нәтижені басқаша қарапайым тәсілмен алуға бола ма?

2.6. ПЕРНЕТАҚТАДАН МӘЛІМЕТТЕРДІ ЕНГІЗУ ЖӘНЕ ЭКРАНҒА ШЫҒАРУ

Мәліметтерді енгізу — бұл сыртқы құрылғылар арқылы жедел жадыға ақпарат жіберу. Негізінен шығарылатын есептің бастапқы берілгендері енгізіледі.

Мәліметтерді шығару — бұл жедел жадыдан сыртқы құрылғыларға (басып шығару, дисплей, магниттік құрылғылар және

т.б.) ақпарат жіберу. Кез-келген есептің нәтижелері шығарылу керек.

Дербес компьютердің негізгі енгізу-шығару құрылғылары пернетақта мен дисплей (монитор экраны) болып табылады. Дәл осы құрылғылардың арқасында негізінен адам мен ДК арасындағы диалог жүзеге асырылады.

Пернетақтадан енгізу процедурасының жазылу форматы келесідей:

```
Read (<енгізу тізімі>)
```

Мұндағы <енгізу тізімі> — бұл үтірмен ажыратылған айнымалылар атауының тізбегі, ал Read (оқу) — стандартты енгізу процедурасын қайтару операторы. Мысалы:

```
Read (a, b, c, d)
```

Бұл оператор орындалған кезде компьютердің жұмысы үзіледі, содан кейін пайдаланушы пернетақтадағы a, b, c, d айнымалы мәндерін бос орын арқылы енгізеді. Енгізілген мәндер экранда көрсетіледі. Енгізу соңында <Enter> пернесі басылады. Мәндерді енгізу Паскаль тілінің синтаксисіне сәйкес қатаң орындалуы керек. Мысалы, программаны енгізу барысында

```
Var T : Real;
    J : Integer;
    K : Char;
Begin
    Read(T, J, K);
```

пернетақта арқылы мына мәндерді енгізу керек

```
253.98 100 G [Enter].
```

Егер программада бірнеше Read операторы болса, онда олар үшін деректер ағыммен енгізіледі, яғни бір Read операторы үшін айнымалы мәндерді оқығаннан кейін, келесі оператордың деректері экрандағы дәл сол жолдың соңына дейін оқылады, осыдан кейін келесі жолға өту орындалады. Мысалы, келесі программаны жазу барысында

```
Var A, B : Integer;
    C, D : Real;
Begin
    Read(A, B);
    Read(C, D);
```

пернетақта арқылы мына мәндерді енгізу керек,

```
18758 34[Enter] 2.62E-02 1.54E+01[Enter].
```

Пернетақтадан енгізу операторы

```
ReadLn(<енгізу тізімі>)
```

түрінде де бола алады.

Мұнда ReadLn (read line-сөзінен) - жолды оқу. Read операторынан айырмашылығы, бір ReadLn операторына арналған мәндер тізіміндегі соңғы нұсқаны оқып болған соң, келесі ReadLn операторына арналған деректер жаңа жолдың басынан оқылады. Егер алдыңғы мысалда, Read операторын ReadLn -мен ауыстырсақ, яғни жазсақ,

```
ReadLn(A, B);
ReadLn(C, D);
```

мәндерді енгізу екі жолда жүзеге асырылады:

```
18758 34 [Enter]
2.62E-02 1.54E+01 [Enter]
```

Экранға шығару операторы (стандартты шығару процедурасын шақыру) келесі түрде жазылады:

```
Write(<шығару тізімі>)
```

Мұнда шығару элементтері тізімі әр түрлі өрнектер типтері болуы мүмкін (көбінесе, тұрақтылар мен айнымалылар), Мысалы:

```
Write(234);                          {Бүтін тұрақты шығарылады}

Write(A + B - 2);                    {Өрнекті есептеу нәтижесі шығарылады

Write(X, Summa, Arg1, Arg2);         {Айнымалы мәндері шығарылады}
```

Бір жолда бірнеше сандарды шығарғанда, олар бір-бірінен бос орындар арқылы бөлінбейді, бұл әрекеттерді программалаушы қадағалаған дұрыс. Мысалы: $I = 1$; $J = 2$; $K = 3$.

```
Write(I, '   ', J, '  ', K)
```

операторының орындалу барысында экранда мынандай жол пайда болады: 1 2 3.
Соңғы символ шығарылғаннан кейін, меңзер сол жолда қалады және экрандағы келесі енгізу меңзер тұрған орыннан басталады.

Экранға шығару процедурасы келесі түрде де болуы мүмкін:

```
WriteLn(<шығару тізімі>)
```

Мұндағы WriteLn (Write line-нан) — жолды жазу. Бұл оператор әрекетінің Write операторынан айырмашылығы тізімдегі соңғы мәнді шығарғаннан кейін меңзер келесі жолдың басына жылжытылады. Параметрлерсіз жазылған WriteLn жолдың ауыстыруын орындайды.

Шығару форматтары. Шығару тізімінде шығару формат нұсқағыштары (пішімдер) болуы мүмкін. Формат экрандағы шығарылған мәннің көрінісін анықтайды және тиісті элементтен қос нүкте арқылы бөлінеді. Егер формат нұсқағышы болмаса, машина алдын ала үнсіз келісім бойынша белгілі бір ережеге сәйкес мәнді шығарады.

Бұдан әрі, қысқаша түрде анықтама түрінде форматталмаған және форматталған әртүрлі типтегі мәндердің ережелері мен мысалдары келтіріледі. Шығару тізімін көрсету үшін біз келесі белгілеулерді пайдаланамыз:

I, P, Q — бүтінсанды өрнектер;
R — нақты типті өрнектер;
B — бульдік типті өрнектер;
Ch — символдық шама;
S — жолдық өрнек;
сан;
* «+» немесе «-» таңбасы;
_ бос орын.

Write процедурасының форматы

I — меңзердің тұрған орнынан бастап I шамасының ондық көрінісі шығарылады:

I мәні	Оператор	Нәтиже
134	Write(I)	134
287	Write(I,I,I)	287287287

I:P —өрістің P ені бойынша шеткі оң позициясына I шамасының ондық көрінісі шығарылады.

I мәні	Оператор	Нәтиже
134	Write(I:6)	_ 134
312	Write((I + I):7)	_ 624

R — ені 18 символдық өріске жылжымалы нүктелі форматында R шамасының ондық көрінісі шығарылады (егер R ≥ 0.0, болса #.##########E*## форматы қолданылады; егер R < 0.0, _-#.##########E*##) форматы қолданылады:

R мәні	Оператор	Нәтиже
715.432	Write(R)	_ 7.1543200000E+02
-1.919E+01	Write(R)	_ -1.9190000000E+01

R:P — P символды ені бойынша өрістің оң жақ шеткі позициясына қалыпты жылжымалы нүктелі форматта R мәнінің ондық көрінісі шығарылады. Оң сандар үшін шығару өрісінің минималды ұзындығы жеті символ, ал теріс сандар үшін сегіз символ болып табылады. Нүктеден кейін кем дегенде бір сан шығарылады:

R мәні	Оператор	Нәтиже
511.04	Write(R:15)	5.110400000E+02
46.78	Write(-R:12)	-4.67800E+01

R:P:Q — P символды ені бойынша өрістің оң жақ шеткі позициясына бекітілген нүктелі форматта R мәнінің ондық көрінісі шығарылады, ондық нүктеден кейін Q саны шығарылады (0 < Q < 24), мұнда санның бөлшек бөлігі қамтылады (егер Q = 0 болса, онда бөлшек бөлім де, ондық нүкте де шығарылмайды; егер Q > 24 болса, онда шығару барысында жылжымалы нүктелі формат қолданылады):

R мәні	Оператор	Нәтиже
511.04	Write(R:8:4)	511.0400
-46.78	Write(R:7:2)	_-46.78

Ch:P — P ені бойынша өрістің шеткі оң позициясына Ch мәні шығарылады:

Ch мәні	Оператор	Нәтиже
'X'	Write(Ch:3)	X
'!'	Write(Ch:2,Ch:4)	! !

S — меңзер тұрған орыннан бастап S мәні шығарылады:

S мәні	Оператор	Нәтиже
'Day N'	Write(S)	Day N
'RRDD'	Write(S,S)	RRDDRRDD

S:P — S мәні P символдарының ені бойынша өрістің шеткі оң позициясына шығарылады:

S мәні	Оператор	Нәтиже
'Day N'	Write(S:10)	Day N
'RRDD'	Write(S:5,S:5)	RRDD RRDD

B — меңзер тұрған орыннан бастап B өрнегінің нәтижесі шығарылады (True немесе False):

B мәні	Оператор	Нәтиже
True	Write(B)	True
False	Write(B, Not B)	FalseTrue

B:P — P символдарының ені бойынша өрістің шеткі оң позициясына бульдік өрнектің нәтижесі шығарылады:

B мәні	Оператор	Нәтиже
True	Write(B:6)	True
False	Write(B:6, Not B:7)	False True

2.7. ЭКРАНҒА СИМВОЛДЫҚ ШЫҒАРУДЫ БАСҚАРУ

Экранға шығару үшін Write және WriteLn процедураларын ғана қолдану программалаушыға шығарылған мәтіннің орналасуын басқаруға өте аз мүмкіндік береді. Мәтінді тек жоғарыдан төменге, солдан оңға қарай ғана енгізуге болады. Сонымен қатар, алдыңғы жолға оралу, басылған мәтінді өшіру, таңбалардың түсін өзгерту

және т.б. әрекеттерін орындау мүмкін емес.

Экранға шығаруды басқарудың қосымша мүмкіндіктерін CRT модулінің процедуралары мен функциялары қамтамасыз етеді. Бұл модульмен қолданушы программасының байланысын орнату үшін сипаттау бөлімінің алдына мына жол жазылуы қажет:

```
Uses CRT
```

CRT модулімен жұмыс істеу барысындағы қажетті ұғымдарды қарастырайық: экран режимдері, экрандағы позиция координаталары, мәтіндік терезе, фон түсі мен символ түсі.

Экран режимдері. Біріншіден, экранға шығару мәтіндік немесе графикалық түрде болуы мүмкін (графикалық дисплейлерде). Біз мұнда тек мәтінді шығару туралы сөз қозғаймыз.

Дисплейлер монохроматикалық (қара және ақ түсті) және түрлі-түсті болуы мүмкін. Монохроматикалық дисплейлер тек ақ-қара режимде жұмыс істей алады, ал түрлі-түсті дисплейлер ақ-қара және әр түрлі түсте жұмыс істей алады. Сонымен қатар, мәтіндік режимдер экранға сыятын символдық жолдар және бағандар санымен ерекшеленеді.

CRT модулінде әрбір режимнің символикалық атауы бар (сипатталған тұрақты) белгілі бір нөмірі болады. Экран режимін орнату үшін,

```
TextMode(<режим нөмірі>)
```

процедурасы қолданылады.

<Режим нөмірі> процедурасын шақыру кезінде сәйкес тұрақты санын да, атауын да көрсетуге болады. Мысалы, келесі екі операторлар эквивалентті:

```
TextMode(1);
TextMode(CO40);
```

Үнсіз келісім бойынша орнатылған экранның бастапқы режимі — CO80 (түрлі-түсті дисплейлерде).

Позиция координаталары. Мәтіндік экрандағы әрбір символдық позиция (X, Y) екі координаталарымен анықталған. X координатасы жолдың позициясы. Жолдағы сол жақ шеткі позиция координаты $X = 1$. Y координатасы символы бар жол нөмірі. Жолдар жоғарыдан төмен қарай нөмірленеді.

Мысалы, 80x25 режиміндегі символ экранның сол жақ жоғарғы бұрышында (1; 1) координатада болады, оң жақ төменгі бұрыштағы символ координаталары — (80; 25), ал экранның ортасындағы

символ координаталары — (40; 13).

Экрандағы меңзерді (X, Y) координатамен орнату үшін, CRT модулінде мынадай процедура бар:

```
GoToXY(X,Y).
```

Мұнда меңзер координаталары Byte типті өрнектермен беріледі.

Мысалға экранды тазалап, экранның ортасына «*» белгісін қоятын программа келтірейік:

```
Uses CRT;
Begin
     ClrScr;
     GoToXY(40,13);
     Write('*')
End.
```

Бұл жердегі қолданылып отырған ClrScr процедурасы экранды тазалау әрекетін орындайды.

Мәтіндік терезе. Символдарды шығару әрекеті орындалатын экрандағы тіктөртбұрышты аймақ *мәтіндік терезе* деп аталады. Бұл терезенің орналасуы тіктөртбұрыштың жоғарғы сол жақ және төменгі оң жақ бұрыштарының координаталары арқылы анықталады. Егер терезе бүкіл экранды алатын болса, онда 80x25 режимінде оның координаталары (1; 1) және (80; 25) болады. Бұл - бастапқы терезе, оның орнын және өлшемін мына процедурамен өзгертуге болады:

```
Window(X1, Y1, X2, Y2).
```

Мұндағы аргументтер — Byte типінің шамалары; (X1, *Y1)* — терезенің жоғарғы сол жақ бұрышының координаталары; (X2, Y2) — терезенің оң жақ төменгі бұрышының координаталары.

Терезе анықталғаннан кейін оның шегінен тыс символдарды шығару ешқандай нәтиже бермейді. Жаңа параметрлермен Window процедурасын қайта шақыру, алдыңғы белгілеулерді жояды.

Түстерді басқару. EGA, VGA және SVGA типтердегі түрлі-түсті дисплейлерде экранның мәтіндік режимінде 16 түс пайдалана аламыз.

CRT модулінде тұрақтылар жарияланған, олардың атаулары ағылшынша түстердің атауы болып табылады және тиісті мәндер осы түстердің реттік нөмірлері болып табылады.

Фонның түсін орнату процедурасы:

```
TextBackGround(Color).
```

Мұндағы аргумент — түс нөмірін тағайындайтын Byte типінің шамасы.

Символ түсін орнату процедурасы:

```
TextColor(Color).
```

Егер фон түсі мәтіндік терезе тазаланғанға дейін тағайындалған болса, онда ол тазаланғаннан кейін осы түспен «толтырылады». Егер фон экран тазаланғаннан кейін орнатылса, онда терезе қара түсті болады (үнсіз келісім бойынша) және тағайындалған фон түсі символдар шығатын орындарға орнатылады.

Кезекпен төрт терезе ашатын және әрбір терезе әр түрлі түспен боялатын программа мысалын келтірейік:

```
Uses CRT;
Begin
    Window(1, 1, 40, 12);
    TextBackGround(White); ClrScr;
    Window(41, 1, 80, 12);
    TextBackGround(Red); ClrScr;
    Window(1, 13, 40, 25);
    TextBackGround(LightRed); ClrScr;
    Window(41, 13, 80, 25);
    TextBackGround(Green); ClrScr;
End.
```

Келесі программа экранның ортасында ақ фондағы алғашқы он бес түстің нөмірлерін көрсетеді және әрбір нөмір өзі білдіретін сәйкес түске боялады:

```
Uses CRT;
Var I : Byte;
Begin
    TextBackGround(White);
    ClrScr;
    GoToXY(1, 12);
    For I := 0 To 14 Do
    Begin
        TextColor(I);
        Write(I:5)
    End
End.
```

CRT модуліндегі мәтіндік экранды басқарудың тағы бірнеше процедураларын сипаттайық:

- ClrEOL процедурасы — меңзер тұрған орыннан терезедегі жолдың бөлігін осы жолдың аяғына дейін өшіреді осыдан кейін бұл жағдайда меңзер орналасуы өзгермейді;
- DelLine процедурасы — тұтас бір жолды жояды нәтижесінде астыңғы жолдар бір жолға жоғары көтеріледі;
- InsLine процедурасы — меңзер тұрған орынның алдына бір бос жол қосады;
- LowVideo, NormVideo, HighVideo процедуралары — символдардың сәйкесінше төмендетілген, қалыпты және жоғарылатылған режимдерін орнатады.

CRT модуліндегі KeyPressed функциясы өте пайдалы, ол орындалған кезде пернетақтаға сауалнама жүргізеді және қандайда бір перненің басылып қалмауын анықтайды. Нәтижесінде, бұл функция кез-келген перне басылғанда True деген логикалық мән береді, және керісінше жағдайда False мәнін береді. Бұл функция көбінесе экрандағы нәтиже терезесінің кідірісін ұйымдастыру үшін қолданылады (программаны орындағаннан кейін Турбо Паскаль экранға редактор терезесін шақырады), ол үшін программаның соңының алдында келесі оператор жазылады:

```
Repeat Until KeyPressed;
```

Бұл кез-келген пернені басқанша, «орнында айналып тұратын» бос цикл. Осы уақытта экранда нәтиже терезесі көрсетіледі. Пернені басқаннан кейін, KeyPressed мәні True мәніне тең болады, цикл аяқталады, орындалу программа соңына шығады және редактор терезесі экранға оралады. Бұл әдіс кез-келген жерде программаның орындалуын кідіру үшін қолданылуы мүмкін.

Жоғарыда көрсетілген программада экранда төрт түрлі түсті терезе алу үшін біз мынадай толықтырулар енгіземіз: төрт түсті экранды орнатқаннан кейін программа орындалуы тоқтатылады және сурет сақталады, содан соң кез-келген пернені басқаннан кейін экран бастапқы күйіне оралады (80x25, қара фон, ақ символдар). Ол үшін программаның аяқталуына дейін келесі операторларды қосу керек:

```
Repeat Until KeyPressed;
Window(1, 1, 80, 25);
TextBackGround(Black);
ClrScr;
```

CRT модулінің басқа да процедуралары мен функцияларының сипатталуы Турбо Паскальдың арнайы әдебиетінде келтірілген.

ЖАТТЫҒУЛАР

1. Бастапқы берілгендер 1.0 және -2.0 болса, келесі программаның нәтижесі қандай болады?

```
Program Roots;
Var B, C, D : Real;
Begin
      Read(B, C);
      D := Sqrt(Sqr(B) - 4 * C);
      WriteLn('x1 =',       (-B * D)/2,
                  'x2 =',     (-B - D)/2)
End.
```

2. 3.4 және 7.9 мәндерін енгізгеннен кейін келесі программаның нәтижесі қандай болады?

```
Program Less;
Var X : Real; T : Boolean;
Begin
      Read(X);
      T := X < Round(X);
      Read(X);
      T := T And (X < Trunc(X));
      WriteLn(T)
End.
```

3. 36, -6 және 2 345 мәндерін енгізгеннен кейін келесі программаның нәтижесі қандай болады?

```
Program ABC;
Var A, B : Integer;
Begin
      Read(A, B, A);
      WriteLn(A, B : 2, A : 5)
End.
```

4. Келесі түрде қолданушымен диалог жүргізе алатын екі бүтін санның қосындысын есептейтін программа құрыңдар (көпнүктенің орны — енгізілетін және шығарылатын сандар):

Екі қосылғышты енгіз:
a = ...
b = ...
Есептеу нәтижесі:
a + b = ...

2.8. ЛОГИКАЛЫҚ ШАМАЛАР, АМАЛДАР, ӨРНЕКТЕР

Тараудың 1.4-бөлімінде логикалық шамалар, логикалық амалдар және логикалық өрнектер туралы айтылып өткен болатын. Логикалық шамалардың типтері екі мәнді ғана қабылдай алады: АҚИҚАТ және ЖАЛҒАН.

Паскальда логикалық мәндер False (F) және True (T) қызметші сөздерімен белгіленеді, ал логикалық типтің идентификаторы ол— Boolean болып табылады.

Boolean типті шамалардан басқа (тұрақтылар мен айнымалылар) False, True логикалық мәндерін қатынас амалдарының нәтижелері де қабылдай алады.

Қатынас амалдары екі операндты салыстыруды жүзеге асырады және олардың арасындағы сәйкес қатынастардың ақиқат немесе жалған екенін анықтайды.

Қатынас амалдарының құрылымы 2.11-суретте көрсетілген.

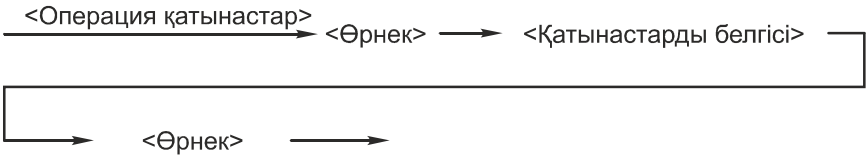
```
<қатынас таңбасы> ::= = (тең) |      <> (тең емес) |
> (үлкен) | < (кіші) | >= (үлкен немесе тең) |
<= (кіші немесе тең).
```

Қатынас амалдарына мысал келтірейік:

```
x < y; a + b >= c/d; abs(m - n) <= 1.
```

қатынас мәндерін есептеу мысалы:

Қатынас	Нәтиже
12 >= 12	True
56 > 10	True
11 <= 6	False



2.11-сурет. Қатынас амалдарының құрылымы

Логикалық амалдар бульдік типтегі операндтарға орындалады. Жалпы үш логикалық амал бар: Not — терістеу; And — логикалық көбейту (конъюнкция); Or — логикалық қосу (дизъюнкция). Осы үш міндетті амалдардан басқа, Турбо Паскальда тағы бір «НЕМЕСЕден БАСҚА» амалы бар, ол Xor қызметші сөзімен белгіленеді. Бұл екі орынды амал болып табылады, егер екі операнд та әр түрлі логикалық мән қабылдаса, нәтижесінде АҚИҚАТ мәнін береді.

Логикалық амалдар кему реті бойынша басымдылықтарына қарай орналасады. Операндтардың әр түрлі мәндері үшін логикалық амалдардың орындалу нәтижесі 2.5-кестеде көрсетілген.

Қатынас амалдарының басымдылықтары ең соңында орындалады, сондықтан егер логикалық амалдардың операндтары қатынастар болса, оларды жақшаға алған дұрыс. Мысалы, $1 < x < 50$ математикалық теңсіздігіне келесі логикалық өрнек сәйкес келеді:

```
(1 <= x) And (x <= 50)
```

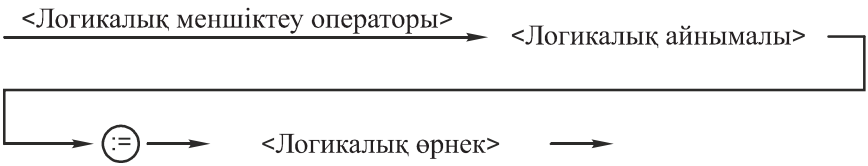
Логикалық өрнек — бұл программалау тілінде жазылған логикалық формула. Логикалық өрнек логикалық амалдар және жақшалармен байланысқан логикалық операндтардан тұрады. Логикалық өрнектерді есептеу нәтижесі бульдік шама болып табылады. (False немесе True). Логикалық операндтар рөлін тұрақтылар, айнымалылар, функиялар және қатынас амалдары атқаруы мүмкін. Бір жеке логикалық операнд логикалық өрнектің қарапайым формасы болып табылады.

Логикалық өрнекке мысал келтірейік, d, *b, c* — логикалық сандар; x, *y* — нақты типтер; *k* — бүтін айнымалы:

```
1) x <2 * y;      2) True;          3)d;
4) Odd(k);        5) Not Not d;     6) Not(x >y/2);
7) d And (x <> y) And b;
8) (c Or d) And (x =y) Or Not b.
```

2.5-кесте. Логикалық амалдардың орындалу нәтижесі

A	*B*	Not A	A And *B*	A Or *B*	A Xor *B*
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T



2.12-сурет. Логикалық меншіктеу операторының құрылымы

d = True, *b* = False, *c* = True, *x* = 3.0, *y* = 0.5, *k* = 5 болғанда есептеу нәтижелері келесідей болады:

1) False; 2) True; 3) True; 4) True;
5) True; 6) False; 7) False; 8) True.

Берілген мысалда Odd(x) логикалық функциясы қолданылған. Бұл егер *x* тақ сан болса - True мәнін қабылдайтын және ол жұп болса - False мәнін қабылдайтын *x* бүтін аргументінің функциясы.

Логикалық меншіктеу операторының құрылымы 2.12-суретте көрсетілген.

Логикалық меншіктеу операторына мысал келтірейік:

1) d := True;
2) b : = (x > y) **And** (k <> 0);
3) c := d **Or** b **And Not**(Odd(k) **And** d).

2.9. ӘР ТҮРЛІ МӘЛІМЕТТЕР ТИПТЕРІН БАЙЛАНЫСТЫРАТЫН ФУНКЦИЯЛАР

Әр түрлі мәліметтер типтерін байланыстыратын функциялар тізімі 2.6-кестеде көрсетілген.

Ord, Pred және Succ функциялары реттік мәліметтер типтеріне ғана, яғни нақты типтерден басқа барлық қарапайым мәліметтер типтеріне қолданылады.

Бүтін санға қолданылған Ord функциясы өз мәнін қайта шығарады. Мысалы:

```
Ord(-35) = -35; Ord(128) = 128
```

Егер аргумент бүтін болса, мысалы, *y* := Pred(x) операторы *y* := *x* – 1 өрнегіне эквивалентті, ал *y* := Succ(x) операторы *y* := *x* + 1 өрнегіне эквивалентті болады.

Символдық типтің аргументі үшін бұл функциялар сәйкесінше кодтау кестесіндегі алдыңғы және кейінгі символдарды береді.

2.6-кесте. Әр түрлі мәліметтер типтерін байланыстыратын стандартты функциялар

Оператор	Аргумент типі	Нәтиже типі	Әрекет
Ord (x)	кез-келген реттік	I	*x*-тің типінде оның реттік мәнін береді
Pred (x)	бірдей	*x* типімен бірдей	*x*-ке қатысты өзінің типінде алдыңғы қатынас мәнін береді
Succ (x)	»	*x* типімен бірдей	*x*-ке қатысты өзінің типінде кейінгі қатынас мәнін береді
Chr (x)	Byte	Char	*x*-тің реттік нөміріндегі символды береді.
Odd (*x*)	I	Boolean	егер *x* — тақ сан болса True, және егер *x* — жұп сан болса False мәнін береді

Латын алфавиті әрқашан код бойынша реттелген, яғни

```
Ord('a') < Ord('b') < ... < Ord('z'),
```

болса онда, мына түрде жазамыз:

```
Pred('b') = 'a'; Succ('b') = 'c'.
```

Сандық литерлерге де қатысы бар:

```
Pred('5') = '4'; Succ('5') = '6'.
```

Егер *x* — символдық шама болса, Chr(x) функциясы Ord(x) функциясына кері функция болып табылады. Мысалы, ASCII коды үшін келесі жазба заңды:

```
Ord('a') = 97; Chr(97) = 'a'.
```

Егер *x* — символдық шама болса, бұл әрекетті мына формуламен сипаттауға болады:

```
Chr(Ord(x)) = x.
```

Кейбір жағдайларда символдық шамаларды сандарға түрлендіру есептерін шешуге тура келеді. Мысалы, '5' литерінен бүтін 5 санын келесі әдіспен алуға болады:

```
N := Ord('5') - Ord('O').
```

Мұндағы *N* — бүтін айнымалы және '5' литерінің коды '0' кодынан бес бірлікке үлкен заңдылығы қолданылып тұр.

Мәліметтердің бульдік типі де реттік болып табылады. Оның екі мәндерінің орналасу реті келесідей: False, True. Осыған байланысты келесі қатынастардың орындалуы заңды:

```
Ord(False) = 0; Succ(False) = True;
Ord(True) = 1; Pred(True) = False.
```

Қызықты мысал келтірейік. *x, y,* z — нақты айнымалылар болсын. Келесі оператор қандай есепті шешетінін анықтау керек:

```
z := x * Ord(x >= y) + y * Ord(y > x).
```

Жауап: z = max(x, y), яғни берілген есепті if.. then.. else шартты операторды қолданылуынсыз шешуге болады деген сөз.

ЖАТТЫҒУЛАР

1. Келесі логикалық өрнектердің мәндерін табыңдар:

а) `K mod 7 = K div 5 - 1`, егер `K = 15;`

ә) `Odd(Trunc( 10 * P))`, егер `P = 0.182;`

б) `Not Odd(n)`, егер `n = 0;`

в) `t And (P mod 3 = 0)`, егер `t = True, P = 10101;`

г) `(x * y <> 0) And (y > x)`, егер `x = 2, y = 1;`

ғ) `a Or Not b`, егер `a = False, b = True.`

2. Келесі меншіктеу операторлары орындалғаннан кейін *d* логикалық айнымалысы қандай мәнге тең болатынын анықтау керек, егер *a* = True және *x* = 1 болса.

а) `d := x < 2;`

ә) `d := Not a Or Odd(x);`

б) `d := Ord(a) <> x.`

3. Егер келесі тұжырымдар ақиқат боса, *t* логикалық айнымалысының мәні True болатындай, және керісінше жағдайда False болатындай меншіктеу операторының орындалу нәтижесін жазыңдар:

а) x, y, z сандарынан екеуі ғана өзара тең;

ә) x — оң сан;

б) x, y, z әрбір саны оң сан;

в) x, y, z сандарының біреуі ғана оң сан;

г) p – q-ға қалдықсыз бөлінеді;

ғ) 5 саны *к* бүтін санының ондық жазбасына кіреді.

2.10. ТАРМАҚТАЛУ АЛГОРИТМДЕРІН ПРОГРАММАЛАУ

Паскальда тармақталудың алгоритмдік құрылымы шартты оператордың көмегімен программаланады:

```
If <шарт> Then <1-Оператор> Else <2-Оператор>;
```

Сонымен қатар шартты оператордың толық емес формасы да қолданылуы мүмкін:

```
If <шарт> Then <Оператор>;
```

Шартты оператордың синтаксистік диаграмма формасындағы қатаң сипатталуы 2.13-суретте көрсетілген.

Шартты оператор шарты логикалық өрнек болып табылады, және ол бірінші болып есептеледі. Шарттың толық формасы үшін егер оның мәні True болса, онда <1-Оператор> (Then-нен кейін) орындалады, ал егер мәні False болса, онда <2-Оператор> (Else-тен кейін) орындалады және толық емес форма үшін (Else-сіз) бірден кезекте тұрған оператор орындалады.

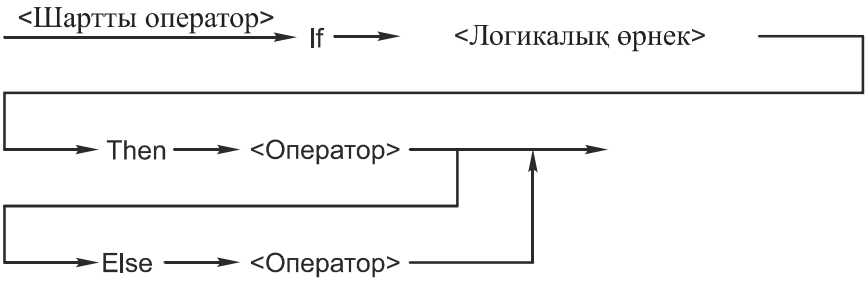
2.1 - Мысал. Қабырғаларының ұзындығы *a, b, c* болатын үшбұрыштың ауданын есептейтін программа құру керек.

Есепті шешу үшін Герон формуласын пайдаланамыз:

$$\sqrt{p(p-a)(p-b)(p-c)}$$

мұндағы $p = (a + b + c)/2$ — үшбұрыш периметрінің жартысы.

Бастапқы берілгендер үшбұрыштың қабырғаларының негізгі қатынастарын қанағаттандыру керек: әрбір қабырғаның ұзындығы қалған екі қабырғалардың ұзындықтарының қосындысынан кем болуы қажет.



2.13-сурет. Шартты операторлың синтаксистік диаграммасы

Бір шартты оператордың ішіне күрделі логикалық өрнектерді жазу мүмкіндігін пайдаланып, дұрыс емес бастапқы берілгендердің барлық нұсқаларын «сүзіп аламыз»:

```
Program Geron;
Var A, B, C, P, S : Real;
Begin
  WriteLn(Үшбұрыш қабырғаларының ұзындығын енгіз:');
  Write('a = '); ReadLn(A);
  Write('b = '); ReadLn(B);
  Write('c = '); ReadLn(C);
  If (A > 0) And (B > 0) And (C > 0) And (A + B > C)
       And (B + C > A) And (A + C > B)
  Then Begin
          P := (A + B + C)/2;
          S := sqrt(P * (P- A) * (P - B) * (P
          - C)); WriteLn('Аудан = ', S)
       End
  Else WriteLn('Бастапқы берілгендер дұрыс емес')
End.
```

2.2-Мысал. Квадрат теңдеуді есептеудің программасын құру керек.

Бұл есепті шешу алгоритмі мен алгоритмдік тілде сипатталуы 1.3 бөлімде (1.8-суретті қараңыз) толық қарастырылған болатын. Бұл алгоритм қабаттасқан тармақталу құрылымын қамтиды. Паскальдағы программасы келесідей болады:

```
Program Roots;
Var A, B, C, D, X1, X2 : Real;
Begin
  WriteLn(квадрат теңдеудің коэффициенттерін енгіз:');
  Write('a = '); ReadLn(A);
  Write('b = '); ReadLn(B);
  Write('c = '); ReadLn(C);
  If (A = 0)
  Then If (B = 0)
         Then If (C = 0)
              Then WriteLn('кез-келген X — шешім')
              Else WriteLn('Шешімі жоқ')
         Else
         Begin
              X := -c/b;
              WriteLn('X = ', X)
         End
  Else
```

```
  Begin d := b * b - 4 * a * c;
        If (d < 0)
        Then WriteLn('Нақты түбірлер жоқ')
        Else
        Begin
             X1 := (-b + sqrt(d))/2/a;
             X2 := (-b - sqrt(d))/2/a;
             WriteLn('X1 = ', X1);
             WriteLn('X2 = ', X2)
        End
  End
End.
```

2.3-мысал. Бес баллдық бағалауды олардың атауына сәйкес ауыстырудың программасын құру: 5 – үздік, 4 – жақсы, 3 – қанағаттанарлық, 2 – қанағаттанарлықсыз.

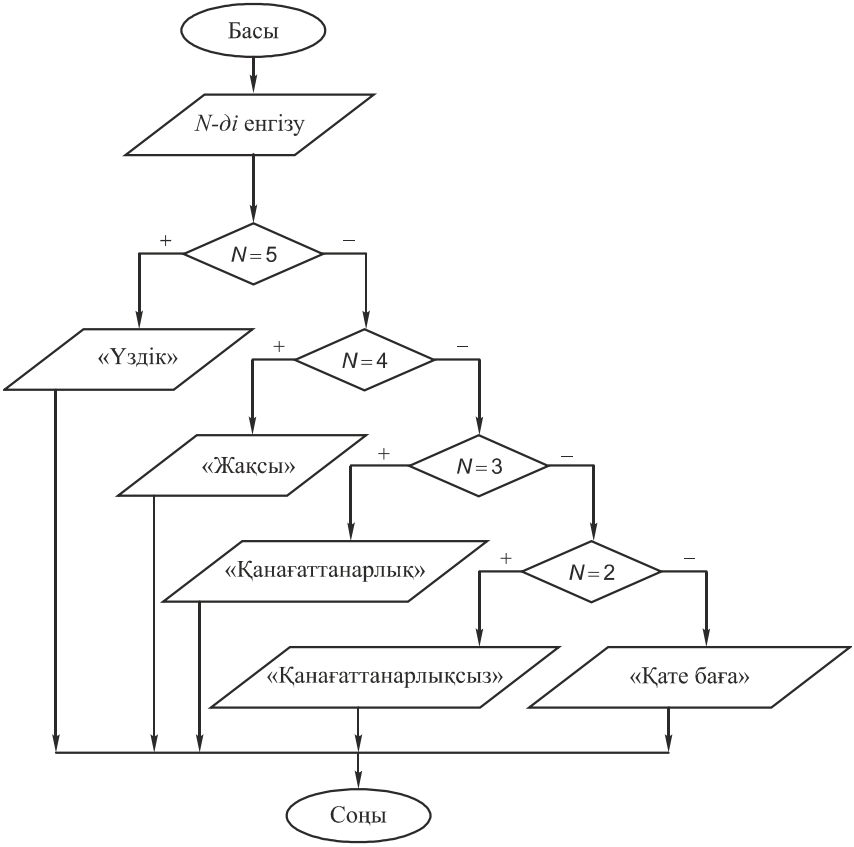
Есепті шешу алгоритмінің блок-сызбасы 2.14-суретте көрсетілген.

Бұл алгоритм қабаттасқан тармақталу құрылымын қамтиды және сәйкес Паскальдағы программасы мына түрде болады:

```
Program Marks-2;
Var N : Integer;
Begin
  WriteLn('Бағаны енгіз:'); ReadLn(N);
  If (N = 5)
  Then WriteLn('Үздік')
  Else If (N = 4)
       Then WriteLn('Жақсы')
       Else If (N = 3)
            Then WriteLn('Қанағаттанарлық')
            Else If (N = 2)
                 Then WriteLn('Қанағаттанарлықсыз');
                 Else WriteLn('Қате баға')
End.
```

Қабаттасқан тармақтар құрылымын бір таңдау операторының көмегімен программалауға болады:

```
Program Marks;
Var N: Integer;
Begin
    WriteLn('Бағаны енгіз:');
    ReadLn(N);
    Case N Of
          5: WriteLn('Үздік');
          4: WriteLn('Жақсы');
```

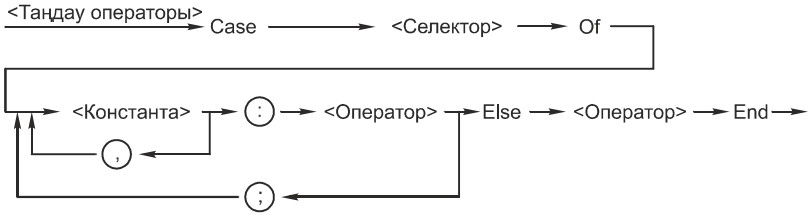


2.14-сурет. Бес баллдық бағалауды олардың атауына сәйкес ауыстырудың блок-сызбасы

```
            3: WriteLn('Қанағаттанарлық');
            2: WriteLn('Қанағаттанарлықсыз')
    Else WriteLn('Ондай баға жоқ')
End;
```

Таңдау операторы форматының синтаксистік диаграмма арқылы сипатталуы 2.15-суретте көрсетілген, мұндағы <Селектор> — бұл кез-келген реттік типтің өрнегі, <Константа> — селектор типіне сәйкес тұрақты шама, ал <Оператор> — кез-келген жай және құрама оператор.

Таңдау операторының орындалуы келесі түрде жүзеге асырылады: селектор — өрнегі есептеледі, одан кейін тұрақтылар



2.15-сурет. Таңдау операторының синтаксистік диаграммасы

тізімінде селектордың алынған нәтижесімен сәйкес келетін мән орналасады, ары қарай берілген тұрақтымен белгіленген оператор орындалады.

Егер мұндай тұрақты табылмаса, Else-тен кейінгі операторға көшу орындалады.

Тұрақтылар тізімінің таңдау операторында қолдану мысалын қарастырайық. Келесі программа егер студент «5», «4», немесе «3» деген бағалар алса, емтиханнан өтті, ал егер «2» деген баға алса, емтиханнан өтпегендігі туралы хабарлама шығарады.

```
Case N Of
      3, 4, 5 : WriteLn('Емтиханнан өтті')
            2 : WriteLn('Емтиханнан өткен
            жоқ')
      Else WriteLn('Ондай баға жоқ');
```

Шартты оператор сияқты таңдау операторы да толық емес формада, яғни Else тармағынсыз қолданылуы мүмкін.

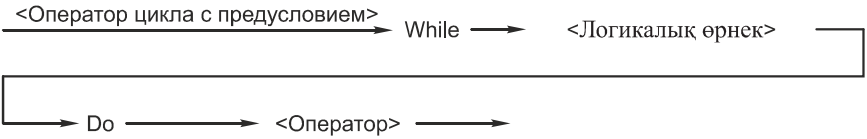
2.11. ЦИКЛДІК АЛГОРИТМДЕРДІ ПРОГРАММАЛАУ

Циклдік құрылымның үш түрі бар: шарты алдын ала берілген цикл (циклдің алғышарты), шарты соңынан берілген цикл (циклдің ілесу шарты) және параметрлі цикл.

Шарты алдын ала берілген цикл. «Әзірше» цикл операторы немесе шарты алдын ала берілген циклдің синтаксистік диаграммасын қарастырайық (2.16-сур.). Бұл жерде алдымен <Логикалық өрнек> есептеледі. Оның мәні True болғанша <Оператор> — цикл денесі орындалады және <Оператор> жай және құрама бола алады.

Мысал ретінде гармоникалық қатардың ε қосындысын есептейтін Паскальдағы программа фрагментін көрсетейік.

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{1}+\cdots$$



2.16-сурет. Алғышартты циклдің синтаксистік диаграммасы.

Кезекті қосылғыш ε-ден кіші болған жағдайда немесе *I* бүтін айнымалысы MaxInt мәніне жеткенде қосындылау тоқтатылады:

```
S := 0;
I := 1;
While (1/I >= Eps) And (I < MaxInt) Do
Begin
  S := S + 1/I;
  I := I + 1
End;
```

Шарты соңынан берілген цикл. «Дейін» цикл операторының немесе шарты соңынан берілген циклдің синтаксистік диаграммасы 2.17-суретте көрсетілген.

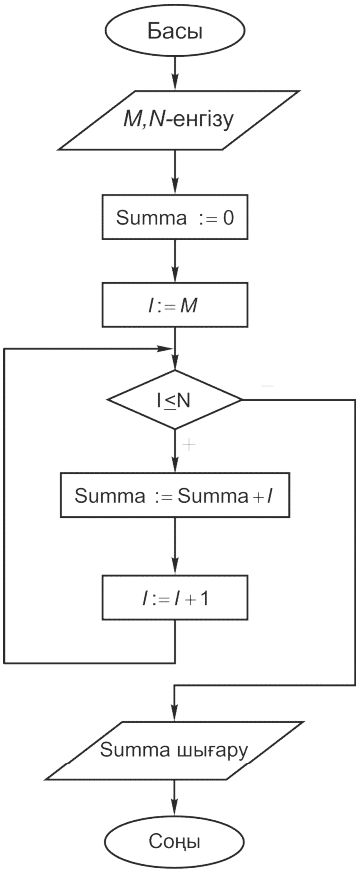
Берілген циклдің орындалуы <Логикалық өрнек> True мәніне тең болғанша жалғаса береді.

Алдыңғы есепті циклдің ілесу шартын қолданып шешетін болсақ, ол мына түрде жазылады:

```
S := 0;
I := 1;
Repeat
  S := S + 1/I;
  I := I + 1
Until (1/I < Eps) Or (I >= MaxInt);
```

Параметрлі цикл. *M*-нан *N-ге* дейінгі бүтін сандарды қосу мысалын қарастырып көрейік:

2.17. Шарты соңынан берілген циклдің синтаксистік диаграммасы



2.18-сурет. Бүтін сандарды қосу алгоритмінің блок-сызбасы

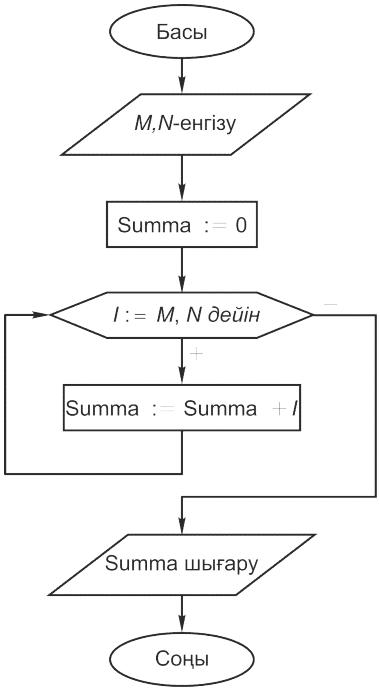
$$Summa = \begin{cases} \sum_{I=M}^{N} I, & \text{егер } M \le N \\ 0, & \text{егер } M > N \end{cases}$$

Бұл есепті шешу алгоритмінің блок-сызбасы 2.18-суретте көрсетілген. Енді «Әзірше» циклін қолданып келесі программаны құрамыз:

```
Program Adding;
Var I, M, N, Summa : Integer;
```

```
Begin
   Write('M =');
   ReadLn(M);
   Write('N =');
   ReadLn(N);
   Summa := 0;
   I := M;
   While I <= N Do Begin
        Summa := Summa + I;
        I := Succ(I)
   End;
   WriteLn('Қосынды тең: ',
End.
```

Енді циклдік құрылымның жаңа түрі – *параметрлі циклмен* жұмыс жасап көрейік. Осы құрылымды қолданып қосындылау алгоритмінің блок-сызбасы 2.19-суретте көрсетілген.



2.19-сурет. Параметрлі циклді қосындылау алгоритмінің блок-сызбасы

Ал Паскальдағы жазбасы мынадай:

```
Program Summering 2;
Var I, M, N, Summa: Integer;
Begin Write('M = ');
      ReadLn(M);
      Write('N =') ;
      ReadLn(N);
      Summa := 0;
      For I := M To N Do
          Summa := Summa + I;
      WriteLn('Қосынды тең: ', Summa)
End.
```

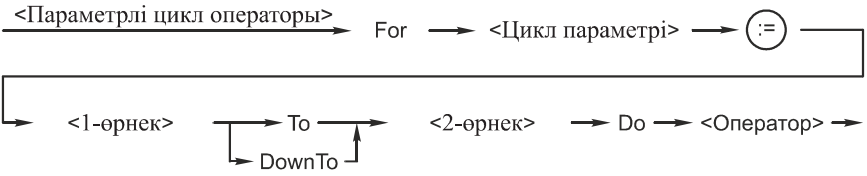
Мұнда бүтін *I* айнымалысы *M* мен N аралығындағы барлық мәндер тізбегін қабылдайды. Әрбір *I* мәнінде цикл денесі орындалады. *I* = *N* болғандағы циклдің соңғы әрекетінде алгоритмді жалғастыру үшін циклден шығу орындалады. Егер *M* < *N* болса, цикл кем дегенде бір рет орындалады, ал *M* > *N* болса, онда цикл орындалмайды.

Келтірілген программада параметрлі цикл қолданылған және оның синтаксистік диаграммасы 2.20-суретте көрсетілген.

<цикл параметрі> ::= <реттік типтегі қарапайым айнымалының атауы>

For операторының орындалуы (бірінші нұсқада To) келесі сызба бойынша жүзеге асады:

1. <1-өрнек> және <2-өрнек> мәндері есептеледі және олар циклге кіргенде бір рет есептеледі.
2. Цикл параметріне <1-өрнек> мәні меншіктеледі.



2.20. Параметрлі циклдің синтаксистік диаграммасы.

3. Цикл параметрінің мәні <2-өрнек> мәнімен салыстырылады. Егер цикл параметрі осы мәннен кем немесе тең болса, онда циклдің денесі орындалады, әйтпесе цикл аяқталады.

4. Цикл параметрінің мәні оның типіндегі келесі мәнге ауысады (бүтін сандар үшін ол 1-ге артады) және осы сызбаның 3-тармағына оралу орын алады.

For циклінің операторы **While** циклін пайдаланған кезде әртүрлі операторларды орындау әрекеттерін біріктіреді: параметрге бастапқы мәнді меншіктеу, оны соңғы мәнмен салыстыру және мәнді келесіге өзгерту.

Бүтін сандар қосындысының нәтижесі қосындылау тәртібіне байланысты емес екені белгілі. Мысалы, қарастырылып отырған есепте, сандарды кері тәртіпте, яғни N-нен M-ге (N>M) қосуға болады. Ол үшін **For** циклдік операторының екінші нұсқасын пайдалануға болады:

```
Summa := 0;
For I := N DownTo M Do
Summa := Summa + I;
```

DownTo «төменге дейін» деп аударылады. Берілген жағдайда цикл параметрі кему ретімен өзгереді, яғни циклдің әрбір қайталануында параметр өзінің мәнін алдыңғы мәнге өзгертеді. (i := pred(i)). $N < M$ болған жағдайда цикл орындалмайды.

For параметрімен жұмыс жасау барысында келесі ережелерді есте сақтаған жөн:

- цикл параметрі real типті бола алмайды;
- цикл денесінде айнымалыны — цикл параметрін өзгертуге болмайды;
- циклден шығу уақытында айнымалы мәні— цикл параметрі анықталмаған болып қалады.

Келесі мысалда, **For** циклінің параметрі ретінде символдық айнымалы мәнін пайдаланамыз. Экранда латын алфавиті әріптерінің ондық кодтарын алу қажет делік. Өздерің білетіндей, кодтау кестесіндегі латын әріптері алфавит бойынша сұрыпталған. Тиісті программаның фрагменті келесідей жазылуы мүмкін:

```
For C := 'a' To 'z' Do
Write (C,' - ', Ord(C));
```

Мұнда C айнымалысының типі char.

Енді латын алфавитінің әріптерінің кері реттегі кодталуын өз беттеріңше программалаңдар ('z'-тен 'a'-ға дейін).

ЖАТТЫҒУЛАР

1. Паскальда биквадрат теңдеуді толық шешудің программасын құрыңдар.
2. Таңдау операторын пайдаланып ай нөмірін енгізгенде сәйкес жыл мезгілін (қыс, көктем, жаз, күз) шығаратын программа құрыңдар.
3. While, Repeat және For цикл операторларын қолдана отырып, N! есептеу программасының үш нұсқасын құрыңдар.
4. Кіші немесе үлкен *z* латын әріптері пайда болғанға дейін символдар тізбегі енгізілетін программа құрыңдар. *W* әрібінің қанша рет кездесетінін есептеңдер.
5. (ln x; e^x), мұндағы $x > 1$, интервалында орналасқан барлық бүтін сандардың квадраттарының қосындысын есептейтін программа құрыңдар.
6. R (R> 0) радиусты және центрі координаталар басындағы шеңберде орналасқан бүтінсанды координаталары бар нүктелер санын есептеңдер.
7. [0; 1] аралықтағы және қадамы 0.1 болатын sin*x* және cos*x* функциялары мәндерінің кестесін келесі түрде шығарыңдар:

x	sin x	cos x
0.0000	0.0000	1.0000
0.1000	0.0998	0.9950
...	...	...
1.0000	0.8415	0.5403

8. Ондық жүйеде барлық үш таңбалы сандарды өсу ретімен шығарыңдар, және онда бірдей сандар болмау керек.
9. $n > 2$ бүтін саны берілген. [2; n] интервалындағы барлық жай сандарды шығаратын программа құрыңдар

2.12. ІШКІ ПРОГРАММАЛАР

1.5-бөлімде көмекші алгоритмдер туралы айтып өткенбіз. Программалау тілдерінде көмекші алгоритмдер *ішкі программалар* деп аталады. Паскальда ішпрограмманың екі түрі бар: *процедуралар* және *функциялар*.

Келесі мысалды қарастырайық: екі натурал сандар *a* және *b* берілген. Ең үлкен ортақ бөлгішті анықтау керек: *a*+ *b,* |a-b|, *a* • *b.* Біз оны ЕҮОБ (a + b, |a- b|, *a* • b) түрінде жазамыз.

Бұл есепті шешу идеясы келесі математикалық фактта болып табылады: егер *x*, *y,* z - үш натурал сандар болса, онда ЕҮОБ(x, *y, z)* = ЕҮОБ(ЕҮОБ(*x*, y), z). Басқаша айтқанда, алдымен екі шаманың ЕҮОБ табуымыз керек, содан кейін алынған нәтиже мен үшінші санның ЕҮОБ табамыз (оны дәлелдеп көріңдер).

Әлбетте, бұл есепті шешудің көмекші алгоритмі екі бүтін сандардың ең үлкен ортақ бөлгішін алу болып табылады. Демек, бұл есеп белгілі Евклид алгоритмінің (1.3-бөлімді қара) көмегімен шығарылады. Оны алгоритмдік тілдегі процедура формасында жазамыз:

```
процедура Евклид (бүт M, N, K);
басы
  әзірше M <> N
  цб
        егер M > N
        онда M : = M - N
        әйтпесе N := N - M
        тс
  цс;
  K := M
соңы.
```

Мұндағы, *M* және *N* - процедураның формалды параметрлері, яғни параметр-аргументтер, *K* – нәтиже-параметрі.

Бастапқы есепті шешудің негізгі алгоритмі келесідей:

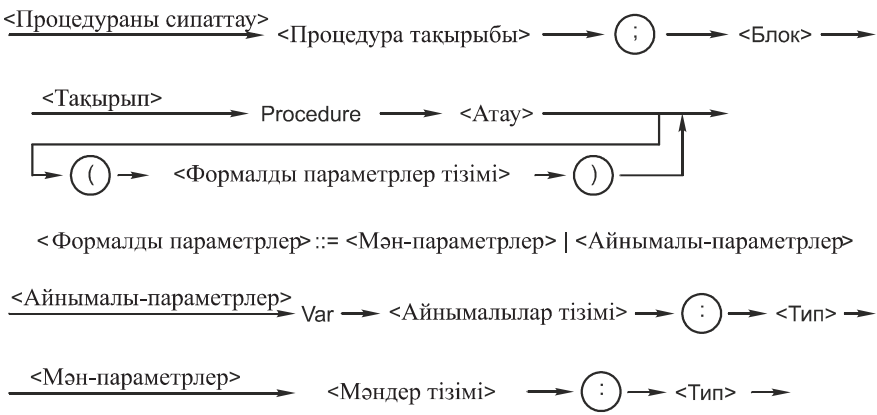
```
алг есеп
бүт a, b, c
басы енгізу (a, b)
     Евклид(a + b, |a - b|, c)
     Евклид(c, a * b, c)
     шығару(c)
соңы
```

Процедуралар. АТ-ден қарағанда процедуралар шақырушы программаға қатысты ішпрограммада болады және Паскальдағы процедуралар ішпрограммаларды сипаттау бөлімінде сипатталады. Турбо Паскальда бастапқы есепті шешу программасы келесі формаға ие болады:

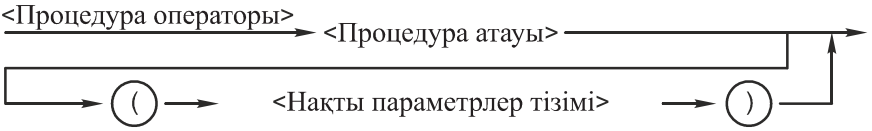
```
Program NOD1;
Var A, B, C : Integer;
Procedure Evklid(M, N : Integer; Var K : Integer);
Begin
   While M <> N Do If M > N Then
   M := M - N Else N := N - M;
   K := M End;
Begin
   Write('A =');
   ReadLn(A);
   Write('B =');
   ReadLn(B);
   Evklid(A + B, Abs(A - B), C);
   Evklid(C, A * B, C);
   WriteLn('ЕҮОБ =', C)
End.
```

Бұл жағдайда негізгі программа мен процедура арасындағы аргументтер мен нәтижелер алмасуы параметрлер (формалды және нақты) арқылы жүзеге асырылады. Алмасудың басқа да механизмі бар – олар төменде талқыланатын ауқымды айнымалылар.

Процедураның сипаттамасының синтаксистік диаграммасы 2.21-суретте көрсетілген.



2.21-сурет. Процедураны сипаттаудың синтаксистік диаграммасы



2.22-сурет. Процедураны шақыру операторының құрылымы.

Бұл процедураның параметрлерінің бар немесе болмауы көрсетілген диаграммадан байқауға болады. Көбінесе, аргументтер параметр-мәндер ретінде көрсетіледі (бірақ олар да параметр-айнымалылар болуы мүмкін), нәтижелерді беру үшін айнымалы-параметрлер пайдаланылады.

Процедура нәтиже ретінде шақырып отырған программаға көптеген мәндер беруі мүмкін (жеке жағдайда - біреу) немесе ол бірде-бір мәнді бере алмауы мүмкін. Енді процедураны шақыру ережелерін қарастырайық. Процедураны шақыру - процедура операторы формасында жүзеге асырылады (2.2.2-суретті қара).

Егер формалды параметрлері бар процедура сипатталған болса, оны шақыру процедураның нақты параметрлері бар операторларымен орындалады. Формалды және нақты параметрлер арасындағы сәйкестік ережелері келесідей: *сан бойынша, ретімен* және *тип* бойынша сәйкестендіру.

Формалды және нақты параметрлердің өзара әрекеттесуінің бірінші нұсқасы *мән бойынша берілу* деп аталады: нақты параметрдің (өрнектің) мәні есептеледі және бұл мән тиісті формальды параметрге меншіктеледі. Формалды және нақты параметрлердің өзара әрекеттесуінің екінші нұсқасы *атау бойынша берілу* деп аталады: процедура орындалған кезде, формалды айнымалы атауы сәйкес нақты айнымалы атауымен алмастырылады. (компиляцияланған программада айнымалы атауына жадының ұяшық мекен-жайы сәйкес келеді).

Қарастырылған мысалда M және N формалды параметрлері параметр-мәндер болып табылады. Мұнда процедура аргументтерін бірінші рет шақыру барысында А + Б және *Abs* (А - В) өрнегі сәйкес келеді, ал екінші рет - С және А • Б мәндері сәйкес келеді. *K* параметрі процедура жұмысының нәтижесі алынатын айнымалы-параметр. Процедураны екі рет шақыру барысында да сәйкес нақты параметр С айнымалысы болып табылады, оның көмегімен негізгі программа нәтижені алады.

Енді бастапқы берілген есепті параметрсіз процедураны қолданып шешудің программа мысалын қарастырайық:

```
Program NOD2;
Var A, B, K, M, N : Integer;
Procedure Evklid;
Begin
   While M <> N Do
   If M > N
   Then M := M - N
   Else N := N - M;
   K := M
End;
Begin
   Write('A =');
   ReadLn(A);
   Write('B =');
   ReadLn(B);
   M := A + B;
   N := Abs(A - B);
   Evklid;
   M := K;
   N := A * B;
   Evklid;
   WriteLn('ЕҮОБ тең', K)
End.
```

Бұл мысалды талдау үшін *сипаттау әрекеттерінің аймағы* ұғымын қарастыруымыз керек.

Кез-келген программа объектісінің (айнымалы, тип, тұрақты және т.б.) сипаттау әрекетінің аймағы - бұл сипаттама орналасқан блок болып табылады. Егер бұл блок басқа (ішкі программаға) блокта орналасса, онда оның сипаттамалары *жергілікті* болады және тек ішкі блокта әрекет ете алады. Сыртқы блокта орналасқан сипаттамалар, ішкі блокқа қатысты *ауқымды* болып табылады. Егер ауқымды сипатталған объект ішкі блокта пайдаланылса, оған сыртқы (ауқымды) сипаттама таратылады.

NOD1 программасында M, N, K айнымалы мәндері процедура ішінде - жергілікті, ал A, B, C айнымалы мәндері - ауқымды айнымалылар болып табылады. Дегенмен, A, B, C айнымалы мәндері процедура ішінде пайдаланылмайды. Сыртқы блок пен процедура арасындағы байланыс параметрлер бойынша жүзеге асырылады.

NOD2 программасында барлық айнымалылар ауқымды болып табылады. Евклид процедурасында жергілікті айнымалылар жоқ (параметрлер де жоқ), сондықтан пайдаланылған M және N

айнымалылары өздерінің мәндерін программаның негізгі блогындағы меншіктеу операторы арқылы алады. Нәтиже экранға шығарылатын ауқымды *K* айнымалысында алынады.

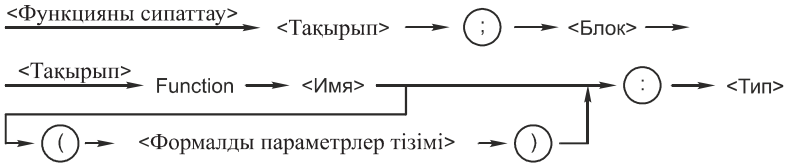
Параметрлер арқылы мәндерді беру механизмін қолдану, процедураны негізгі программадан тәуелсіз әмбебап етеді. Дегенмен, кейбір жағдайларда мәндерді ауқымды айнымалы арқылы беру, әсіресе үлкен көлемдегі ақпаратпен жұмыс істейтін процедураларды пайдалануда ыңғайлы болып табылады. Бұл жағдайда ауқымды өзара әрекеттесу компьютердің жадысын үнемдейді.

Функциялар. Енді ішпрограмма функцияның не екенін анықтайық. Ішкі функцияның нәтижесі скаляр (қарапайым) шама болса, әдетте функция қолданылады. Нәтиже типін функция типі деп атайды. Турбо Паскальда жолдық типтегі функцияларға рұқсат етіледі. Функция сипаттамасының синтаксистік диаграммасы 2.23-суретте көрсетілген.

Процедурадағы сияқты функцияда формалды параметрлер тізімінде оның аргументі болып табылатын айнымалы-параметрлер мен параметр-мәндер болуы мүмкін. Бұл жағдайда, егер аргументтер ауқымды түрде берілсе, тізімдегі параметрлер болмауы да мүмкін.

Функцияны пайдалану арқылы туындаған бастапқы есепті шешу программасы келесі түрде болады:

```
Program NOD3;
Var A, B, Rez: Integer;
Function Evklid(M, N : Integer) : Integer;
Begin
  While M <> N Do
  If M > N Then M
  := M - N Else N
  := N - M;
  Evklid : = M
End;
Begin
    Write('A = ');
```



2.23-сурет. Функцияны сипаттаудың синтаксистік диаграммасы

```
    ReadLn(A);
    Write('B = ');
    ReadLn(B);
    Rez := Evklid(Evklid(A + B, Abs(A - B)), A * B);
    WriteLn('NOD равен ', Rez)
End.
```

Бұл мысалда, функцияның процедурадан айырмашылығы - функция денесінде *нәтиже айнымалыға функцияның атауымен меншіктеледі*.

Функцияны шақыру - бұл өрнектегі операнд болып табылады және келесі формада жазылады:

```
<функция атауы> (<нақты параметрлер тізімі>)
```

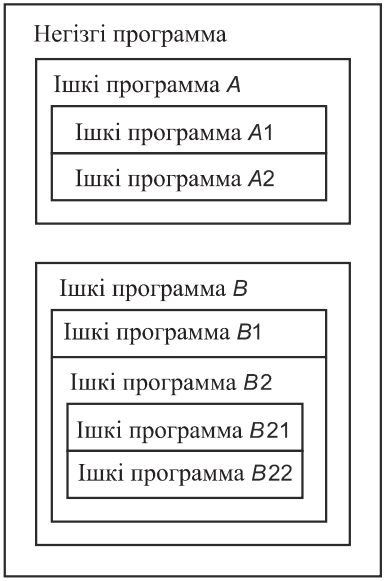
Функцияның формалды және нақты параметрлері арасындағы сәйкестік ережелері процедураға ұқсас. Қарастырылған программаларды салыстыра отырып, NOD3 программасының басқаларына қарағанда белгілі бір артықшылықтары бар екендігі туралы қорытынды жасауға болады. Функция бір меншіктеу операторын орындау арқылы нәтижені алуға мүмкіндік береді. Сондай-ақ, мұнда көрсетілгендей, функцияны шақыру барысында нақты аргумент функцияның өзі болуы мүмкін.

Стандартты Паскаль ережелеріне сәйкес, ішпрограммадан шақырып отырған программаға оралу ішпрограмманың орындалуы соңына жеткенде (соңы - end) орындалады. Алайда, Турбо Паскальда кез-келген жерде ішпрограммадан шығуға мүмкіндік беретін құрал бар. Бұл Exit оператор-процедурасы. Мысалы, берілген нақты сандардың ең үлкенін анықтау функциясы келесідей сипатталуы мүмкін:

```
Function Max(X, Y : Real): Real;
Begin
  Max := X;
  If X > Y Then Exit Else Max := Y
End;
```

Сипаттау әрекеттерінің аймағы туралы тағы да сөз қозғайық. Паскальда келесі ереже қатаң түрде орындалады: кез-келген программа объектісі (тұрақты, айнымалы, тип, т.б.) программада пайдаланбас бұрын ол сипатталуы керек. Басқаша айтқанда, объектінің сипаттамасы программаның басқа фрагменттерінде бірінші рет пайда болмауы керек. Бұл ереженің ішкі программаларға да қатысы бар.

Шартты программада ішпрограммаларды сипаттаудың өзара орналасу құрылымы 2.24-суретте көрсетілген.



2.24-сурет. Программа құрылымының мысалы

Осы сызбаны пайдаланып сипаттау әрекеттерінің аймағы туралы сұрақты талқылайық.

Кез-келген ішпрограмма оның сипатталуының әрекеттер аймағы шегінде ғана қолданылуы мүмкін. Мысалы, А және В ішпрограммаларының әрекеттер аймағы - негізгі программа болып табылады, сондықтан негізгі программаға А және В ішпрограммасын шақыра аламыз. Өз кезегінде, В ішпрограммасына А ішпрограммасын шақыра аламыз, алайда А ішпрограммасына В ішпрограммасын шақыра алмаймыз, өйткені А сипаттамасы В сипаттамасының алдында орналасқан. А1 және А2 ішпрограммалары А ішпрограммасында локализацияланған және тек сол ішпрограммада ғана пайдаланылуы мүмкін, және А2-ге А1-ді шақыруға болады, ал А1-ге А2-ні шақыру мүмкін емес.

В1 ішпрограммасына А ішпрограммасын шақыра аламыз, өйткені оның сипаттамасы В1-ге қатысты ауқымды, бірақ А1-ді шақыра алмаймыз, өйткені А1-дің сипаттау әрекеттерінің аймағы В ішпрограммасы блоктарына таралмайды.

*В*22 ішпрограммасына В21, В1 және А ішпрограммаларын шақыра аламыз. (Себебін өз беттеріңше анықтаңдар).

Егер сыртқы (ауқымды) және ішкі (жергілікті) блокта бір атау сипатталса, онда соңғы сипаттау (жергілікті) ішкі блоктың шегінде бірінші блокты жауып қалады. Екі программа қарастырайық:

```
Program Example1;
Var X : Integer;
Procedure P;
Var X: Integer;
Begin WriteLn('X = ', X)
End;
Begin X := 1;
  P
End.
```

```
Program   Example2;
Var X : Integer;
Procedure P;
Begin
    WriteLn('X = ', X)
End;
Begin X : = 1;
   P
End.
```

Жұмыстың нәтижесінде бірінші программа X = ... нәтижесін шығарады, онда көп нүктенің орнында *x*-тің белгісіз шамасына сәйкес келетін белгілі бір еркін мән болады.

Екінші программа X = 1 нәтижесін шығарады.

Бірінші жарты программада *X* айнымалысы ауқымды да, жергілікті де болып сипатталып тұр, бірақ процедура ешқандай мән меншіктелмеген жергілікті айнымалы мәнін шығарады. Мұнда *X* идентификаторы екі түрлі жады ұяшығына сәйкес келетін екі түрлі шамаларды белгілеген.

Екінші программада X айнымалысы барлық программа үшін біреу болып табылады және ауқымды сипатталған, сондықтан оған негізгі программада 1 мәні меншіктелген және ол ішпрограммаға беріледі.

ЖАТТЫҒУЛАР

1. Ішкі және сыртқы радиустардың мәндері бойынша сақинаның ауданын есептеуге арналған программаның екі нұсқасын құрыңдар: процедурасы және функциясы бар ішпрограмманы қолданып шеңбердің ауданын есептеу.

2. Үшбұрыштың төбелерінің координатасынан екі нүкте арасындағы кесіндіні есептеудің ішпрограммасын пайдалана отырып, оның периметрін есептеңдер.

3. Үш бүтін сан берілген. Ішкі программаны пайдаланып қайсысының сандарының қосындысы үлкен болатынын анықтаңдар.

4. Төбелерінің координаталары бойынша дөңес төртбұрыштың ауданын анықтаңдар, кесіндінің ұзындығын есептеуге ішпрограмма-функцияны және үшбұрыштың ауданын есептеудің Герон формуласына ішпрограмма-процедураны пайдаланып есептеңдер.

2.13. РЕКУРРЕНТТІ ТІЗБЕКТЕРДІ ЕСЕПТЕУ

Рекуррентті тізбек. Рекуррентті тізбек ұғымы математикадан белгілі. Ол келесі түрде енгізіледі: *к* тізбегі белгілі болсын: $a_{1,\ldots,}a_k$

сандық тізбектің басы болып табылады. Бұл тізбектің келесі элементтері мына формулалар бойынша есептеледі:

$$a_{k+1} = F(a_1, \ldots, a_k);\ a_{k+2} = F(a_2, \ldots, a_{k+1});$$

$$a_{k+3} = F(a_3, \ldots, a_{k+2}) \ldots,$$

мұндағы F(...) — k аргументінің функциясы.

Төмендегі формула

$$a_i = F(a_{i-1}, a_{i-2}, \ldots, a_{i-k})$$

рекуррентті деп аталады, ал k шамасы — *рекурсия тереңдігі* деп аталады.

Басқаша айтқанда *рекуррентті тізбек* дегеніміз — бұл сандардың шексіз қатары, мұнда бастапқы k –дан басқа, әрқайсысы алдыңғы сандар арқылы есептеледі.

Рекуррентті тізбекке арифметикалық және геометриялық прогрессиялар мысал болады.

$$a_1=1,\ a_2=3,\ a_3=5,\ a_4=7,\ a_5=9,\ \ldots;$$

$$a_1=1,\ a_2=2,\ a_3=4,\ a_4=8,\ a_5=16,\ \ldots;$$

Берілген арифметикалық прогрессияның рекуррентті ормуласы:

$$a_i = a_{i-1} + 2.$$

Берілген геометриялық прогрессияның рекуррентті формуласы:

$$a_t = 2a_{t-1}.$$

Екі жағдайда да рекурсия тереңдігі 1-ге тең (мұндай тәуелділікті бір қадамды рекурсия деп те атайды). Жалпы алғанда, рекуррентті тізбек бастапқы мәндер жиынтығымен және рекуррентті формуламен сипатталады. Барлығын бір тармақты формулаға біріктіруге болады, арифметикалық прогрессия үшін оның формасы келесі түрде болады:

$$a_i = \begin{cases} 1, & \text{егер } i = 1; \\ a_{i-1} + 2, & \text{егер } i > 1, \end{cases}$$

ал геометриялық прогрессия үшін -

$$a_i = \begin{cases} 1, & \text{егер } i = 1; \\ 2a_{i-1}, & \text{егер } i > 1, \end{cases}$$

Сандық тізбегі түрінде жазсақ,

1; 1; 2; 3; 5; 8; 13; 21; 34; 55 …,

математикада Фибоначчи сандары деп аталатын, үшінші элементтен бастап әрбір сан алдыңғы екі мүшесінің қосындысына тең болатын, тереңдігі 2-ге тең рекуррентті тізбек болып табылады (екі қадамды рекурсия). Оны тармақталған түрде жазайық:

$$f_i = \begin{cases} 1, & i = 1,\ i = 2; \\ f_{i-1} + f_{i-2}, & i > 2. \end{cases}$$

Рекуррентті тізбектер түсінігін енгізу бізге белгілі есептерге жаңа көзқараспен қарауға мүмкіндік береді. Мысалы, n бүтін санының факториалын келесі сандар қатарының n-ші элементінің мәні ретінде қарастыруға болады:

$$a_0 = 1;\ a_1 = 1!;\ a_2 = 2!;\ a_3 = 3!;\ a_4 = 4!;\ \ldots$$

Мұндай тізбектің рекуррентті сипаттамасы келесі түрде болады

$$a_i = \begin{cases} 1, & i = 0; \\ a_{i-1}\, i, & i > 0. \end{cases}$$

Рекуррентті тізбектерді есептеуді программалау. Рекуррентті тізбектермен келесі есептер байланысты:

1) тізбектің берілген (*n*-ші) элементін есептеу;

2) тізбектің белгілі бір бөлігін математикалық өңдеу (мысалы, қосындыны есептеу немесе алғашқы *n*-мүшелердің көбейтіндісі);

3) белгілі бір қасиеттерді қанағаттандыратын берілген тізбек кесіндісіндегі элементтер санын анықтау;

4) белгілі бір талаптарды қанағаттандыратын бірінші элементтің нөмірін анықтау;

5) жадыда тізбек элементтерінің белгілі санын есептеу және сақтау.

Бұл тізім өзінің толықтығы туралы талап етпестен, ең көп таралған есептер түрлерін қамтиды. 1 ... 4 тапсырмаларда бір мезгілде сандық қатардағы элементтер жиынтығын жадыда сақтау қажет емес: *оның элементтері бір айнымалыда бір-бірін алмастырып тізбектелген күйде пайда болуы мүмкін.*

2.4-Мысал. Арифметикалық прогрессияның *n*-ші элементін есептеу қажет (2.1).

Берілген есепті шешу программасы:

```
Var N, I: 0..MaxInt;
  A : Real;
Begin
  Write('N = ');
  ReadLn(N);
  A := 1;
  For I := 2 To N Do
      A := A + 2;
  WriteLn('A(', N:1,  ')= ', A:6:0)
End.
```

Рекуррентті формула $a_t = a_{t-1} + 2$ келесі операторға өтті $A := A + 2$.

2.5-Мысал. Прогрессияның қосынды формуласын қолданбай, геометриялық прогрессияның алғашқы n элементтерін қосындылау қажет (2.2). Берілген есепті шешу программасы:

```
Var N, I: 0..MaxInt;
  A, S : Real;
Begin
  Write('N = '); ReadLn(N);
  A := 1;
  S : = A;
  For I : = 2 To N Do
  Begin
      A := 2 * A;
      S := S + A
  End;
  WriteLn('Қосынды тең ', S:6:0)
End.
```

Тереңдігі екіге тең рекурренті тізбекті есептеу барысында, бір айнымалыны ғана пайдалануға келмейтінін төмендегі мысалдан көруге болады.

2.6-Мысал. Фибоначчидің алғашқы n(n > 3) сандарын шығару және олардың ішінде қанша жұп сандар бар екенін анықтау қажет.

Берілген есепті шешу программасы мынадай:

```
Var N, I, K, F, F1, F2 : 0.. MaxInt;
Begin
  F1 := 1; F2 := 1;
  K := 0;
  WriteLn('F(1) = ', F1,  ' F(2) = ', F2);
  For I := 3 To N Do
  Begin
       F := F1 + F2;
```

```
        WriteLn('F(', I : 1, ') = ', F);
        If Not Odd(F) Then K := K + 1;
        F1 := F2; F2 := F
    End;
    WriteLn('Тізбектегі жұп сандар тең ', K)
End.
```

Екі қадамды рекурсияны есептеуге үш айнымалы қажет, себебі келесі элементті есептеу үшін алдыңғы екі элементтердің мәндерін есте сақтау қажет.

2.7-Мысал. Берілген нақты x және кіші шама ε үшін (например, ε = 0,000001) келесі қатардың қосындысын есептеу қажет және ε-нің шамасынан асатын тек қосылғыштарды пайдалану керек

$$1 + x + \frac{x^2}{2!} + \frac{x^2}{3!} + \cdots,$$

Шексіз тізбектердің қосындысы e^x шекті мәніне тең екені математикадан белгілі, яғни e = 2.71828 ... - натурал логарифмінің негізі. Осы тізбектегі элементтер нөлге ұмтылатын сандардың кему ретін білдіретіндіктен, қосындылауды бірінші қосылғышқа дейін ε мәнінен аспайтын абсолюттік мәнде орындау қажет.

Бұл өрнектегі қосылғыштарды былай белгілесек,

$$a_0 = 1;\ a_1 = x;\ a_2 = \frac{x^2}{2!};\ a_3 = \frac{x^3}{3!}, \ldots,$$

онда i-ші элемент үшін жалпы формула келесі түрдегідей болады:

$$a_i = \frac{x^i}{i!}.$$

Бұл тізбектің элементтері арасынан рекуррентті тәуелділіктің бар екендігін көру қиын емес. Оны интуициялық түрде табуға болады, бірақ формалды түрде шығару керек. Рас, бұл үшін рекурсияның бір қадамды екенін және әрбір келесі элемент алдыңғы элементті қандайда бір көбейткішке көбейту арқылы алынатынын байқау қажет, яғни

$$a_i = K a_{i-1}$$

жалпы формуланы пайдаланып:

$$\frac{x^l}{i!} = K \frac{x^{i-1}}{(i-1)!},$$

бұдан

$$K = \frac{x^i/i!}{x^{i-1}/(i-1)} = \frac{x}{i},$$

шынымен,

$$a_0 = 1;\ a_1 = a_2\frac{x}{2};\ a_3 = a_2\frac{x}{3}\ldots$$

Осыдан, берілген рекуррентті тізбек келесі түрде сипатталады:

егер $i = 0$;

$$a_i = \begin{cases} 1, & \text{егер } i = 0; \\ a_{i-1}\dfrac{x}{i}, & \text{егер } i > 0. \end{cases}$$

Қарастырылған есептің программасын құрайық:

```
Var A, X, S, Eps : Real;
    I : Integer;
Begin
  Write('X = '); ReadLn(X);
  Write('Epsilon = '); ReadLn(Eps);
  A := 1; S := 0; I := 0;
  While Abs(A) > Eps Do
  Begin
    S := S + A;
    I : = I + 1;
    A := A * X/I
  End;
  WriteLn('Қатардың қосындысы тең ', S : 10 : 4)
End.
```

Бір қадамды рекурренті тізбектің мәні бір айнымалымен есептеліп тұр.

Бұл программа циклінің әрбір қайталануы S мәнін нәтижеге жақындатады (оның жазбасындағы маңызды сандарды нақтылайды). Мұндай есептеу процесі математикада *итерациялық процесс* деп аталады. Тиісінше, итерациялық есептеу үдерісін іске асыратын циклдер *итерациялық циклдер* деп аталады. Оларды ұйымдастыру үшін, While немесе Repeat операторларын пайдаланамыз.

2.8-Мысал. *N* берілген натурал саны мен *x* нақты саны үшін (x > 0) келесі өрнекті есептеу керек:

$$\sqrt{x+\sqrt{x+\ldots+\sqrt{x}}}.$$

Бұл жағдайда рекурренттілік байқалмайды. Оны индукция әдісімен тауып көрейік. Ізделіп отырған өрнек келесі тізбектің N-ші элементі делік,

$$a_1 = \sqrt{x};\ a_2 = \sqrt{x + \sqrt{x}};\ a_3 = \sqrt{x + \sqrt{x + \sqrt{x}}}\ \ldots,$$

бұдан келесі байланыс байқалады:

$$a_2 = \sqrt{x + a_1};\ a_3 = \sqrt{x + a_2}; \ldots;\ a_i = \sqrt{x + a_{i-1}}.$$

Енді бұл есепті шешу қиын емес:

```
Var A, X : Real; I, N : Integer;
Begin
  Write('X = '); ReadLn(X);
  Write('N = '); ReadLn(N);
  A := Sqrt(X);
  For I := 2 To N Do
       A := Sqrt(X + A);
  WriteLn('ЖАУАП:  ', A)
End.
```

Барлық келтірілген есептерді Паскальдағы функцияны рекурсивті түрде анықтау арқылы шешуге болады.

Паскальдың рекурсивті ішпрограммалары туралы толығырақ 2.17-бөлімде қарастырылады. Арифметикалық прогрессияның сипаттамасынан рекуррентті тізбек түрінде берілген элементті есептеуге арналған функцияны анықтау әдісі тікелей орындалады.

Жалпы жағдай үшін a_0 бастапқы мәнімен және айырымы d болатын арифметикалық прогрессияны анықтайық:

$$a_i = \begin{cases} a_0, & \text{егер } i = 1; \\ a_{i-1} + d, & \text{егер } i > 1. \end{cases}$$

Сәйкес ішпрограмма-функция келесі түрде жазылады:

```
Function Progres(A0, D : Real; I : Integer): Real;
Begin
  If I = 1
  Then Progres := A0
  Else Progres := Progres(A0, D, I_1) + D
End;
```

Келесі программа экранға мәндерін Fibon рекурсивті функциясы анықтайтын Фибоначчидің алғашқы 20 санын шығарады:

```
Var K : Byte;
Function Fibon(N : Integer) : Integer;
Begin
  If (N = 1) Or (N = 2)
  Then Fibon := 1
  Else Fibon := Fibon(N - 1) + Fibon(N - 2)
End;
Begin
  For K := 1 To 20 Do WriteLn(Fibon(K))
End.
```

Рекурсивті функцияларды пайдалану есептеудің баяулауына әкелетінін айта кету керек. Сонымен қатар, рекурсивті шақырулардың «маршруттары» еске сақталынатын стек ұзындығының жетіспеушілігімен проблема туындауы мүмкін.

Рекуррентті тізбектер көбінесе эволюциялық есептердің түрлерін шешу үшін жиі пайдаланылады, яғни белгілі бір уақыттың ішінде дамып келе жатқан белгілі бір есептер үдерісі бақыланады. Осындай есепті қарастырайық.

2.9-Мысал. Емдеу диетасы кезінде науқастың салмағы 30 күн ішінде 96 кг-нан 70 кг-ға дейін азайды. Науқастың дене салмағы диета басталғаннан бастап k күннен кейін k = 1, 2, ..., 29 үшін қаншаға тең болғанын есептеу керек.

i-ші күнгі науқастың дене салмағын p_i (*i* = 0, 1, 2,., 30) деп белгілейік. Есептің шартынан $p_0 = 96$ кг, $p_{30} = 70$ кг екені белгілі. *K* — бір күндегі массаның азаюының пропорционалды коэффициенті болса,

$$p_1 = p_0 - Kp = (1-K)\,p_0;\ p_2 = (1-K)p_1;$$
$$p_3 = (1-K)p_2;\ \ldots;\ p_i = (1-K)p_i - 1$$

яғни келесі рекуррентті формуламен сипатталған тізбек аламыз:

$$p_i = \begin{cases} 96, & \text{егер } i = 0; \\ (1-K)p_{i-1}, & \text{егер } 1 \le i \le 30. \end{cases}$$

$p_{30} = 70$ шартын пайдаланып және «кері» қоюды орындау арқылы *K* коэффициентін табуға болады:

$$p_{30} = (1-K)p_{29} = (1-K)^2 p_{28} = (1-K)^3 p_{27} = \ldots =$$
$$= (1-K)^{30}\,p_0 = (1-K)^{30} \cdot 96.$$

$(1 - K)^{30} \cdot 96 = 70$ теңдігінен:

$$1 - K = \left(\frac{70}{96}\right)^{\frac{1}{30}}.$$

Ары қарай бұл есептің программасы мынадай болады:

```
Var I : Byte;
    P, Q : Real;
Begin
  P := 96;
  Q := Exp(1/30 * Ln(70/96));
  For I := 1 To 29 Do
  Begin
       P := Q * P;
       WriteLn(I, '-ші күн ', P : 5 : 3,  ' кг')
  End
End.
```

ЖАТТЫҒУЛАР

1. Келесі әрекеттерден рекуррентті тізбек анықталған:

$$a_i = \begin{cases} 1, & \text{егер } i = 0; \\ a_{i-1} i + \dfrac{1}{i}, & \text{егер } i > 0. \end{cases}$$

Берілген n натурал саны үшін a_n мәнін анықтаңдар.

2. Мына тізбек берілген:

$$a_i = \begin{cases} 1, & \text{егер } i = 0, i = 1; \\ a_{i-2} + \dfrac{a_{i-1}}{i-1}, & \text{егер } i > 1. \end{cases}$$

1-ден 20-шы элементтердің көбейтіндісін есептеу қажет.

3. Рекурренттілікті пайдаланып Горнер формуласы бойынша 10-шы дәрежелі көпмүшенің қосындысын есептеңдер:

$$10x^{10} + 9x^9 + 8x^8 + \ldots + 2x^2 + x = (((((((((10x + 9)x + 8)x + \ldots + 2)x + 1)x,$$

мұндағы x — берілген нақты сан.

4. Берілген x нақты саны үшін және N натурал саны үшін келесі тізбекті есептеңдер:

$$x/(1 + x/(2 + x/(3 + x/(\ldots /(N + x))\ldots).$$

5. 10^5 мәнінен асатын сандық қатардың барлық мүшелерін шығару және есептеу.

$$1; \frac{1}{2!}; \frac{1}{3!}; \ldots; \frac{1}{N!},$$

6. $y = \sqrt{x}$ функциясын тізбектің шекті мәні ретінде есептеуге болады, бұл функцияны келесі рекуррентті формула бойынша анықтау керек:

$$y_k = \frac{1}{2}\left(y_{k-1} + \frac{x}{y_{k-1}}\right), k = 1,2, \ldots \text{ үшін}$$

Мұнда y_0 бастапқы мәні ($\sqrt{x}$-ке жақын) еркін беріледі. Дәлдікпен жақындаған е түбірінің мәні ретінде $|y_k - y_k - 1| < e$ шарты орындалатын y_k алынады. Есептеудің сәйкес программасын құрыңдар.

2.14. ТУРБО ПАСКАЛЬДЫҢ ГРАФИКАЛЫҚ ҚҰРАЛДАРЫ

Әзірге біз компьютер экранын тек символдық ақпараттарды - сандар мен мәтіндерді шығару үшін қолдандық. Алайда Турбо Паскаль бейнелерді, сызбаларды, функциялар графиктерін, диаграммаларды, яғни компьютерлік графика деп аталатынның барлығын экранға шығаруға мүмкіндік береді.

Стандартты Паскальда графиканы шығару қамтамасыз етілмеген. Дегенмен, компьютерлердің әртүрлі типтерінде графиканы шығаруға арналған түрлі программалық құралдары бар: арнайы деректер жиынтығы, функциялар, процедуралар. Бұл жағдайда қандай да бір программалау тілінде графиканы жүзеге асырудың кез-келген нұсқасына тән жалпы түсініктер мен құралдар бар. Осындай негізгі құралдарды қарастырайық.

IBM PC үшін Турбо Паскальдың төртінші нұсқасынан бастап Graph модулінде ұйымдастырылған қуатты графикалық кітапхана пайда болды. 2-қосымшада осы модульдің негізгі компоненттерінің сипаттамасы анықтама формасында берілген. Graph модулін қосу үшін программаның басында мына жолды жазу керек

```
Uses Graph;
```

Экранның графикалық режимдері. Графикалық бейнелерді шығару үшін, экранды графикалық режимдердің біріне ауыстыру керек. Графикалық режимде программаның көмегімен әр пиксельдің — экранның нүктелік элементі күйін бақылауға болады.

Графикалық режимдер мыналармен ерекшеленеді:

- графикалық тордың өлшемі (M x N, мұндағы M — көлденеңінен орналасқан нүктелер саны; N — тігінен орналасқан нүктелер саны);
- түрлі-түстілігі (экранға шығарылатын түстер саны).

Рұқсат етілген режимдер монитор типіне және компьютерде қолданылатын сәйкес графикалық драйверге тәуелді болады.

Экранның графикалық режимін орнату үшін сәйкес процедуралар қолданылады. Graph модулінде экранның графикалық режимін орнату үшін келесі тақырыпты жазу қажет:

```
Procedure InitGraph(Var Driver, Mode : Integer;
Path : String);
```

Мұнда бүтін айнымалы Driver - графикалық драйвердің түрін анықтайды, бүтін айнымалы Mode - графикалық драйвердің жұмыс істеу режимін орнатады және Path - графикалық драйвер файлын іздестіру жолын қамтитын String типтегі шама.

Драйверлердің типтерін және режимдерін анықтайтын Graph модулінің тұрақтылар тізімдері 2-қосымшада келтірілген (Қ2.1 және Қ2.2 кестелерін қараңыз).

VGA драйверімен жұмыс істеу үшін графикалық VGAHi режимін инициализациялайтын программа мысалы (VGA мониторы):

```
Uses Graph;
Var Driver, Mode : Integer;
Begin
   Driver : = У^Л;{Драйвер}
   Mode : = VGAHi;{Жұмыс режимі}
   InitGraph(Driver, Mode, 'C:\TP\BGI');
```

Мұнда VGA мониторы үшін egavga.bgi драйвері бар файлы C: \ TP \ BGI каталогында орналасқандығын көрсетеді. VGAHi режимі 16 түстер палитрасы бар 640 x 480 графикалық торға сәйкес келеді.

Сондай-ақ, программа мәтініне өзгертулер енгізбестен әр түрлі мониторлармен жұмыс жасауға мүмкіндік беретін драйвер типін және режимін орнатуды автоматты түрде анықтауға болады:

```
Driver := Detect;
InitGraph(Driver, Mode, 'C:\TP\BGI');
```

Сонымен қатар ең жоғары қабілетті кеңейтімі мен түсі бар режим автоматты түрде орнатылады. Жұмысты графикалық режимде аяқтағаннан кейін экранның мәтіндік режиміне оралуымыз керек.

Graph модулінде мәтін режиміне оралу процедурасы:

```
Procedure CloseGraph;
```

Фон түсі және сурет түсі. Түрлі-түсті мониторда экранның түсін өзгертуге болады. Экранның орнатылған түсі фон түсі деп аталады. Бұл фондағы сурет әр түрлі сызықтарды қолданумен салынады: түзулер, шеңберлер, тіктөртбұрыштар, сынық сызықтар және т.б. Бұл сызықтардың түстері әр түрлі болуы мүмкін.

EGA, VGA типті мониторларына арналған 16 түсті палитраны анықтайтын тұрақтылар атаулары қосымшаның Қ2.3-кестесінде келтірілген.

Фон түсін орнату процедурасының тақырыбы:

```
Procedure SetBkColor(Color : Word);
```

Мұнда Color — фон түсінің нөмірін анықтайтын бүтін типті өрнек.

Түзу сызық түсін орнату процедурасының тақырыбы:

```
Procedure SetColor(Color : Word);
```

Егер сызық түсі нөмірі ретінде 0 көрсетілген болса, онда ол әрдайым фон түсімен (көрінбейтін сызық) сәйкес келеді.

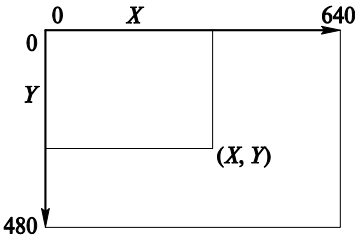
Егер графикалық экранды тазалау қажет болса (суретті өшіріп тастау) келесі атауды қамтитын экранды тазалау процедурасын қолданамыз:

```
Procedure ClearDevice;
```

Бұл процедураның орындалу нәтижесінде экран орнатылған фон түсімен толтырылады.

Графикалық координаталар. Графикалық торда әрбір пиксельдің орналасуы оның координаталарын көрсету арқылы анықталатыны сөзсіз. Экрандағы графикалық осьтердің орналасуы 2.25-суретте көрсетілген. *X* – көлденең осі солдан оңға қарай, *Y* — вертикаль осі жоғарыдан төмен қарай бағытталған.

VGAHi режиміне сәйкес шекті графикалық координаталар 2.25-суретте көрсетілген.



2.25-сурет. Экранды координаталарының жүйесі

Келесі екі бүтінсанды функциялардың көмегімен берілген драйверге сәйкес осьтер бойынша максималды координаталарды анықтауға болады:

```
Function GetMaxX;
Function GetMaxY;
```

Графикалық терезе. Бейнелерді шығару аумағы экранда кез келген тіктөртбұрышпен шектелуі мүмкін. Мұндай аймақ графикалық терезе деп аталады. Экрандағы графикалық терезенің орнын анықтайтын процедура бар.

Графикалық терезені өңдеу процедурасының тақырыбы:

```
Procedure SetViewPort(X1, Y1, X2, Y2 : Integer;
Clip : Boolean);
```

Мұнда (X1, Y1) — терезенің сол жақ жоғарғы бұрышының координаталары; *(X2,* Y2) — терезенің оң жақ төменгі бұрышының координаталары; Clip —фигуралар шектегіші.

Егер Clip = True болса, онда барлық сызбалар тек терезе шегінде жүзеге асырылады, әйтпесе олар терезе шегінен шығып кетулері мүмкін.

Терезені орнатқан соң оның ішіндегі нүктелердің координаталары оның жоғарғы бұрышынан бастап саналады.

Графикалық меңзер ұғымы бар, ол символдық меңзерден қарағанда экранда көрінбейді. Графикалық меңзер экрандағы ағымды орынды көрсетеді. Графикалық режимге кіргенде, ағымдағы орынның координаттары (0, 0) болады.

Графикалық меңзердің координаталарын орнату процедурасы:

```
Procedure MoveTo (X, Y : Integer);
```

Мұнда X, Y - терезенің немесе экранның жоғарғы сол жақ бұрышына (егер терезе орнатылмаған болса) орнатылатын меңзер координаталары.

Нүктені қою процедурасы - бейне алудың негізгі процедурасы болып табылады, себебі кез-келген сурет нүктеден құралады. Жарық нүктесінің орналасуы экрандағы координаталары және түстері арқылы анықталады.

Графикалық экранда нүкте қоюдың процедура тақырыбы

```
Procedure PutPixel(X, Y : Integer; Color : Word);
```

Мұнда X, *Y* — нүкте координаталары; Col**or** — нүкте түсі.

2.10-Мысал. Экранның ортасында 100x100 өлшемде графикалық терезені орнататын, фоны сары түспен боялатын және төрт позиция бойынша орналасатын көк нүктелермен толтырылу керек программа келесідей болады:

```
Uses Graph;
Var Driver, Mode : Integer;
     X, Y, X1, Y1, X2, Y2, Xc, Yc : Integer;
Begin
{---Графикалық режимді инициализациялау -}
     Driver : = Detect;
     InitGraph(Driver, Mode, 'C:\TP\BGI');
{---Экран центрі координаталарын анықтау --}
     Xc := GetMaxX Div 2;
     Yc : = GetMaxY Div 2;
{---Графикалық терезенің координаталарын анықтау---}
     X1 := Yc - 50
     Y1 := Yc - 50
     X1 := Yc + 50
     Y1 := Yc + 50
{---Графикалық терезені орнату ----}
     SetViewPort(X1, Y1, X2, Y2, True);
{---Фон түсін және экранды тазалауды орнату -}
     SetBkColor(Yellow);
     ClearDevice;
{---Терезедегі нүктелерді орналастыру---}
     For X := 1 To 25 Do
        For Y := 1 To 25 Do
        PutPixel(4 * X, 4 * Y, Blue);
{---<Enter>-ді басқанға дейін экранда суретті кідірту--}
     ReadLn;
{---Графикалық режимнен мәтіндік режимге шығу ---}
     CloseGraph;
End.
```

Графикалық примитивтер. Кез-келген бейнені нүктелерден құрастыра аламыз, бірақ нүкте қою процедурасын пайдаланып күрделі бейне немесе сызбаларды алуды программалау өте ыңғайсыз және қиын. Кез-келген графикалық пакетте негізгі геометриялық фигураларды салу процедуралары бар: түзу сызықтар, шеңберлер, эллипстер, тіктөртбұрыштар және т.б. Мұндай фигуралар *графикалық примитивтер* деп аталады.

Graph модуліндегі графикалық примитивтерді салудың бірнеше негізгі процедураларын қарастырайық:

■ (X1, Y1) және (X2, Y2) түзу сызықтың нүктелерінің координаталары:

```
Procedure Line(X1, Y1, X2, Y2 : Integer);
```

■ Меңзер тұрған орыннан бастап (X, Y) координатасына дейінгі түзу сызық:

```
Procedure LineTo(X, Y : Integer);
```

■ Ағымдағы нүктеден қашықтығы координаталар өсімшесіне *(DX, DY)* сәйкес болатын жаңа нүктеге дейін түзу сызық сызады:

```
Procedure LineRel(DX, DY : Integer);
```

■ сол жақ жоғарғы (X1, Y1) және оң жақ төменгі (X2, Y2) төбелерінің координаталары бойынша тіктөртбұрыш сызады:

```
Procedure Rectangle(X1, Y1, X2, Y2 : Integer);
```

Радиусы пиксель бойынша *R* болатын және центрі (X, *Y)* нүктесінде орналасқан шеңбер:

```
Procedure Circle(X, Y : Integer; R : Word);
```

■ Радиусы - R, бастапқы бұрышы - BegA, соңғы бұрышы - EndA болатын, центрі (X, Y) нүктесінде орналасқан шеңбер доғасы (бұрыштар *X* осінен сағат тіліне қарама-қарсы градуспен өлшенеді):

```
Procedure Arc(X, Y : Integer; BegA, EndA, R :Word);
```

■ Горизонталь радиусы - RX және вертикаль радиусы - *RY*, бастапқы бұрышы - BegA, соңғы бұрышы - EndA болатын, центрі (X, Y) нүктесінде орналасқан эллипс доғасы:

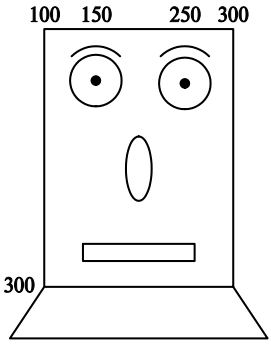
```
Procedure Ellipse(X, Y : Integer; BegA, EndA, RX, RY :
Word);
```

2.11-Мысал. Роботтың басын сызатын программа құрайық (2.26-сур.).

Бұл суретте екі тіктөртбұрыш, екі шеңбер, екі доға, эллипс, үш түзу сызық және екі қара нүкте қолданылады. Барлық координаталарды, сурет элементтерінің өлшемдерін алдын ала анықтап, программа құрамыз:

```
Uses Graph;
Var Driver, Mode : Integer;
Begin
{----графикалық режимді инициализациялау }
  Driver := Detect;
```

2.26-сурет. Робот басының сызбасы.




```
  InitGraph(Driver, Mode, 'C:\TP\BGI');
  SetColor(White); {Суреттің ақ түсі}
  SetBkColor(Black); {Фонның қара түсі}
  Rectangle(100, 100, 300, 300); {Басы}
  Circle(150, 170, 30); {Сол жақ көз}
  Circle(250, 170, 30); {Оң жақ көз}
  Arc(150, 170, 45, 135, 40); {Сол жақ қас}
  Arc(250, 170, 45, 135, 40); {Оң жақ қас}
  Ellipse(200, 250, 0, 359, 10, 20); {Мұрын}
  Rectangle(130, 280, 270, 290); {Ауыз}
  MoveTo(100, 300); {Солға қарай төмен орналасу}
  LineTo(50, 350); {Үш}
  LineTo(350, 350); {Сызықтар}
  LineTo(300, 300); {Мойын}
  PutPixel(150, 170, Black); {Сол көз қарашығы}
  PutPixel(250, 170, Black); { Оң көз қарашығы}
  ReadLn; {Кідіріс}
  CloseGraph; {Графикадан шығу}
End.
```

Берілген мысалда барлық сызықтар қалыңдығы тұрақты болып табылады. Graph модулі сызықтың стилін (біртұтас, пунктирлі, нүктелі және т.б.) және оның қалыңдығын басқаруға мүмкіндік береді. Ол үшін SetLineStile процедурасын (2.4Қ кестені қара) қолданамыз.

Бояу және толтыру. Графикалық примитивтердің ішінде боялған аймақтар бар. Бояу түсі SetColor процедурасымен анықталады. Сонымен қатар, бояу суретін (бояу типін) басқаруға болады. Бұл біртұтас бояу, сирек нүктелермен толтыру, крест сызықтары, штрихтар және т.б. Бояуды толтыру тұрақтылары қосымшаның 2.5Қ-кестесінде көрсетілген.

Толтыру типі (Fill) мен түсті (Color) анықтайтын процедура бар:

```
Procedure SetFillStyle(Fill, Color : Word);
```

Берілген бұрыштардың координатасы бойынша тіктөртбұрышты аймақты бояумен толтыру процедурасы:

```
Procedure Bar(X1, Y1, X2, Y2 : Integer);
```

Сызықпен қоршау (SetLineColor, SetLineStyle) және эллипсті бояу процедурасы (SetFillStyle):

```
Procedure FillEllips(X, Y, RX, RY : Integer);
```

Сызықпен қоршау және эллипс секторын бояу процедурасы:

```
Procedure Sector(X, Y : Integer; BegA, EndA, RX, RY: Word);
```

Сызықпен қоршау және шеңбер секторын бояу процедурасы:

```
Procedure PieSlice(X, Y : Integer; BegA, EndA :Word);
```

Сызықпен бекітілген кез-келген аймақты бояуға болады. Ол үшін осы аймақтағы белгілі бір нүктенің (X, *Y)* координаталарын және бекітілген сызық түсін (Border) көрсету керек. Сәйкес процедура келесідей:

```
Procedure FloodFill(X, Y : Integer; Border : Word);
```

Graph модулі графикалық экранда мәтіндерді көрсетуге мүмкіндік береді. Бұл мәселені егжей-тегжейлі талқыламаймыз (қажетті ақпаратты арнайы әдебиеттен табуға болады), біз тек бір ғана мәтіндік процедураны мысалға келтіреміз, ол арқылы белгілі бір (X, Y) позициядан графикалық терезеге символдық жол (Txt) шығарылады:

```
Procedure OutTextXY(X, Y : Integer; Txt : String);
```

Мысалы, 2.26-суреттің астына БҰЛ РОБОТ жолын енгізу үшін, программаға мына операторды қосу керек:

```
OutTextXY(195, 400, 'БҰЛ РОБОТ');
```

Функция графигін сызу. Компьютерлік графика қосымшаларының бірі математикалық есептеулердің нәтижелерін

көрнекі түрде ұсынуды қамтамасыз етеді. Сызылған функция графигі, диаграмма, кеңістіктегі тәуелділіктерді орналастыру деңгейлерінің сызықтары және басқалары есептеулердің нәтижелерін неғұрлым айқын, анық, түсінікті етеді.

Енді математикалық графиканың ең қарапайым нұсқасы – функция графигін сызуды қарастырайық.

Экран дисплейінде $y = F(x)$ функция графигін құруды есептеу үшін келесі кезек бойынша әрекет жасаған дұрыс.

1. Сызылатын графиктің аргумент мәндерінің шекараларын X_{min} (төменгі шек) және X_{max} (жоғарғы шек) белгілеулері арқылы анықтау.

2. Аргумент мәндерінің берілген облысы үшін Ymin және Ymax функциясының шекті мәндерін анықтау. Бұл мәндердің дәл болуы міндетті емес. Оларды бағалауға болады.

3. Графикалық терезенің шекарасын орнату, себебі оның ішіне график сызылады: [Xg_{min}, Xg_{max}], [^ SW Yg_{max}]. Графикалық координаталарда тік ось төмен бағытталғандықтан, Yg_{min}> Ygmax болады.

Сонымен, екі координаталық жүйе бар екенін білдік: (X, Y) - математикалық координаталар (әдебиетте «әлемдік координата» термині жиірек қолданылады) және (Xg, Yg) - графикалық координаталар.

Графикалық және математикалық координаталарға қатысты формулаларды алу қиын емес:

$$Xg = Xg_{min} + \left[\frac{Xg_{max} - Xg_{min}}{X_{max} - X_{min}}(X - X_{min})\right];$$

$$Yg = Yg_{min} + \left[\frac{Yg_{max} - Yg_{min}}{Y_{max} - Y_{min}}(Y - Y_{min})\right]$$

Мұнда тік жақшалар бүтін мәнге дейін дөңгелектеуді білдіреді (Round функциясы).

Функция графигін тұрғызу нүктелік немесе кесік-сызықты әдіспен жүзеге асады. *Нүктелік әдіс* график максималды түрде жақын орналасқан нүктелер тізбегі түрінде сызылады, яғни аргумент мәнінің [Xg_{min}, Xg_{max}] аралықтағы сәйкес *Y*-координаталарға нүктелерді орналастыру арқылы «попиксельді» таңдау орындалады.

Кесік-сызықты әдіс (X_i,Y_i) мәндер тізбегі есептеліп, ΔX математикалық қадамы тағайындалады:

$$X_i = X_{max} + i\Delta X; Y_i = F(X_i);$$
$$i = 0, 1, \dots, n; n = \frac{X_{max} - X_{min}}{\Delta X}$$

График (X Y_i), (X_i + 1, Yi + 1) нүктелері арқылы өтетін түзудің кесінділері түрінде сызылады.

2.12-Мысал. Нүктелік әдісті пайдаланып, $x \in [0; 2\pi]$ үшін y = sinx функциясының графигін сызудың программасын құру керек.

Есептің шарты бойынша $X_{min} = 0$, $X_{max} = 2\pi$. Бұл шектерде sin$x$ функциясы (-1; 1) аралығында өзгереді, сондықтан $Y_{min} = -1$, $Y_{max} = 1$.

Графикалық терезенің шекараларын орнатайық: $Xg_{min} = 10$; $Xg_{max} = 200$; $Yg_{min} = 140$; $Yg_{max} = 40$.

График нүктелер тізбегі түрінде келесі математикалық координаталар бойынша тұрғызылады:

$$Xi = X_{min} + ih;\ Y_i = sin(X_i);\ i = 0, \ldots, 190.$$

h қадамы графикалық тордың қадамына сәйкес минималды мүмкіндігі бойынша таңдалады:

$$h = \frac{X_{max} - X_{min}}{Xg_{max} - Xg_{min}} = \frac{2\pi}{190} = \frac{\pi}{95}.$$

Математикалық координаталарды графикалық координаталарға түрлендіру формулалары келесі түрде болады:

$$Xg = 10 + \left[\frac{200 - 10}{2\pi} X\right] = 10 + \left[\frac{95}{\pi}\right];$$
$$Xg = 140 + \left[\frac{40 - 140}{2}(Y + 1)\right] = 90 - [50Y].$$

Функция графигімен бірге координата осьтері сызылады. X осі $Yg = 90$ координатасын, ал Y осі — $Xg = 10$ координатын қамтиды.

График тұрғызудың программасы:

```
Uses Graph;
Var Driver, Mode : Integer;
    X : Real; Xg, Yg, I : Integer;
Begin
{ --  Графикалық режимді инициализациялау- }
    Driver := Detect;
    InitGraph(Driver, Mode, 'C:\TP\BGI');
    SetColor(White); {Ақ түсті сызық} SetBkColor(Black);
    {Қара түсті фон}
    Line(10, 90, 200, 90); {Ось X}
    Line(10, 20, 10, 160); {Ось Y}
{ --  Функция графигін сары нүктелермен құру------ }
    X : = 0;
    For I := 0 To 190 Do
    Begin Xg := 10 + Round(95/Pi * X);
      Yg := 90 - Round(50 * Sin(X));
      PutPixel(Xg, Yg, Yellow);
```

```
        X := X + Pi/95
    End;
{----Функцияны жазу, осьтерді белгілеу---}
    OutTextXY(15, 30, 'Y');
    OutTextXY(205, 90, 'X');
    OutTextXY(130, 40, 'Y =
    Sin(X)'); Readln; {Кідіріс}
    CloseGraph; {Графикадан шығу}
End.
```

ЖАТТЫҒУЛАР

1. «Жұлдызды аспан» программасын құру: қара терезеге белгілі бір пернені басқанда программа жұмысын аяқтайтын кездейсоқ орындарда ақ нүктелерді орналастырыңдар.
2. Жаңа жұлдыз пайда болғанда, жанып тұрған жұлдыздардың сөнуін (фон түсін бояу) көрсететін «Жұлдызды аспан» программасын өзгертіңдер.
3. 2.11-мысалының программасына өзгертулер енгізіңдер. Нәтижесінде Робот әр түрлі түстерге боялу керек.
4. Сызықты және басқа да графикалық примитивтерді пайдаланып үйдің программасын құрыңдар.
5. Экранда шахмат тақтасын құратын программа құрыңдар.
6. $y = F(x)$ функциясының графигін нүктелік әдіспен құру үшін әмбебап процедура жасаңдар. Процедураның келесі параметрлері болуы керек. X_{min}, X_{max}, Y_{min},Y_{max}, Xg_{min}, Xg_{max}, Yg_{min}, Yg_{max}.

$F(x)$ функциясы ішпрограмма-функцияда сипатталады.

7. Анықталу облысын зерттеп және координаталық осьтердің орналасуын таңдап, алдыңғы есептің процедурасын қолдану арқылы экранда келесі функция графиктерін салу программасын құрыңдар:

а) $y = \frac{1}{x+1}$; б) $y = \frac{x+3}{x-2}$; в) $y = 1 + \frac{2}{x} + \frac{3}{x^2}$

2.15. СИМВОЛДЫҚ ЖОЛДАР

Құрылымдық сандарға қатысты мәліметтер типін қарастырайық. Ол — жолдық тип (жол). Жолдық мәліметтер типі Турбо Паскальда бар, ал стандартты Паскальда жоқ екенін айта кету керек.

Жол — бұл символдар тізбегі. Әрбір символ 1 байт орын алады (код ASCII). Жолдағы символдар саны оның ұзындығын береді. Жол ұзындығы 0-ден 255 аралығында орналасады. Жолдық шамалар тұрақтылар және айнымалылар бола алады.

Жолдық тұрақты — бұл апострофқа алынған символдар тізбегі. Мысалы:

```
'ПАСКАЛЬ программалау тілі'
'IBM PC — computer',
'33-45-12'.
```

Жолдық айнымалы — айнымалаларды сипаттау бөлімінде келесі түрде сипатталады:

```
Var <идентификатор>:String[<жолдың максималды ұзындығы>].
```

Мысалы:

```
Var Name : String[20].
```

Жолдың ұзындығы сипаттауда көрсетілмеуі мүмкін. Бұл жағдайда ол максималды түрде 255 символды қамтиды деген сөз. Мысалы:

```
Var Slovo : String.
```

Жолдық айнымалы сипаттамада көрсетілген ұзындыққа қарағанда жадыда 1 байттан көп орын алады. Өйткені бір (нөлдік) байт *ағымдық жол ұзындығының* мәнін қамтиды. Егер жолдық айнымалыға ешқандай мән меншіктелмесе, оның ағымдағы ұзындығы нөлге тең.

Жолды символдармен толтырғанда, оның ағымдағы ұзындығы ұлғаяды, бірақ ол максималды сипатталған мәннен аспауы керек.

Жолдың ішіндегі символдар 1-ден бастап индекстеледі (нөмірленеді). Әрбір жеке символ тік жақшаның ішіне алынған жолдың атауымен нөмірленеді. Мысалы:

```
Name[5], Name[i], Slovo[k+1].
```

Индекс оң тұрақты, айнымалы және бүтін типті өрнек болуы мүмкін. Индекстің мәні сипаттама шегінен шығып кетпеуі қажет.

String типі мен стандартты Char типі сәйкес болып келеді. Жолдар мен символдар бір өрнекте қолданыла береді.

Жолдық өрнектер жолдық тұрақтылардан, айнымалылардан, функциялардан және амал таңбаларынан жасалады. Жолдық мәліметтерде тіркеу мен қатынастар амалдарына рұқсат етіледі.

Тіркеу амалдары (+) бірнеше жолдарды бір нәтиже шығаратын жолға біріктіру үшін қолданылады. Жолдық тұрақтыларды да, айнымалыларды да тіркеуге болады. Мысалы:

```
'ЭВМ'+' IBM'+' PC'.
```

Нәтижесінде мынадай жол алынады:

```
'ЭВМ IBM PC'.
```

Нәтиже алынатын жолдың ұзындығы 255 символдан асып кетпеуі керек.

Қатынас амалдары (=, <, >, <=, >=, <>) екі жолды салыстырады, нәтижесінде логикалық шама алынады. (True немесе False). Қатынас амалдары тіркеу амалдарына қарағанда төменірек басымдыққа ие. Жолды салыстыру бірінші сәйкес келмейтін символға дейін солдан оңға қарай орындалады. Егер символдардың кодталу кестесіндегі бірінші сәйкес емес символ нөмірі үлкен болса, онда сол жол үлкен болып саналады.

Егер жолдар әртүрлі ұзындықтарға ие болса, бірақ ортақ бөлікте символдар сәйкес келсе, қысқарақ жол ұзын жолдан кішірек деп есептеледі. Егер жолдар ұзындықтары бірдей болса және бірдей символдарды қамтыса, онда олар өзара тең болады. Мысалы:

Өрнек	Нәтиже
'cosml' < cosm2'	True
'pascal' > 'PASCAL'	True
'Кілт ' <> 'Кілт'	True
'MS DOS' = 'MS DOS'	True

- Copy(S, Poz, N) функциясы — жолдан *N* символды ұзындықтағы *S* ішкі жолын Poz позициясынан бастап шығарады. Мұндағы, *N* және Poz — бүтінсанды өрнектер. Мысалы:

S мәні	Өрнек	Нәтиже
'ABCDEFG'	Copy(S, 2, 3)	'BCD'
'ABCDEFG'	Copy(S, 4, 4)	'DEFG'

- Concat (S1, S2, ..., *SN)* функциясы — *S*1,..., *SN* жолдарын бір жолға тіркеуді орындайды (кон катенация) Мысалы:

Өрнек	Нәтиже
Concat('AA', 'XX', 'Y')	'AAXXY'

- Length(S) функциясы — S жолының ағымдағы ұзындығын анықтайды. Мысалы:

S мәні	Өрнек	Нәтиже
'test-5'	Length(S)	6
'(A+B)*C'	Length(S)	7

- Pos (S1, S2) функциясы - S1 ішкі жолының S2 жолында бірінші пайда болуын анықтайды. Нәтижесінде S1 ішкі жолының бірінші символы орналасқан позиция нөміріне тең бүтін сан алынады. Егер S2-де S1 ішкі жолы табылмаса, нәтиже 0-ге тең болады. Мысалы:

*S*2 мәні	Өрнек		Нәтиже
'abcdef'	Pos('cd'	S2)	3
'abcdcdef'	Pos('cd'	S2)	3
'abcdef'	Pos('k',	S2)	0

- Delete(S, Poz, N) процедурасы — Poz позициясынан бастап *S* жолынан *N* символдарды жояды. Мысалы:

S бастапқы мәні	Оператор	Соңғы мән
'abcdefg'	Delete(S,3,2)	'abefg'
'abcdefg'	Delete(S,2,6)	'a'

Процедураның орындалу нәтижесінде ағымдағы айнымалыдағы *S* жолдың ұзындығы қысқарады.

- Insert(S1, S2, Poz) процедурасы — S2 жолына S1 жолын Poz позициясынан бастап орналастырады. Мысалы:

Бастапқы S2	Оператор	Соңғы S2
'ЭВМ PC'	Insert('IBM-',S2,5)	'ЭВМ IBM-PC'
'2-сур.'	Insert('N',S2,6)	' 2 -N сур.'

2.13-Мысал. «ВЕЛИЧИНА» сөзінен «НАЛИЧИЕ» сөзін алу программасы:

```
Program Slovo_1;
Var Sl1, Sl2 : String[10];
Begin
   sl1 := 'ВЕЛИЧИНА';
   sl2 := Copy(Sl1,7,2) + Copy(Sl1,3,4) + Sl1[2];
   WriteLn(Sl2)
End.
```

2.14-Мысал. «СТРОКА» сөзінен «СЕТКА» сөзін алу программасы:

```
Program Slovo_2;
Var Sl : String[10];
Begin
  Sl := 'СТРОКА';
  Delete(Sl,3,2);
  Insert('E',Sl,2);
  WriteLn(Sl)
End.
```

2.15-Мысал. N жұлдызшалардан тұратын символдық жол жасау программасын құру (мұндағы, N — бүтін сан, $1 < N < 255$):

```
Program Stars;
Var A : String;
  N, I : Byte;
Begin
  Write ('Жұлдызшалар санын енгізу');
  ReadLn(N);
  A : = '';
  For I := 1 To N Do
       A : = A + ' *';
  WriteLn(A)
End.
```

Мұнда A айнымалысына алдымен бос жолдың (' ') мәні меншіктеледі, содан кейін оларға жұлдызшалар қосылады.

2.16-Мысал. «!» символына дейінгі символдық жолда сандарды есептеу программасы:

```
Program C;
Var S : String;
     K, I : Byte;
Begin
     WriteLn('Жолды енгіз');
     ReadLn(S);
     K := 0;
     I : = 1;
     While (I <= Length(S)) And (S[I] <> '!') Do
     Begin
        If (S[I] >= '0') And (S[i] <= '9')
        Then       K := K + 1;
        I := I + 1
     End;
     WriteLn (' "!" символына дейінгі сандар тең ', K)
End.
```

Бұл программада айнымалы *K* сандарды санауыш рөлін атқарады, ал айнымалы I - цикл параметрінің рөлін атқарады. Циклдің орындалуы жолда «!» символы шыққан кезде немесе жолдың соңына келгенде (егер жолда мұндай символ болмаса) тоқтатылады. Егер '0' < S[I] < '9' қатынасы ақиқат болса, онда S [I] символы сан болады.

2.17-Мысал. Символдар жолы берілген: 'A © B ='. Мұндағы, A және B ондық сандар; © белгісі амалдар (+, -, *) белгілерінің бірін білдіреді. Басқаша айтқанда, екі бірмәнді сандармен және теңдік белгісі бар арифметикалық өрнек берілген, мысалы: '5 + 7 ='. Машина осы өрнекті есептеп, «=» белгісінен кейін нәтижені шығару қажет. Бүтін сандармен ғана жұмыс жасау үшін бөлу амалы қарастырылмайды.

Мұндай есепті шешу программасы *интерпретатор* деп аталады. Интерпретатор жолдың мазмұнын ашып, сәйкес арифметикалық амалды орындауы керек. Бастапқы символдық ақпараттан ол сандық ақпаратпен жұмыс істеуге ауысуы керек.

Егер жол көрсетілген форматқа сәйкес тек төрт символдан құралған деп есептесек, онда оны шешу программасы оңай болады:

```
Program Interpretator;
Var Str : String[4];
  A, B : 0..9;
  C : -100..100;
Begin
{----Сандық символдарды цифрларға айналдыру----   }
  A := Ord(Str[1]) - Ord('0');
  B := Ord(Str[3]) - Ord('0');
{---Арифметикалық амалдарды орындау---}
Case Str[2] of
'+' : C:=A+B;
'-' :  C:=A-B;
'*' : C:=A*B;
  End;
{ ---Нәтиже шығару ----- }
  WriteLn(C:2)
End.
```

Бұл программада Str [2] символдық шама таңдау операторындағы селектордың рөлін атқарады. Егер ол «+»-ке тең болса, операторы орындалады: C = A + B; егер «-»-ке тең болса, C: = A - B операторы орындалады; егер «*»-ке тең болса, C:=A*B операторы орындалады. Str [2] кез-келген басқа мәндері үшін, меншіктеу операторларының ешқайсысы орындалмайды және C айнымалысының мәні анықталмай қалады.

Бұл программада Case операторының толық емес формасы жазылған, яғни Else тармағынсыз қолданылып тұр.

Қате символ алынғандығы туралы хабарлама шығаратын Str[2]-гі программа келесі түрде болады:

```
Case Str[2] Of
   '+' : C := A + B;
   '-' : C := A - B;
   '-' : C := A * B
Else WriteLn('Амал таңбасы қате!')
End;
```

Interpretator программасы қалай орындалатынын талқылайық.

Жолды енгізген соң сандық символдар сәйкес ондық сандарға ауысады. Одан кейін амал таңбасы интерпретацияланады. Осы интерпретацияға байланысты үш арифметикалық амалдың біреуі орындалады. Ары қарай нәтиже «=» белгісінен кейін экранға шығады.

ЖАТТЫҒУЛАР

1. Тіркеу амалын және Copy функциясын пайдаланып «дисковод» сөзінен «воск» сөзін алу программасын құрыңдар.
2. Delete және Insert процедураларын пайдаланып «операция» сөзінен «правило» сөзін алу программасын құрыңдар.
3. Берілген сөзде бірінші және соңғы символды «*» символына ауыстыру программасын құрыңдар.
4. Берілген сөздегі бірінші символ мен соңғы символдың орындарын ауыстыру программасын құрыңдар.
5. Берілген сөзде қанша символ бар, сонша «!» белгісін тіркеу программасын құрыңдар (мысалы, «АЛАҚАЙ» жолынан «АЛАҚАЙ!!!!!!» сөзін алу).
6. Берілген жолдағы әрбір символдан кейін бос орын қою программасын құрыңдар.
7. Енгізілген сөздің әрбір әрібін екі рет жазу программасын құрыңдар.
8. Енгізілген жолды кері жазу программасын құрыңдар (Мысалы, «ДИСК» сөзін «КСИД» деп жазу).
9. Берілген жолдағы барлық бос орындарды алып тастау программасын құру керек.
10. Бүтін санды жазу барысында жолды сәйкес бүтін типті шамаға ауыстыру программасын құрастыру

2.16. ЖИЫМДАР

Күнделікті және ғылыми тәжірибеде кесте түрінде берілген ақпараттармен жиі кездесеміз.

2.7-кесте. Температуралардың сызықты кестесі												
Жыл айлары	1	2	3	4	5	6	7	8	9	10	11	12
Темпе-ратура, °С	-21	-18	-7,5	5,6	10	18	22,2	24	17	5,4	-7	-18

Белгілі бір жылға арналған орташа айлық температура мәндері туралы ақпарат 2.7-кестеде көрсетілген. Реттелген сандардың тізбегі қамтылған мұндай кесте *сызықтық* деп аталады. Егер осы деректерге математикалық өңдеудің қандай да бір түрі талап етілсе, онда әдетте оларды белгілеу үшін индексті символика енгізіледі. Мысалы, T_1 - қаңтар айының температурасы (бірінші ай), T_5 – мамыр және т.с.с. Тұтас алғанда, кестеде келтірілген мәндер жиынтығы келесідей белгіленеді:

$$\{Ti\}, i = 1, \dots, 12.$$

Элементтердің реттік нөмірлері (1, 15, і және т.б.) *индекстер* деп аталады. Индекстелген шамалар математикалық өңдеу үшін және оларды жазу кезінде қолдануға ыңғайлы. Мысалы, орташа жылдық температураны есептеу формуласы мынадай:

$$T_{cp} = \frac{1}{12}\sum_{i=1}^{12} T_i$$

Енді, мысалға, 10 жыл ішінде, 1981 жылдан 1990 жылға дейінгі аралықта орташа айлық температура туралы ақпаратты жинау қажет екендігін елестетіп көрейік. Ол үшін бағандар - жылдарға және жолдар - айларға сәйкес келетін *тіктөртбұрышты кестені* (2.8-кесте) пайдаланамыз.

2.8-кесте. Температуралардың тіктөртбұрышты кестесі												
Жыл	Жыл айлары											
	1	2	3	4	5	6	7	8	9	10	11	12
1981	-23	-17	-8,4	6,5	14	18,6	25	19	12,3	5,6	-4,5	-19
1982	-16	-8,5	-3,2	7,1	8,4	13,8	28,5	21	6,5	2	-13	-20
1983	-9,8	-14	-9,2	4,6	15,6	21	17,8	20	11,2	8,1	-16	-21
:	:	:	:	:	:	:	:	:	:	:	:	:
1990	-25	-9,7	-3,8	8,5	13,9	17,8	23,5	17,5	10	5,7	-14	-20

Дұрыс таңдалған мәліметтердің жазылу формасы қажетті ақпарат алудың ыңғайлылығы мен жылдамдығын қамтамасыз етеді. Мысалы, 2.8-кестеден көрсетілген онжылдықта ең суық қаңтар қай жылы болғанын немесе қай айда ең тұрақсыз температуралық режим анықталғанын оңай байқауға болады.

Осындай кестеде сақталған мәндер үшін екі индекстік белгілеуді қолданған ыңғайлы. Мысалы, H_{19812} - 1981 жылғы ақпандағы температура және т.с.с. Бұл жағдайда кестені құрайтын мәліметтердің толық жиынтығы математикалық түрде келесідей болуы мүмкін:

$$\{H_{i,j}\};\ I = 1981\ \ldots\ 1990;\ j = 1,\ \ldots,\ 12.$$

Мұндай мәліметтерді өңдеу екі индексті шамаларды қолдану арқылы жүзеге асырылады. Мысалы: 10 жыл ішіндегі наурыз айының орташа температурасы,

$$H_{\text{ср.март}} = \frac{1}{10}\sum_{i-19}^{1990} H_{i,3}$$

ал барлық 10 жыл ішіндегі орташа температура:

$$H_{\text{ср.март}} = \frac{1}{10}\sum_{i-1981}^{1990}\left[\frac{1}{12}\sum_{j-1}^{12} H_{i.j}\right] = \frac{1}{120}\sum_{i-198}^{1990}\sum_{j-1}^{12} H_{i,j}.$$

Паскальдағы кестелердің баламасы – *жүйелі тип* немесе *жиым* деп аталатын мәліметтердің құрылымдық типі. Кесте сияқты, жиым ортақ атауы бар нөмірленген біртипті мәндердің жиынтығы болып табылады. Жиым элементтері индекстелген айнымалылар ретінде белгіленеді. Индекстер жиым атауынан кейін тік жақшаларда жазылады. Мысалы:

```
T[1], T[5], T[i], H[1981, 9], H[i, j]
```

Сызықтық кестеден тұратын жиым *бірөлшемді* деп аталады, ал тіктөртбұрышты кестелі жиым - *екіөлшемді* деп аталады. Программаларда өлшемдері одан да көбірек жиымдар пайдалануы мүмкін.

Жиым элементтерінің типі оның *негізгі типі* болып табылады. Әрине, қарастырылған температура жиымдарының негізгі типі - бұл нақты тип (Real).

Жиымдарды сипаттау. Жүйелі типтегі айнымалы сипаттау бөлімінде келесі түрде жазылады:

```
Var <идентификатор> : Array [<индекс типі>] Of <тип
компонент>
```

Көбінесе индекстер интервалдық типте қолданылады. Мысалы, жоғарыда келтірілген орташа айлық температурасының бірөлшемді жиымы мына түрде сипатталады:

```
Var T : Array [1..12] Of Real;
```

Жиымның сипаттамасы оның жадыда орналасуын және программада одан әрі пайдалану ережелерін анықтайды. Жиымның кезекті элементтері кезекті жад ұяшықтарында орналасады (T [1], T [2] және т.б.) және индекс мәндері 1 ... 12 аралығынан аспауы керек. Индекс ретінде сәйкес типтегі кез-келген өрнек қолданылуы мүмкін. Мысалы, T [i + j], T [m div 2].

Индекс типі Integer-ден басқа кез-келген скалярлы реттік тип болуы мүмкін. Мысалы, программа келесі сипаттамаларды қамтуы мүмкін:

```
Var Cod : Array[Char] Of 1..100;
    L : Array[Boolean] Of Char;
```

Берілген программада жиым элементтерінің келесі белгілеулері рұқсат етілген:

```
Cod['x']; L[True]; Cod[Chr(65)]; L[a > 0].
```

Кейбір жағдайларда, саналатын мәліметтер типін индекс ретінде пайдалану ыңғайлы. Мысалы, бір мектептегі төрт оныншы сынып оқушылары туралы мәліметтер келесі жиымда сақталуы мүмкін:

```
Type Index = (A, B, C, D);
Var Class 10 : Array[Index] Of Byte;
```

Егер, мысалы, Class_10 [A] элементі 35-ке тең болса, онда 10 «А» сыныбында 35 адам бар екенін білдіреді. Мұндай индекстеу программаны көрнекті қылып көрсетеді.

Көбінесе құрылымдық мәліметтер типіне сипаттау бөлімінде атау меншіктеледі, содан кейін ол айнымалыларды сипаттау бөлімінде пайдаланылады:

```
Type Mas1 = Array [1..100] Of Integer;
       Mas2 = Array [-10..10] Of Char;
Var Num : Mas1; Sim : Mas2;
```

Бірөлшемді жиымдарда элементтер типтері скалярлы болатынын айтып келеміз.

Көпөлшемді жиым - Паскальда элемент типтері жиым болып табылатын бірөлшемді жиым сияқты сипатталады (жиымдардың жиымы). Мысалы, жоғарыда келтірілген тіктөртбұрышты кестені мынадай жиымда сақтауға болады:

```
Var H : Array[1981..1990] Of Array[1..12] Of Real;
```

Бұл жиымның кейбір элементтерінің белгілеулеріне мысал келтірейік:

```
H[1981][1]; H[1985][10]; H[1990][12].
```

Екіөлшемді жиым элементтерін белгілеудің эквивалентті формасы жиі қолданылады:

```
H[1981, 1]; H[1985, 10]; H[1990, 12].
```

Мұндағы, H[1981] айнымалысы кестенің барлық бірінші жолын білдіреді, яғни 1981ж. барлық температуралар жиымы.

Келтірілген екіөлшемді жиымды сипаттаудың эквивалентті нұсқасы:

```
Type Month = Array [1..12] Of Real;
       Year = Array [1981..1990] Of Month;
Var H : Year;
```

Бұл жиымның қысқартылған сипаттау түрі мынадай:

```
Var H : Array[1981..1990, 1..12] Of Real;
```

Үшөлшемді жиымды элементтері екіөлшемді жиым болып табылатын бірөлшемді жиым ретінде анықтауға болады. Мысалы:

```
Var A : Array[1..10, 1..20, 1..30] Of Integer;
```

Бұл 10 • 20 • 30 = 6000 бүтін сандардан тұратын және 6000• 2 = 12,000 байт жадты қамтитын жиым сипаттамасы. Паскальда жиымның өлшеміне жоғарғы шек жоқ. Дегенмен, Паскальдың әрбір нақты жүзеге асырылуында, жиымдарға бөлінген жады көлемі шектелген. Турбо Паскальда бұл шектеу 64 Кб құрайды.

Математикаға ұқсас, бірөлшемді сандық жиымдар көбінесе *векторлар* деп аталады, ал екі өлшемді жиымдар *матрицалар* деп аталады.

Паскаль динамикалық жиымдарды, яғни өлшемі программаны орындау барысында анықталатын жиымдарды пайдалануға жол бермейді. Жиымның өлшемін өзгерту программа мәтінін өзгерту мен қайта компиляция жасау арқылы жүзеге асады. Осындай өзгерістерді жеңілдету үшін тұрақтылар бөліміндегі индекс параметрлері анықталды:

```
Const Imax = 10; Jmax = 20;
Var Mas : Array[1..Imax, 1..Jmax] Of Integer;
```

Енді Mas жиымының өлшемін және осы өлшемдермен байланысқан барлық программа операторларын өзгерту үшін программада бір жолды – тұрақты бөліміне өзгертулер енгізу ғана жеткілікті.

Жиымдардағы біртұтас әрекеттер. Мұндай әрекеттер келесі жағдайларда ғана рұқсат етіледі:

- бір жиымның мәндерін екінші жиымға меншіктеу;
- «тең» және «тең емес» қатынас амалдарында.

Екі жағдайда да жиымдар бірдей типте болуы керек (индекс типтері мен элемент типтері). Мысалы:

```
Var P, Q : Array[1..5,1..10] Of Real;
P := Q
```

меншіктеу амалын орындау барысында *P* жиымының барлық элементтері Q жиымының сәйкес элементтеріне тең болады.

Жоғарыда айтылғандай, көпөлшемді жиымдарда индексі бар айнымалы бүтін жиымды білдіре алады. Мысалы, егер *H*-кестедегі 1989 жылғы деректерді 1981 жылмен бірдей жасау керек болса (тоғызыншы жолға бірінші жолдың мәні меншіктелсе), онда бұл келесідей болуы мүмкін:

```
H[1989]:= H[1981].
```

Бұл кестедегі жолдардың мәндерін ауыстыру сол типтегі үшінші айнымалының көмегімен жүзеге асырылады:

```
P := H[1989]; H[1989]:= H[1981]; H[1981]:= P;
```

Мұндағы, *P* келесі түрде сипатталған:

```
Var P : Array [1..12] Of Real;
```

Программада жиымдарды өңдеу компонентті түрде жасалады. Жиымдарға мәндерді енгізу мысалын келтірейік:

```
For I := 1 To 12 Do
     ReadLn(T[I]);
For I := 1 To IMax Do
For J := 1 To JMax Do
      ReadLn(Mas[I, J]);
```

Мұнда әрбір келесі мән жаңа жолдан енгізіледі. Әр жолға енгізу үшін Read операторы қолданылады.

Циклде индексті айнымалы бойынша жиым мәнін шығару ұйымдастырылады. Мысалы:

```
For I := 1 To 12 Do Write(T[I]:8:4);
```

Келесі программа фрагменті матрицаның экранға әр жолда шығуын көрсетеді:

```
For I := 1 To IMax Do
Begin
   For J : = 1 To JMax Do
        Write(Mas[I,J]:6);
   WriteLn
End;
```

Матрицаның келесі жолын басып шығарғаннан кейін, параметрсіз WriteLn операторы меңзерді жаңа жолдың басына жылжытады. Соңғы мысалда экрандағы матрица JMax 12-ден аспаса, (себебін өз беттеріңмен түсіндіріңдер) тіктөртбұрышты кестенің сол қалпы түрінде алынады.

Жиымдарды өңдеудің бірнеше программа мысалдарын көрсетейік.

2.18-Мысал. Келтірілген орташа айлық температура жиымының Г[1 ...12] орташа жылдық температура мен осы мәннен айлық ауытқуларды анықтау керек.

Берілген есептің программасы:

```
Program Example;
Cont N = 12;
```

```
Type Vec = Array [1..N] Of Real;
Var T, Dt : Vec;
      St : Real;
      I : Integer;
Begin { ---Бастапқы берілгендерді енгізу-----}
      WriteLn('Температура кестесін енгіз');
      For I := 1 To N Do
      Begin
           Write(I : 2, ':');
           ReadLn(T[I])
      End;
{ -- Орташа температураны есептеу----- }
      St := 0;
      For I : = 1 To N Do
           St := St + T[I];
           St := St/N;
{ -- Кестенің орташа мәннен ауытқуын есептеу----- }
      For I : = 1 To N Do
           Dt[I] := St - T[I];
{ -- Нәтижелерді шығару-- }
      WriteLn('Орташа температура тең ', St:6:2);
      WriteLn;
      WriteLn('Орташа температурадан ауытқу:');
      For I : = 1 To N Do
      WriteLn(I:1, ':  ', Dt[I]:6:2)
End.
```

Бұл программаны пайдалана отырып, кез-келген бірөлшемдегі нақты жиым үшін орташа мән мен одан ауытқу векторын есептеуге болады. Жиым өлшемін тұрақтылар бөлімін түзету арқылы ғана реттеуге болады.

2.19-Мысал. Келтірілген температура жиымынан максималды элементті таңдау қажет, яғни ең жоғарғы температура мен оған сәйкес келетін ай нөмірін таңдау керек.

Бұл есепті шешуге арналған алгоритм идеясы келесідей: нақты айнымалы TMax мәнінде максималды температураны алу үшін, алдымен оған Г [1] жиымының бірінші мәні енгізіледі. Содан кейін, TMax мәні кезекпен температура жиымының қалған элементтерімен салыстырылады және TMax-тан үлкен әрбір мән осы айнымалыға меншіктелінеді. Ең жылы айдың нөмірін алу үшін NumMax бүтін айнымалысына температура жиымының элемент нөмірін TMax-ке оның мәнін енгізген сайын жіберіп отыру керек.

Берілген есептің программасы:

```
TMax := T[1];
NumMax := 1;
For I : = 2 To 12 Do
```

```
  If T[I] > Tmax
  Then
  Begin
       TMax := T[I];
       NumMax := I
End;
```

Егер температура жиымында максималды бірнеше мән болса, онда NumMax-қа осы элементтердің бірінші нөмірі алынады. Соңғы мерзімді алу үшін If операторындағы «>» қатынас амалын «>=» амалына өзгерту керек.

2.20-Мысал. Жиымды сұрыптау қажет, яғни *N* элементті бірөлшемді X жиымында, мәндерді олардың өсу ретімен орналастыру қажет: $X_1 \le X_2 \le ... \le X_N$.

Алгоритмдерді сұрыптаудың бірнеше әдістері бар. *Көпіршікті әдіс* деп аталатын алгоритмді сипаттайық.

Жиым элементтерінің іргелес жұптарын жүйелі түрде орындау жүзеге асады: X_j және X_2, X_2 және X_3, ..., X_{N-1} және X_N. Нәтижесінде максималды мән X_N-ге ауысады. Сосын X_1 және X_2 элементтерінің тізбегі қалғанша, X_{N-1} және т.б. алғанға дейін сол процедура қайталанады. Бұл алгоритм екі қабаттасқан цикл құрылымына ие, ал ішкі цикл айнымалы (азаятын) ұзындықта болады.

Берілген есептің программасы:

```
For I := 1 To N _1 Do
For K := 1 To N _I Do
  If X [K] > X [K + 1]
  Then
  Begin
       A := X[K];
       X[K] := X[K + 1];
       X[K + 1] := A
  End;
```

2.21-Мысал. Бұрын сипатталған 10 жыл ішіндегі орташа айлық температураның екіөлшемді жиымынан ең жылы жаздың қай жылы болғандығын анықтау керек, яғни жаз айларындағы ең үлкен орташа температурасы болған жыл.

Есепті шешу алгоритмі идеясы келесідей: біріншіден, *S* векторына 10 жыл бойғы жаз айларының орташа температурасын аламыз, содан кейін осы вектордың ең үлкен элементінің нөмірін табамыз. Бұл ізделініп отырған жыл болады.

Есептің программасы мынадай:

```
Program Example_2;
Type Month = Array[1..12] Of Real;
     Year = Array[1981..1990] Of Month;
```

```
Var H: Year;
    S: Array[1981..1990] Of Real;
    I, J, K: Integer;
Begin { ---Пернетақтадан берілгендерді енгізу--}
  For I := 1981 To 1990 Do
  For J := 1 To 12 Do
  Begin
      Write(J : 2, '.', I : 4,':       ');
      ReadLn(H[I, J])
  End;
{ --Жазғы орташа температураларының векторын есептеу--}
  For I := 1981 To 1990 Do
  Begin
      S[I] := 0;
      For J:= 6 To 8 Do
          S[I] := S[I] + H[I, J];
          S[I] := S[I]/3
  End;
{ --Ең жылы жазғы жылды анықтау ------------- }
  K := 1981;
  For I := 1982 To 1990 Do
      If S[I] > S[K] Then K := I;
  WriteLn('Ең жылы жаз', K-ші жылы болды')
End.
```

ЖАТТЫҒУЛАР

8. 10 х 15 өлшемді нақты матрицасының барлық жолдарының элементтер мәндерін кему ретімен орналастыру.

9. 5 х 5 өлшемді бүтінсанды матрицаны транспонирлеу, яғни негізгі диагональға қатысты жолдар мен бағандардың орындарын ауыстыру.

10. 10 х 10 екіөлшемді матрицасында сәйкес келетін жолдарды анықтау.

2.17. РЕКУРСИВТІ ІШКІ ПРОГРАММАЛАР

Рекурсия - көмекші алгоритмді, яғни ішкі программаны ұйымдастыру тәсілі, бұл ішкі программа (процедура немесе функция) орындалу барысында өзін-өзі шақырады.

Рекурсивті объект - бұл ішінара анықталған кез-келген объект.

Мысалы, келесі екілік кодты анықтау рекурсивті болып табылады:

```
<екілік код> ::= <екілік сан> | < екілік код > < екілік
сан >
< екілік сан > ::= 0|1
```

Рекурсивті анықтамада міндетті түрде шектеу болуы тиіс, яғни *шекаралық шарт*, одан шығып кеткенде рекурсивті шақырулардың инициациясы тоқтатылады.

Кейбір математикалық функциялардың рекурсивті анықтамаларына мысалдар келтірейік.

Рекурсияның классикалық мысалына факториалды анықтау сәйкес келеді. Бір жағынан факториал: n! = 1 • 2 • 3 •... • n түрінде анықталса, екінші жағынан оны келесі түрде анықтауға болады:

$$n! = \begin{cases} 1, & \text{егер } n \le 1; \\ (n-1)!\, n, & \text{егер } n > 1. \end{cases}$$

Мұнда шекаралық шарт $n \le 1$ болып табылады. Берілген n натурал санының мөлшерін қайтаратын *K(n) функциясын рекурсивті түрде анықтау* келесі түрде болады:

$$K(n) = \begin{cases} 1, & \text{егер } n < 10; \\ K(n/10) + 1, & \text{егер } n \ge 10 \end{cases}$$

Берілген натурал санның цифрларының қосындысын есептейтін *S(n)* функциясын рекурсивті анықтауды өз беттеріңше орындаңдар.

$0 < m < n$ бойынша C(m, n) функциясын рекурсивті анықтау үшін биномды коэффициентті есептеу келесі түрде болады:

$$C\,0 — C^{n} — 1^{*}\, c^{m} — C^{m} + c^{m}\text{-}1$$
$$^{C}n \sim {}^{C}n \sim {}^{1}\,{}^{C}n \sim {}^{C}n\text{-}1\ {}^{C}\text{n-1}\ \bullet$$

Рекурсивті ішпрограмманы шақыру кез-келген басқа ішпрограмманы шақырумен бірдей. Әрбір жаңа рекурсивтік шақыруда ішпрограмманың жаңа көшірмесі барлық жергілікті айнымалылармен жадыда жасалады. Мұндай көшірмелер шекаралық шартқа жеткенге дейін құрылады. Шекаралық шарт болмаған жағдайда мұндай көшірмелердің шексіз көбеюі программаның төтенше аяқталуына, яғни, стектің толтырылуына әкеп соғады.

Шекаралық шартқа жеткен рекурсивті ішпрограмманың барлық жаңа көшірмелері *рекурсивтің түсуі* деп аталады. Компьютердің жадында бір уақытта болатын рекурсивті ішпрограмманың көшірмелерінің ең көп саны *рекурсияның тереңдігі* деп аталады. Ең бірінші рекурсивтік шақырылған ішпрограммадан бастап рекурсиялық ішпрограммалардың жұмысының аяқталуы *рекурсивтік көтерілу* деп аталады.

Рекурсиялық ішпрограммалардағы әрекеттерді орындау келесі жолдардың бірімен жүзеге асырылуы мүмкін:

Рекурсивті көтерілу	Рекурсивті түсу	Рекурсивті көтерілу және рекурсивті түсу

Begin
P;
операторлар;
End;
	Begin
	операторлар;
	P
	End;
		Begin
		операторлар;
		P;
		операторлар
		End;

Мұндағы P — рекурсивті ішпрограмма.

Рекурсивті ішпрограммадағы әрекеттер рекурсивті сатылардың бірінде немесе екеуінде бірден орындалуы мүмкін. Іс-әрекеттерді ұйымдастырудың тәсілі алгоритмнің логикасы арқылы дамиды.

Жоғарыда берілген рекурсивтік анықтамаларды Паскальдағы функциялар мен процедуралар түрінде жүзеге асырамыз.

2.22-Мысал. Факториалды рекурсивті анықтаудың программада жүзеге асырылуы:

```
{Паскальдағы функция}
Function Factrial (N : Integer) : Extended;
Begin
   If N <= 1
   Then Factorial := 1
   Else Factrial := Factorial(N - 1) * N
End;

{Паскальдағы процедура}
Procedure Factorial (N : Integer; Var F : Extended);
Begin
   If N <= 1
   Then F : = 1 Else Begin
                Factorial(N - 1, F);
                F := F * N
           End
End;
```

2.23-Мысал. *N* берілген натурал сандағы цифрлардың өлшемін қайтаратын *K(N)* функциясын рекурсивті анықтаудың программада жүзеге асырылуы:

```
{Паскальдағы функция}
Function K(N : Longint) : Byte;
Begin
   If N < 10
   Then K : = 1
   Else K := K(N Div 10) + 1
End;

{Паскальдағы процедура}
Procedure K(N : Longint; Var Kol : Byte)
Begin
   If N < 10
   Then Kol := 1 Else Begin
                K(N Div 10, Kol);
                Kol := Kol + 1
           End
End;
```

2.24-Мысал. Биномды коэффициенттерді рекурсивті есептеудің программада жүзеге асырылуы:

```
{Паскальдағы функция}
Function C(m, n : Byte) : Longint;
Begin
   If (m = 0) Or (m = n)
   Then C := 1
   Else C := C(m, n - 1) + C(m - 1, n - 1)
End;
```

```
{ Паскальдағы процедура}
Procedure C(m, n : Byte; Var R : Longint);
Var R1, R2: Longint;
Begin
   If (m = 0) Or (m = n)
   Then R : = 1 Else Begin
                C(m,n - 1, R1);
                C(m - 1, n - 1, R2);
                R : = R1 + R2
             End
End;
```

2.25-Мысал. Сызықтық жиым элементтерінің қосындысын есептеу.

Бұл есепті шешу үшін келесі тұжырымды қолданамыз: егер элементтер саны 0-ге тең болса, қосынды да 0-ге тең. Егер элементтер қосындысы 0-ге тең болмаса, барлық алдыңғы элементтердің қосындысына соңғы элементті қосу.

Бұл есептің программасы мынадай:

```
{Паскальдағы программа}
Program Rec2;
Type LinMas = Array[1..100] Of Integer;
Var A : LinMas;
      I,    N : Byte;

{Рекурсивті функция}
Function Summa(N : Byte; A : LinMas): Integer;
Begin
      If N = 0
      Then Summa : = 0
      Else Summa := A[N] + Summa(N - 1, A)
End;

{Негізгі программа}
Begin
       Write('Жиым элементтерінің саны? ');
       ReadLn(N);
       Randomize;
       For I := 1 To N Do Begin
              A[I] := -10 + Random(21);
              Write(A[I] : 4)
       End;
       WriteLn;
       WriteLn('Қосынды:     ', Summa(N, A))
End.
```

2.26-Мысал. Берілген жол палиндром болып табыла ма, соны анықтау, яғни солдан оңға қарай және оңнан солға қарай оқылуы бірдей болатын жол. Шекаралық шарт: егер жол бос болса немесе бір

символдан тұрса, ол – палиндром болып табылады.

Бұл есепті шешу идеясы жолды солдан оңға қарай және оңнан солға қарай қарап шығу мен сәйкес символдарды салыстыруды бір уақытта орындау болып табылады. Егер белгілі бір уақытта символдар сәйкес келмесе, онда жол палиндром болып табылмайтындығы жөнінде хабарлама шығарылады, ал егер жолдың ортасына жеткен уақытта сәйкес символдар табылса, онда жол палиндром болады.

```
Program Palindrom;
{Рекурсивті функция}
Function Pal(S : String): Boolean;
Begin
   If Length(S) <= 1
   Then Pal := True
   Else Pal := (S[1] = S[Length(S)]) And
            Pal(Copy(S, 2, Length(S) - 2 ) ) ;
End;
Var S : String;

{Негізгі программа}
Begin
   Write('Жолды енгіз:          ');
   ReadLn(S);
   If Pal(S)
   Then WriteLn('Жол палиндром болып табылады')
   Else WriteLn('Жол палиндром болып табылмайды')
End.
```

Берілген натурал сан палиндром болып табылатындығын өз беттеріңше анықтаңдар.

Қорытындылай келгенде, рекурсияны қолдану әдемі программалау әдісі болып табылады. Сонымен бірге, көптеген тәжірибелік есептерді шешуде, бұл әдіс тиімсіз, себебі ол программаның орындалу уақытын көбейтеді және жиі рекурсивті түсудің ішпрограммасының көшірмелерін сақтауға арналған жадыда көп орынды талап етеді. Сондықтан іс жүзінде рекурсивті алгоритмдерді итеративті алгоритмдермен алмастырған орынды.

ЖАТТЫҒУЛАР

1. Рекурсивті көмекші алгоритмдерді қолданып өздерің есеп құрастырыңдар.
2. Келесі программа не үшін апатты жағдайда тоқтап қалатынын анықтаңдар:

```
Function Stroka : String;
Var C : Char;
```

```
Begin
   Write('Кезекті символды енгіз:          '); ReadLn(C);
   Stroka := Stroka + C
End;
```

Бұл алгоритмнің қай кезеңінде әрекеттер орындалады?

3. x^n шамасын төмендегі формула бойынша есептейтін pow(x, n) рекурсивті функциясын нақты x (x ∈ 0) –тен және n – нен анықтау:

$$x^n = \begin{cases} 1 & \text{егер } n = 0; \\ \dfrac{1}{x^{-n}} & \text{егер } n < 0; \\ x \cdot x^{n-1} & \text{егер } n > \end{cases}$$

4. n және m натурал сандары берілген. Рекурсивті процедураны қолданып ЕҮОБ(n, m) табыңдар. Мұндағы, ЕҮОБ(n,m) = ЕҮОБ(m, r), r — n –ді m-ге бөлгендегі қалдық.

2.18. ЖИЫНДАР

Математиканың іргелі салаларының бірі - жиындар теориясы. Бұл теорияның математикалық аппаратының кейбір аспектілері Паскальда *жиын типті* мәліметтер арқылы (жиындар) жүзеге асырылады.

Жиын - біртұтас қарастырылатын элементтердің бірдей типтерінің жиынтығы. Паскальда тек ақырғы жиындар болуы мүмкін. Турбо Паскальда жиын 0-ден 255-ке дейінгі элементтерді қамтуы мүмкін.

Жиынның жиымнан айырмашылығы, оның элементтері нөмірленбеген және реттелмеген болады. Жиынның әрбір жеке элементі айқындалмайды және онымен жеке ешқандай әрекет орындалмайды. Әрекеттер тек тұтас жиындармен ғана орындалады.

Жиын элементтерінің типі базалық тип деп аталады. Базалық тип Real типті қоспағанда, кез-келген скаляр тип бола алады.

Жиын конструкторы. Жиынның нақты мәндері төртбұрышты жақшаға алынған элементтердің тізімін ұсынатын жиын конструкторы арқылы беріледі. Элементтердің өздері тұрақты немесе базалық типті өрнектер болуы мүмкін.

Конструкторды пайдаланып көрсететін жиындардың кейбір мысалдары:

[3, 4, 7, 9, 12] — бес бүтін сандардан тұратын жиын;
[1..100] —1- ден 100-ге дейінгі бүтін сандар жиыны;
['a','b','c'] — *a, b, c* үш литері бар жиын;
['a'..'z','?','!'] — латынның кіші әріптері және «?» мен «!» белгілері бар жиын.

«[]» символы бос жиынды білдіреді, яғни бұл жиында ешқандай элемент жоқ деген сөз.

Конструктордың ішінде элементтердің жазылу реті маңызды емес. Мысалы, [1, 2, 3] және [3, 2, 1] — бұл эквивалентті жиындар.

Жиынның әрбір элементі тек бір рет есептеледі, сондықтан [1, 2, 3, 4, 2, 3, 4, 5] және [1..5] эквивалентті жиындар болып табылады.

Жиындық типті айнымалылар келесі түрде сипатталады:

Var <идентификатор> : Set Of <базалық тип>.

Мысалы:

```
Var A, D : Set Of Byte;
    B : Set Of 'a'..'z';
    C : Set Of Boolean;
```

Енгізу операторы арқылы мәндерді жиындық айнымалыға енгізе алмаймыз және шығару операторы арқылы оларды шығаруға болмайды. Жиындық айнымалы тек келесі түрдегі меншіктеу операторының орындалу нәтижесінде ғана нақты мәнді ала алады:

<Жиындық айнымалы> := <Жиындық өрнек>

Мысалы:

```
A := [50,100,150,200];
B := ['m','n','k'];
C := [True,False];
D := A;
```

Сонымен қатар, өрнектер жиындардағы амалдардың орындалуын қамтуы мүмкін.

Жиындарға қолданылатын амалдар. Паскальда математикалық жиын теориясының негізгі амалдары жүзеге асырылады: бірігу, қиылысу, айырым. Барлық осы амалдарда операндтар мен нәтижелер бірдей базалық типтегі жиындық шамалар.

A және B *екі жиынның бірігуі дегеніміз* - кемінде біреуі A немесе B жиынына енетін элементтерден тұратын жиын. Паскальдағы бірігу амалы белгісі «+».

Екі жиынның бірігу нәтижесі сызба түрінде 2.27, а - суретте көрсетілген. Мысалы:

```
[1.2.3.4]   +[3,4,5,6] ^[1,2,3,4,5,6].
```

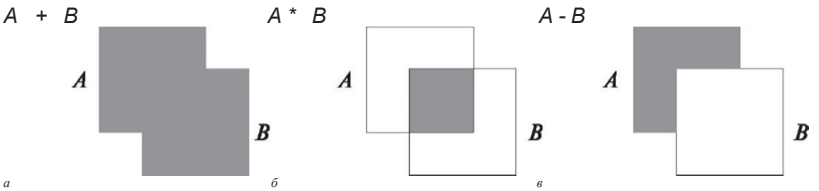
А және *В жиындарының қиылысуы дегеніміз* – бір уақытта *А* және *В* жиынына кіретін элементтерден тұратын жиын (2.27-сурет, *б*). Мысалы:

```
[1.2.3.4]   * [3,4,5,6] ^[3,4].
```

А және *В жиындарының айырымы* дегеніміз – *В* жиынына кірмейтін *А* жиынының элементтерінен тұратын жиынды айтамыз (2.27-сурет, в). Мысалы:

```
[1,2,3,4]   - [3,4,5,6] ^ [1,2]
[3,4,5,6] - [1,2,3,4] ^ [5,6]
```

Бірігу және қиылысу амалдары алмастырымды, ал айырым амалы – алмастырылмайтыны көрініп тұр.



а *б* *в*

2.27-сурет. Жиындарға қолданылатын амалдар: *а* — бірігу; *б* — қиылысу; *в* — айырым

2.9-кесте. Жиындарға қолданылатын қатынас амалдары

Қатынас	Нәтиже	
	True	False
A = *B*	*А* және *В* жиындары сәйкес келеді	қарсы жағдайда
А <> *В*	*А* және *В* жиындары сәйкес келмейді	өзі
А <= *В*	*А* –ның барлық элементтері *В-да* жатады	»
А >= *В*	*В* –ның барлық элементтері *А -да* жатады	»

Жиындарды бір-бірімен салыстыруға болады, яғни олар үшін *қатынас амалдары* анықталады. Қатынас нәтижесі, белгілі

болғандай, True немесе False логикалық шамалары болып табылады. Жиындар үшін «>» және «<» амалдарын қоспағанда, басқа барлық қатынас амалдары қолданылады.

Жиындарға қолданылатын қатынас амалдары 2.9-кестеде сипатталған. Мұнда А және В жиындарының элементтері бірдей типті деп есептеледі.

Қатынас амалдарына бірнеше мысалдар келтірейік.

М айнымалысы программада келесі түрде сипатталған делік:

```
Var M : Set Of Byte;
```

Операторлар бөлімінде оған мән меншіктеледі.

```
M := [3,4,7,9];
```

Осыдан қатынас амалдары келесі нәтижелерді береді:

```
M = [4,   7, 3, 3, 9,]        -          True;
   M <>[7, 4, 3, 9, ] -                   False;
            co        л п к              True;
                              < п  s     True;
         M >= [1..10]         -           False;
              M <= [3..9]  -             True.
```

Кіру амалы жиын мен скалярлы шама арасында типі жиынның базалық типімен сәйкес келетін байланыс орнатады, яғни егер *x* — берілген типтің скалярлы шамасы, ал *M* — жиын болса, онда кіру амалы келесі түрде жазылады:

x In M.

Егер *x* мәні М жиынына кірсе (тиісті болса), бұл жерде нәтиже True логикалық шамасы болады, және қарсы жағдайда—False болады. Келтірілген жиын үшін,

4 In M - True,
5 In M — False болады.

Жиындарға бірнеше мысалдар келтірейік.

2.27-Мысал. Символдық жол берілген. Ондағы тыныс белгілерінің санын анықтау керек (. — , * : ! ?).

Есептің сәйкес программасы:

```
Program P1;
Var S: String; I, K: Byte;
Begin
   ReadLn(S); K := 0;
   For I := 1 To Length(S) Do
```

```
    If S[I] In ['.',  '-',  ',',  ';',  ':',  '!',  '?']
    Then K := K + 1;
    WriteLn('Тыныс белгілерінің саны тең ', K)
End.
```

Бұл мысалда элементтердің символдық типі жиындық тұрақты қолданылып тұр. Бірақ, бұл есепті If операторына мынадай ұзақ (S[I] = '.') Or (S[I] = '—') және т.с.с. логикалық өрнекті жазу арқылы жиындарды қолданбай да шығаруға болады. Ал жиындарды қолдану - программа жазбасын қысқа етіп көрсетуге ыңғайлы.

2.28-Мысал. Латынның тек бас әріптері қамтылған екі символдық жол берілген. *S*1 және *S*2 жолдарының ортақ символдары алфавиттік ретте және қайталанбайтындай *S*3 жолына кіретін жаңа жолды құрастыру керек.

Есептің программасы:

```
Program P2;
Type Mset = Set Of 'a'..'z';
Var S1, S2, S3 : String;
                        MS1, MS2, MS3 : Mset;
    C : Char;
Procedure SM(S : String; Var MS : Mset);
{ процедура S жолының барлық символдарын қамтитын MS
жиынын қалыптастырады}
Var I: Byte;
Begin MS := [];
                    For I := 1 To Length(S) Do
                        MS := MS + [S[I]]
End;
Begin {Бастапқы жолдарды енгізу}
  ReadLn(S1); ReadLn(S2);
{ S1 және S2 жолдарының символдарынан MS1 және MS2
жиындарының қалыптастырылуы}
  SM(S1, MS1); SM(S2, MS2);
{Жиындардың қиылысуы - MS3 жиынында ортақ
элементтердің шығуы}
  MS3 := MS1 * MS2;
{ S3 нәтиже жолының пайда болуы}
  S3 := '';
  For C := 'a' To 'z' Do
      If C In MS3 Then S3 := S3 + C;
  WriteLn('Нәтиже:  ', S3)
End.
```

2.29-Мысал. 2-ден *N* –ге дейінгі ($1 < N < 255$) натурал сандар тізбегінен барлық жай сандар таңдалып алынатын программа құру керек.

Белгілі «Эратосфен Решето» деп аталатын алгоритмі бар, бұл

алгоритмнің негізі мынада:

1) сандар тізбегінен минималды мән – жай сан таңдалады;
2) тізбектен таңдалған мәнге еселі барлық сандарды жояды;
3) егер барлық еселі сандарды жойғаннан кейін тізбек бос болып қалмаса, 1-ші пункт орындалады

Осындай алгоритмге мысал қарастырайық, $N = 15$ (мұнда таңдалған жай сандардың асты сызылған):

2 3 4 5 6 7 8 9 10 11 12 13 14 15
3 5 7 9 11 13 15
5 7 11 13
7 1113
11 13
13

Бұл есепті шешу программасында мәліметтердің жиындық типін қолдану ыңғайлы:

```
Program Eratosfen;
Const N = 201;
{мүмкін кез-келген мән 1 < N < 256}
Var A, B : Set Of 2..N;
K, P : Integer;
Begin
{бастапқы А жиынын қалыптастыру; В— ізделіп отырған жай
сандар жиыны, алдымен — бос жиын}
      A := [2..N];
      B := [];
      P := 2;
      Repeat
{А жиынынан минималды санды іздеу}
            While Not (P In A) Do P := P + 1;
{В жиынына табылған санды кіргізу}
            B := B + [P];
            K := P;
{А жиынынан Р еселі сандарды алып тастау}
            While K <= N Do Begin
                  A := A - [K];
                  K : = K + P;
            End
      Until A = [];
{Нәтиже шығару, яғни В жиынынан барлық сандарды өсу
ретімен шығару}
      For P := 2 To N Do If P In B Then WriteLn(P)
End.
```

Бұл әдемі программа, алайда оны $N > 255$ шарты бойынша қолдануға болмайды. Себебі Турбо Паскальда жиындардың максималды өлшемі 255.

Жоғарыда айтылғандай, мәндерді тікелей жиынға енгізе алмаймыз. Дегенмен, программалаушыға осындай қажеттілік

туындауы мүмкін. Бұл жағдайда INSET процедурасын пайдалануға болады.

Мысалы, символдық базалық типті жиында негізгі программада SetChar типі ауқымды түрде жарияланған деп болжасақ,

```
Type SetChar : Set Of Char;

Procedure INSET(Var M : SetChar);

Var I, N : Byte; C : Char;
Begin
   Write('Жиынның өлшемін көрсет:'); ReadLn(N);
   M := [];
   For I := 1 To N Do
   Begin
         Write(I:1,'-M элемент:     '); ReadLn(C);
         M := M + [C]
   End;
   WriteLn('Енгізу аяқталды!')
```

End; Негізгі программада жиынға мәнді енгізу үшін, мысалға SIM атаулы мән, ол үшін INSET(SIM) операторын жазу жеткілікті. Нәтижесінде мәндерді диалогтық енгізу пайда болады.

Келесі меншіктеу операторларының орындалуы нәтижесіндегі *P* айнымалысының мәнін анықтау керек. Мұндағы $i = 3$ және $j = 5$:

а) P := [I + 3, J Div 2, J..Sqr(I) - 3] ;
ә) P :=[2 * I..J];
б) P :=[I, J, 2 * I, 2 * J] ?

2. Келесі қатынастардың мәнін анықтаңдар:

а) [2] <> [2,2,2];
ә) ['a','b']= ['b','a'];
б) [4,5,6] = [4,5,6];
в) ['c','b']=['c'..'b'];
г) [2,3,5,7] <= [1..9];
ғ) [3,6..8] <= [2..7,9];
д) Trunc(3.9)In[1,3,5];
е) Odd(4)In [].

ЖАТТЫҒУЛАР

1. Программада келесі сипаттамалар келтірілген:

```
Var P: Set Of 0..9; I, J : Integer;
```

3. Өрнектердің мәнін табыңдар:

а) [1,3,5] + [2,4]; ә) [1,3,5] * [2,4];
б) [1,3,5] - [2,4]; в) [1..6] + [3..8];
г) [1..6] *[3..8]; ғ) [1..6] - [3..8];

д) [] + [4]; е) [] * [4];
ж) [] - [4].

4. Натурал санның ондық жүйеде әр түрлі цифрлы сандардың санын есептеуге арналған программа құрастыру.

5. $n^2 + m^2$ түрде берілген, және n, $m > 0$ болатын 1... 255 аралығындағы барлық бүтін сандарды өсу ретімен шығаратын программа құрыңдар.

6. Кіші орыс әріптерінен жол берілген. Көрші әріптердің арасында – үтір, соңғы сөзден кейін – нүкте қойылған. Әріптерді келесі алфавиттік ретте орналастырудың программасын құрыңдар:

а) әрбір сөзге кіретін барлық дауысты дыбыстарды;
б) бірде-бір сөзге кірмейтін барлық дауыссыз дыбыстарды;
в) бір сөзге ғана кіретін барлық дауыссыз дыбыстарды;
г) бір сөзден артық сөздерге кіретін барлық дауысты дыбыстарды.

2.19. ФАЙЛДАР

«Файл» терминімен сендер бұрыннан таныссыңдар. Бұл ұғымды алдымен құрылғылардың ішкі жадысындағы ақпараттармен байланыстырады.

Паскальда файл ұғымының екі анықтамасы бар:

- ішкі құрылғыдағы атауы бар ақпарат (ішкі файл);
- Паскаль-программада файлдық типтегі айнымалы (ішкі файл).

Программада осы объектілер арасында байланыс орнатылады. Нәтижесінде программаны ішкі файлмен бірге орындау кезінде туындайтын барлық ақпараттардың сыртқы файлда көшірмесі пайда болады. Файл элементтерімен тек екі амалды орындай аламыз: файлдан оқу, файлға жазу.

Айнымалының файлдық типі — саны алдын ала анықталмаған (программаны орындағанға дейін) бір типті элементтердің жиынтығын қамтитын құрылымдық тип.

Файлдық айнымалының сипатталуы:

Var <айнымалы атауы> : File Of <элемент типі >;

Мұндағы, <элемент типі> файлдық типтен басқа кез-келген тип. Мысалы:

```
Fi | File Of Integer;
Fr | File Of Real;
Fc | File Of Char;
```

Файл *маркер соңы* (М.с) деп аталатын арнайы кодпен аяқталатын 0-ден бастап нөмірленген элементтердің дәйекті тізбегі (Эл.) ретінде ұсынылуы мүмкін:

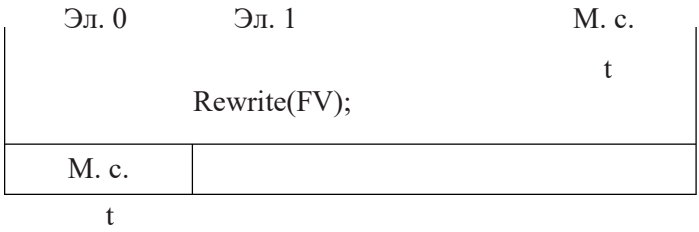
Эл. 0	Эл. 1		Эл. *N*	М. с.

Дәл сол уақыттағы файлда сақталған элементтердің саны *ағымдағы ұзындық* деп аталады. Ағымды өңдеу үшін (жазу немесе оқу) файл элементінің мекен-жайын сақтайтын арнайы жады ұяшығы бар. Бұл мекен-жай файлдың *нұсқағышы* немесе *терезесі* деп аталады.

Файлға жазуды бастау үшін оны ашу керек. Бұл Rewrite (FV) процедурасымен қамтамасыз етіледі (FV - файлдық айнымалы атауы). Бұл жағдайда нұсқағыш файлдың басына орнатылады. Егер бұрын файлда ақпарат болса, ол жойылады.

Rewrite процедурасы орындалуын сызбалық түрде былай көрсетуге болады:

Дейін:

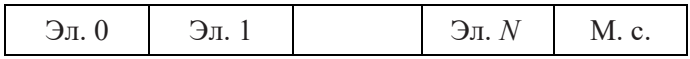


Төменгі жақтағы бағыттауыш сызығы нұсқағыш орнын білдіреді.

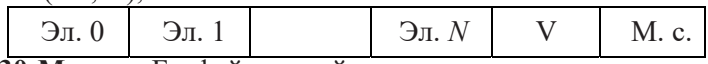
Файлға жазу Write (FV, V) процедурасы арқылы орындалады; (мұндағы, V - FV файл типіндегі айнымалы мән). Жазу терезе орнатылған жерде (файлдың нұсқағышы) орын алады. Алдымен мән жазылады, содан кейін нұсқағыш келесі орынға жылжиды. Егер файлдың соңына жаңа элемент қосылса, соңғы маркер жылжиды.

Бұл процедураның орындалу сызбасы:

Дейін:



Write(FV, V);

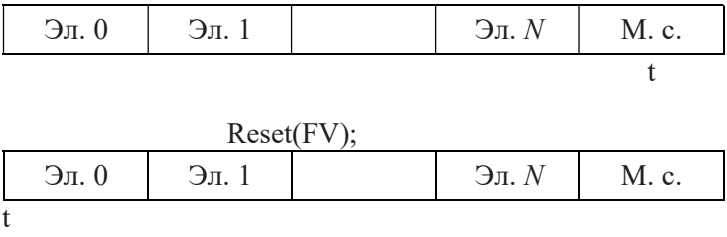


2.30-Мысал. Fx файлдық айнымалысына пернетақтадан 20 нақты сандар енгізу керек. Бұл есепті шешу программасы келесі түрде болады:

```
Var Fx : File Of Real;
        X : Real; I: Byte;
    Begin
      Rewrite(Fx);
      For I := 1 To 20 Do Begin
            Write('? '); ReadLn(X);
            Write(Fx, X)
      End
    End.
```

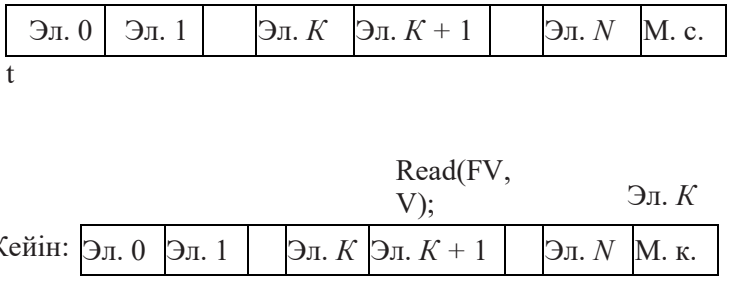
Файл элементтерін *оқу үшін оны алдымен ашып алу* керек. Бұл әрекет Reset(FV) процедурасы арқылы жүзеге асырылады. Нәтижесінде нұсқағыш файлдың басына орналасады. Сонымен қатар, файлда барлық ақпарат сақталады.

Берілген процедураның орындалу сызбасы келесідей:



Файлдан оқу Read(FV, V) процедурасы арқылы орындалады; (мұндағы, V —FV файл типіндегі айнымалы мән). Ағымдағы файл элементінің мәні V айнымалысына жазылады да, нұсқағыш келесі элементке көшеді.

Берілген процедураның орындалу сызбасы келесідей:



Файл элементтеріне рұқсат *тізбектелген* немесе *тікелей* болуы мүмкін. Стандартты Паскальда элементтерге тек тізбектелген рұқсат қарастырылған.

Тізбектелген рұқсат принципі: файлдың *n*-ші жазбасын оқу үшін алдымен оның барлық 1-ден (n -1)-ге дейінгі алдыңғы жазбаларын оқып алу қажет.

2.31-Мысал. *X* айнымалысына нақты *Fx* файлының 10-шы элементін алу керек.

Программасы келесідей:

```
Program A;
Var Fx : File Of Real;
      X : Real;
Begin
  Reset(Fx);
  For I := 1 To 10 Do Read(Fx, X)
End.
```

Eof(FV) функциясы (end of file) файлдың маркері соңын тексереді. Бұл егер нұсқағыш маркер соңында тұрса, True мәнін, ал қарсы жағдайда False мәнін қабылдайтын логикалық функция.

2.32-Мысал. 2.31-мысалда көрсетілген Fx файлындағы барлық сандардың қосындысын есептеу қажет.

Бұл есептің программасы мынадай:

```
Reset(Fx);
Sx := 0;
While Not Eof(Fx) Do
Begin
  Read(Fx, X);
  Sx := Sx + X
End;
```

Сонымен қатар, берілген есепті Repeat операторының көмегімен де шешуге болады:

```
Repeat
   Read(Fx, X);
   Sx := Sx + X
Until Eof(Fx);
```

Екінші нұсқада егер файл бос болса, оны оқуда қателер туындауы мүмкін, бірінші нұсқада мұндай қателер орын алмайды.

Сыртқы файлдар. Турбо Паскальда барлық сыртқы құрылғылар (дисплей, пернетақта, принтер, дискілер және т.б.) деректерді ұйымдастырудағы файл құрылымы мен логикалық құрылғылар ретінде қарастырылады. Барлық магниттік емес сыртқы құрылғылар – бірфайлдық құрылғылар. Басқаша айтқанда, олардың әрқайсысымен стандартты аты бар бір файл байланысқан және олар ЭЕМ-нің мәтіндік (символдық) ақпараттарын ішкі жадымен алмастыруға бағытталған.

Логикалық құрылғылардың стандартты атауларын Паскаль жұмыс істеп тұрған операциялық жүйе анықтайды. MS DOS жүйесінде келесі атаулар анықталған:

CON (консоль) — пернетақтадан енгізу мен экранға шығару әрекеттерімен байланысқан логикалық құрылғы;

PRN (принтер) — басып шығару құрылғысымен байланысқан файлдың логикалық атауы;

AUX — ДК-дің басқа да машиналармен байланысы кезінде пайдаланылатын байланыс каналының атауы;

INPUT — пернетақтамен байланысқан стандартты енгізу құрылғысы. Пернетақтадан енгізілген символдар экран дисплейінде көрініп тұрады;

OUTPUT — стандартты экранға шығару құрылғы.

Магнитті диск (МД) — көпфайлдық құрылғы. Мұнда стандартты файлдармен (жүйелік) қатар, қолданушы құрған файлдар сақталады. Магнитті дискте файлдың кез-келген типтері сақтала береді. Мұндағы файлдар оқу режимімен қатар, жазу режимінде де

пайдаланыла береді.

Дисктегі файлдар тізімі диск директориясында (каталог) сақталады. Каталог экранға DIR жүйелік командасының көмегімен шақырылады. Толық түрде, каталогта файл идентификаторлары сақталатын жады мөлшері және файл жасалған күн мен уақыт болады.

Файл идентификаторы файл атауы мен типінен тұрады:

<файл атауы>.<файл типі>

Файлдың атауы 1-ден 8-ге дейінгі латын әріптерінен және сандардан тұрады; файл типі — файлда сақталатын (1-ден 3-ке дейінгі символдар) ақпараттардың мінездемесін көрсететін міндетті емес элемент.

Мысалы:

PROGRAM.PAS — файлдағы Паскальдағы программа мәтіні;

NUMBER. DAT — сандық мәліметтер файлы;

NAMES.TXT — мәтіндік файл.

Турбо Паскальда файлдық айнымалы мен ішкі файлды байланыстыру үшін келесі *тағайындау процедурасы* қолданылады:

Assign (<файлдық айнымалының атауы>, <ішкі файл идентификаторы>).

Мұндағы, <ішкі файл идентификаторы> — жолдық шама (тұрақты немесе айнымалы). Мысалы:

```
Assign(Fi, 'Number.dat');
```

Assign және Rewrite процедураларының орындалуынан кейін атауы директорияға кіретін жаңадан ішкі файл құрылады.

Егер файл оқу үшін ашылса (Assign және Reset), онда каталогта атауы көрсетілген ішкі файл болу керек. Қарсы жағдайда қате туралы хабарлама шығады.

Программада файлдармен жұмысты *аяқтау* келесі процедура қолданылады:

Close(<файлдық айнымалының атауы>);

Мысалы:

```
Close(Fi).
```

Сонымен, файлды құру мен оны толтырудағы орындалатын әрекеттер:

- файлдық айнымалыны сипаттау;
- файлдың типімен бірдей айнымалыны сипаттау;

- тағайындауды орындау (Assign);
- жазу үшін файлды ашу (Rewrite);
- файлға мәліметтер жазу (Write);
- файлды жазу (Close).

2.33-Мысал. Бірнеше күн ішіндегі орташа температураларын есептейтін файл құру. Енгізу ақпаратындағы цифрлардың санын алдын ала көрсету міндетті емес. Енгізу соңы болып табылатын белгілі бір шартты мәнді өзіміз белгілей аламыз, мысалы 9 999 саны.

Есептің программасы:

```
Program Task1;
Var Ft : File Of Real; T : Real;
Begin
  Assign(Ft,'Temp.dat'); Rewrite(Ft);
  WriteLn('Берілгендерді енгіз. Белгі соңы - 9999') ;
  ReadLn(T);
  While T <> 9999 Do Begin
        Write(Ft, T); Write('? '); ReadLn(T)
  End;
  WriteLn('Енгізу аяқталды!');
  Close(Ft)
End.
```

Бұл программаның нәтижесінде дискте енгізілген ақпараттар сақталған Temp.dat атты файл құрылады.

Файлдан кезекті мәліметтерді оқу үшін мына әрекеттерді де орындау керек:

- файлдық айнымалыны сипаттау;
- файлдың типімен бірдей айнымалыны сипаттау;
- тағайындауды орындау (Assign);
- оқу үшін файлды ашу (Reset);
- циклде файлдарды оқу (Read);
- файлды жабу (Close).

2.34-Мысал. Temp.dat файлында сақталған мәндер үшін орташа температураны анықтау.

Есептің программасы:

```
Program Task2;
Var Ft : File Of Real;
      T, St : Real; N : Integer;
Begin Assign(Ft, 'Temp.dat');
  Reset(Ft);
```

```
    St := 0;
    While Not Eof(Ft) Do
    Begin
          Read(Ft, T);
          St := St + T
    End;
    N := FileSize(Ft);
    St := St/N;
    WriteLn('Орташа температура',N:3,   ' күнде тең', St:7:2,'
    градус');
    Close(Ft)
End.
```

Бұл программада файл өлшемін анықтайтын келесі процедура қарастырылған:

FileSize (<файлдық айнымалы атауы>);

Бұл функцияның орындалу нәтижесі - ағымдағы файл ұзындығына тең бүтін сан.

Ескерту. Стандартты Паскаль тіліне сәйкес, Rewrite операторы ашқан файл тек ақпаратты жаза алады және Reset операторы арқылы ашылған файлды тек оқу үшін пайдалануға болады. Турбо Паскальда оқу үшін (Reset) ашылған файлға жазу (Write) рұқсат етіледі, бұл файлдарды модификациялау үшін белгілі бір ыңғайлылықты тудырады.

Мәтіндік файлдар. Бұл файлдардың ең жиі қолданылатын түрлері. Жоғарыда айтылғандай, магниттік емес сыртқы құрылғылар (логикалық) тек мәтіндік файлдармен жұмыс істейді. Паскаль және басқа да программалау тілдеріндегі программалардың мәтіндері бар файлдар мәтіндік болып келеді. Электронды байланыс арналары арқылы берілетін түрлі құжаттар мен ақпараттар да мәтіндік файлдар болып табылады.

Программада мәтіндік типтегі файлдық айнымалы төмендегідей сипатталады:

Var <идентификатор> : text;

Мәтіндік файл жолдарға бөлінген символдық тізбектен тұрады. Әрбір символды (S) жолы арнайы кодпен — жол соңының маркерімен (Ж.с.м.). аяқталады. Тұтас файл файл соңының маркерімен аяқталады (Ф.с.м.). Сызба түрінде оларды мына түрде сипаттаймыз:

S1	*S2*		*Sk1*	Ж.с.м	S1	S2		*Sk2*	Ж.с.м		Ф.с.м

Әрбір таңба ішкі кодта (ASCII) ұсынылған және 1 байт орынды

алады. Дегенмен, мәтіндік файл символдық файлдан тек жолдарға бөлумен ғана ерекшеленбейді: мәтіндік файлға жаза аламыз, сонымен қатар кез-келген түрдегі ақпаратты оқуға болады. Егер бұл ақпарат символдық болса, онда оқу немесе жазу барысында ол символдық формадан ішкі формаға және керісінше түрленеді.

Мәтіндік файлды құруға немесе мәтіндік редактордың көмегімен түрлендіріп, экран дисплейінен көруге және оны басып шығаруға болады.

Паскальдың программаларында мәтіндік файлдармен жұмыс істеу барысында Read және Write процедураларымен қатар, ReadLn және WriteLn процедуралары қолданылады.

FV атаулы файлдан жолды оқитын және оқылған ақпаратты енгізу тізімдерінің айнымалыларына белгілеп отыратын процедура келесідей:

ReadLn(FV, <енгізу тізімі>).

FV файлына шығару тізімдеріндегі мәндерді жазатын және жол соңының маркерін тағайындайтын процедура:

WriteLn(FV, <шығару тізімі>).

Eoln (FV) функциясы мәтіндік файлда жолдың соңын (end of line — жолдың соңы) табу үшін пайдаланылады. Бұл файлдың нұсқағышы жол соңының маркеріне жеткен болса, True мәнін қабылдайтын және қарсы жағдайда - False мәнін қабылдайтын логикалық функция.

Read және ReadLn операторларын файлдық айнымалы атауын көрсетпестен пайдалану, стандартты Input файлынан (пернетақтадан енгізу) оқуды білдіреді. Write және WriteLn операторларын файл айнымалысының атауынсыз пайдалану стандартты Output файлына (экранға шығару) жазуды білдіреді. Біз бірнеше рет операторлардың осы нұсқаларын қолдандық. Енгізу және шығару файлдары, кез-келген программа жұмыс істеп жатқанда, тиісінше оқу және жазу мақсатында ашылады деп есептеледі.

Пернетақтадан енгізу барысында жол соңының маркері <Enter> пернесін басқанда пайда болады.

ReadLn процедурасы енгізу тізімінсіз қолданылуы мүмкін. Бұл жағдайда оқылатын файлдағы ағымдағы жол өткізілмейді.

WriteLn процедурасын шығару тізімінсіз пайдалану бос жолды шығаруды білдіреді (файлда жол соңының маркері орналасады).

Мәтіндік файлға жазғанда, шығару тізімінде форматтар болуы мүмкін. Форматтардың әрекеттері экранға деректерді шығаруды сипаттау барысында қарастырылған болатын (2.6 бөлімін қара). Форматтар кез-келген басқа құрылғымен байланысқан мәтіндік

файлдарды шығарған кезде да тура сол сияқты жұмыс істейді.

2.35-Мысал. Note.txt атаулы файлында мәтін бар деп есептейік. Осы мәтіндегі жолдардың санын анықтау керек.

Есепті шешу программасы келесідей:

```
Var Note : Text; K : Integer;
Begin
  Assign(Note, 'Note.txt');
  Reset(Note);
  K := 0;
  While Not Eof(Note) Do
  Begin
    ReadLn(Note);
    K := K + 1
  End;
  WriteLn('Жолдар саны тең ', K);
  Close(Note)
End.
```

Бұл есепте қолданылған ReadLn(Note) операторы Note мәтіндік файлынан жолдарды қандайда бір айнымалыға кіргізбей «парақтайды».

2.36-Мысал. Note.txt мәтіндік файлындағы ең ұзын жолды анықтау.

Есептің программасы келесідей:

```
Var Note : Text;
      Max, K : Integer; C: Char;
Begin
  Assign(Note, 'Note.txt');
  Reset(Note);
  Max : = 0;
  While Not Eof(Note) Do
  Begin
    K := 0;
    While Not Eoln(Note) Do
    Begin
      Read(Note, C);
      K := K + 1
    End;
    If K > Max Then Max := K;
    ReadLn(Note)
  End;
  WriteLn('Ең үлкен жолда', Max, 'белгі бар');
  Close(Note)
End.
```

Мұнда әрбір жол символдар бойынша саналады. *K* айнымалысында жолдағы символдардың санауышы жұмыс істеп

тұрады, ал Max айнымалысына осы санауыштың ең үлкен мәні алынады.

2.37-Мысал. Келесі есепті шешу қажет. F күшінің әсерімен және V бастапқы жылдамдықтағы тұтқыр ортада M массалы дене жылжып келеді. Қозғалыс кедергісі K коэффициентті жылдамдық квадратына пропорционалды. Қозғалыс траекториясының бес бақылау нүктелерінен өту уақытын анықтау керек. Бұл жағдайда бастапқы нүктеден арақашықтық белгілі.

DATE.TXT файлында мәтіндік редактордың көмегімен бастапқы берілгендер келесі кестедегідей дайындалған болсын:

Бастапқы берілгендер				
M (кг) 36,3	*F* (Н) 2000	*V* (м/с) 50,5	*K* (кг/м) 0,5	
Бақылау нүктелердің координаталары (м)				
X(1) 10	X(2) 100	X(3) 150	X(4) 1 000	X(5) 3 000

M, F, V, *K* нақты айнымалыларына және X[1..5] жиымына сандық мәліметтерді енгізу керек, есептеулер жүргізіп, нәтижелерді Г[1..5] жиымында алу қажет. Сонымен қатар, дискідегі Result.Txt атаулы мәтіндік файлға да нәтижелерді алу керек.

Келесі программада есептеулер көрсетілмеген, тек бастапқы берілгендерді енгізу мен нәтижелерді шығару фрагменті сипатталған:

```
Var M, F, V, K : Real; I : Integer;
      T, X : Array[1..5] Of Real;
      FR, FD : Text;
      Begin
          Assign(FD, 'DATE.TXT'); Reset(FD);
          Assign(FR, 'Result.txt'); Rewrite(FR);
{ ---- Алғашқы үш жолды өткізу ------- }
          ReadLn(FD); ReadLn(FD); ReadLn(FD);
{ ---- Берілгендерді енгізу  }
          ReadLn(FD, M, F, V, K);
{ ---- Үш жолды өткізу --------- }
          ReadLn(FD); ReadLn(FD); ReadLn(FD);
{ ---- Берілгендерді енгізу  }
          For I := 1 To 5 Do Read(FD, X[I]);

          .....
          {Программаның есептеу бөлімі}
          ……
```

```
{ ---- Экранға және FR файлына нәтижелерді шығару--    }
        WriteLn('Нәтижелер '); WriteLn;
        WriteLn(FR, ' Нәтижелер'); WriteLn(FR);
        WriteLn(' T(1) T(2) T(3) T(4) T(5)');
        WriteLn(FR, ' T(1) T(2) T(3) T(4) T(5)');
        For I := 1 To 5 Do Begin
                Write(T[I]:8:2); Write(FR, T[I]:8:2)
        End;
        Close(FD); Close(FR)
    End.
```

Нәтижелер Result.Txt файлында сақталады. Оларды экраннан көріп, қағаз бетіне шығаруға болады. Бұл файлдағы мәліметтер басқа есептердің бастапқы берілгендеріне сәйкес болған жағдайда, қажет сол программаға кіріс файл ретінде қолданыла алады.

ЖАТТЫҒУЛАР

1. Нақты сандар файлы берілген. Осы файлдағы 0-дік мәндердің санын анықтаңдар.
2. Бүтін санды екі файл берілген. Осы сандардың тепе-теңдігін анықтау.
3. Бірдей өлшемдегі екі символдық файлдар берілген. Олардың арасында ақпарат алмасуды жүзеге асыру қажет.
4. Ішкі мәтіндік файл берілген. Оның ең қысқа жолын анықтау қажет.
5. T бос емес мәтіндік файлының мәліметтерін жол бойынша шығаратын Lines(T) процедурасын сипаттау. Шығару барысында әрбір жолдың басына төрт позиция мен бос орын алынатын, оның реттік нөмірін жазу қажет.
6. T мәтіндік файлында бос орындармен жазылған нақты сандардың бос емес тізбегі бар. Осы сандардың ішіндегі үлкенін анықтайтын Max(T) функциясын сипаттау.

2.20. МӘЛІМЕТТЕРДІҢ АРАЛАС ТИПТЕРІ

Барлық қарастырылған құрылымдық типтер бір типті шамалардың жиынтығы болып табылады.

2.28-сурет. Бір деңгейлі ағаш

Студент анкетасы. Аты-жөні.
Жынысы. Мерзімі. Мекен-жайы.
Курс. Топ. Шәкіртақы

Мәліметтердің аралас типтері — түрлі типтегі компоненттердің (өрістердің) белгіленген санынан тұратын құрылымдық тип. Оның басқаша атауы – *жазба.*

Әдетте жазбада бір объектіге қатысты әртүрлі атрибуттардың жиынтығы болады. Мысалға, университет студенті туралы жеке мәліметтерді 2.28-суретте көрсетілген ақпараттық құрылым түрінде ұсынуға болады, ол *бір деңгейлі ағаш* деп аталады. Паскальда бұл ақпарат Record (жазба) типінің айнымалысында сақталуы мүмкін. Типті тағайындау мен тиісті айнымалы мәндерді төмендегідей сипаттай аласыз:

```
Type Anketa1 = Record
                 FIO : String[50];
                 Pol : Char;
                 Dat : String[16];
                 Adres : String[50];
                 Curs : 1..5;
                 Grup : 1..10;          {Өрістер}
                 Stip : Real End;
Var Student : Anketa1;                  {Жазбалар}

                                        {немесе элементтер
```

Мұндай жазбаға сәйкес ағаш та *бір деңгейлі* деп аталады.

Жазбаның әрбір элементін *құрама атауды* пайдаланып шақыруға болады:

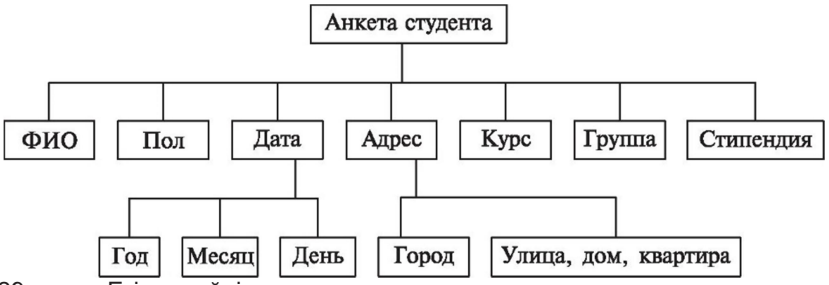
```
<айнымалы атауы>.<өріс атауы>
```

Мысалы, Student.FIO; Student.Dat және т.с.с.

Мысалы, «Курс» өрісіне 3 мәнін меншіктеу керек болса, онда ол келесі түрде жүзеге асырылады:

```
Student.Curs := 3;
```

Жазба өрістері кез-келген типті бола алады, әсіресе олар өздері жазба болуы мүмкін. Мұндай әрекеттер



2.29-сурет. Екі деңгейлі ағаш

Студент анкетасы. Аты-жөні. Жынысы. Мерзімі. Мекен-жайы. Курс. Топ. Шәкіртақы. Жыл. Ай. Күн. Қала. Көше, үй, пәтер.

көпдеңгейлі ағашты (бір деңгейден көп) көрсету керек болған жағдайда қолданылады. Мысалы, студент туралы осы ақпараттарды екі деңгейлі ағаш ретінде де көрсетуге болады (2.29-сурет).

Мұндай деректерді ұйымдастыру, мысалы, студенттердің туған жылы немесе оқушылар тұратын қаланы таңдау арқылы ақпарат жасауға мүмкіндік береді. Бұл жағдайда тиісті жазбаның сипаттамасы келесі формамен орындалады:

```
Type Anketa2 = Record
                  FIO: String[50];
                  Pol: Char;
                  Dat: Record
                          God: Integer;
                          Mes: String[10];
                          Den: 1..31
                          End;
                  Adres:    Record
                          Gorod: String[20];
                          UlDomKv: String[30];
                          End;
                  Curs: 1..5;
                  Grup: 1..10;
                  Stip: Real
               End;
Var Student: Anketa2;
```

Екінші деңгейде орналасқан жазбалар өрісі үш құрама атаумен идентификацияланады. Мысалы,

```
Student. Dat.God; Student. Adres.Gorod.
```

Аралас типтің сипатталу құрылымы 2.30-суретте көрсетілген.

Программада жазбалардың жиымдары қолданылуы мүмкін. Егер

факультетте 500 студент бар болса, олар туралы мәліметтерді жиымда елестетуге болады:

```
Var Student : Array[1..500] Of Anketal;
```

Бұл жағдайда тізімдегі 5-ші студенттің жылы туралы ақпарат Student[5].Dat.God айнымалысында сақталады.

Кез-келген жазбаны өңдеу (оның ішінде енгізу мен шығару) элемент бойынша жүзеге асады. Мысалы, 500 студент туралы ақпаратты енгізуді келесі жолмен орындауға болады:

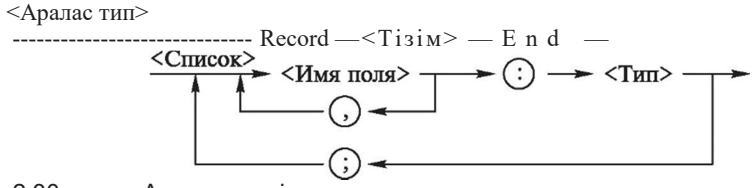
```
For I := 1 To 500 Do
With Student[I] Do
Begin
    Write(Аты-жөні:    '); ReadLn(FIO);
    Write('Жынысы):          '); ReadLn(Pol);
    Write('Туған күні:                '); ReadLn(Dat);
    Write('Мекен-жайы:   '); ReadLn(Adres);
    Write('Курс:         '); ReadLn(Curs);
    Write('Топ:              '); ReadLn(Grup);
    Write('Шәкіртақы (руб.):              '); ReadLn(Stip)
End;
```

Бұл мысалда жалғау операторы қолданылған, ол келесі түрде сипатталады:

```
With <жазба типті айнымалы> Do <оператор>;
```

Бұл оператор With сөзінен кейін «Жазба» типіндегі айнымалы атауын бір рет көрсетіп, өріс атауларымен қарапайым айнымалы ретінде жұмыс істеуге мүмкіндік береді.

Паскальдағы «Жазба» типі айнымалысында программаның орындалу барысында өзгеретін өріс құрамасы болуы мүмкін. Бұл мүмкіндік жазбаның нұсқалық бөлігін пайдалану арқылы жүзеге асырылады (толығырақ Паскаль тілінің арнайы әдебиеттерінен қарауға болады).



2.30-сурет. Аралас типті сипаттау құрылымы

Тізім. Өріс атауы. Тип

Жазба файлдарымен жұмыс. Көбінесе, жазбалар

компьютерлік ақпараттық жүйелерді құрайтын файл элементтері ретінде пайдаланылады. Жазба файлдарымен жұмыс істейтін программалардың мысалдарын қарастырайық.

2.38-Мысал. FM.DAT файлын құру қажет. Бір студенттік топтың емтихан парағы берілген. Файл жазбалары келесі элементтерден тұрады: Аты-жөні; сынақ кітапшасының нөмірі; баға.

Бұл есепті шешу программасы келесі түрде болады:

```
Program Examen;
Type Stud = Record
                    FIO : String[30];
                    Nz : String[6];
                    Mark : 2..5 End;
Var Fstud : File Of Stud;
   S : Stud;
   N, I : Byte;
Begin
   Assign(Fstud, 'FM.DAT'); Rewrite(Fstud);
   Write('Группадағы студенттер саны ');
   ReadLn(N);
   For I := 1 To N Do
   Begin
        Write(I:1,'-ft, ФИО '); ReadLn(S.FIO);
        Write('Сынақ кітапшасының нөмірі:        '); ReadLn(S.Nz);
        Write('Баға:        '); ReadLn(S.Mark);
        Write(Fstud, S)
   End;
   WriteLn('Файлды өңдеу аяқталды!');
   Close(Fstud)
End.
```

Келесі мысалға көшпес бұрын, жасалынған файлдың өңдеуіне байланысты, біз файлдармен жұмыс істеуге арналған тағы бір құралды қарастырамыз.

Файлдық жазбаларға тікелей қол жеткізу. Стандартты Паскаль тілінде файл элементтеріне тек тізбекті қатынасуға рұқсат етіледі. Турбо Паскальда жүзеге асырылған қосымша функциялардың бірі - файл жазбаларына тікелей қол жеткізу.

Жоғарыда айтылғандай, файлдың элементтері файлға кіргізу тәртібімен нөлден бастап нөмірленеді. Файл элементінің нөмірін белгілей отырып, оған нұсқағыш орнатуға болады, содан соң бұл элементті оқуға немесе қайта жазуға болады.

Көрсетілген файл элементіне нұсқағышты орнату келесі процедурамен орындалады

```
Seek(FV, n).
```

Мұндағы, FV — файлдық айнымалының атауы, n — элементтің реттік нөмірі.

Осы процедураға қатысты мысал қарастырайық.

2.39-Мысал. 2.38 мысалдан алынған файл бар деп есептейік. Кейбір студенттер емтиханды қайта тапсырып басқа баға алды делік. Осы қайта тапсырылған емтихан нәтижесін студенттің нөмірі мен кейінгі алған бағасын сұрайтын программаны сол файлға кіргізу қажет. 9999 санын енгізгенде программа жұмысын тоқтатады.

Есептің программасы:

```
Program New_Marks;
Type Stud = Record
                    FIO : String[30]; Nz : String[6];
                    Mark : 2..5 End;
Var Fstud: File Of Stud;
      S : Stud;
      N : Integer;
Begin
  Assign(Fstud, 'FM.DAT');
  Reset(Fstud);
  Write('Емтихан парағы? ');
  ReadLn(N);
  While N <> 9999 Do Begin
        Seek(Fstud, N - 1);
        Read(Fstud, S);
        Write(S.FIO, ' баға? ');
        ReadLn(S.Mark);
        Seek(Fstud, N - 1);
        Write(Fstud, S);
        Write('Емтихан парағы? ');
        ReadLn(N);
  End;
  WriteLn('Жұмыс аяқталды!');
  Close(Fstud)
End.
```

Бұл мысалды талқылайық. Емтихан парағындағы студенттер тізімі 1-ден бастап нөмірленген. Ал файлдағы жазбалар 0-ден бастап нөмірленеді. Осыдан, егер n — емтихан парағындағы нөмір болса, онда файлдағы сәйкес жазба нөмірі n – 1 болады. n – 1 жазбасын оқығаннан кейін, нұсқағыш келесі n –ші жазбаға ауысады. Тура сол орынға түзетілген жазбаны қайта жазу үшін нұсқағыш орнатуын қайталайды.

ЖАТТЫҒУЛАР

1. Ұшақтардың рейстері туралы ақпараттары бар жазбаларды сипаттау.

2. Д. И. Менделеевтің химиялық элементтер кестесі сақталған жазбалар жиымын сипаттаңдар. Жиымды толтырудың программасын құрыңдар.

3. Кешенді санды екі элементті жазба ретінде қарастырып, кешенді сандармен арифметикалық амалдар орындаудың процедурасын құрыңдар.

4. Қоймадағы бөлшектер туралы келесідей ақпараттар бар: атауы, саны, бағасы. Төмендегі есептерді орындайтын программа құрыңдар:

а) қоймадағы бөлшектер туралы файлды ақпаратпен толтыру;

ә) бөлшектердің жалпы бағасын есептеу;

б) ең көп қай бөлшектер екенін және ең аз бөлшектер түрін анықтау ;

в) берілген типті бөлшектердің қоймадағы бар-жоғын және саны қанша екенін анықтау;

г) қоймадан осы бөлшектердің белгілі бір мөлшерін шығарғаннан кейін файлға өзгертулер енгізу. Қандайда бір бөлшектердің қоймадан толығымен таңдалатын болса, оны жазбадан жою керек.

2.21.

НҰСҚАҒЫШТАР ЖӘНЕ МӘЛІМЕТТЕРДІҢ ДИНАМИКАЛЫҚ ҚҰРЫЛЫМЫ

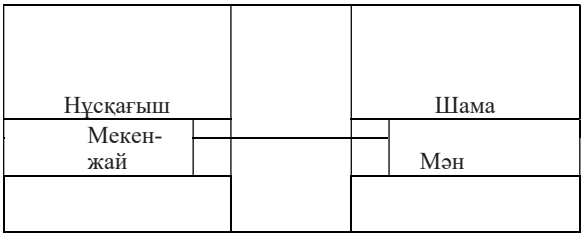
Бұрын тек статикалық мәліметтерді өңдеуді программалауды қарастырған болатынбыз.

Статикалық шамалар дегеніміз компиляция кезінде бөлінетін және программаның барлық жұмысы уақытында сақталатын жад.

Паскальда деректерге жад бөлудің тағы бір тәсілі бар – ол динамикалық. *Динамикалық* шамалар дегеніміз оған сәйкес программаның орындалуы кезіндегі бөлінетін жад.

Статикалық түрде бөлінген жедел жад бөлігі *статикалық* жад деп, ал динамикалық түрде бөлінген жедел жад бөлігі *динамикалық* жад деп аталады.

Динамикалық шамаларды қолдану программалаушыға көптеген қосымша мүмкіндіктер береді.



2.31-сурет. Нұсқағыш әрекетінің сызбасы

Статикалық жад **Динамикалық жад**

Біріншіден, динамикалық жадты пайдалану өңделетін мәліметтердің көлемін ұлғайтуға мүмкіндік береді. Екіншіден, қандайда бір мәліметтердің қажеттілігі программаның аяғына дейін керек болмай қалған жағдайда, ол алып тұрған жадыны басқа ақпараттар үшін босатуға болады. Үшіншіден, динамикалық жадты пайдалану айнымалы көлеміндегі мәліметтер құрылымын жасауға мүмкіндік береді.

Динамикалық шамалармен жұмыс істеу тағы бір мәліметтер типі – *сілтемелі* типін қолданумен байланысты.

Нұсқағыштар. Сілтемелі типті шамаларды *нұсқағыштар* деп атаймыз.

Нұсқағыш динамикалық жадтағы белгілі бір типтің шамасы сақталатын өрістің мекен-жайын қамтиды. Нұсқағыштың өзі статикалық жадта сақталады (2.31-сурет).

Мекен-жай - бұл шама орналасқан жады өрісінің бірінші байтының нөмірі. Өріс өлшемі әлбетте тип бойынша анықталады.

Сілтемелі типінің (нұсқағыштың) мәні айнымалыларды сипаттау бөлімінде келесідей сипатталады:

Var <идентификатор> : <сілтемелі тип>;

Стандартты Паскальда әр нұсқағыш тек бір типтің шамасына сілтеме жасай алады. Ол осы нұсқағыш үшін *базалық тип* болып саналады. Базалық типтің атауы келесі форматтағы сипаттамада көрсетілген:

<сілтемелі тип> := ^<тип атауы>;

Нұсқағыштарды сипаттауға мысалдар келтірейік:

```
Type Massiv = Array[1..100] Of Integer;
 Var
        P1: ^integer;
        P2: ^Char;
        PM: ^Massiv;
```

Мұндағы *P1* — бүтін типтегі динамикалық шамаға нұсқағыш; *P2* — символдық типтегі динамикалық шамаға нұсқағыш;
PM — типі бөлімінде көрсетілген динамикалық жиымға нұсқағыш.

Динамикалық шамалардың өздері программада сипаттауды қажет етпейді, себебі компиляция кезінде оларға жады бөлінбейді. Компиляция уақытында жад тек статикалық шамаларға бөлінеді. Нұсқағыштар – статикалық шамалар болғандықтан, олар сипаттауды талап етеді.

Нұсқағышпен байланысқан динамикалық шамаға жад NEW стандартты процедурасының орындалу нәтижесінде бөлінеді. Бұл процедураны келесі форматта шақырамыз:

```
NEW(<нұсқағыш>);
```

Бұл оператордың орындалу нәтижесінде динамикалық шама пайда болады деп саналады, оның атауы мына түрде жазылады:

```
<динамикалық шама атауы> ::= <нұсқағыш>^.
```

Программада нұсқағыштың осыған дейін келтірілген сипаттамалары болсын және келесі операторлар бар делік:

```
NEW(P1); NEW(P2); NEW(PM);
```

Оларды динамикалық жадта орындағаннан кейін P1^ пт, P2^ пт, PM^ идентификаторларына ие үш мәнге (екі скаляр және бір жиым) сәйкес орын бөлінеді. Мысалы, P1^ - P1 нұсқағышына сілтеме жасалған динамикалық айнымалы мәнді белгілеу.

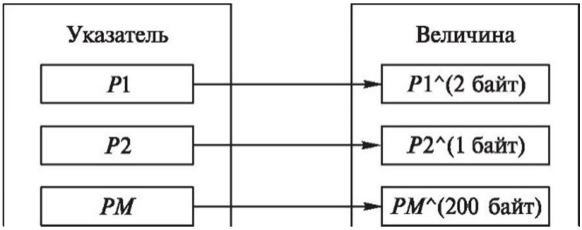
Динамикалық шамалар мен олардың нұсқағыштары арасындағы байланыс 2.32-суретте көрсетілген.

Динамикалық айнымалылармен ары қарай сәйкес типті статикалық айнымалы мәндері сияқты жұмыс істейміз, яғни оларға мәндерді меншіктеуге болады, оларды өрнектердегі операндтар ретінде және ішпрограммалар параметрлері ретінде және т.б. пайдалануға болады. Мысалы, егер P1^ айнымалысына 25 санын, P2^ айнымалысына «W» символының мәнін меншіктеп, ал PM^ жиымын 1-ден 100-ге дейінгі сандармен толтыру қажет болса, онда оны келесі түрде жүзеге асырамыз:

```
P1^ := 25; P2^ := ' W ' ;
For I := 1 To 100 Do PM^[I] := I;
```

NEW процедурасынан басқа нұсқағыштың мәні келесі меншіктеу операторымен де анықталады:

```
<нұсқағыш> := <сілтемелі өрнек>;
```



2.32-сурет. Нұсқағыштардың динамикалық шамалармен байланысы.

Нұсқағыш. Шама

Сілтемелі өрнек ретінде мыналарды пайдалануға болады:

- нұсқағыш;
- сілтемелі функцияны (яғни мәні нұсқағыш болатын функция);
- Nil тұрақтысын.

Nil — алдын ала сақталған тұрақты, бос сілтемені білдіреді, яғни ештеңені нұсқамайтын сілтеме. Меншіктеу кезінде нұсқағыштың базалық типтері мен сілтемелі өрнек бірдей болуы керек. Nil тұрақты мәнін кез-келген базалық типтегі нұсқағышқа меншіктеуге болады.

Мәнді меншіктемес бұрын сілтемелі айнымалы (меншіктеу операторын немесе NEW процедурасын пайдалану арқылы) анықталмаған болады.

Нұсқағыштарды енгізуге және шығаруға рұқсат етілмеген.

Мысал қарастырайық. Программада келесі нұсқағыштар сипатталған болсын:

```
Var D, P : ^Integer;
    K : ^Boolean;
```

Онда меншіктеу операторларына рұқсат беріледі

```
D := P; K := Nil,
```

және типтер сәйкестік принципі орындалу керек. Оператор *K* := *D* операторы қате болып саналады, себебі сол және оң жақтағы базалық типтер әр түрлі.

Егер динамикалық шама нұсқағышын жоғалтса, ол «қоқысқа» айналады. Программалауда жадыда орын алатын керек емес ақпараттар осылай аталады.

Программада алдын ала нұсқағыштармен сипатталған операторлар бөлімінде төмендегі ақпараттар жазылған:

```
NEW(D); NEW(P);
{Екі бүтін айнымалыға динамикалық жадтан орын бөлінді. Нұсқағыштар
сәйкес мәндерін алды.}
D^ := 3; P^ := 5;
{Динамикалық айнымалыларға мәндер меншіктелді}
P : = D;
{P және D нұсқағыштары 3-ке тең бір шамаға сілтенді}
WriteLn(P^, D ^ ) ; {3 саны екі рет шығады}
```

Осылайша, 5-ке тең динамикалық мән нұсқағышты жоғалтып, қол жетімсіз болды. Дегенмен, ол жадыда орын алады. Бұл «қоқыс» пайда болуының мысалы. P: = D операторын жүзеге асыру нәтижесі 2.33-суретте көрсетілген.

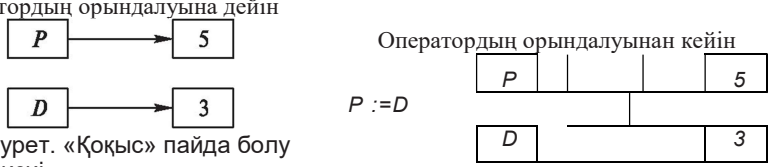
Паскальда қажеттілігі жоғалып кеткен деректерді жоюға мүмкіндік беретін стандартты процедура бар. Оның форматы:

```
DISPOSE(<нұсқағыш>);
```

Мысалы, P^ динамикалық айнымалысы қажет болмаса, DISPOSE(P) операторы оны жадтан жояды, одан кейін P нұсқағышының мәні анықталмаған болады. Жадты үнемдеу әсері үлкен көлемдегі деректерді жою арқылы алынады.

MS DOS операциялық жүйесімен жұмыс істейтін Турбо Паскаль нұсқаларында бір программаның мәліметтеріне 64 Кбайт жады бөлінген. Бұл статистикалық жады аймағы болып табылады.

Ақпараттардың үлкен жиымымен жұмыс істеу қажет болған жағдайда бұл жады жетпей қалуы мүмкін. Динамикалық жадының



2.33-сурет. «Қоқыс» пайда болу механизиі

көлемі үлкенірек (жүздеген килобайт), сондықтан оны пайдалану өңделетін ақпарат көлемін үлкейтуге мол мүмкіндік береді.

Динамикалық деректермен жұмыс программаның орындалуын баяулатады, себебі, шамаға қол жеткізу екі кезеңде жүзеге асырылады: алдымен нұсқағыш анықталады, содан кейін мән анықталады. Жиі кездесетін жағдайдағыдай, «Келеңсіздіктердің сақталу заңы» әрекет етеді, яғни жад көлемінің ұлғаюындағы ұтымдылық, уақыт жоғалтумен өтеледі.

2.40-Мысал. 10 000 саннан тұратын нақты жиым құрып, және оны 0-ден 1-ге дейінгі кездейсоқ сандармен толтыру қажет. Жиымның орташа мәнін есептеу керек. Динамикалық жадты тазалап, бүтін типті 10 000 саннан тұратын жиым құрып, оны -100-ден 100-ге дейінгі кездейсоқ сандармен толтырып, орташа мәнін есептеу қажет.

Есепті шешу программасы:

```
Program Sr;
Const NMax = 10000;
Type Diapazon = 1..NMax;
       MasInt = Array[Diapazon] Of Integer;
       MasReal = Array[Diapazon] Of Real;
Var Plint : ^MasInt;
      PReal : ^MasReal;
      I, Midint : longInt;
      MidReal : Real;
Begin
   MidReal := 0; MidInt := 0;
   Randomize;
   NEW(PReal);
   {Нақты жиымға жад бөлу}
   {Жиымды есептеу және қосындылау}
   For I := 1 To NMax Do Begin PReal^I] := Random;
            MidReal := MidReal + PReal^I]
   End;
   DISPOSE(PReal);{Нақты жиымды жою}
    NEW(PInt); {Бүтін жиымға жадтан орын бөлу}
   { Жиымды есептеу және қосындылау}
   For I : = 1 To NMax Do Begin
            Pint^[I] := Random(200) - 100;
            MidInt := MidInt + Pint^[I]
   End;
   {Орташа мәнді шығару}
   WriteLn('Бүтін орташа тең:', MidInt Div Max);
   WriteLn('Нақты орташа тең: ', (MidReal / NMax):10:6)
End.
```

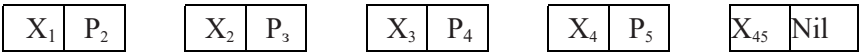
Байланысқан тізімдер. Енді динамикалық жадыда айнымалы өлшеміндегі деректер құрылымын қалай жасауға болатынын қарастырайық.

Келесі мысалды талқылайық. Физикалық тәжірибе барысында құрылғының (мысалы, термометр) көрсеткіштері бірнеше рет алынады және одан әрі өңдеу үшін компьютерлік жадқа жазылады. Қанша өлшеулерді жүргізілетіні алдын-ала белгісіз.

Мұндай деректерді өңдеу үшін сыртқы жадты (файлдарды) пайдаланбасақ, оларды динамикалық жадта орналастыру қажет. Біріншіден, динамикалық жад статистикалық деректерге қарағанда көбірек ақпарат сақтауға мүмкіндік береді. Екіншіден, динамикалық жадта бұл сандарды байланыстырылған тізімде реттеуге болады, ол жиым сияқты сандардың алдын-ала нұсқауын қажет етпейді. Байланыстырылған тізім дегеніміз не? Ол сызбада келесі түрде ұсынылған:

Тізімнің әрбір элементі екі бөліктен тұрады: ақпараттық (x_1, x_2, ...) және келесі элементке нұсқағыш (p_2, p_3, •••). Соңғы элемент Nil нөлдік көрсеткішке ие. Осы типтің байланыстырылған тізімі *бірбағытты тізбек* деп аталады.

Орындалған тапсырманың ақпараттық бөлігінде келесі нақты сандар бар: x_1 - бірінші өлшеудің нәтижесі; x_2 екінші өлшеу нәтижесі; x_3 - үшінші өлшеу нәтижесі; x_4 - соңғы өлшеудің нәтижесі. Байланысқан тізімде жаңа ақпарат пайда болған кезде, ол толықтырыла алады. Толықтырылу тізімнің соңына жаңа элемент қосу арқылы орын алады және соңғы элементтегі Nil мәні тізбектің жаңа элементіне сілтеме арқылы ауыстырылады:



Байланысқан тізім жадта артық орын алмайды. Жад кіріс ақпарат үшін қажетті мөлшерде жұмсалады.

Программада тізбектің элементтерін көрсету үшін аралас деректер типі - жазба пайдаланылады. Біздің мысалда осындай элементтің типі келесідей болуы мүмкін:

```
Type Pe = ^Elem;
     Elem = Record
              T : Real;
              P : Pe
            End;
```

Мұндағы Pe — Elem айнымалы типіне сілтемелі тип. Бұл атаумен аралас тип белгіленген. Ол екі өрістен тұрады: T — температура мәнін сақтайтын нақты шама, және P — Elem типінің динамикалық шамасына нұсқағыш.

Бұл сипаттамада Паскаль тілінің негізгі принциптерінің бірі бұзылады, оған сәйкес кез-келген программа объектісіне оның сипаттамасынан кейін ғана сілтеме жасалуы керек. Шын мәнінде, Pe типі Elem типі бойынша анықталады, ол өз кезегінде Pe типі бойынша анықталады. Дегенмен, Паскальда бұл ережеге сілтеме типіне қатысты бір ғана ерекшелік бар. Программа үзіндісі дұрыс болып саналады.

2.41-Мысал. Мәліметтерді енгізу барысында байланысқан тізімді құру программасы:

```
Type Pe = ^TypElem;
        TypElem = Record
                        T : Real; P : Pe
                  End;
Var Elem, Beg : Pe;
      X : Real; Ch : Char;
Begin
{Тізім басының мекен-жайын анықтау және сақтау}
   NEW(Elem);
   Beg := Elem;
   Elem^.P := Elem;
{Мәндерді тізімге диалогтық түрде енгізу және элементтер арасында байланыс орнату}
While Elem^.P <> Nil Do
Begin
   Write('Санды енгіз:          '); ReadLn(Elem^.T);
   Write('Енгізуді қайталау? (Y/N)'); ReadLn(Ch);
   If (Ch = 'n') Or (Ch = 'N')
   Then Elem^P := Nil
   Else Begin
            NEW(Elem^. P);
            Elem : = Elem^ P
        End
End;
WriteLn('Енгізу аяқталды');
{Алынған сандық тізбекті шығару}
WriteLn('Шығаруды қайта бақылау');
Elem := Beg;
Repeat
   WriteLn(N, ':', Elem^.T : 8 : 3);
   Elem := Elem^.P
```

```
Until Elem = Nil
End.
```

Мұндағы Beg сілтемелі айнымалысы тізбектің басындағы мекенжайды сақтау үшін пайдаланылады. Тізбекті кез-келген өңдеу оның бірінші элементінен басталады. Программа өңделіп жатқан кезде тізбектің қалай жүретіні көрсетіліп отырады (бұл мысалда тізімнің ақпараттық бөлігін басып шығару кезінде).

Бірбағытты тізбек - байланыстырылған тізімнің қарапайым нұсқасы. Сондай-ақ, программалау тәжірибелерінде екібағытты тізбектері (әрбір элементтің келесі және алдыңғы элемент тізіміне нұсқағышы болса) және екілік ағаштар қолданылады. Мұндай деректер құрылымдары динамикалық деректер құрылымдары деп аталады. Осындай мәліметтер құрылымы *динамикалық* деп аталады.

Әдетте программалауда жиі пайдаланылатын деректер құрылым стек болып табылады. *Стек* бұл – басы төбесі деп аталатын және элементтер құрамы үнемі өзгеріп тұратын байланысқан тізбек. Әрбір жаңадан келген элемент тізбектің төбесіне орнатылады. Стектен элементтерді жою төбесінен бастап орындалады.

Стек - сақиналары таяққа бекітілген балалар пирамидасына ұқсас. Әрбір жаңа сақина үстінде орналасады. Сақиналарды жоғарыдан ғана алып тастай аламыз. Стек мазмұнын өзгерту принципі көбінесе «соңғы рет келді - бірінші кетті» бойынша жүзеге асырылады.

2.42-Мысал. Элементті стекке қосу (INSTEK) процедурасын құрастыру қажет және егер осы элементтер символдық шамалар болса, элементті стектен алып тастау (OUTSTEK) керек.

Процедураларда негізгі программада ауқымды түрде жарияланған Pe мәліметтер типі қолданылады:

```
Type Pe = "TypElem;
      TypElem = Record
                     C : Char; P : Pe
                End;
```

Келесі INSTEK процедурасында Sim аргументі стек параметріне кіретін символдардан тұрады. Сілтемелі айнымалы Beg процедураға кірген кезде және одан шығу кезінде стектің төбесіне нұсқағышты қамтиды. Демек, бұл параметр аргумент және процедура нәтижесі болып табылады. Стек бос болған жағдайда, Beg нұсқағышы Nil-ге тең болады.

```
Procedure INSTEK(Var Beg : Pe; Sim : Char);
```

```
Var X : Pe;
Begin New(X);
        X^.C := Sim;
        X^.P := Beg;
        Beg : = X
    End;
```

Келесі OUTSTEK процедурасында Beg параметрі алдыңғы процедурадағы рөлді атқарады, яғни соңғы символды жойғаннан кейін оның мәні Nil болады. Әрине, егер стек бос болса, онда ештеңені өшірудің қажеті жоқ. Flag логикалық параметрі осы жағдайды тануға мүмкіндік береді: егер процедурадан шыққанда Flag = True болса, жою әрекеті орындалды, және егер Flag = False болса, бұл стек бос және ештеңе өзгерген жоқ деген сөз.

```
Procedure OUTSTEK(Var Beg : Pe; Var Flag : Boolean);
Var X : Pe;
Begin
    If Beg = Nil Then Flag
    := False Else Begin
                Flag := True;
                X : = Beg;
                Beg := Beg^.P;
                DISPOSE(X)
            End
End;
```

Бұл процедурадан кейін өшірілген элементтермен жадыда орын босап, артынан «қоқыс» қалмайды.

ЖАТТЫҒУЛАР

1. Программада мынадай сипаттамалар бар.

Var P, Q : ^integer; R : ^Char;

Келесі операторлардың қайсысы қате екенін анықтау және себебін түсіндіру:

а) P := Q; ә) Q := R; ә) P := Nil; б) R := Nil;
в) Q := P^; ғ) P^ := Nil; г) R^ : = P^; ғ) Q^ := Ord(R^);
д) If R <> Nil Then R^ := M1^; е) If Q > Nil Then Q^ := P^;
ж) If P = Q Then Write(Q); з) If Q <> R Then Read(R^.

2. Программада мынадай сипаттамалар бар

```
Type A = ^Char;
```

```
        B = Record F1 : Char; F2 : A End;
Var P : ^B; Q: A;
```

Барлық нұсқағыштар мәні мен динамикалық айнымалылардың мәнін келесі операторлардың орындалуынан кейін анықтау:

```
NEW(Q); Q^ := '7';
NEW(P); P^.F1 := Succ(Q^); P^.F2 := Q;
```

3. Келесі программадағы қателерді табыңдар:

```
Program Example;
Var A, B : ^Integer;
Begin If A = Nil Then Read(A);
   A^ := 5; B := Nil; B^ := 2;
   NEW(B); Read(B^); WriteLn(B, B^);
   NEW(A); B := A; DISPOSE(A); B^: = 4
End.
```

4. Магниттік дискідегі файлда динамикалық жадыға 10 000 саннан тұратын нақты жиымды кіргізудің программасын құру, сондай-ақ оның ішіндегі бірінші максималды элементінің мәнін және нөмірін іздеу.

5. Сандық жиымының өлшемі алдын ала анықталмаған жағдайда (файл өлшемі бойынша анықталады: файл өлшемі динамикалық жадтың өлшемінен аспайды деп есептеледі) төртінші жаттығуды орындаңыз. Динамикалық жадтағы деректерді байланыстырылған тізім ретінде ұйымдастыру.

6. Байланыстырылған тізім – латынның кіші әріптерінен құралған символдық тізбекті береді. Төмендегі процедураны немесе функцияны сипаттаңыз:

а) тізімнің бос болуын анықтау;
б) тізімдегі барлық символдарды бір кіріс символдарын екінші кіріс символдарына алмастыру;
в) бос емес тізімдегі бірінші және соңғы элементтердің орындарын ауыстыру;
г) тізім элементтері алфавит бойынша сұрыпталғанын анықтау;
д) тізімге қай әріп ең көп рет кіретінін анықтау.

4. 2.42-мысалдан INSTEK және OUTSTEK процедураларының жұмысын тексеретін мәтіндік программа құру.

5. Кезек дегеніміз «бірінші келді – бірінші шықты» принципі орындалатын динамикалық құрылым. Элементті кезекке қою және оны кезектен алып тастау процедурасын жазу. Осы процедураны тексеретін мәтіндік программа құру қажет.

2.22. СЫРТҚЫ ІШПРОГРАММАЛАР МЕН МОДУЛЬДЕР.

Стандартты Паскальда жеке компиляцияланған және ары қарай әзірлеуші өзі ғана пайдаланбайтын программалаушының кітапханаларын (Fortran және басқа да жоғары деңгейлі программалау тілдеріне қарағанда) әзірлеу және қолдау құралдары жоқ. Егер программалаушының көбірек тәжірибесі болса және кейбір ішпрограммаларды жаңа қосымшаларды жазу үшін пайдаланса, бұл ішпрограммаларды толығымен жаңа мәтінге қосуы керек болады.

Турбо Паскальда бұл шектеулер ең алдымен, сыртқы ішпрограммаларды енгізу, екіншіден, модульдерді құру және пайдалану болып табылады. Екі жолды да қарастырайық.

Сыртқы ішпрограммаларды енгізу. Бұл жағдайда әрбір процедураның немесе функцияның бастапқы мәтіні бөлек файлда сақталады және қажет болған жағдайда, арнайы компилятор директивасын пайдаланып жасалған программаның мәтініне қосылады.

Бұл әдіс келесі бүтін сан арифметикалық есеп үлгісімен суреттеледі: натурал n саны берілген, бұл санның бірінші және соңғы сандарының қосындысын табу қажет.

Есепті шешу үшін натурал сандардағы цифрлардың санын есептейтін функцияны қолданамыз. Мысалы:

```
Function Digits(N : LongInt): Byte;
Var Kol : Byte;
Begin
    Kol := 0;
    While N <> 0 Do
    Begin
        Kol := Kol + 1;
        N := N Div 10
    End;
    Digits : = Kol
End;
```

Бұл мәтінді файлда .inc кеңейтілімінде сақтайық (Турбо Паскальдағы сыртқы ішпрограммалар кеңейтілімі), Мысалы: digits. inc.

Сондай-ақ натурал санды натурал дәрежеге шығару функциясын жазайық(a^n):

```
Function Power(A,N:LongInt):LongInt;{файл power.inc}
Var I, St : LongInt;
```

```
Begin
    St := 1;
    For I := 1 To N Do
            St := St * A;
            Power := St
End;
```

Сипатталған функцияларды пайдаланып есепті шешу үшін негізгі программаны құрайық:

```
Program Example1;
Var N, S : Integer;
{$I digits.inc} {Жазбадағы цифрлардың сандарын есептейтін digits.inc файлынан сыртқы функцияны қосу}
{$I power.inc} {A санын  N дәрежеге шығаруды есептейтін power.inc файлынан сыртқы функцияны қосу}
Begin
    Write('Натурал санды енгіз:                    '); ReadLn(N);
{ N санының соңғы сандарын анықтау үшін, бұл санды 10-ға бөлудің қалдық бөлігі алынады және бірінші санды анықтау үшін  N санының жазбасындағы цифрлар санынан 1-ге кем дәрежеге шығарылған N саны 10-ға бөлінеді (разрядтарды нөмірлеу 0-ден басталады}
    S := N Mod 10 + N Div Power(10, Digits(N) — 1);
    WriteLn('Қосынды тең:                  ', S)
End.
```

{$I <файл атауы>} компилятор директивасы берілген программа мәтіні орнына көрсетілген атауы бар файл мазмұнын қоюға мүмкіндік береді.

Магниттік дискіде .inc кеңейтілімі бар файлдарды жинақтап ішпрограммалардың жеке кітапханасын қалыптастыруға болады.

Сыртқы процедуралар қарастырылған есептегі функциялар сияқты өздері қолданылатын программада құрылады және қалыптасады.

Модульдерді құру және пайдалану. Модульдердің құрылымын, оларды өңдеу, компиляциялау және пайдалануды қарастырайық.

Модуль – оларды қолданатын программадан тәуелсіз өңделетін және сақталатын ресурстар (функциялар, процедуралар, тұрақтылар, айнымалылар, типтер және т.б.) жиынтығы. Модульдердің сыртқы ішпрограммалардан айырмашылығы – ол процедуралар мен функциялардың үлкен жиынтығын қамтиды. Сонымен қатар, программаны өңдеуге қажетті басқа да ресурстары

бар. Модульдің негізгі идеясы құрылымдық программалау принципі болып табылады. Турбо Паскальдың стандартты модульдері бар (SYSTEM, CRT, GRAPH және т.б.), олар туралы анықтама ақпарат қосымша 1 ...3-те келтірілген.

Модуль құрылымы келесідей:

```
Unit <модуль атауы>; {Модуль тақырыбы}
Interface
    {Интерфейсті бөлік}
Implementation
    {Жүзеге асыру бөлімі}
Begin
    {Модульді инициализациялау бөлімі}
End.
```

Unit қызметші сөзінен кейін, модульдің атауы жазылады, (әрі қарай әрекеттерге ыңғайлы болу үшін) бұл атау осы модуль бар файлмен сәйкес келуі керек. Сондықтан (MS DOS-та жасалғандай) атауда сегізден артық символ болмауы керек.

Интерфейс бөлімі модуль қосылған кезде программалаушыға қол жетімді болатын барлық ресурстарды жариялайды. Ішпрограммаларға бұл жерде тек толық тақырып көрсетіледі.

Implementation бөлімінде бұрын жарияланған барлық ішпрограммалар сипатталады. Сонымен қатар, онда өздерінің тұрақтылары, айнымалылары, типтері, ішпрограммалары болуы мүмкін, олар қосымша сипатқа ие және негізгі ішпрограммаларды жазу үшін пайдаланылады. Интерфейс бөлімінде жарияланған ресурстардан қарағанда, Implementation бөліміндегі қосымша жарияланғандардың барлығы модульдерді қосқанда қол жетімсіз болады. Негізгі ішпрограмманы сипаттау барысында, барлық тақырыпты қайта жазбастан, оның атауын көрсету жеткілікті, содан кейін денені жазамыз.

Инициализация бөлімі (көбінесе болмайтын бөлім) программаны модуль арқылы іске қосылғаннан кейін дереу орындалуы керек операторларды қамтиды.

Модульді әзірлеу мен пайдаланудың мысалын келтірейік. (Төменде қарастырылған есеп өте қарапайым болғандықтан, біз программаның мәтінін толық түсініктемелермен жазамыз.)

2.43-Мысал. Ішпрограммалар жиынын модуль түрінде жүзеге асыру қажет, *P/Q* (мұндағы *P* — бүтін сан, *Q* — натурал сан) қарапайым бөлшектермен келесі амалдар орындалады: қосу, азайту, көбейту, бөлу, қысқарту, *N* дәрежеге шығару (мұндағы *N* — натурал сан), сонымен қатар, қатынас амалдарын жүзеге асыратын

функциялар (тең, тең емес, үлкен немесе тең, кіші немесе тең, үлкен, кіші).

Бөлшек келесі типте болуы керек:

```
Type Frac = Record
                P : Integer;
                Q : 1..32767
            End;
```

Өңделген модульдің көмегімен келесі есептерді шығару керек:

1. А жиымы берілген. Оның элементтері - қарапайым бөлшектер. Барлық элементтердің қосындысы мен арифметикалық ортасын табу керек. Жауабын қысқармайтын бөлшек түрінде көрсету.

2. А жиымы берілген. Оның элементтері - қарапайым бөлшектер. Элементтерді өсу реті бойынша сұрыптау қажет. Ішпрограммалар жиынының модуль түрінде жүзеге асырылуы:

```
Unit Droby;
Interface
  Type
     Natur = 1..High(LongInt);
     Frac = Record
               P : LongInt;
               Q : Natur
            End;
Procedure Sokr(Var A : Frac);
Procedure Summa(A, B : Frac; Var C : Frac);
Procedure Raznost(A, B : Frac; Var C : Frac);
Procedure Proizvedenie(A, B : Frac; Var C : Frac); Procedure
Chastnoe(A, B: Frac; Var C : Frac);
Procedure Stepen(A : Frac; N : Natur; Var C : Frac); Function
Menshe(A, B : Frac): Boolean;
Function Bolshe(A, B : Frac): Boolean;
Function Ravno(A, B : Frac): Boolean;
Function MensheRavno(A, B : Frac): Boolean;
Function BolsheRavno(A, B : Frac): Boolean;
Function NeRavno(A, B : Frac): Boolean;
{Модульді жүзеге асыру бөлімі}
Implementation
{Екі санның ең үлкен ортақ бөлгіші – алдын ала
жарияланбаған қосалқы функция}
Function NodEvklid(A, B : Natur): Natur;
Begin
  While A <> B Do
     If A > B Then
        If A Mod B <> 0 Then A := A Mod B
                        Else A := B
      Else
```

```
            If B Mod A <> 0 Then B := B Mod A
                                    Else B := A;
        NodEvklid := A
End;
{Бөлшекті қысқарту}
Procedure Sokr;
Var M, N : Natur;
Begin
   If A.P <> 0
   Then
   Begin
      If A.P < 0
         Then M := Abs(A.P)
         Else M := A. P; { A.P – LongInt сияқты типтерді
                                қысқарту}
         N := NodEvklid(M, A.Q);
         A.P := A.P Div N;
         A. Q : = A. Q Div N
   End
End;
Procedure Summa; {Бөлшектердің қосындысы}
Begin
   {Бөлшектің бөлімі}
   C.Q := (A.Q * B.Q) Div NodEvklid(A.Q, B.Q);
   {Бөлшектің алымы}
   C.P := A.P * C.Q Div A.Q + B.P * C.Q Div B.Q;
   Sokr(C)
End;
Procedure Raznost; {Бөлшектер айырымы}
Begin
   {Бөлшектің бөлімі}
   C.Q := (A.Q * B.Q) Div NodEvklid(A.Q, B.Q);
   {Бөлшектің алымы}
   C. P := A. P * C. Q Div A. Q - B. P * C. Q Div B. Q;
   Sokr(C)
End;
Procedure Proizvedenie; {Бөлшектерді көбейту}
Begin
   { Бөлшектің бөлімі }
  C. Q : = A. Q * B. Q;
   { Бөлшектің алымы }
  C. P : = A. P * B. P;
  Sokr(C)
End;
Procedure Chastnoe; {Бөлшектерді бөлу}
Begin
    {Бөлшектің бөлімі}
```

```
    C.Q := A.Q * B.P;
    {Бөлшектің алымы}
    C. P : = A. P * B. Q;
    Sokr(C)
End;
Procedure Stepen; {Бөлшекті дәрежеге шығару}
Var I : Natur;
Begin
    C.Q : = 1;
    C.P : = 1;
    Sokr(A);
    For I := 1 To N Do
        Proizvedenie(A, C, C)
End;
Function Menshe;{Бөлшектер арасындағы «<» қатынасы}
Begin
    Menshe : = A. P * B. Q < A. Q * B. P
End;
Function Bolshe; {Бөлшектер арасындағы «>» қатынасы}
Begin
    Bolshe := A.P * B.Q > A.Q * B.P
End;
Function Ravno; { Бөлшектер арасындағы «=» қатынасы}
Begin
    Ravno := A.P * B.Q = A.Q * B.P
End;
Function BolsheRavno; { Бөлшектер арасындағы «>» қатынасы}
Begin
    BolsheRavno := Bolshe(A, B) Or Ravno(A, B)
End;
Function MensheRavno; {Бөлшектер арасындағы «<» қатынасы}
Begin
    MensheRavno := Menshe(A, B) Or Ravno(A, B)
End;
Function NeRavno; { Бөлшектер арасындағы «<>» қатынасы}
Begin
    NeRavno := Not Ravno(A, B)
End;
{Модульді инициализациялау бөлімі}
Begin
End.
```

Модульді өңдеу барысында келесі тәртіптерді есте сақтаған жөн:

1) модульді жобалау, яғни негізгі және қосалқы ішпрограммалар мен басқа да ресурстарды анықтау;

2) модульдің компоненттерін сипаттау;

3) әрбір ішпрограмманы дұрыстау, одан кейін модуль мәтініне

«жабыстыру».

Өңделген программа мәтінін DROBY.PAS файлында сақтайық және Турбо Паскальмен бірге қойылған ішкі компиляторды пайдаланып модульді компиляциялайық. Бұл команда мына түрде болады: TPC DROBY.PAS. Егер мәтінде синтаксистік қателер болмаса, DROBY. TPU файлын аламыз, әйтпесе қатесі бар жолы көрсетілген сәйкес хабарлама шығарылады. Компиляцияның басқа нұсқасы: Турбо Паскаль мәзірінен Compile/Destination Disk, сосын — Compile/Build таңдау.

Енді өңделген модульді қолдану үшін программаға қосуға болады.

Жиым бөлшектерін қосындылау программасы мына түрде жазылады:

```
Program Sum;
Uses Droby;
Var A : Array[1..100] Of Frac;
      I, N : Integer;
      S : Frac;
Begin
   Write(' Жиым элементтері санын енгіз:                    ');
   ReadLn(N);
   S.P := 0; S.Q := 1; {Бастапқы қосынды нөлге тең}
   For I := 1 To N Do {Бөлшектерді енгізіп, қосындылаймыз}
   Begin
      Write(' Алымды енгіз', I, '-ші бөлшек:                ');
      ReadLn(A[I].P);
      Write('бөлімді енгіз ', I, '-ші бөлшек:                 ');
      ReadLn(A[I].Q);
      Summa(A[I], S, S);
   End;
   WriteLn('Жауап:    ', S.P, '/', S.Q)
End.
```

Екінші есепті — жиымды сұрыптауды оқырман өз бетінше шығару керек.

Жоғарыда келтірілген мысалдан көріп отырғандай, модульді қосу үшін Uses қызметші сөзі пайдаланылады, одан кейін модульдің атауы көрсетіледі. Бұл жол бірден программа тақырыбынан кейін жазылады. Егер бірнеше модульдерді қосу қажет болса, олар үтірлермен бөлінеді.

Модуль ресурстарын пайдаланған уақытта программалаушы шақырылған ішпрограммалардың қалай жұмыс жасайтындығы туралы білу міндетті емес. Ішпрограммалар мен олардың сипаттамаларын, яғни олардың атаулары мен параметрлерін білу жеткілікті. Бұл принцип бойынша барлық стандартты модульдермен жұмыс істеу жүзеге асырылады. Сондықтан егер программалаушы модульдерді тек жеке пайдалану үшін жасамаса, ресурстардың қосылу кезінде барлық толық сипаттамаларды орындау қажет.

ЖАТТЫҒУЛАР

1. Осы тақырыптағы Droby модулінің сипаттамаларын қолданып, келесі есепті шешіңдер. А жай бөлшектер жиымы берілген. Барлық бөлшектердің қосындысын тауып, нәтижесін қысқармайтын бөлшек ретінде көрсетіңдер. Барлық бөлшектердің арифметикалық ортасын есептеңдер және нәтижесін қысқармайтын бөлшек түрінде беріңдер.
2. Осы тақырыптағы Droby модулінің сипаттамаларын қолданып, келесі есепті шешіңдер. А жай бөлшектер жиымы берілген. Оларды өсу ретімен сұрыптаңдар.

БАҚЫЛАУ СҰРАҚТАРЫ

1. Паскаль программалау тілін қай жылы және қандай мақсатпен кім ойлап тапты?
2. Паскальдағы программа неше бөліктен тұрады? (Турбо Паскаль нұсқасы)?
3. Паскальдың алфавитіне қандай символдар кіреді?
4. Идентификаторлар қандай ережеге сәйкес құрылады? Идентификаторлардың дұрыс және қате нұсқаларына мысал келтіріңдер.
5. Қандай мәліметтер типтері реттелген деп аталады және оларды қандай ортақ қасиет біріктіреді?
6. Мәліметтердің қарапайым типтерінің құрылымдық типтерден қандай ерекшеліктері бар? Паскаль деректерінің құрылымдық типтерін атаңдар (Турбо Паскаль).
7. Саналатын және интервалды типтер сипаттау бөлімінде қалай беріледі? Мысал келтіріңдер.
6. Қандай жағдайда арифметикалық амалдың нәтижесі бүтін типті шама болады? Мысал келтіріңдер.
7. Паскальда бүтін және нақты мәндегі көрсеткіштерді қалай дәрежеге шығарамыз?
8. Паскальдың қандай логикалық амалдары мен оларды орындау

тәртібін (ақиқаттық кесте) білесіңдер?

9. Паскальдың амалдарын олардың басымдылықтары бойынша орналастырыңдар: арифметикалық, логикалық, қатынас амалдары.
10. Қандай операторлардың көмегімен программада пернетақтадан мәліметтерді енгізу және шығару орындалады? Олардың жазылу форматын көрсетіңдер.
11. Мәліметтерді шығару форматын қандай тәсілмен басқаруға болады? Әр түрлі мәліметтер типтеріне мысал келтіріңдер.
12. Тармақталу операторының синтаксистік диаграммасын сызыңдар және оның орындалу тәртібін көрсетіңдер.
13. Таңдау операторының синтаксистік диаграммасын сызыңдар және оның орындалу тәртібін көрсетіңдер.
14. Шарты алдын ала берілген цикл операторының синтаксистік диаграммасын сызыңдар және оның орындалу тәртібін көрсетіңдер.
15. Шарты соңынан берілген цикл операторының синтаксистік диаграммасын сызыңдар және оның орындалу тәртібін көрсетіңдер.
16. Параметрлі цикл операторының синтаксистік диаграммасын сызыңдар және оның орындалу тәртібін көрсетіңдер.
17. Циклдің үш нұсқасын пайдаланып есеп құрастырыңдар: «Әзірше» циклі, «Дейін» циклі, параметрлі цикл. Программасын құрыңдар.
18. Функция мен процедураның айырмашылығы неде? Функция мен процедураның көмегімен шешілетін есеп құрастырыңдар.
19. процедураның көмегімен шешілетін, бірақ функциямен шешілмейтін есеп құрастырыңдар.
20. Ішпрограмманың формалды және нақты параметрлері дегеніміз не? Параметр-айнымалылар мен параметр-мәндердің сипатталуы мен ішпрограммаға шақыруда айырмашылығы неде? Мысал келтіріңдер.
21. Формалды және нақты параметрлердің сәйкестіктерін айтыңдар.
22. Ауқымды және жергілікті сипаттаулардың айырмашылықтары неде? Ауқымды айнымалылар арқылы ішпрограммаларға мәліметтер жіберу мысалын келтіріңдер.
23. Символдық жол дегеніміз не? Оның жиым символынан айырмашылығы қандай?
24. Паскальда жолдармен жұмыс жасауға арналған қандай амалдар, функциялар және процедуралар бар?
25. Программадағы жүйелі типті шамалар қалай сипатталады (Жиымдар)?
26. Жиымның өлшемі және мөлшерлігі дегеніміз не?
27. Жиым элементтері қалай идентификацияланады?

28. Көпөлшемді жиым элементтері компьютер жадысында қандай тізбекте орналасады?
29. Тұтас жиымға қандай әрекеттер орындауға болады (элементтеріне емес)?
30. Компьютер экранына матрицаның (екіөлшемді жиым) жол бойынша шығарылуына программа үзіндісін көрсетіңдер.
31. Сыртқы және ішкі файлдар дегеніміз не?
32. Қандай оператордың көмегімен файлдық айнымалы (ішкі файл) мен сыртқы файлдың арасында сәйкестік орнатылады? Форматын жазыңдар.
33. Қандай операторлардың көмегімен файлды оқу үшін ашу мен жазуға болады?
34. Файлды жабу операторының орындалуынан кейін қандай жағдай орнайды?
35. Типтелген файлдар мен мәтіндік файлдардың айырмашылығы қандай?
36. Қандай операторлардың көмегімен типтелген және мәтіндік файлдарға енгізу мен шығару орындалады? Мысал келтіріңдер.
37. Файлдардағы тізбектелген және тікелей қол жеткізуді қолданудың айырмашылықтары неде?
38. Мәліметтердің аралас типтері дегеніміз не және олар программада қалай сипатталады? Аралас типті шама қалай сипатталады (жазба)? Мысал келтіріңдер.
39. Жазба элементтері қалай идентификацияланады? Мысал келтіріңдер.
40. Статикалық және динамикалық шамалар және статикалық және динамикалық жадтардың бір-бірінен айырмашылығы қандай?
41. Нұсқағыш дегеніміз не? Программада қалай сипатталады? Динамикалық айнымалылар қалай идентификацияланады?
42. Динамикалық шамаға жадыда қандай оператордың көмегімен орын бөлінеді?
43. Алдын ала динамикалық шамаға жадтан бөлінген орын қандай оператордың көмегімен босатылады?
44. Байланысқан тізім дегеніміз не? Ол қалай жасалады? Стек дегеніміз не?
45. Сыртқы және ішкі ішпрограммалардың айырмашылығы неде? Сыртқы ішпрограммаларды құрудың қандай тәсілдерін білесіңдер?
46. Модуль дегеніміз не? Оның құрылымы қандай?
47. Модульдің ресурстарын қолдану үшін программада қандай әрекет орындау қажет?

АЛГОРИТМДЕРДІ ҚҰРУ ТӘСІЛДЕРІ

Бұл тараудан білесіңдер:

- күрделі программаны құрастырудағы тізбектелген детализация технологияларын Паскаль программалау тілінде жүзеге асыру туралы;
- программаны түзету және тестілеудің толық жүйесін құрастыруды;
- рекурсивті функцияның көмегімен Ханой мұнарасы туралы есеп қалай программаланатынын;
- іріктеу есептері дегеніміз не және барлық нұсқаларды толық іріктеуді қалай оңтайландыруға болатынын;
- іріктеудің рекурсивті алгоритмі және лабиринтті өту есебінің үлгісі бойынша іріктеудің рекурсивті алгоритмі туралы;
- алгоритмдердің күрделілігінің критерийлері туралы;
- мәліметтерді сұрыптаудағы есептің қойылымы туралы;
- қосудың сұрыптау алгоритмі дегеніміз не екенін;
- Э. Хоардың жылдам сұрыптау алгоритмінің мәнісін;
- алгоритмдер күрделілігінің бағасын.

Сендер үйренесіңдер:

- күрделі есептерді программалауда тізбектелген детализация әдісін қолдануды;
- комбинаторлық есептерді шешуде рекурсивті алгоритмдерді қолдануды;
- алгоритмдердің күрделілігін бағалауды.

3.1.

ТІЗБЕКТЕЛГЕН ДЕТАЛИЗАЦИЯ ӘДІСІ

Құрылымдық программалау негіздеріне сәйкес тізбектелген детализация әдісі күрделі алгоритмдерді жобалауды қабылдау болып табылады. Осы әдістің мәні (1.6-тармақты қара) келесідей: алдымен бастапқы тапсырма талданады, одан ішесептер таңдап алынады және осындай ішесептер иерархиясы жасалады (3.1-сурет). Ары қарай негізгі алгоритмнен (негізгі программадан) бастап алгоритмдер (немесе программалар) құрылады, содан кейін қарапайым командалардан тұратын алгоритмдер (ішкі программалар) алынғанша, тізбекті деңгейді тереңдете отырып қосалқы алгоритмдер құрастырылады.

'A+*B*=' түріндегі бастапқы символдық жолды шешу үшін Interpretator программасы келтірілген 2.17-мысалға оралайық.

Interpretator программасының 2.15 бөлімдегіден әлдеқайда әмбебап ететін талаптарын қарастырайық.

1. *a* және *b* операндтары MaxInt шегіндегі көпмәнді бүтін оң сандар болуы мүмкін.

2. Жолдың элементтері арасында, сонымен қатар оның басы мен аяғында бос орындар болуы мүмкін.

3. Мәтінге қарапайым нұсқамен шектелетін синтаксистік бақылау жүргізуді жүзеге асыру қажет: жолдар тек сандардан, амал таңбаларынан, «=» белгісінен және бос орындардан ғана тұруы керек.

4. Семантикалық бақылау жүргізілуі қажет: Жол *'a © b =*' сызбасы бойынша құрылуы керек. Егер қандайда бір элемент жоқ болса немесе олардың тәртібі бұзылса, қате шығады.

5. Операндтар мен нәтиже мәндерінің диапазонын бақылау жүзеге асырылу қажет (мәндер MaxInt шегінен асып кетпеулері керек).

Келтірілген тізімнен программа қарапайым болмайтыны түсінікті. Оны тізбектелген детализация әдісін пайдаланып құратын боламыз. Алгоритмді есептерді шешу кезеңдерінің сызықты тізбегі ретінде қарастырудан бастайық.

1. Жолды енгізу.

2. Синтаксистік бақылау (қажет емес символдардың бар-жоғын анықтау).

3. Семантикалық бақылау (өрнектің дұрыстығын анықтау).

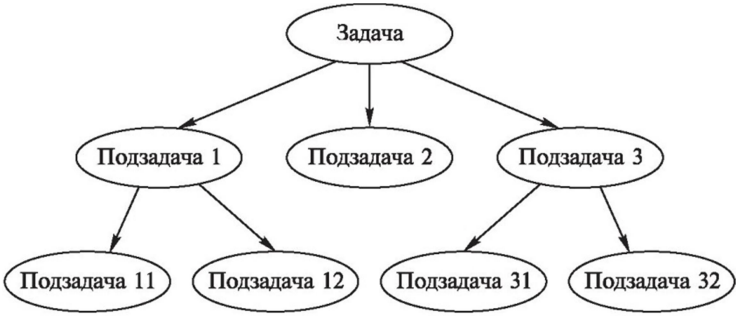


Рис. 3.1. Иерархия подзадач

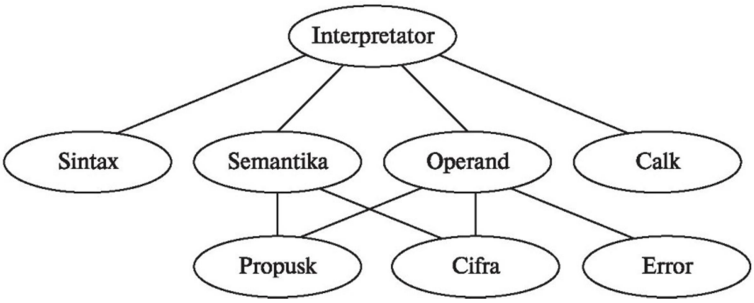
3.1-сурет. Ішесептер иерархиясы
Есеп. 1-ішесеп. 2-ішесеп. 3-ішесеп. 11-ішесеп. 12-ішесеп. 31-ішесеп. 32-ішесеп.

4. Операндтарды бөліп көрсету, олардың рұқсат етілген мәндер диапазонына және бүтін санға айналу сәйкестігін тексеру.

5. Амалдарды орындау, рұқсат етілген мәндер диапазонына сәйкестігін тексеру.

6. Нәтиже шығару.

2 ... 5 кезеңдерін, бірінші деңгейдің ішкі есептері ретінде қарастырамыз және сәйкесінше Sintax, Semantika, Operand, Calc деп атаймыз. Оларды жүзеге асыру үшін келесі ішкі есептерді шешу қажет: артық бос орындарды (Propusk) және символдық цифрді бүтін санға түрлендіру (Cifra). Операндтарды бөліп алу кезінде максималды рұқсат етілген мәндерден (Error) асатын операндтарды алдымен танып алу керек. Жалпылама айтқанда, барлық айтылғандарды сызба түрінде көрсетсек, программалық модульдер құрылымы сәйкес келетін ішкі есептер құрылымын аламыз (3.2-сурет). *Детализацияның алғашқы қадамы.* Алдымен барлық қажет ішпрограммалардың тек тақырыптарын (спецификациясын) бөліп көрсетейік.



3.2-сурет. Программалық модульдер құрылымы

Ішкі программаның орнына түсініктемелер жазамыз (мұндай ішпрограмманың түрі бітеуіш деп аталады). Осыдан кейін негізгі программа бөлігін жазамыз:

```
Program Interpretator;
Type PozInt = 0..MaxInt;
Var Line : String;
    A, B : PozInt;
    Znak : Char;
    Flag : Boolean;
    Rezult : Integer;
Procedure Propusk(Stroka:String; M:Byte; Var N:Byte);
Begin
{------ Stroka жолында қатар тұрған бос орындарды M нөмірлі символдан бастап өткізеді. N айнымалысы бос орын болып табылмайтын бірінші символды алады---}
End;
Function Cifra(C : Char): Boolean;
Begin
{------ Егер C символы сан болса, оның мәні - True болады, және қарсы жағдайда - False болады}
End;
Function Sintax(Stroka : String): Boolean;
Begin
{------ Синтаксистік бақылау. Егер жолда сандардан басқа символдар, амал таңбалары, '=' белгісі мен бос орындар болмаса, онда оның мәні - True болады және қарсы жағдайда - False болады}
End;
Function Semantika(Stroka : String): Boolean;
Begin
```

```
  {------ Жол құрылымын 'a © b =' форматына сәйкестігін тексереді. Егер
  сәйкес болса, оның мәні - True, әйтпесе – False болады}
  End;
  Procedure Operand(Stroka : String; Var A, B : PozInt; Var Z : Char; Var
        Flag : Boolean);
  Begin
  {------ Операндтарды бөліп шығарады. Рұқсат етілген мәндер
  диапазонына сәйкестігін тексереді. Егер тексеру нәтижесі дұрыс болса,
  Flag айнымалысы True мәнін қабылдайды, әйтпесе False мәнін
  қабылдайды. Операндтарды бүтін оң сандарға түрлендіреді. Z  айнымалысына
  амал таңбасы алынады}
End;
Procedure Calc(A, B : PozInt; Znak : Char; Var Rez: Integer; Var Flag : Boolean);
Begin
{ --------Амалдардың нәтижесін есептейді. Нәтиженің MaxInt : MaxInt диапазонында
орналасуын тексереді.  Flag — тексеру нәтижесінің белгісі. Амал нәтижесі Rez
айнымалысына алынады}
End;
Begin {---------- Программаның негізгі бөлігінің басы----------}
    WriteLn(' Өрнекті енгізу:                    ');
    WriteLn;
    Read(Line) ;
    If Not Sintax(Line)
    Then WriteLn(' Рұқсат етілмейтін символдар')
    Else If Not Semantika(Line)
    Then WriteLn('Қате өрнек')
    Else Begin
            Operand(Line, A, B, Znak, Flag);
            If Not Flag
            Then WriteLn('Тым үлкен операндтар')
            Else Begin
                Calc(A,B,Znak,Rezult,Flag);
                If Not Flag
                Then WriteLn('Үлкен нәтиже')
                Else WriteLn(Rezult)
                End
            End
    End.
```

Детализацияның екінші қадамы — ішпрограммаларды құру:

```
Procedure Propusk(Stroka:String; M:Byte; Var N:Byte);
Begin
    N := M;
```

```
    While (Stroka[N] = '                    ') And (N < Length(Stroka))
Do
      N := N + 1
End;
Function Cifra(C : Char): Boolean;
```

```
Begin
    Cifra := (C >= '0') And (C <= '9')
End;
Function Sintax(Stroka : String): Boolean;
Var I : Byte;
Begin
    Sintax := True;
    For I := 1 To Length(Stroka) Do If Not ((Stroka[I] = ' ') Or Cifra(Stroka[I])
                    (Stroka[I] = '+') Or (Stroka[I] = '—' (Stroka[I] = '*') Or
(Stroka[I] = '=' Then Sintax := False End;
Function Semantika(Stroka : String): Boolean; Var I : Byte;
Begin
    Semantika := True;
    Propusk(Stroka, 1, I);
    If Not Cifra(Stroka[I])
    Then
            Begin
                    Semantika := False;
                    Exit
            End;
    While Cifra(Stroka[I]) Do I : = I + 1;
    Propusk(Stroka, I, I);
    If Not((Stroka[I] = '+') Or (Stroka[I] = '—') Or (Stroka[I] = '*'))
    Then
    Begin
            Semantika := False;
            Exit
    End;
      I : = I + 1;
      Propusk(Stroka, I, I);
      If Not Cifra(Stroka[I])
      Then
              Begin
                      Semantika := False;
                      Exit
              End;
      While Cifra(Stroka[I]) Do
```

Or)
Or
))

```
                I := I + 1;
        Propusk(Stroka, I, I);
        If Stroka[I] <> '='
        Then
                Begin
                        Semantika := False;
                        Exit
                End;
        I := I + 1;
        Propusk(Stroka, I, I);
        If I < Length(Stroka)
        Then
                Begin
                        Semantika := False;
                        Exit
                End
End;
Procedure Operand(Stroka : String; Var A, B : PozInt; Var Z: Char; Var Flag : Boolean);
Var P, Q : String;
    I : Byte;
    Code : Integer;
Function Error(S : String): Boolean;
{Егер S жолында сан болса, онда функцияның мәні максималды саннан асатын
True болады, және қарсы жағдайда – False болады}
        Const Lim = '32767';
        Var I: Byte;
Begin
        If Length(S) > 5 Then Error : = True Else
        If Length(S) < 5
                Then Error := False Else Error : =
                S > Lim
End;
Begin
        Flag : = True;
        I : = 1;
        Propusk(Stroka, I, I);
        Q := '';
        While Cifra(Stroka[I]) Do
                Begin
                        Q := Q + Stroka[I];
```

```
                    I := I + 1
            End;
    If Error(Q)
    Then
            Begin
                    Flag := False;
                    Exit
            End;
    Propusk(Stroka, I, I);
    Z := Stroka[I];
    I := I + 1;
    Propusk(Stroka, I, I);
    P := '';
    While Cifra(Stroka[I]) Do
            Begin
                    P := P + Stroka[I];
                    I := I + 1
            End;
    If Error(P)
    Then
            Begin
                    Flag := False;
                    Exit
            End;
    Val(Q, A, Code);
    Val(P, B, Code)
{Val стандартты процедурасы цифрлық жолды сәйкес санға түрлендіреді; Code –
қате коды}
End;
Procedure Calc(A, B : PozInt; Znak : Char;
    Var Rez : Integer; Var Flag : Boolean);
    Var X, Y, Z: Real;
Begin
    Flag := True;
    Y : = A;
    Z : = B;
    Case Znak Of
            '+': X := Y + Z;
            '-': X := Y - Z;
            '*': X := Y * Z
    End;
    If Abs(X) > MaxInt
    Then
```

```
                False;
                        Flag := Exit
            End;
    Rez     : Round(X)
End;
```

Error функциясы Operand процедурасында ішкі түрде анықталған, себебі ол тек осы процедурада қолданылатынын ұмытпаған жөн. Оның басқа программалық модульдер үшін қажеттілігі жоқ.

Соңында ішкі программалардың мәтіндерін негізгі программамен біріктіріп, Interpretator программасының жұмыс нұсқасын аламыз. Осыдан кейін оны компьютерге енгізе аламыз.

Программаны түзету және тестілеу. Жазылған программаның бірден дұрыс шыққанына сену қиын. (бұл мүмкін болса да, бірақ есептің күрделенуімен дұрыс болу ықтималдылығы төмендейді). Жұмыстың соңғы және нақты нәтижесіне жеткенге дейін программа *түзету* процесінде жүреді.

Программадағы қателер «тілдік» және алгоритмдік болуы мүмкін. Біріншісі әдетте, Паскальдан компиляторды табуға көмектеседі. Бұл программалау тілінің ережелерін бұзумен байланысты қателер. Олар *компиляциялық уақыт қателіктері* деп те аталады, себебі олар дәл компиляция уақытында анықталады. Компилятордың өзі сол немесе басқа формада қолданушыға қатенің сипаты мен программа мәтініндегі орны туралы хабарлама береді. Кезекті қатені түзеткеннен кейін қолданушы компилляцияны қайталайды. Осылайша, бұл барлық қателер жойылғанға дейін жалғасады.

Алгоритмдік қателер әртүрлі салдарға әкелуі мүмкін. Мысалы, мүмкін емес әрекеттер: нөлге бөлу, теріс санның квадрат түбірі, жол шекарасынан тыс индекстің шығуы және т.б. Бұл *орындау уақытының қателіктері* және олар программаны орындаудағы тоқтатуларға әкеп соғады. Әдетте, мұндай қателерді табуға көмектесетін жүйелік программалық құралдар бар.

Сондай-ақ, алгоритмдік қателер программаны орындауға кедергі келтірмейтін жағдайлар болуы мүмкін. Программа соңына дейін орындалады, кейбір нәтижелер жасалады, бірақ олар дұрыс болмай шығады. Оның нақты түрде алгоритмін түзетіп және дұрыстығына талдау жасау үшін *тестілеу* жүргізіледі, яғни нәтижелері алдын ала белгілі есепті шешу нұсқасы – тест орындалады. Әдетте, бір тестілеу нұсқасымен программаның дұрыстығын дәлелдеу мүмкін емес. Сондықтан программалаушы тест жүйесін ойластырып және сәйкесінше бүкіл программаны толық тексеру үшін тестілеу

жоспарын жасауы керек.

Жоғарыда айтылғандай, кез-келген нұсқадағы сапалы программа дұрыс емес болып аяқталмауы керек. Interpretator программасының тесттері, егер бастапқы жол дұрыс енгізілсе, әрдайым дұрыс нәтижелер алынып, ал қателер (синтаксистік, семантикалық, мәндердің рұқсат етілген диапазоннан шығуы) болған жағдайда тиісті хабарламаларды көрсетеді. Біздің программаның тестілеу жоспары, мысалы, келесідей болуы мүмкін:

№	Енгізу	Нәтижесі
1	25 + 73 =	98
2	5 274 - 12 315 =	- 7 041
3	614 * 25 =	15 350
4	25,5 + 31,2 =	Рұқсат етілмеген символдар
5	5 = 6 - 1	Қате өрнек
6	72 315 - 256 =	Өте үлкен операндтар
7	32 000 * 100 =	Үлкен нәтиже

Барлық тестілеудің дұрыс өтуі программаның дұрыстығының ең қажетті шарты болып табылады. Алайда бұл шарт әрдайым жеткіліксіз екенін ескертеміз. Программа неғұрлым күрделі болса, тестілеу жоспарын жасау соғұрлым қиынырақ болады. Тәжірибе көрсеткендей, тіпті «фирмалық» программалардың өзінде амалдарды орындау барысында қателер табылған, яғни тестілеу программасы өте маңызды және сонымен бірге өте күрделі әрекет болып табылады.

3.2. РЕКУРСИВТІ ӘДІСТЕР

Рекурсивтік әдістердің мәні есептің өз-өзіне әрекеті болып табылады. Жоғарыда айтылғандай, Паскальда рекурсивті алгоритмдерді іске асыратын функциялар мен процедураларды анықтаудың рекурсивті мүмкіндігі бар. Дегенмен, есепті шешу үшін рекурсивті алгоритм жасау оңай емес.

Әдебиетте «Ханой мұнарасы» атауымен белгілі классикалық есепті қарастырайық (3.3 сур.).

3.3-сурет. Ханой мұнарасы туралы есеп

Тұғырда (А деп атайық) өзінің табанынан төбесіне дейін азаятын дискілерден тұратын пирамида бар. Осы пирамиданы сол қалпында В алаңына жылжыту қажет. Бұл жұмысты орындау кезінде келесі шектеулерді сақтау қажет:

- пирамиданың үстіңгі жағынан бастап бір-бірден дискілерді алу;
- дискті бөліктің табанына немесе өзінен үлкен дискке ғана қоюға болады;
- көмекші ретінде С тұғырын пайдалануға болады.

«Ханой мұнарасы» деген атау аңызға байланысты, оған сәйкес, ежелгі уақытта Ханой храмының монахтары 64-диск мұнарасын осы ережелерге сәйкес жылжытуға міндеттенді. Монахтар әлі де жұмыс істеп жатыр және ұзақ уақыт бойы жұмыс істейді!

Бұл есепті екі диск үшін шешу қиын емес. Дискінің орын ауыстыруларына белгілеу енгізейік, мысалы, А-дан В тұғырына дейінгі ауыстырылу - А ^ В болсын, осы жағдайдың алгоритмін жазып алайық:

$$A \text{ ^ C; } A \text{ ^ B; } C \text{ ^ } B,$$

яғни тек үш қадам жасалды.

Үш диск үшін есепті шешу алгоритмі ұзағырақ болады:

$$A \text{ ^ B; } A \text{ ^ C; } B \text{ ^ C; } A \text{ ^ B; } C \text{ ^ A; } C \text{ ^ B; } A \text{ ^ B, яғни}$$

енді жеті қадам болды.

K дисктер үшін *N* қадамдарының санын анықтау келесі рекуррентті формула бойынша есептеледі:

$$N(1) = 1; N(k) = 2 \times N(k - 1) + 1.$$

Осылайша, қадамдар санын аламыз N(10) = 1023, N(20) = 104 857, а N(64) = = 1 844 6744 073 709 551615, яғни Ханой монахтарына осыншама қадам жасау керек деген сөз. Осы санды оқып көріңдер.

Енді машина монахтар жұмысының алгоритмін есептейтін және оны кез-келген N дисктер саны үшін шығаратын программа құрастырайық. А тұғырында N дискілері бар деп есептейік. Онда есепті шешу алгоритмі төмендегідей болады:

1. Егер $N = 0$ болса, онда ешқандай әрекет жасаудың қажеті жоқ.
2. Егер $N > 0$ болса, онда:

- N - 1 дискілерді *С-ға* В арқылы ауыстыру;
- А –дан *В (А* ^ В)-ға дискті ауыстыру;
- N - 1 дисктерді *С* –дан *В* –ға А арқылы ауыстыру.

Алгоритмінің екінші пунктін орындау барысында дисктерді ауыстырудың үш кезеңін 3.4–суретте көрсетілгендей кезекпен аламыз.

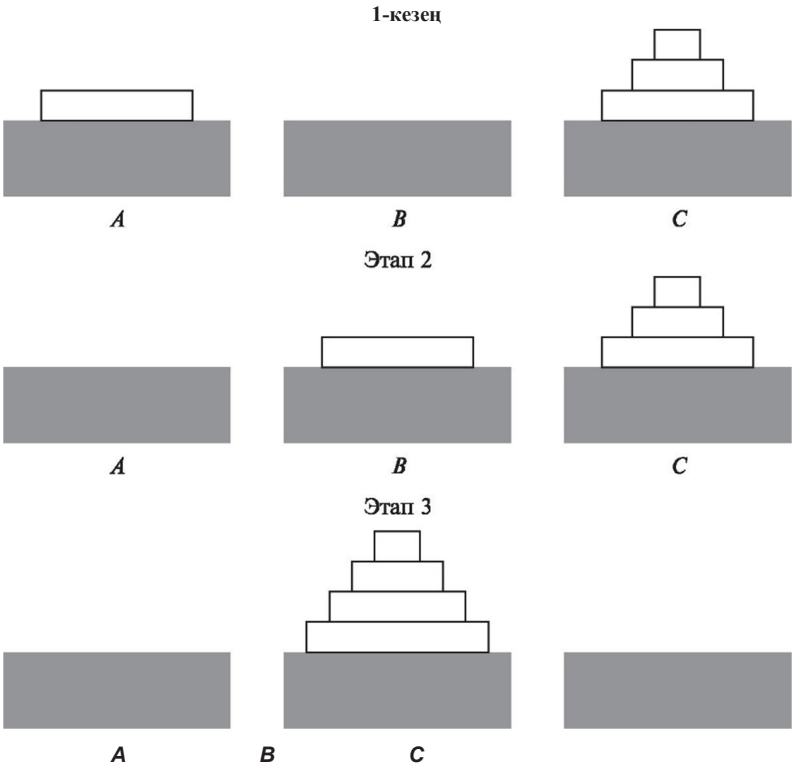
Алгоритмнің сипаттамасы нақты рекурсивтілікте болып табылады. N дискілерінің ауыстырылуы N-1 дискілерін ауыстыру арқылы сипатталады. «Алгоритмнің өзіне рекурсивті сілтемелерінің осы тізбегінен шығу жолы қайда?» деген сұрақ туындаса, жауабы оның тривиалды мазмұны оғаш болып көрінетін алгоритмнің бірінші пунктінде сипатталған.

Енді Паскальда осы есепті шешуге арналған программаны құрайық. Бұл есепте Hanoi рекурсивті процедурасы бар. Оның орындалуы $N = 0$ болған жағдайда тоқтайды. Бұл процедураны шақыру барысында тұғырлардың 1, 2, 3 нөмірлерімен белгіленген нақты атаулары пайдаланылады. Сондықтан ауыстырылу тізбегі келесі түрде болады:

1 ^ 2; 1 ^ 3; 2 ^ 3 және т.с.с.

Есептің программасы мына түрде болады:

```
Program Monahi;
Var N: Byte;
Procedure Hanoi(N: Byte; A, B, C: Char);
Begin
   If N > 0 Then
   Begin Hanoi(N - 1, A, C, B);
            WriteLn(A, '=>', B);
            Hanoi(N - 1, C, B, A)
   End
End;
Begin
   WriteLn('Дисктер санын енгіз:          ');
   ReadLn(N);
   Hanoi(N, '1',     '2',    '3') End.
```



3.4-сурет. Дисктердің ауыстырылуының үш кезеңі

(2-кезең. 3-кезең.)

Бұл ең ғажайып программалардың бірі! Оны машинада орындап көріңдер. Қадамдардың N санының артуымен қатар оның қалай өзгеретінін қадағалаңдар, ол үшін программаға қадамдар санағышын қосуға болады және соңында оның мәнін немесе қадамдардың реттік нөмірлері бойынша шығаруға болады.

3.3. ІЗДЕУ ЕСЕПТЕРІНДЕГІ ІРІКТЕУ ӘДІСТЕРІ

Ақпараттарды іздеу кезіндегі туындайтын мәселелермен байланысты кейбір есептерді қарастырайық. Бұл программалаудың классикалық әдебиетінде толығырақ жазылған есептердің ауқымды

класы болып табылады.

Іздеу есебінің негізгі мәні мынада: ЭЕМ жадысында сақталған ақпараттан белгілі бір шарттарды (критерийлерді) қанағаттандыратын қажетті мәліметтерді таңдау.

Осыған ұқсас есептерді біз қарастырған болатынбыз. Мысалы, сандық жиымдағы максималды санды іздеу, мәліметтер файлындағы қажетті жазбаны іздеу және т.б. Мұндай іздеу мәліметтер құрылымының барлық элементтерін іріктеу арқылы жүзеге асырылады.

Құрылымның барлық элементтеріне қарап, *толық іріктеу* деп аталатын - «фронталды» іздестіру әдісі бар. Алайда, ол сирек қолданылады.

Мысал қарастырайық. Бірөлшемді *X* жиымында нақты сандар осінде жатқан *N* нүктелердің координаталары берілген. Нүктелер нөмірленген. Олар *X* жиымындағы тізбекке сәйкес келеді. Координата басынан ең алыс нүктедегі бірінші нүктенің нөмірін анықтау қажет.

Толық іріктеу арқылы шешілетін модулі бойынша X жиымының ең үлкен элементінің санын анықтау, біз үшін белгілі есеп екенін түсіну оңай:

```
Const N = 100;
Var X : Array[1..N] Of Real; I, Number :1..N;
Begin
{ ----- X жиымын енгізу ----- }

Number : = 1;
For I := 2 To N Do
     If Abs(X[I]) > Abs(X[Number])
     Then Number := I;
     WriteLn(Number)
End.
```

Мұнда бірөлшемді жиым элементтерінің толық іріктеуі бір циклдық құрылымды қолдану арқылы орындалады.

Ал енді сол бастапқы деректермен бірге ара қашықтықтары ең үлкен екі нүктелерді анықтаймыз.

Іріктеу әдісін қолданып, бұл есепті келесі жолмен шешуге болады: N берілгендерден барлық жұп нүктелерді іріктеп, олардың ең үлкен (координаталық айырымдардың ең үлкен модулі) аралығындағы нөмірлерін анықтау. Мұндай толық іріктеу екі қабаттасқан цикл арқылы жүзеге асырылады:

```
Number1 : = 1;

Number2 : = 2;
```

```
For I : = 1 To N Do For J := 1 To
    N Do
        If A b s ( X [ I ]  — X [ J ] > Abs(X[Number1] - X[Number2])
        Then
        Begin
            Numberl := I;
            Number2 := J
        End;
```

Есепті бұлай шешу рационалды емес болып табылады. Мұндағы әрбір нүктелер жұптары екі рет қаралады, мысалы, $I = 1$, $J = 2$ және $I = 2$, $J = 1$. $N = 100$ болғанда циклдердің орындалуы 100 х 100 = 10000 рет қайталанады.

Егер қайталауды алып тастасақ, программаның орындалуы тездетіледі. Сонымен қатар, I және J мәндерінің сәйкестігін алып тастау қажет. Сонда циклдердің қайталану саны $N = \frac{N-1}{2}$ болады, яғни $N = 100$ болғандағы қайталау саны 4 950 болады.

Алдыңғы есептегі қайталауды алып тастау үшін ішкі циклдің басындағы 1-ді $(I + 1)$ - ге өзгерту қажет. Сондағы программа келесідей болады:

```
Number1 : = 1;
Number2 : = 2;
For I := 1 To N Do
    For J := I + 1 To N Do
        If A b s ( X [ I ]  —  X [ J ] ) > Abs(X[Number1] —
X[Number2])
        Then
        Begin
            Numberl := I;
            Number2 : = J
        End;
```

Қарастырылған алгоритм нұсқасын *қайталаусыз іріктеу* деп атайық.

Ескерту. Әрине бұл есепті басқа жолмен де шешуге болар еді. Бірақ дәл бұл жағдайда іріктеу алгоритмін қолдану қажет болды. Сызықта емес жазықтықта немесе кеңістікте орналасқан нүктелер үшін іріктеу алгоритмін құруды іздеу аздаған мәселелерді туғызады.

Бұл жағдайда алгоритм ұзындық айнымалылары бар үш қабаттасқан циклдерден тұрады:

```
For I := 1 To N Do
   For J := I + 1 To N Do
         For K := J + 1 To N Do
               If X[I] + X[J] + X[K] = 10 Then
               WriteLn(X[I], X[J], X[K]);
```

Енді X жиымынан қосындылары 10-ға тең болатын сандардың барлық топтарын таңдау керек делік. Топтардағы цифрлардың саны 1-ден N-ге дейінгі кез-келгені болуы мүмкін. Бұл жағдайда іріктеу нұсқаларының саны күрт артып, алгоритмнің өзі тривиалды емес болып кетеді.

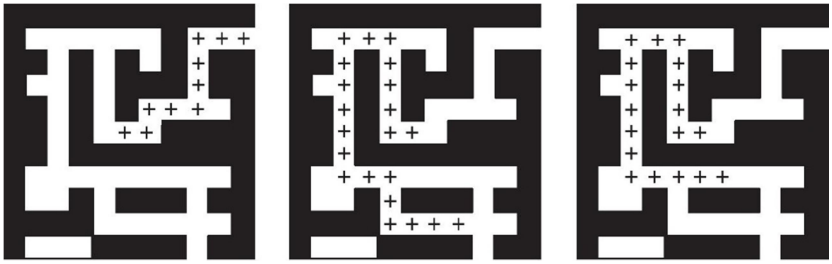
Бұл нені білдіреді? Машина тез жұмыс істейді! Сонда да санап көрейік. N объектілерінің әртүрлі топтарының саны (соның ішінде бос) - 2^N құрайды. N = 100 болғанда, 2^{100} және 10^{30} болады. Демек, секундына миллиардтаған амалдар орындайтын жылдамдықтағы компьютер бұл іріктеуді шамамен 10 жыл бойы орындайды. Тіпті орын ауыстыру қайталауын алып тастаған жағдайда, тіпті осындай іріктеу алгоритмі іс жүзінде оны мүмкін етпейді.

Мұндай есептердің практикалық шешілу жолы, іріктеуден болашағы жоқ нұсқалы есептердегі шартты алып тастау тәсілдерін табу болып табылады.

Кейбір есептер үшін оны төменде сипатталған алгоритмді қолдану арқылы жүзеге асыруға болады.

Лабиринттен өтіп кету есебінің мысалын келтіре отырып, *қайтымды іріктеу* алгоритмін қарастырамыз (3.5-сурет).

Лабиринт берілген. Одан шығу жолын табу керек. Ішінде тек горизонталь және вертикаль бағытта ғана орын ауыстыруға болады. Лабиринттің ортаңғы нүктесінен шығудың барлық нұсқалары 3.5-суретте(а, ә, б) көрсетілген.



а *б* *в*

3.5-сурет. Лабиринттен шығудың барлық нұсқалары

Бұл есепті шешу программасын құру барысында екі сұрақ туындайды: берілгендері қалай ұйымдастыруға болады және алгоритмді қалай құрамыз?

Лабиринт формасы туралы ақпаратты символдық типтегі *N x* N өлшемді LAB квадрат матрицасында сақтаймыз. Мұндағы, *N* —тақ сан (орталық нүкте болу үшін). Лабиринт пішініне тор орнатылады. Оның әрбір торына қабырға немесе жол сәйкес келу керек. Матрица торда келесідей толтырылады: шығу элементтеріне бос орын сәйкес келеді, ал қабырғаға белгілі бір символ, мысалы, «М» сәйкес келеді. Лабиринттегі қозғалыстар «+» таңбасымен белгіленсін. Мысалы, Сурет - 3.5, *б* LAB матрицасының толтырылуына сәйкес келеді. Ол келесі формадағыдай көрсетілген:

M	M	M	M	M	M	M	M	M	M	M
M		+	+	+			M			
M	M	+	M	+	M		M		M	M
M		+	M	+	M	M	M		M	M
M	M	+	M	+	M					M
M	M	+	M	+	+		M	M	M	M
M	M	+	M	M	M	M	M	M	M	M
M	M	+	+	+						M
M			M	+	M	M	M		M	M
M	M	M	M	+	+	+	+	+		M
M				M	M	M	M	+	M	M

Талдау үшін бастапқы деректер – лабиринт пішіні («+» таңбасы жоқ бастапқы LAB матрицасы); талап етілген нәтиже – орталық нүктеден шығудың барлық мүмкін траекториялары (әрбір жолға траекториясы «+» мен белгіленген LAB матрицасы) шығарылады.

Қайтымды іріктеу алгоритмін сонымен қатар *таңдау әдісі* деп те атайды және оның мәні келесі түрде болады:

1. Траекторияның әрбір келесі нүктесінен қозғалыс мүмкін бағыттары бірдей ретпен қарастырылады. Көру әр уақытта сағат тіліне қарсы (оң-жоғарғы-сол-төменгі) орындалатынына келісеміз; алғашқы көрінген бос көрші ұяшыққа қадам жасалады; қадам жасалатын ұяшық крестпен белгіленеді.

2. Егер кезекті тордан шығу жолы болмаса, бір қадам артқа қайтып, сол нүктеден басқа қарастырылмаған жолдарды таңдау қажет. Артқа қайтқан уақыттағы ұяшық бос орынмен белгіленеді.

3. Қадам жасалған кезекті ұяшық лабиринттің шетіне келсе

(шығу жолы), табылған жол жауап ретінде басып шығарылады.

Бұл есептің программасын тізбектелген детализациясы әдісімен құрамыз:

```
Program Labirint;
Const N = 11; {лабиринт өлшемі N X N ұяшық}
Type Field = Array[1..N,                    1..N] Of Char;
Var LAB : Field;
Procedure GO(LAB : Field; X, Y : Integer);
Begin
{лабиринт центрінен шетке жол іздеу – әрбір табылған жол басып шығарылады}
End;
Begin
    {Лабиринтті енгізу}
    GO(LAB, N Div 2 + 1, N Div 2 + 1) {Ортасынан бастаймыз}
End.
```

GO процедурасы (X, *Y)* координаталы ұяшыққа қадам жасайды. Егер осы ұяшық лабиринттен шығу орнында тұрса, онда жүріп өтілген жол баспаға шығарылады, қарсы жағдайда алдын ала орнатылған тізбекке сәйкес көрші торға қадам жасалады. Егер ұяшық бітеу болса, онда бір қадам артқа жылжиды. Осы барлық айтылғандарға байланысты процедураға рекурсивті мінездеме тән екенін байқаймыз.

Алдымен процедураның жалпы сызбасын детализациясыз жазып көрейік:

```
Procedure GO(LAB : Field; X, Y : Integer);
Begin
    If {торкөз (X, Y) бос}
    Then
    Begin
            { (X, Y) ұяшығына қадам жасау}
            If {Лабиринттің шетіне жеттік}
            Then {Табылған жол басып шығарылады }
            Else { Шартты тізбекке байланысты көрші ұяшыққа қадам жасауға
            тырысу} {Бір қадам артқа жылжу}
    End
End;
```

Табылған қозғалыс траекторияларын шығару үшін PRINTLAB процедурасы құрылады.

Нақты түрдегі программа мынадай болады:

```
 Program Labirint;
Const N = 11; {N X N тор көздердің лабиринт өлшемі}
Type Field = Array[1..N,                    1..N] Of Char;
Var LAB : Field; X, Y : Integer;
Procedure PRINTLAB(LAB : Field);
{Лабиринттегі табылған жолды басып шығару}
Var X, Y : Integer;
Begin
  For X := 1 To N Do Begin
     For Y := 1 To N Do
          Write(LAB[X, Y]);
          WriteLn
  End;
  WriteLn End; {Басып
шығарулар}
Procedure GO(LAB : Field; X, Y : Integer);
Begin
  If LAB[X, Y] = '                ' {Егер тор көз бол болса}
  Then
  Begin
     LAB[X, Y] := '+'; {Қадам жасалады}
     If (X = 1) Or (X = N) Or (Y = 1) Or (Y = N) {Шеті}
     Then
          PRINTLAB(LAB){Табылған жол басып шығарылады}
     Else
          Begin {келесі қадамды іздеу}
               GO(LAB, X + 1, Y);
               GO(LAB, X, Y + 1);
               GO(LAB, X - 1, Y);
               GO(LAB, X, Y - 1)
          End;
     LAB[X, Y] := '                ' {Артқа қайту}
  End
```

```
End; {GO процедуралары}
Begin {Негізгі программа}
          {Лабиринтті енгізу}
  For X := 1 To N Do
  Begin
    For Y := 1 To N Do
          Read(LAB[X, Y]);
          ReadLn
  End;
  GO(LAB,N Div 2 + 1, N Div 2 + 1){Ортасынан бастаймыз}
End.
```

Бұл процедураны рекурсивті анықтауды қолданудағы тағы бір әдемі программа мысалы.

Бұл программаның алгоритм сызбасы қайтымды іріктеу әдісіне тән. Осындай алгоритмдер, мысалы, шахмат тақтасының фигуралары жүрісі міндеттерін шешеді және сол сияқты оңтайлы таңдаулары бар көптеген есептерді (жолаушының есептері, жолдардың оңтайлы құрылысы, және т.б.) шешуге көмектеседі.

Ескерту. LAB жиымын қолданғандықтан параметр-мән ретінде GO процедурасында ЭЕМ-де программаны орындау барысында жадымен кейбір қиындықтар туындауы мүмкін. Мұндай жағдайда жиымның ауқымды беруіне көшкен дұрыс.

3.4.

МӘЛІМЕТТЕРДІ СҰРЫПТАУ ТӘСІЛДЕРІ ЖӘНЕ АЛГОРИТМДЕРДІҢ КҮРДЕЛІЛІГІ

Алгоритмдер күрделілігі. Біріншіден, біз алгоритмнің күрделілігін бағалау мәселесін талқылаймыз. Дәстүрлі түрде алгоритмнің күрделілігінің дәрежесін ол қолданатын компьютерлік ресурстардың көлемі бойынша бағалауға болады: процессор уақыты мен жады. Осыған байланысты алгоритмнің *уақыттық және көлемдік күрделілігі* ұғымдары енгізіледі.

Уақыттық күрделілік программаның интерактивті режимін немесе нақты уақыттағы басқару есептерін қамтитын тапсырмалар үшін өте маңызды болып табылады. Көбінесе техникалық құрылғыларды басқаратын программаны жасайтын программалаушы есептеулердің дәлдігі мен программа уақыты арасындағы ымыраға ие болуы керек. Әдетте, дәлдік көбейіп, уақыттың ұлғаюына әкеледі.

Өңделетін деректер көлемі ЭЕМ-нің жад көлемінің шегіне жеткен кезде программаның көлемдік күрделілігі критикалық дәрежеге жетеді. Заманауи компьютерлер үшін осы мәселенің маңыздылығы олардың ЖЕСҚ көлемінің ұлғаюына және көп деңгейлі сақтау құрылғылары жүйесін тиімді пайдалануға байланысты төмендейді. Бұл программада өте үлкен, іс жүзінде шектеусіз жады (виртуалды жад) бар, ал негізгі жадтың жетіспеушілігі дискімен алмасуға байланысты жұмыстың белгілі бір баяулауына әкеледі. Осындай алмасу кезінде жоғалған уақытты барынша азайтуға мүмкіндік беретін әдістер бар: дискіден негізгі жадқа қажетті мәндерді тасымалдауға мүмкіндік беретін кэш-жады мен программа командаларын аппараттық қарауды алға жылжыту. Айтылғандардан, сыйымдылықтың күрделілігін барынша азайту басты мәселе емес, сондықтан біз алгоритмдердің негізгі уақыттық күрделілігіне қызығушылық танытатынымыз туралы қорытынды жасауға болады.

Программаның орындалу уақыты орындалатын амалдар санына пропорционалды. Әрине, уақыт бірлігінде де (c) ол процессор жұмысының жылдамдығына тәуелді. Алгоритмнің уақыттық күрделілік көрсеткіші компьютердің техникалық сипаттамаларына инвариантты болу үшін, оны қатысты бірліктермен өлшейміз. Әдетте уақыттың күрделілігі орындалатын амалдар санымен бағаланады.

Әдетте, алгоритмнің уақыттық күрделілігі бастапқы деректерге, оның шамасына және көлеміне тәуелді болып табылады. Алгоритмнің уақыттық күрделілік параметрін T_a таңбасы бойынша және бастапқы деректерді V әрпімен сипаттайтын кейбір сандық параметрді белгілесек, уақыттық күрделілікті $T_a(V)$ функциясы ретінде ұсына аламыз. V параметрін таңдау есептің шешілетін түріне немесе шешуге қолданылатын алгоритм түріне байланысты болады.

3.1-Мысал. Оң бүтін санның факториалын есептеудегі алгоритмнің уақыттық күрделілігін бағалау керек.

Есепті шешу программасы:

```
Function Factorial(x : Integer) : Integer;
Var m, i : Integer;
Begin
   m := 1;
   For i := 2 To x Do m := m * i;
   Factorial := m
End;
```

Программада х-тің берілген мәнінде орындалатын амалдардың жалпы санын есептеп шығарайық. M:=1 операторы бір рет орындалады; циклдің денесі (екі амал: көбейту және меншіктеу) (х - 1) рет орындалады; Factorial := m меншіктеуі бір рет орындалады. Егер амалдардың әрқайсысын күрделілік бірлігі ретінде алсақ, алгоритмнің уақыттық күрделілігі 1 + 2 (х-1) + 1 = 2х болады, мұндағы х мәні *V* параметрі ретінде қабылдануы керек. Уақыттың күрделілік функциясы келесі түрде болады:

$$T_a(V) = 2V.$$

Бұл жағдайда, уақыттық күрделілік сызықты түрде берілгендер параметрлеріне — Factorial функциясы аргументі мәніне тәуелді.

3.2-Мысал. Екі вектордың скаляр көбейтіндісін есептеу қажет: $A = (a_1, a_2, ..., a_к)$, $B = (b_1, b_2, ..., b_k)$.

Есептеу программасы:

```
AB := 0;
For i := 1 To k Do AB := AB + A[i] * B[i];
```

Бұл есепте енгізілетін берілгендер көлемі $n = 2k$. Орындалатын амалдар саны $1 + 3k = 1 + 3(n/2)$. Бұл жерде $V = k = n/2$ қабылдауға болады. Алгоритм күрделілігі *A* және *B* вектор элементтер мәніне тәуелсіз. 3.1-мысалдағыдай, уақыттық күрделіліктің мәліметтер параметрлеріне сызықты түрде тәуелділігі жөнінде айтуға болады.

Екі теориялық мәселені әдетте алгоритмнің уақыттық күрделілігі параметрімен байланыстырады. Бірінші мәселе келесі сұрақтың жауабын табу болып табылады: есептің шешілуін алгоритмді дамыту арқылы уақыттық күрделілігі мәндерінің қандай шегіне жетуге болады? Бұл шектеу есептің өзіне байланысты және, сонымен қатар ол өзінің сипаты болып табылады.

Екінші мәселе алгоритмдерді уақыттық күрделілігімен жіктеуге байланысты. V өскен сайын $T_a(V)$ функциясы да өседі. Ол қаншалықты жылдам өседі? T_a-ның V-ға сызықтық тәуелді (мысалы, қарастырылған мысалдарда), квадраттық тәуелділік пен полиномды деп аталатын жоғары дәрежедегі тәуелділігі бар алгоритмдер бар. Күрделілігі кез-келген полиномнан да жылдам өсіп бара жатқан алгоритмдер бар. Алгоритм зерттеушілері жиі шешетін мәселе келесі сұрақ болып табылады: бұл есептер үшін полиномды алгоритм мүмкін бола ма?

Мәліметтерді сұрыптаудағы есептердің қойылымы. Алгоритмдер екі категорияға бөлінеді: сандық және сандық емес. *Сандық алгоритмдер* математикалық есептеулерге арналған: формулалар бойынша есептеу, теңдеулерді шешу, мәліметтерді статистикалық өңдеу және т.б. Мұндай алгоритмдерде негізгі

өңделген мәліметтер типі сан болып келеді. *Сандық емес алгоритмдер* әр түрлі берілгендермен жұмыс жасайды: символдық, графикалық, мультимедиялық ақпараттар. Бұл категорияға жүйелік программалау алгоритмдері (трансляторлар, операциялық жүйелер), мәліметтер базасын басқару жүйелері, желілік программалық қамтамасыз етулер және т.б. жатады.

Екінші категориядағы программалық өнімдер үшін деректерді сұрыптау алгоритмдері жиі пайдаланылады, яғни белгілі бір негізде ақпараттың сұрыпталуы. Негізінен бүкіл программаның тиімділігі осы алгоритмдердің, әсіресе олардың орындалу жылдамдығының тиімділігіне байланысты болып келеді.

Ішкі сұрыптау алгоритмдері - ішкі жадыда сұрыптау және сыртқы сұрыптау алгоритмдері - сұрыптау файлдары бар. Бұдан әрі біз ішкі сұрыптауды қарастырамыз.

Әдетте, сұрыпталған деректер жиымда орналасады. Ең қарапайым жағдайда бұл сандық массивтер. Дегенмен, сандық емес алгоритмдер үшін массивтерді сұрыптау да жиі кездеседі. Мәндері бойынша сұрыпталған өріс сұрыптау кілті деп аталады және ол әдетте сандық типке ие. Мысалы, сұрыпталған жазбалар жиыны екі өрістен тұрады: өнімнің атауы және қоймадағы тауардың саны. Паскаль программасы бойынша оның сипаттамасы мынадай форманы құрайды:

```
Const n = 1000;
Type Tovar = Record
                                    Name : String;
                              Key : Integer
                         end;
Var A : Array[1..n] Of Tovar;
```

Мәліметтерді сұрыптау A[i]. key кілті мәнінің өсу реті немесе кему реті бойынша жүзеге асырылады.

Ары қарай программалардың барлық мысалдарында программада бұрын сипатталған сипаттамалар программада ғаламдық түрде болады және олардың әрекеттер аймақтары сұрыптау процедураларына таралады.

«Көпіршікті» әдіс бойынша сұрыптау алгоритмі 2.16-тарауда қарастырыған болатын. Енді қарапайым қосу арқылы сұрыптау және жылдам сұрыптауды талқылаймыз.

Қарапайым қосу арқылы сұрыптау алгоритмі. Алгоритмнің кейбір кезеңдерінде жиымның сол жақ 1-ден (*i*-1)-ші элементтері сұрыпталған, ал оның оң бөлігі *i*-ші элементтен бастап *n*-ші элементіне дейін сұрыпталмайды. Алгоритмнің келесі қадамы

жиымның сол жағын бір элементке кеңейту және тиісінше, оң жағын азайту болып табылады. Бұл үшін оң жақтың бірінші элементі (i индексі) сол жақтағы сәйкес орынға реттілігі сақталған күйде орналастырылады. Процесс бір элементтен тұратын A[1] жиымының сол жағынан басталады және оның оң жағы босағанда процесс аяқталады.

Қарапайым қосу арқылы сұрыптау программасы келесідей:

```
Procedure StraightInsertion;
Var i, j : Integer; x : Tovar;
Begin              For i := 2 To n Do
Begin x := A[i]; {x айнымалысында сол бөліктегі тиісті орынға қою керек мән
сақталады}
        j := i - 1; {Сол бөліктің оң жағы}
        While (x.key < A[j].key) and (j >= 1) Do
        Begin A[j + 1]              := A[j]; j := j - 1; {Жиымның сол бөлігінде оңнан
        солға қарай A[i] элементі қосылу керек орынға дейін «Тесіктің» жылжуы}
        End;
        A[j + 1]: = x { сол жақтағы «тесікке» A[i] қосу }
    End
End;
```

Енді қарапайым қосу арқылы сұрыптау алгоритмінің күрделілігін бағалайық. Уақыттық күрделілігі сұрыпталатын жиым өлшеміне және оның бастапқы күйіне, яғни элементтердің реттелуіне байланысты. Егер бастапқы жиым кілттер мәндерінің қажетті реті бойынша сұрыпталса, уақыттық күрделілік минималды болады (бұл жағдайда өсу ретімен). Уақыттық күрделіліктің максималды мәні бастапқы жиымның қарама-қарсы реттілігіне сәйкес келеді, яғни бұл жағдайда негізгі мәндер кілт мәндерінің кему ретімен орналасқан болса, бұл максималды мән болады. Әдетте, сұрыптау алгоритмдерінің уақыттық күрделілігі элементтерді беру санымен бағаланады.

Алгоритмнің минималды уақыттық күрделілігін бағалайық. Егер жиым сұрыпталған болса, While циклі денесі бірде-бір рет орындалмайды. Процедураның орындалуы келесі цикл жұмысына жалғасады:

```
For i := 2 To n Do Begin
    x := A[i];
    j := i - 1;
    A[j + 1]        := x
End;
```

For циклінің денесі *n*-1 рет орындалғандықтан, элементтің берілу саны $M_{min}= 2(n - 1)$ және кілттерді салыстыру саны $C_{min}=n–1$ болады.

Алгоритмнің күрделілігі бастапқы жиым кему ретімен реттелген болса, барынша максималды болады. Содан кейін әрбір элемент A [r] жиымының басына «қуылады», яғни бірінші орынға орнатылады. While циклі i = 2 үшін бір рет орындалады, i = 3 болса 2 рет, және n-1 үшін i = n рет орындалады.

Мәндерді беру жазбасының жалпы саны

$$M_{max} = 2(n - 1) + \sum_{i=2}(i - 1) = 2(n - 1) + n(n - 1)/2 = (n^2 + 3n - 4).$$

Нақты жағдайда қолайлы болып *алгоритмнің күрделілігіне орташа баға* беріледі. Оны есептеу үшін бастапқы жиымның барлық элементтері кездейсоқ сандар және олардың мәндері олардың нөміріне ешқандай қатысы жоқ деп санауға тиіспіз. Бұл жағдайда x.key<A [j] шартын тексеру нәтижесі While циклінде кездейсоқ болады.

Әрбір нақты *i* мәніне арналған While циклінің орташа саны i/2-ге тең болады, яғни кезекті элементке сәйкес орын табылғанша жиымды жартысына дейін қарап шығу керек. Содан кейін берудің орташа санын есептеудің формуласы (орташа күрделілік балл) мынадай болады: Орташа санды беруді есептеу формуласы келесі түрде болады (күрделіліктің орташа бағасы):

$$M_{op} = 2(n - 1) + \sum_{i=2} i = 2(n - 1) + (n - 2)(n - 1)/4 = (n^2 + 9n - 10).$$

Алгоритм күрделілігін максималды және орташа бағалау n – сұрыпталатын жиымның өлшемі бойынша параметрге қатысты квадратты (екінші дәрежелі полином) болып табылады.

Жылдам сұрыптау алгоритмі. Бұл алгоритмді Э.Хоар әзірледі. Ол жылдам сұрыптау алгоритмінде қолданылады:

- сұрыпталатын жиымды екі бөлікке бөлу — сол және оң бөлік;
- сол жақтың барлық элементтері оң жағындағы элементтерден аспайтын екі бөліктің (ішкі жиымдар) өзара реттелуі;
- ішкі жиым жалпы жиым қалай сұрыпталса, ол да солай сұрыпталатын рекурсия.

Жиымды екі бөлікке бөлу үшін, кілттің кейбір тосқауыл мәнін

таңдау керек. Бұл мән тек бір шартты қанағаттандыруы тиіс: берілген жиымның мәндер диапазонында болуы керек (ең аз және ең үлкен мәндер арасында). Кедергі ретінде жиымның кез-келген элементі кілтінің мәнін таңдауға болады, мысалы, бірінші, соңғы немесе ортасында орналасқан.

Ары қарай кедергіден кіші болатын сол жақ ішкі жиымда барлық кілтті элементтер, ал оң жақта – кедергіден үлкен кілтті элементтер болатындай етіп жасау қажет. Содан кейін, жиымды солдан оңға қарай қарап шығып, «кедергіден» кіші болатын бірінші кілтті элементтің позициясын табу керек және оңнан солға қарай қарап, «кедергіден» аз бірінші кілтті элементті табу керек. Осы мәндердің орындарын алмастырып, алмастырылатын келесі жұп элементтеріне дейін қарсы қозғалысты жалғастырамыз. Процесс сол және оң көріністің индексі сәйкес келгенше жалғасуы керек. Сәйкес келетін орын екі өзара реттелген ішкі жиым арасындағы шекара болады. Әрі қарай, алгоритм әр ішкі жиымға (сол және оң) рекурсивті түрде жеке-жеке қолданылады. Нәтижесінде, одан ары бөлінбейтін өзара реттелген бір элементті *n* жиымнан жиынтық аламыз. Бұл жиынтық толықтай реттелген жиымды құрайды, яғни сұрыптау аяқталады.

Мәліметтерді жылдам сұрыптау программасы келесі түрде болады:

```
Procedure Quicksort;
Procedure Sort(L, R : Integer);
Var i, j, bar : Integer; w : Tovat;
Begin
    i : = L; j : = R;
    bar := A[(L+R)div2]. key; {Кедергіні орнату}
    Repeat
        {Алмастыру үшін элементті сол жақтан іздеу}
        While A[i].key < bar Do i : = i + 1;
        { Алмастыру үшін элементті оң жақтан іздеу }
        While A[j]. key > bar Do j := j - 1;
        {Элементтерді алмастыру және жиым бойынша жылжу}
        If i <= j Then Begin
            w := A[i]; A[i] : = A[j]; A[j] := w; i := i + 1; j := j - 1
        End;
    Until i > j;
    {Өзара реттелген ішжиымдар қалыптасты} {Сол жақ ішжиымды сұрыптау}
        If L < j Then Sort(L,j);
            {Оң жақ ішжиымды сұрыптау}
            If i < R Then Sort(i,R);
End; {Sort}
```

```
Begin
   Sort(1,n)
End;{Quicksort}
```

Жылдам сұрыптау алгоритмінің уақыттық күрделілігін зерттеу - өте мұқият жұмыс, сондықтан оны біз қарастырмаймыз. Тек осы талдаудың соңғы нәтижесін береміз. Жиымның *n*-өлшемінің функциясы ретінде Т-ның уақытша күрделілігі келесі формула бойынша шамаланған тәртіпте көрсетіледі:

$$T(n) = O(nln(n)).$$

Мұнда математикалық O(x) белгілеуі - x ретілігінің шамасы пайдаланылады. Сондықтан жылдам сұрыптау алгоритмінің уақыттық күрделілігі nln (n) реттілігінің шамасы болып табылады, ол $n^{1\ 2}$ қарағанда *n* оң бүтін сандары үшін аз екендігін ұмытпайық (қарапайым қосудың сұрыптау алгоритмі n^2 ретті күрделік). Сонымен, n мәні неғұрлым үлкен болса, бұл айырмашылық соғұрлым үлкен болады. Мысалы:

$n = 10,$	$n^2 = 100,$	$nln(n) = 23,03;$
$n = 100,$	$n^2 = 10000,$	$nln(n) = 460,52.$

ЖАТТЫҒУЛАР

ә) Бір-бірінен ең алыс жатқан екі нүктені табыңдар;

б) Ең үлкен периметрлі үшбұрыштың төбелерінде жатқан үш нүктені табыңдар;

в) екі жақын нүктені тауып, олардың ортасындағы кесінді, ішінде барлық нүктелер жатқан шеңбердің радиусы болып табылатын және осы нүктелердің қайсысы осы шеңбердің центрі болатынын анықтаңдар.

2. Labirint программасын лабиринт центрінен шығуға ең қысқа жолды табатындай етіп өзгертіңдер.

3. Шахмат тақтасындағы жылқы әрбір өрісті тек бір рет басып өтетіндей тақтаның барлық өрістерінен өтетіндей программа құрыңдар.

4. Шахмат тақтасында 8 ферзьді бір-біріне қауіп төндірмейтіндей етіп орналастыратын программа құрыңдар.

---

1. *N* нүктелердің жазықтықтағы декарттық координаталары берілген. Келесі есептерді шешу программасын құрыңдар:

а) Бір-біріне ең жақын жатқан екі нүктені табыңдар;

БАҚЫЛАУ СҰРАҚТАРЫ 3 4 5 6 7 8 9 10 11 12

3 Программалауда тізбектелген детализация әдісінің мәнісі қандай?
4 Программаны түзету қалай орындалады?
5 Ханой мұнарасы туралы есептің мәні неде? Ханой мұнарасы есебіне программалауда рекурсияны қалай қолдануға болады?
6 Толық іріктеу есебі дегеніміз не?
7 Қайтымды іріктеу алгоритмі дегеніміз не? Лабиринттен өтуге мысал келтіріңдер.
8 Алгоритмдер күрделілігінің қандай критерийлері бар?
9 Полиномды уақыттық күрделілік алгоритмдері дегеніміз не?
10 Алгоритмдерді сұрыптаудағы қарапайым қосудың негізгі идеясы қандай?
11 Алгоритмді жылдам сұрыптау қалай жүзеге асады?
12 Жылдам сұрыптау алгоритмінің күрделілік реті қандай?

4-тарау

ОБЪЕКТІГЕ-БАҒЫТТАЛҒАН ПРОГРАММАЛАУ

Бұл тарауда білесіңдер:

- объектіге-бағытталған программалау дегеніміз не екенін;
- объектіге-бағытталған программалау тарихын;
- объектіге-бағытталған программалаудың негізгі ұғымдарын — объект, объект жағдайы, объект әдісі, инкапсуляция, класс, мұрагерлік, полиморфизм;
- Object Pascal тіліндегі объектілерді сипаттау және қолдану ережелерін ;
- Delphi біріктірілген программалау ортасының тарихы мен тағайындалуын;
- Delphi интерфейсінің негізгі элементтерін;
- Delphi-дің негізгі компоненттері мен қасиеттерін;
- оқиғалы-басқарушы программалау дегеніміз не және ол Delphi-де қалай жүзеге асырылатынын;
- Delphi-де программалық қосымшаларды өңдеу кезеңдерін;
- класстардың иерархиялары туралы.

Сендер үйренесіңдер:

- Object Pascal-да қарапайым программалардың объектілерін сипатталуы мен қолданылуын;
- Delphi программалау ортасында жұмыс істеуді;
- Delphi қосымшалары үшін қарапайым интерфейс құруды;
- интерфейс объектілері үшін олардың қасиеттерін анықтау және әдістерді программалауды;
- Delphi құралдарының көмегімен Windows –тың қарапайым қосымшаларын жасауды.

4.1. ОБЪЕКТІГЕ-БАҒЫТТАЛҒАН ПРОГРАММАЛАУ ДЕГЕНІМІЗ НЕ?

ОБП тарихы. Программалаудағы объектілі көзқарастың негізін қалаушылар норвегиялық Оле Йохан Дал және Кристен Нюгорт - Симул тілінің авторлары. Симула тарихы 1962 жылы Монте-Карло әдісінің программалық жасақтамасын модельдеуге арналған Simulation Language жобасы арқылы басталды. 1965 жылы осы жобаның авторлары деректерді өңдеуді процедуралармен біріктіруді ойлап тапты. Тілге мультипроцессорлық жұмыстың имитациясы және моделдеудің жаңа құралдары, сондай-ақ «класс» және «объект» терминдері енгізілді. Сонымен қатар, мұрагерлік технологиясы пайда болды, яғни Симула шығарушылары тілге класс атауларын префикс түрінде көрсете отырып, ортақ қасиетті әртүрлі класстарды қолдану мүмкіндігін ұсынды.

Алан Кэй программаны өңдеудің жаңа тұжырымдамасын енгізді, оған сай тізбекті түрде орындалатын нұсқаулар жиынтығы асинхронды хабар алмасу арқылы бір-бірімен байланысатын объектілердің көпөлшемді өзара әрекеттесу ортасына ауыстырылуы мүмкін болды. Бұл идеяны ол бастапқыда Biological System деп аталған және BASIC-те модельденген SmallTalk жаңа тілінде жүзеге асырды, бұл тіл содан кейін ассемблерге енгізілді. «Объектіге-бағытталған программалау» термині SmallTalk тілінде алғаш рет қолданылды. Тілдің ең танымал нұсқасы SmallTalk 80 болып табылады.

1980 жылы Бьерн Страуструп процедуралық программалау тілін Си класстар концепциясымен толықтырды, ал 1983 жылы бұл жаңа өнімге С ++ деген нақты атауы берді. Содан бері С ++ объектіге-бағытталған программалаудың ең танымал тілі болды.

Кейінірек объектіге-программалау мүмкіндіктері басқа да тілдерге енгізіле бастады. Турбо Паскальда ОБП құрал-саймандары 5.5 нұсқасынан бастап пайда болды. ОБП элементтері Фортран-90 нұсқасынан бастап Фортранға енгізілді. Визуалды әдістерімен ОБП технологиясын біріктірген қазіргі заманғы программалау тілдері Delphi, Visual Basic, Java.

ОБП-дың негізгі ұғымдары. Объектіге-бағытталған парадигманың негізгі идеясы берілгендер мен оларды өңдейтін процедураларды біртұтас – *объектіге* біріктіру болып табылады. *Объектіге-бағытталған программалау* — бұл программаның әрқайсысы белгілі бір классты (мысалы, ерекше типтегі деректер) іске асыруға және мұрагерлік қағидаттарына негізделген иерархияны қалыптастыратын объектілер жиынтығы түрінде

программаны ұсынуға негізделген программалау методологиясы. Сонымен қатар, объект барлық қасиеттердің жиынтығы мен олардың ағымдағы мәндерімен қоса, берілген объект әрекеттері үшін қолайлы жиынтық ретінде сипатталады.

ОБП енгізудің және қолданудың әр түрлі ерекшеліктеріне әртүрлі көзқарастардың болуына қарамастан, үш негізгі (базалық) ұғым өзгеріссіз қалады:

- инкапсуляция (Encapsulation);
- мұрагерлік (Inheritance);
- полиморфизм (Polymorphism).

Бұл ұғымдар ОБП-дың негізінде жатыр.

Программалауға процедуралық көзқараста түпкілікті нәтижеге қол жеткізу үшін алгоритмнің әрбір қадамын сипаттау керек. Объектіге-бағытталған тәсілмен келіп түскен шақыруға жауап беру объекттің жұмысы болып табылады. Ол үшін есепті стандартты формаға қойып, жауап алу жеткілікті.

Объект үш бөліктен тұрады:

- атауы;
- жағдайы (айнымалы жағдайы);
- әдістер (амалдар).

Жалпылама анықтама беруге болады: ОБП объекті — бұл айнымалы жағдайлары мен олармен байланысқан әдістер (амалдардың) жиынтығы. Олар объектінің қоршаған ортамен өзара қатынасын анықтайды.

Объект әдістері - бұл жарияланулары объектінің сипаттамасына енгізілген және әрекеттерді орындайтын процедуралар мен функциялар. Объектінің жағдайын шақырулармен басқару мүмкіндігі оның әрекетін анықтайды. Бұл әдістер жиыны *объектінің интерфейсі* деп аталады.

Инкапсуляция — бұл деректерді басқаратын мәліметтер мен әдістерді біріктіретін және сыртқы кедергілерден немесе дұрыс пайдаланбаудан қорғайтын механизм болып табылады. Бұл әдістер мен деректер біріктірілген кезде, объект пайда болады.

Инкапсуляцияны қолдану арқылы объектіге қатысты деректерді және осы деректерге тікелей қол жеткізген кезде туындайтын қателерден қорғаймыз. Сонымен қатар, осы механизмді қолдану программа кодындағы мүмкін қателерді анықтауға жиі көмектеседі, бұл олардың іздеу және түзету процесін айтарлықтай жеңілдетеді. Инкапсуляция дегеніміз деректерді жасыру, яғни оларды қорғау дегенді білдіреді. Инкапсуляцияны қолдану объектінің элементтеріне қолжетімділіктің төмендеуіне

әкеледі, бұл оның ішкі элементтерін (айнымалы мәндерін) өзгерту әдістерін шақырумен байланысты. Дегенмен, қазіргі заманғы есептеуіш техниканың даму деңгейінде, бұл шығындардың тиімділігі маңызды рөл атқармайды.

Мұрагерлік - бұл бір объект басқа объектінің қасиеттерін иелене алатын және тек соған тән ерекшеліктерді қоса алатын процесс. Нәтижесінде объект типтерінің иерархиясы жасалады, онда деректер өрістері мен «бабалар» әдістері автоматты түрде деректердің өрістері және «ұрпақтар» әдісі болып табылады.

Мұрагерліктің мағынасы мен әмбебаптылығы - жаңа объектіні әр жолы қайтадан сипаттауға тура келмейді, тек оның «ата-анасын» (базалық класс) көрсете саламыз және жаңа класстың ерекшеліктерін сипаттай аламыз. Нәтижесінде, жаңа объект «ата-ана» класының барлық қасиеттеріне және өзінің жеке ерекшеліктеріне ие болады.

Мысалы, төрт дөңгелекті барлық көліктер үшін әмбебап болатын «Көлік құралы» базалық класын жасауға болады. Бұл класс, дөңгелектердің қалай қозғалатынын, бұрылатынын, тоқтайтынын және т.б. әрекеттерінің барлығын «біледі». Содан кейін, осы классқа негізделген «Жеңіл автомобиль» деп аталатын класс жасай аламыз. Жаңа құрылған класс «Көлік құралдары» класынан мұраға қалғандықтан, олардың барлық басты ерекшеліктері де мұраға қалдырылды, яғни дөңгелектердің қалай қозғалуын қайта сипаттаудың қажеті жоқ, тек олар автомобильдерге тән ерекшеліктерді қосамыз. Сонымен қатар, «Көлік құралы» класына негізделген «Жүк машиналары» класын құруға болады, яғни олардың өздеріне тән ерекшеліктерін сипаттап және осы жаңа класс негізінде жүк көліктерінің арнайы ішкі класын сипаттауға және т.б. әрекеттерді жасауға болады. Осылайша, алдымен қарапайым шаблон, содан кейін қиындату мен нақтылау арқылы біртіндеп күрделі шаблондар жасалады.

Полиморфизм - бұл әр түрлі техникалық есептерді шешу үшін бірдей атауды пайдалануға мүмкіндік беретін қасиет. Полиморфизм иерархиядағы типтерге бір атаулы әдіс әр түрлі туыстық байланысқан объектілерге қолданылуы мүмкін әдістерді анықтайды. Жалпы, полиморфизм тұжырымдамасы - «бір интерфейс - әдістер жиынтығы», яғни, полиморфизм программалардың күрделілігін төмендетуге көмектеумен қатар, ол ортақ класстар әрекеттеріне бір интерфейсті қолдануға мүмкіндік береді. Осыған орай, жағдайға байланысты нақты әрекетті таңдау компиляторға жүктеледі.

Мысалы, машина қалай қозғалу керектігін, қалай өзгеретінін,

оның сигналдары қалай көрінетінін және т.б. сипаттайтын «Автомобиль» класы болсын. Мұнда «Берілістерді алмастыру» әдісі сипатталған. Айта кетейік, «Автомобиль» класының бұл әдісі автоматты беріліс қорабын сипаттайды. Механикалық жылдамдықты беріліс қорабы бар «Спорттық автокөлік» класын сипаттау қажет. Әрине, жаңа класс үшін барлық әдістерді қайта сипаттауға болады. Дегенмен, оның орнына «Спорттық автокөлік» класы «Автокөлік» класынан мұраланғанын көрсетуімізге болады, сондықтан ол «ата-ана» класы үшін сипатталған барлық қасиеттер мен әдістерге ие. Қажетті жалғыз істеу керек әрекет - бұл жылдамдықты механикалық қорапты берілу үшін «Берілістерді алмастыру» әдісін қайта жазу керек. Нәтижесінде, сіз «Берілістерді алмастыру» әдісін шақырғанда, ата-ана класс әдісі емес, жаңадан жасалған әдіс орындалады.

ОБП жұмыс істеу механизмін бұл жағдайларда осылайша сипаттауға болады, бір немесе басқа класс әдісін шақыру барысында алдымен класстың өзінен әдісті іздестіру қажет. Егер әдіс табылса, ол орындалады және оны іздеу аяқталады. Егер әдіс табылмаса, «ата-ана» класын шақырамыз және одан аталған әдісті іздеуіміз керек. Егер әдіс табылса, онда ол орындалады іздеу тоқтатылады, ал табылмаса, іздеу иерархиялық ағашпен түбіріне (жоғарғы класс) қарай іздеуді одан әрі жалғастырады. Бұл мысал ерте байланыстыру механизмі деп аталады.

4.2. ТУРБО ПАСКАЛЬДАҒЫ ОБЪЕКТІЛЕР

Объектілерді сипаттау. Инкапсуляция. Турбо Паскальда объектілерді «object» сөзімен сипаттаймыз. «Object» типі — бұл өрістер мен әдістерді қамтитын мәліметтер құрылымы. Объектілі типтің сипатталуы келесі түрде жазылады:

```
Type        <объект типінің идентификаторы> = Object
      <өріс>;

      <өріс>;
      <әдіс>;

      <әдіс>;
End;
```

Өрісте деректердің типі мен атауы сақталған. *Әдістер* — бұл

объектілі типтің декларациясы ішінде жарияланған процедуралар мен функциялар. Сонымен қатар, олар объектілерді құратын және жоятын ерекше процедуралардан тұрады (конструкторлар мен деструкторлар). Объектілі типтің сипаттауы ішінде әдісті жариялау тақырыптан ғана тұрады (модульдегі Interface бөлімі сияқты).

4.1-Мысал. «Алым мен бөлімнің ЕҮОБ», «Қысқару» және «Натурал дәреже» әдістерімен «Жай бөлшек» объектісін сипаттайық:

```
Type Natur = 1..32767;
     Frac = Record P : Integer; Q : Natur End;
     Drob = Object A : Frac;
                   Procedure NOD(Var C : Natur);
                   Procedure Sokr;
                   Procedure Stepen(N : Natur; Var C : Frac);
     End;
```

Объектілі типті сипаттау инкапсуляция қасиетін көрсетеді.

Сипатталған объектімен оның әдістерін жүзеге асыратын және көрсетілген әдістерді шақыратын жұмысты көрсетейік (бұл жағдайда кейбір қосымша әдістер қажет болады):

```
Type Natur = 1..MaxInt;
     Frac = Record P : Integer; Q : Natur End;
{---------Объектілі типті сипаттау---------------------------}
     Drob = Object
                 A : Frac;
                 Procedure Vvod; {Бөлшекті енгізу}
                 Procedure NOD(Var C : Natur); {ЕҮОБ}
                 Procedure Sokr;
                 Procedure Stepen(N : Natur; Var C : Frac);
                    Procedure Print; {Бөлшекі шығару}
                 End;
{---------Объект әдістерін сипаттау-----------------------}
Procedure Drob.NOD;
Var M, N : Natur;
Begin M := Abs(A.P); N := A.Q;
  While M <> N Do
  If M > N
  Then If M mod N <> 0 Then M := M mod N Else M := N Else If N mod M <> 0
  Then N := N mod M Else N := M; C := M
End;
Procedure Drob.Sokr;
Var N : Natur;
Begin
```

```
    If A. P <> 0
    Then Begin
            Drob.NOD(N);
            A.P := A.P div N; A.Q := A.Q div N
    End
    Else A. Q := 1 End;
Procedure Drob.Stepen;
Var I : Natur;
Begin
    C. P := 1; C. Q := 1;
    For I := 1 To N Do
            Begin C.P := C.P * A.P; C.Q := C.Q * A.Q
            End;
End;
Procedure Drob.Vvod;
Begin
    Write('Бөлшектің алымын енгіз:          '); ReadLn(A.P);
    Write(' Бөлшектің бөлімін енгіз:          '); ReadLn(A. Q);
End;
Procedure Drob.Print;
Begin WriteLn(A.P, '/', A.Q) End;
{-------- Негізгі программа---------------------- }
Var Z : Drob; F : Frac;
Begin
    Z.Vvod; {Бөлшекті енгізу}
    Z.Print; {Енгізілген бөлшекті басып шығару}
    Z.Sokr; {Енгізілген бөлшекті қысқарту}
    Z. Print; {Қысқартылған бөлшекті баспаға шығару}
    Z.Stepen(4, F); {Енгізілген бөлшекті 4-ші дәрежеге шығару}
    WriteLn(F.P,'/', F.Q)
End.
```

Қарастырылған мысалды талқылайық.

Біріншіден, әдістерді іске асыру объектіні жариялаудан кейін сипаттама бөлімінде жүзеге асырылады және әдісті қолданғанда, оның тақырыбын параметрлердің тізіміндегі өзінің қатысы бар объект типін ғана сипаттау арқылы көрсету жеткілікті. Бұлардың барлығы, қосқанда қол жетімді ресурстар алдымен Интерфейс бөлімінде жарияланатын, содан кейін Implementation бөлімінде жүзеге асырылатын модуль құруды еске түсіреді. Іс жүзінде, объектілер мен олардың әдістері көбінесе модуль түрінде жүзеге асырылады.

Екіншіден, объектіге қолданылатын барлық әрекеттер оның әдісінің көмегімен ғана орындалады.

Үшіншіден, объект типінің жеке бір данасымен жұмыс істеу үшін, айнымалыларды сипаттау бөлімінде сәйкес типтегі айнымалы (немесе айнымалылар) жариялану керек. Мысалда статикалық объектілердің жариялануы басқа айнымалылардың жариялануынан айырмашылығы жоқ және оларды программада пайдаланылуы жазбаларды еске салады.

Мұрагерлік. Объект типтерін иерархияға тізіп қоюға болады. Бір объект типі басқа объект типінің компоненттерін иеленуі мүмкін. Мұра етілетін объект «ұрпақ» деп аталады, ал мұраланған объект - «баба» деп аталады. Егер «баба» біреудің «мұрагері» болса, онда «ұрпақ» осы өрістер мен әдістерді мұраға алады. Айта кету керек, мұра тек объектілер типтеріне ғана тән.

«Ұрпақ» типінің сипатталуы ерекше:

<ұрпақ атауының типі> = Object <баба атауының типі>);

Ары қарайғы жазба кәдімгідей.

Еске сала кетейік, өрістер ешқашанда мұра етілмейді, сондықтан жаңа өрістерді жариялағанда, олардың атауларының бірегейлігін бақылап отыру керек, әйтпесе жаңа өріс атауының мұраланған өрістің атымен сәйкестігі қатені тудырады. Бұл ереже әдістерге қолданылмайды, бірақ бұл туралы әрі қарай қарастырамыз.

4.2-Мысал. «Есептеуіш» объектілі типін «Қосу», «Азайту», «Көбейту», «Бөлу» (кейбір орындаушыны модельдеу) әдістері арқылы және одан туындаған «Тәжірибелі есептеуіш» типінің жаңа әдістері «Дәреже», «*n*-ші дәрежелі түбір» арқылы сипаттау қажет:

```
Type BaseType = Double;
        Vichislitel = Object A,
        B, C : BaseType;
        Procedure Init; {өрістерді енгізу немесе
        инициализациялау}
        Procedure Slozh;
        Procedure Vich;
        Procedure Umn;
        Procedure Delen
End;
        NovijVichislitel = Object(Vichislitel)
        N : Integer;
        Procedure Stepen;
        Procedure Koren
End;
```

Жоғарыда айтылғандардың барлығын қорытындылай келе, мұрагерліктің ережелері келесідей болады:

- «ұрпақ» типті ақпаратты өрістер мен әдістер олардың «ата-

анасынан» иерархияның аралық деңгейлері санына қарамастан барлық типтерін мұраға алады;

- «ата-ана» типтердің әдістері мен өрістеріне рұқсат кез-келген «ұрпақ» типтері шегінде сипатталғандай жүзеге асырылады;
- ешқандай «ұрпақ» типтерінде қандайда бір «ата-ана» типтері өрістерімен сәйкес келетін өріс идентификаторлары пайдаланылмайды. Бұл ереже әдістер тақырыптарында көрсетілген нақты параметрлер идентификаторларына да қатысы бар;
- «ұрпақ» типі өзінің әдістеріндегі және өріс ақпараттарындағы еркін санды анықтай алады;
- «ата-ана» әдісіндегі кез-келген мәтіннің өзгеруі оны шақырған «ұрпақ» типінің барлық әдістеріне автоматты түрде әсерін тигізеді;
- ақпараттық өрістерге қарсы «ұрпақ» типтеріндегі әдіс идентификаторлары «ата-ана» типіндегі әдіс атауларымен сәйкес келуі мүмкін. Осыған орай, «ұрпақ» типіндегі біратаулы әдіс «ата-ана» типі сияқты болуы мүмкін. Осындай әдістің атауын көрсету барысында «ата-ана» типі емес, «ұрпақ» типі шақырылады.

Мұраланған әдістерді шақыру келесі принциптер арқылы жүзеге асырылады:

- әдісті шақыру барысында компилятор алдымен атауы объект типінің ішінде орналасқан әдісті іздейді;
- егер объект типінде шақыру операторында көрсетілген әдіс анықталмаса, компилятор осы атаулы әдісті іздеу барысында жоғары қарай «ата-ана» типіне көтеріледі;
- егер мұраланған әдіс табылып және оның мекен-жайы көрсетілсе, шақырылған әдіс «ұрпақ» типіндегідей емес, «ата-ана» типінде қалай анықталып және компиляцияланғандай жұмыс жасайды. Егер осы мұраланған «ата-ана» типті басқа типтер шақырса, онда жоғарыда орналасқан «ата-ана» типтері шақырылады.

Полиморфизм. Жоғарыда айтылғандай, полиморфизм (әртүрлілік) әр класстың өзінің функционалдық рөліне ие болатын туыстық объектілердің типтері үшін класс немесе әдістердің бірнеше класстарын анықтауды қамтиды. Бір класстың әдістері әдетте жалпы атпен беріледі.

4.3-мысал. «Дөңес төртбұрыш» объектілі типті «ата-аналық» тип бар болсын (өріс типтері—реті бойынша берілген «төбелер координаталары») және одан туындаған типтер: «Ромб», «Параллелограмм» және «Шаршы». Көрсетілген фигураларға

«Бұрыштарды есептеу» (градуспен), «Диагональдарды есептеу», «Қабырға ұзындықтарын есептеу», «Периметрді есептеу», «Ауданды есептеу» әдістерін қолдану.

Программа сипаттамасы мынадай:

```
Type BaseType = Double;
     FourAngle = Object
                    x1, y1, x2, y2, x3, y3, x4, y4,
                    A, B, C, D, D1, D2,
                    Alpha, Beta, Gamma, Delta,
                    P, S : BaseType;
                    Procedure Init;
                    Procedure Storony;
                    Procedure Diagonali;
                    Procedure Angles;
                    Procedure Perimetr;
                    Procedure Ploshad;
                    Procedure PrintElements;
                End;
     Parall =       Object(FourAngle)
                    Procedure Storony;
                    Procedure Perimetr;
                    Procedure Ploshad;
                End;
     Romb = Object(Parall)
                    Procedure Storony;
                    Procedure Perimetr;
                End;
   Kvadrat = Object(Romb)
                    Procedure Angles;
                    Procedure Ploshad;
             End;
Procedure FourAngle.Init;
Begin
     Write('Берілген төртбұрыш төбелерінің координаталарын
     енгіз:      ');
     ReadLn(x1, y1, x2, y2, x3, y3, x4, y4);
End;
Procedure FourAngle.Storony;
Begin        A:= Sqrt(Sqr(x2 - x1) + Sqr(y2 - y1)
             B:= Sqrt(Sqr(x3 - x2) + Sqr(y3 - y2)
             C:= Sqrt(Sqr(x4 - x3) + Sqr(y4 - y3)
             D:= Sqrt(Sqr(x4 - x1) + Sqr(y4 - y1)
End;
Procedure FourAngle.Diagonali;
Begin
     D1 := Sqrt(Sqr(x1 - x3) + Sqr(y1 - y3));
```

```
        D2 := Sqrt(Sqr(x2 - x4) + Sqr(y2 - y4));
End;
Procedure FourAngle. Angles;
      Function Ugol(Aa, Bb, Cc : BaseType): BaseType; Var VspCos,
      VspSin: BaseType;
      Begin
            VspCos := (Sqr(Aa) + Sqr(Bb) -
                  Sqr(Cc))/(2 * Aa * Bb);
            VspSin := Sqrt(1 - Sqr(VspCos));
            If Abs(VspCos) > 1e-7
            Then Ugol := (ArcTan(VspSin/VspCos) +
                  Pi * Ord(VspCos < 0))/Pi * 180
            Else Ugol := 90
      End;
Begin Alpha := Ugol(D, A, D2);
        Beta := Ugol(A, B, D1);
        Gamma := Ugol(B, C, D2);
        Delta := Ugol(C, D, D1);
End;
Procedure FourAngle.Perimetr;
Begin P : = A + B + C + D End;
Procedure FourAngle.Ploshad;
Var Per1, Per2 : BaseType;
         Begin Per1 := (A + D + D2)/2; Per2 := (B + C + D1)/2;
               S := Sqrt(Per1*(Per1 - A) * (Per1 - D) *
                (Per1 - D2)) + Sqrt(Per2 * (Per2 - B) *
        (Per2 - C) * (Per2 - D1))
End;
Procedure FourAngle.PrintElements;
  Begin
       WriteLn('Қабырғалар: ', A:10:6, B:10:6, C:10:6, D:10:6,
  'Бұрыштар: ', Alpha:10:4, Beta:10:4, Gamma:10:4, Delta:10:4,
  'Периметр:   ', P:10:6, 'Аудан:            ', S:10:6, 'Диагональдар:
  ', D1:10:6, D2: 10: 6)
  End;
  Procedure Parall.Storony;
  Begin        A := Sqrt(Sqr(x2 - x1) + Sqr(y2 - y1));
               B := Sqrt(Sqr(x3 - x2) + Sqr(y3 - y2));
               C : = A; D: = B
  End;
  Procedure Parall. Perimetr;
  Begin P := 2 * (A + B) End;
  Procedure Parall.Ploshad;
  Var Per : BaseType;
  Begin
       Per := (A + D + D2)/2;
            S := 2 * Sqrt(Per * (Per - A) * (Per - D) * (Per - D2))
  End;
```

```
Procedure Romb.Storony;
Begin
      A := Sqrt(Sqr(x2 - x1) + Sqr(y2 - y1));
      B : = A; C : = A; D : = A
End;
Procedure Romb.Perimetr;
Begin P : = 2 * A End;
Procedure Kvadrat. Angles;
Begin          Alpha := 90; Beta := 90; Gamma := 90;
               Delta := 90;
End;
Procedure Kvadrat. Ploshad;
Begin S := Sqr(A) End;
{Основная программа}
Var obj : Kvadrat;
Begin
      obj.Init; obj.Storony;
      obj.Diagonali; obj.
      Angles; obj.Perimetr;
      obj.Ploshad;
      obj.PrintElements
End.
```

Осылайша, фигурадағы сәйкес элементті есептеу өз әдісін шақыру арқылы жүзеге асырылады.

ЖАТТЫҒУЛАР

1. Әдістерін көрсете отырып, келесі объектілерді сипаттаңдар:
а) «Телефон қоңырауы» — телефон нөмірі, сөйлесу мерзімі, ұзақтығы, қала коды және т.с.с.;
ә) «Жол жүру» — пойыз нөмірі, белгіленген пункті, жүру күндері, келу уақыты, тұру уақыты және т.с.с.;
б) «Кинотеатр» — кинофильм атауы, сеанс, билет бағасы, көрермендер саны, және т.б.

Объектілі типтегі әдістерді жүзеге асыру.

2. 4.1-мысалдағы Drob объектісін «Оң бөлшек» (логикалық типтегі функция), «Дұрыс бөлшек» (логикалық типтегі функция), «Ақырғы ондық бөлшек» (логикалық типтегі функция), «Бөлшек периоды» (егер сәйкес ондық бөлшек соңғы болып табылмаса, ол бар болып табылады, жалпы нәтижесі – «ұзын» сан) әдістерімен толықтырыңдар. Бұл әдістерді жүзеге асырыңдар және тексеріңдер.

3. Бірінші жаттығуда көрсетілген «ұрпақ» объектілерін сипаттаңдар. Объектілі типтегі жарияланған әдістерді жүзеге асырыңдар.

4. 4.2-мысалдағы «Есептеуіш» объектісінің әдістерін жүзеге асырыңдар.

5. 4.3-мысалдағы объектілер жиынын «Тіктөртбұрыш» объектісімен, сонымен қатар параллелограмм болып табылмайтын (трапеция және т.б.) басқа да дөңес төртбұрыштармен толықтырыңдар және сәйкес әдістермен оларды жүзеге асырыңдар.

6. «Бейнелеу», «Жылжыту», «Өлшемдерін өзгерту», «Тығу» әдістерін ретімен қолданып, келесі объектілердің иерархиясын құрыңдар:

а) координаталар — нүкте — көлденең сызық —тіккөлденең айқас;

ә) координаталар — нүкте —45° бұрыштағы көлбеу сызық — көлбеу айқасу;

б) координаталар — нүкте — шеңбер — доға (Arc процедурасы);

в) координаталар — нүкте — эллипс (FillEllipse процедурасы) — эллипстік доға (Ellipse процедурасы);

г) координаталар — нүкте — шеңбер — сектор (PieSlice процедурасы);

ғ) координаталар — нүкте — сектор (PieSlice процедурасы) — эллипстік доға (Ellipse процедурасы);

д) координаталар — нүкте — тіктөртбұрыш (Rectangle процедурасы) — үшөлшемді жолақ (Bar3D процедура);

е) координаталар — нүкте— штрихталған эллипс (FillEllipse процедурасы) — штрихталған сектор (Sector процедурасы);

ж) координаталар — нүкте — шеңбер — штрихталған сектор (Sector процедурасы);

з) координаталар — нүкте — шеңбер — эллипстік доға(Ellipse процедурасы).

4 DELPHI ПРОГРАММАЛАУДЫҢ БІРІКТІРІЛГЕН ОРТАСЫ

Delphi тарихы мен мақсаты. Delphi, Borland әзірлеген біріктірілген программалау ортасы - бұл Паскаль дамуының нәтижесі болып табылатын Турбо Паскальдың дамуының нәтижесі. Бастапқыда Паскаль толығымен процедуралық тіл болды. Турбо Паскальдың 5.5 нұсқасынан бастап объектіге-бағытталған программалау құралдары енгізілді. Delphi-дегі тіл Турбо Паскаль 7.0 нұсқасы конструкциясына негізделген Object Pascal объектіге-бағытталған программалау тілі болып табылады. Ресейде Borland Delphi 1993 жылдың аяғында пайда болды және бірден танымал болды.

Delphi программалау ортасы Windows операциялық жүйесінің қосымшаларын тез және жылдам жасауға мүмкіндік береді, сондықтан ол RAD (Rapid Application Development) деп аталады.

Delphi-дің өңдеу процесі өте оңайлатылған болып табылады. Ең алдымен, ол әдетте, программаны өңдеу уақыты шамамен 80% құрайтын, интерфейсті құру болып табылады. Программалаушы тек қажетті компоненттерді Windows терезесінде орналастыруы керек (Delphi-де оны пішін деп атайды) және олардың қасиеттерін арнайы құрал - Object Inspector арқылы баптай алады. Осы құралдың көмегімен компоненттердің *оқиғаларын* өңдеу процедурасымен байланыстыра аламыз (батырманы басу, тізімдегі элементті тышқанмен таңдау және т.б.) және нәтижесінде қарапайым программа дайын болады. Ал әзірлеуші өз кезегінде қажетті жөндеуіш құралдарын (процессордың командаларын кезең-кезеңімен орындау), ыңғайлы контекстік-анықтамалық жүйені, жоба бойынша бірлескен жұмыстарды жүргізу құралдарын және т.б. әрекеттерін орындай алады.

Delphi Internet және Intranet-қосымшаларды жасауға мүмкіндік береді. Мәліметтерге қол жеткізу үшін Borland DataBase Engine пайдаланылады.

Delphi Pascal тілі (бұрын — Object Pascal), Delphi-де қолданылатын тіл, ол Borland компаниясының арқасында объектіге-бағытталған программалаудың барлық талаптарын қолдай отырып, үнемі кеңейтіліп және толықтырылып келеді.

Қатаң түрде жазылған тілдерде класстар қарапайым мұраны ғана қолдайды, бірақ интерфейстерде бірден бірнеше «бабалар» болуы мүмкін. Тіл ерекшеліктерінің санына ерекше жағдайларды өңдеуге қолдау көрсету (exceptions), сондай-ақ шамадан тыс жүктеу әдістері мен ішкі программаларды жатқызуға болады (overload). Сонымен

қатар, кез-келген деректер құрылымын жадта сақтауға мүмкіндік беретін ашық жиымдар, нұсқалар және нұсқалық жиымдары бар.

Delphi-де жеке компоненттерімізді жасауға, оларды импорттауға, сондай-ақ жоба үлгілерін құратын жоба шаблондары мен шеберлерін жасауға болады. Delphi әзірлеушіге қосымшаларды (немесе сыртқы программаларды) біріктірілген Delphi (IDE) қабықшасымен байланыстыратын интерфейсті ұсынады.

Интерфейс. Delphi программалау ортасының терезесі Windows-та көруге болатын басқа орталардан ерекшеленеді, себебі ол SDI (Single Document Interface) деп аталатын сипаттамаға сәйкес келеді және бірнеше бөлек орналасқан терезелерден тұрады.

Delphi программалау ортасының терезесі 4.1-суретте сипатталған. Үстіңгі жолда Windows-тың кез-келген басқа қосымшаларындағы сияқты мәзір орналасқан. Сол жақта қасиеттер мен оқиғаларды көрсететін пішіннің компоненттері ағашы мен компоненттер инспекторы бар, терезенің ортасында – пішін, ал пішіннің артында код редакторы орналасқан. Мәзір қатарының астында құрал-саймандар тақтасы мен компоненттер палитрасы орналасқан.

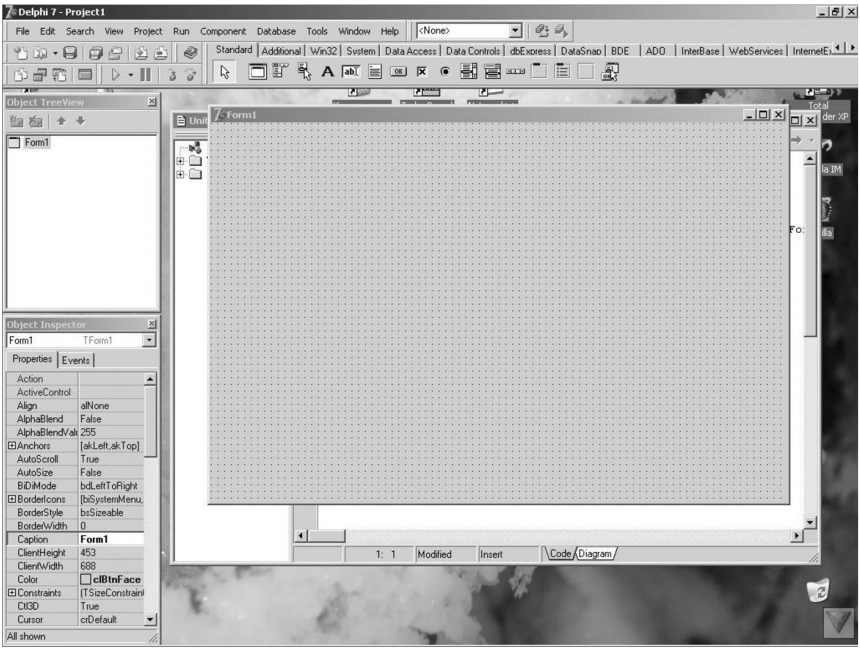
Delphi интерфейсінің элементтері толығырақ 4.2-суретте, ал кейбір батырмалардың мақсаты мен құралдар тақтасы 4.3-суретте көрсетілген.

Пішін. Windows қосымшаларының негізгі терезелерінің барлық қасиеттеріне ие: өлшемді жиек, белгі, тақырып, батырмалар, [Жиыру], [Жаю], [Жабу] (4.4-сур.) және ол тышқанмен басқарылады.

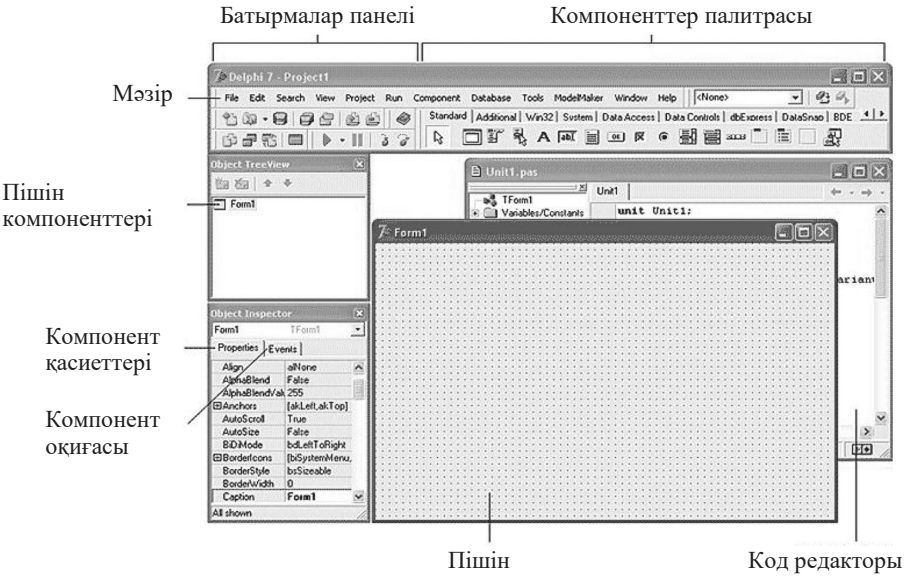
Пішінді жобалаушы терезесі (Form Designer) - дайындау құралы, яғни өңделетін қосымша терезелер бірінің макеті. Пішін Delphi-дің негізгі интерфейс элементі болып табылады. Бұл қосымшаның функционалдығын анықтайтын басқа компоненттерден тұратын контейнер ретінде әрекет ететін кез - келген программаға тән визуалды компонент болып табылады.

Пішіннің жобалаушысы программаларды әзірлеу кезінде келесі әрекеттерді орындауға мүмкіндік береді:

- пішінге компоненттерді қою;
- пішінді және оның компоненттерін өзгерту;
- Delphi Pascal –да компоненттердің өңдеушілерін редакторлар кодында орналасқан программамен байланыстыру.



4.1-сурет. Delphi терезесі



4.2-сурет. Delphi интерфейсінің элементтері

4.3-сурет. Құралдар тақтасы батырмаларының қызметі

Жалпы программаны өңдеуді Delphi-де келесі түрде елестетуге болады: Компоненттер палитрасынан тышқанның көмегімен компонентті таңдау қажет (батырма, жазу, мәтін редакторы және т.б.), оны пішінге орналастыру және *Қасиеттер* аймағынан оның қасиеттерін көрсету қажет.

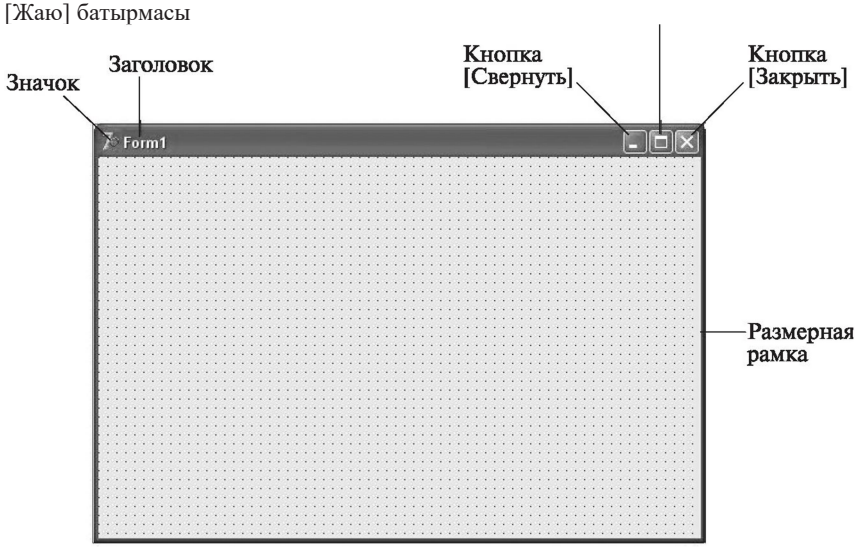


Рис. 4.4. Элементы формы

4.4-сурет. Пішін элементтері.
(Белгіше. Тақырып. [Жиыру] батырмасы. [Жабу] батырмасы. Өлшемдік жиек.)

Delphi ортасы пішіннің ішіндегі мазмұнын талдайды және сәйкес программалық код құрады, ал программалаушыға тек есепті шешу үшін оқиғаларға жауап бергені жеткілікті.

DELPHI КОМПОНЕНТТЕРІ. КОМПОНЕНТТЕР ҚАСИЕТТЕРІ

Delphi компоненті - оның сыртқы көрінісін және күйін анықтайтын қасиеттер жиынтығы, сондай-ақ оның мінез-құлқын анықтайтын әдістер мен оқиғалар жиынтығы бар функционалды элемент. Программаны әзірлеудегі компоненттерді пайдалану тұжырымдамасы объектіге-бағытталған программалау әдіснамасымен тікелей байланысты. Бұл жағдайда компоненттердің көмегімен объектілер көрсетіледі, яғни стандартты сұхбат терезелері, батырмалар, тізімдер және т.б. көрсетіледі. Әрбір компонент өзіндік әрекеттер жиынтығын қабылдайды.

Delphi-дің барлық компоненттері TComponent класынан туындайды. Мұнда компоненттердің ортақ қасиеттері мен әдістері инкапсулирленген.

Компонент идентификаторы басқа Delphi объектілерінің идентификаторлары, сондай-ақ айнымалы мәндер, тұрақты және т.б. сияқты бірдей ережелерге сәйкес жасалған. Үнсіз келісім бойынша компоненттің атауы программалау ортасы арқылы көрсетіледі, бірақ оны программалаушы өз қалауы бойынша қайта атауына болады.

Барлық компоненттер кейбір сипаттамаларға қасиеттері бойынша топтастырылады, ал компоненттердің бүкіл жиынтығы компоненттердің палитрасын құрайды. Әр компоненттің тобына компоненттер палитрасындағы парақ сәйкес келеді .

Компонентті пішінге орналастыру үшін келесі әрекеттерді орындау керек:

- компоненттер палитрасындағы керек парақты таңдау;
- керек компонентті таңдау;
- компоненттің пішіндегі орнын белгілеу (белгілеген жерге компонент орналасады);
- компонентке қажет өлшемдерді ені мен ұзындығы бойынша оның бұрыштарынан созу арқылы орнату қажет.

Біріктірілген өңдеу ортасының негізгі компоненттері.

Әрбір парақтағы компоненттердің жиынтығы мен реттілігі программалаушының қалауымен жанама мәзір арқылы бапталынады.

Standard парағының кейбір компоненттерін сипаттайық. 4.5-сурет.

4.5-сурет. Standard парағы

Frames — пішінге жиек қояды.

TMainMenu — өңделетін программаға негізгі мәзір қоюға мүмкіндік береді. Мәзірді құру үшін:

- TMainMenu-ді пішінге орналастыру;
- Мәзір дизайнерін объектілер Инспекторындағы Items қасиеті арқылы шақыру;
- Мәзір дизайнерінде мәзір пункттерін анықтау.

TPopupMenu — жанама мәзір құруға мүмкіндік береді.
Барлық көрінетін объектілерде керек мәзір таңдалатын PopupMenu қасиеті бар

TLabel — экрандағы мәтінді құру батырмасы.

Ш П TEdit — енгізуге арналған Windows-тың стандартты басқару элементі. Бұл мәтіннің қысқа фрагментін көрсетуге және программаны орындау барысында мәтін енгізуге арналған компонент.

TMemo — TEdit-тің басқа нұсқасы. Үлкен мәтіндермен жұмыс істеуге арналған. Сөздерді тасымалдай алады, Clipboard-қа мәтін фрагменттерін сақтайды, қалпына келтіреді және редактордың басқа да негізгі функцияларын орындайды. Мәтіннің шектеулі көлемі — 32 Кбайт.

TButton — программаны орындау уақытында батырманы басқанда қандай-да бір әрекет орындалады.

TCheckBox —сыртқы жазуы бар төрт бұрышты батырма, оған белгілі бір әрекетті таңдауды білдіретін қанат белгісі болады.

TRadioButton — бірнеше опцияның біреуін таңдауға мүмкіндік беретін батырма.

TListBox — жүгіртпе тізімін орналастырады.

Я TComboBox — TListBox батырмасының енгізу өрісіне ақпарат енгізе алатын және осы батырманың баламасы болып табылады. TComboBox –тың бірнеше типтері бар, соның ішіндегі танымалы төгілмелі тізім.

TScrollbar — объектілерді түзету барысында мәтінді көру үшін автоматты түрде пайда болатын айналдыру жолағы.

Additional парағы 4.6-суретте көрсетілген. Қолданушы интерфейсінің қосымша компоненттері орналасатын парақ. Оның кейбір компоненттерін сипаттайық.

TBitBtn - TButton батырмасының баламасы, үстіне сурет (glyph) қоюға болады.

Бұл компоненттің бірнеше алдын-ала анықталған типтері бар: bkClose, bkOK және т.б., оларды тадағанда сәйкес тип қабылданады.

TTabSet — көлденең беттер, көпбетті терезелерді құруда TNoteBook компонентімен бірге қолданылады.

TNoteBook —TTabSet компонентімен бірге көпбетті диалог құру үшін қолданылады. Әрбір бетте өзінің объектілер жиыны орналасады.

TOutline — байланысқан мәліметтердің иерархиялық қатынастарын құратын компонент. Мысалы, директория ағаштары.

System парағы 4.7-суретте көрсетілген. Ол жүйелік сервистерге қол жеткізетін компоненттер жиындары орналасқан парақ.,

Компоненттер қасиеттері және қасиеттер арқылы басқару. Жоғарыда айтылғандай, компонент қасиеті оның жағдайының айнымалы мәндері мен көрінісін анықтайды. Әрине, көптеген компоненттердің қасиеттерінің тізімі бір жағынан қайталанады, ал екінші жағынан әрбір компонентте өзіндік бірегей қасиеттері бар. Кейбір стандартты компоненттердің қасиеттерін сипаттайық.

Пішін компоненті (TForm объект типі) программаның негізі болып табылады. Пішіннің қасиеттерін программаның терезе көрінісі анықтайды (4.1-кесте).

Компонент **Label** компоненті пішіннің үстіне мәтін шығарады (4.2-кесте).

У -кШ
Е

4.6-сурет. Additional парағы

4.7-сурет. System парағы

4.2-кесте. Мәтін қасиеттері

Қасиет	Сипаттама
Name	Пішін атауы. Программада пішінді басқару үшін және оның компоненттеріне қол жеткізу үшін қолданылады
Caption	Тақырып мәтіні
Top	Пішіннің жоғарғы шекарасынан экранның жоғарғы шекарасына дейінгі аралық
Left	Пішіннің сол шекарасынан экранның сол шекарасына дейінгі аралық
Width	Пішін ені
Height	Пішін биіктігі
ClientWidth, ClientHeight	Пішіннің оң және сол шекараның енін қоспағанда жұмыс аймағының сәйкес ені мен биіктігі
BorderStyle	Қарапайым (bsSizeable), жіңішке (bsSingle) немесе жоқ (bsNone) шекара түрі. Егер терезе қалыпты шекара болса, онда программа жұмыс істеп тұрғанда, пайдаланушы өлшемін тышқанмен өзгерте алады. Терезенің өлшемін жұқа жиекпен өзгерту мүмкін емес. Егер шекара болмаса, атаусыз терезе көрсетіледі, программаның жұмысы кезінде оның позициясы мен мөлшері өзгермейді
BorderIcons	Терезені басқару батырмалары. Қасиет мәні программаның жұмыс істеуі барысында пайдаланушыға қандай терезе басқару түймешіктері қол жетімді болатынын анықтайды. Қасиет мәні мәндерді нақтылау қасиеттеріне меншіктеу арқылы орнатылады: biSystemMenu, biMinimize, biMaximize және BIHelp. biSystemMenu қасиеті [Жиыру] батырмасы мен жүйелік мәзір батырмасының қол жетімділігін анықтайды, biMinimize - [Жиыру] батырмасына қол жетімділік, biMaximize - [Жаю] батырмасына қол жетімділік, biHelp - анықтама ақпаратының батырмасына қол жетімділік.
Icon	Сұхбат терезесі тақырыбының белгісі. Шығару батырмасын білдіреді.
Color	Фон түсі, оның атауын көрсету немесе операциялық жүйенің ағымдық түс сызбасына байланысу арқылы көрсетілуі мүмкін. Соңғы жағдайда түстер байланыстыру компоненті арқылы таңдалған ағымдағы түс сызбасымен анықталады және операциялық жүйенің түс сызбасы өзгерген кезде өзгертіледі
Font	Пішіннің үстінде компоненттермен үнсіз келісім бойынша қолданылатын қаріп.
Canvas	Графиканы шығаруға арналған орын

4.1-кесте. Пішін қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Программада компонентке және оның қасиеттеріне қол жеткізу
Caption	Көрінетін мәтін
Left	Шығару өрісінің сол жақ шекарасынан пішіннің оң жақ шекарасына дейінгі аралық
Top	Шығару өрісінің жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Шығару өрісінің биіктігі
Width	Шығару өрісінің ені
AutoSize	Өріс өлшемі оның ішіндегі мазмұнымен анықталу белгісі
Wordwrap	Ағымдық жолға сыймаған сөздер автоматты түрде келесі жолға ауыстырылатын белгі (AutoSize қасиетінің мәні False болуы керек)
Alignment	Өрістің ішінде мәтінді тегістейді: сол жақ шеті бойынша (taLeftJustify), орта (taCenter) немесе оң жақ шеті бойынша (taRightJustify)
Font	Мәтінді көрсетуге арналған қаріп. Қасиеттері символдардың сызылуын (Font. Name), өлшемін (Font. Size) және түсін (Font. Color) анықтайды
ParentFont	Өзінде орналасқан компонент қарібінің сипаттамасын мұраға алған пішін қарпі, егер қасиет мәні True болса, мәтін пішін үшін орнатылған қаріптер жиынын көрсетіледі
Color	Мәтін шығару аймағының фон түсі
Transparent	Мәтін шығару аймағының фон түсін көрсетуді басқарады. Егер мәні True болса, аймақ мөлдір болады (Color қасиеті бойынша шығару аймағы ешқандай түске боялмайды)
Visible	Мәтінді (False) жасыруға немесе оны көрсетуге мүмкіндік береді (True)

Edit компоненті символдар жолдарының мәтінді енгізіп-түзету өрісі (4.3-кесте).

4.3-кесте. Енгізу-түзету өрісінің қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Программада компоненттерге және олардың қасиеттеріне қол жеткізу үшін қолданылады
Text	Енгізу және түзету өрісіндегі мәтін
Left	Компоненттің сол жақ шекарасынан пішіннің оң жақ шекарасына дейінгі аралық
Top	Компоненттің жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Өріс биіктігі
Width	Өрістің ені
Font	Енгізілетін мәтіннің қарпі
ParentFont	Өзінде орналасқан компонент қарібінің сипаттамасын мұраға алған пішін қарпі. Егер қасиет мәні True болса, пішіннің Font қасиетін өзгерту барысында Font компонентінің қасиеттері автоматты түрде өзгереді.
Enabled	Түзету өрісіндегі мәтінді өзгерту мүмкіндігін шектеу үшін пайдаланылады. Егер қасиет мәні False болса, түзету өрісіндегі мәтінді өзгерту мүмкін емес
Visible	Компонентті (False) жасыруға немесе оны көрсетуге мүмкіндік береді (True)

Button компоненті командалық батырма болып табылады (4.4-кесте).

Memo компоненті бірнеше жолдан тұратын мәтінді түзетуге мүмкіндік береді (4.5-кесте).

4.4-кесте. Командалық батырманың қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Программада компоненттерге және олардың қасиеттеріне қол жеткізу үшін қолданылады
Caption	Батырмадағы мәтін
Left	Батырманың сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Батырманың жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық

4.5-кесте. Memo қасиеті

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Text	Біртұтас болып қаралатын Memo өрісіндегі мәтін..
Lines	Өріс ішіндегі мазмұнға сәйкес келетін жолдар жиымы. Жолға рұқсат нөмір бойынша алынады. Жолдар нөлден бастап нөмірленеді
Lines. Count	Memo өрісіндегі жолдардың саны
Left	Өрістің сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Өрістің жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Өріс биіктігі
Width	Өріс ені
Font	Енгізілетін мәтінді көрсетуге арналған қаріп
ParentFont	Қаріп қасиетінің «ата-аналық» пішіннен алған мұрасы

RadioButton компоненті (4.6-кесте) жағдайы топтың басқа батырмаларының жағдайы арқылы анықталатын тәуелді батырма болып табылады. Егер сұхбат терезесінде бірнеше ауыстырып-қосқыштар топтарын ұйымдастыру керек болса, онда әрбір топты RadioGroup батырмасымен сипаттау қажет.

CheckBox компоненті ауыстырып-қосатын тәуелсіз батырма (4.7-кесте).

ListBox компоненті қажет элементті таңдауға болатын тізімді құрады (4.8-кесте).

ComboBox компоненті түзету өрісіне пернетақта арқылы немесе дайын тізімнен мәліметтерді енгізуге мүмкіндік береді (4.9-кесте).

BitBtn компоненті Button сияқты командалық батырма болып табылады. Алайда, ол әлдеқайда әмбебап және кестені қамтуы мүмкін (4.10-кесте).

ScrollBar компоненті — терезе ішіндегі мазмұнды жүгіртіп көруге арналған және белгілі бір интервалдан мәндерді таңдауға болатын айналдыру жолағы (4.11-кесте).

4.6-кесте. Радиобатырманың қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Caption	Батырманың оң жағында орналасқан мәтін
Checked	Батырманың түрі, жағдайы:егер батырма белгіленсе, онда Checked = True; егер батырма белгіленбесе, онда Checked = False болады.
Left	Жалаушаның сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Жалаушаның жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Түсіндірме мәтіннің шығару өрісі биіктігі
Width	Түсіндірме мәтіннің шығару өрісі ені
Font	Түсіндірме мәтінді көрсететін қаріп
ParentFont	Қаріп қасиетінің «ата-аналық» пішіннен алған мұрасы

Таблица 4.7. Ауыстырып-қосқыштың қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Caption	Жалаушаның оң жағындағы мәтін
Checked	Жалаушаның түрі, жағдайы:егер жалауша белгіленсе, онда Checked = True; егер жалауша белгіленбесе, онда Checked = False болады.
State	Жалауша жағдайы. Checked қасиетінен айырмашылығы, орнатылған, жойылған және аралық жағдайларды ажырата алады. Жалаушаның жағдайын тұрақтылар анықтайды: cbChecked (орнатылған); cbGrayed (сұр, анықталмаған жағдай); cbUnChecked (жойылған)
AllowGrayed	Жалаушаның аралық жағдайда болуын анықтайды: егер AllowGrayed = False болса, онда жалауша тек орнатылған немесе жойылған болады; егер AllowGrayed = True болса, онда аралық жағдай болады
Left	Жалаушаның сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Жалаушаның жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Түсіндірме мәтіннің шығару өрісі биіктігі
Width	Түсіндірме мәтіннің шығару өрісі ені
Font	Түсіндірме мәтінді көрсететін қаріп
ParentFont	Қаріп қасиетінің «ата-аналық» пішіннен алған мұрасы

Окончание табл. 4.10

Таблица 4.7. Ауыстырып-қосқыштың қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Items	Тізім элементтері — жолдар жиымы
Count	Тізім элементтерінің саны
Қасиет	Сипаттама
Sorted	Кезекті элементті қосқаннан кейін автоматты түрде сұрыптау (True) қажеттілігі белгісі.
ItemIndex	Таңдалған элемент нөмірі (тізім элементтері 0-ден бастап нөмірленеді). Егер тізімде ешқандай элемент таңдалмаса, қасиет мәні -1 болады
Left	Тізімнің сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Тізімнің жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Тізім өрісі биіктігі
Width	Тізім өрісі ені
Font	Тізім элементтерінің мәтінін көрсететін қаріп
ParentFont	Қаріп қасиетінің «ата-аналық» пішіннен алған мұрасы

4.9-кесте. ComboBox қасиеті

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Text	Енгізу-түзету өрісіндегі мәтін
Items	Тізім элементтері — жолдар жиымы
Count	Тізім элементтерінің саны
ItemIndex	Таңдалған элемент нөмірі (тізім элементтері 0-ден бастап нөмірленеді). Егер тізімде ешқандай элемент таңдалмаса, қасиет мәні -1 болады
Sorted	Кезекті элементті қосқаннан кейін автоматты түрде сұрыптау (True) қажеттілігі белгісі.
DropDownCount	Ашылған тізімдегі элементтер саны. Егер элементтер саны DropDownCount-тан үлкен болса, онда тігінен айналдыру жолағы пайда болады

Қасиет	Сипаттама
Left	Компоненттің сол жақ шекарасынан пішіннің сол жақ шекарасына дейінгі аралық
Top	Компоненттің жоғарғы шекарасынан пішіннің жоғарғы шекарасына дейінгі аралық
Height	Компонент биіктігі(енгізу-түзету өрістері)
Width	Компонент ені
Font	Тізім элементтерінің мәтінін көрсететін қаріп
ParentFont	Қаріп қасиетінің «ата-аналық» пішіннен алған мұрасы

4.10-кесте. BitBtn қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Caption	Батырмада көрінетін мәтін
Font	Мәтінді көрсетуге арналған қаріп
Left	Пішіннің сол жақ шекарасынан компоненттің сол жақ шекарасына дейінгі аралық
Top	Пішіннің жоғарғы шекарасынан компоненттің жоғарғы шекарасына дейінгі аралық
Width	Компонент өрісінің ені
Height	Компонент өрісінің биіктігі
Enabled	Батырманың қол жетімділік белгісі: егер қасиет мәні True болса батырма қол жетімді, ал егер False болса, қол жетімсіз.
Visible	Пішіннің үстіндегі батырманың көріну қасиеті: егер қасиет мәні True болса, батырма көрінеді, қарсы жағдайда көрінбейді
Glyph	Батырманың үстінде көрінетін сурет (сурет файлы)
NumGlyphs	Glyph қасиетінде белгіленген сурет файлындағы бейнелер саны

Қасиет	Сипаттама
Layout	Батырмадағы сурет пен мәтіннің өзара орналасуын анықтайды. Ол келесі мәндерді қамтуы мүмкін: blGlyphLeft — сурет мәтіннің сол жағына, blGlyphRight — оң жағына, blGlyphTop — үстіңгі жағына, blGlyphBottom — астыңғы жағына орналасады
Spacing	Сурет пен мәтін аралығы. аралық пиксельдермен беріледі

4.11-кесте. Айналдыру жолағының қасиеттері

Қасиет	Сипаттама
Name	Компонент атауы. Программада компоненттің қасиеттеріне қол жеткізу үшін қолданылады.
Enabled	Батырманың қол жетімділік белгісі: егер қасиет мәні True болса айналдыру қол жетімді, ал егер False болса, қол жетімсіз.
Kind	Айналдырудың орналасуын анықтайды: көлденең немесе тігінен
Max	Жүгіртпенің орналасуына максималды мән береді
Min	Жүгіртпенің орналасуына минималды мән береді
Visible	Пішіннің үстіндегі батырманың көріну қасиеті: егер қасиет мәні True болса, айналдыру көрінеді, қарсы жағдайда көрінбейді
LargeChange	Жолақты тышқанмен шерткенде немесе [PageUp] және [PageDown] батырмаларын басқанда жүгіртпе қанша нүктеге ауысатынын көрсетеді
Left	Пішіннің сол жақ шекарасынан компоненттің сол жақ шекарасына дейінгі аралық
Top	Пішіннің жоғарғы шекарасынан компоненттің жоғарғы шекарасына дейінгі аралық
Width	Компонент өрісінің ені
Height	Компонент өрісінің биіктігі

Компоненттер қасиеттерін Properties парағындағы Object Inspector терезесінде өзгертуге болады. Form компонентінің қасиеттерін өзгерту туралы 4.8-суретте көрсетілген.

Компоненттер қасиеттерін баптау туралы ары қарай біріктірілген ортада айтылатын болады.

Компоненттер оқиғалары және оқиғалар негізінде процедураларды анықтау. Жоғарыда айтылғандай, оқиғалар программа жұмыс істеп тұрған кездегі компоненттің мінез-құлқын анықтайды. Пайдаланушы әрекеттеріне реакция бір немесе басқа программа кодының орындалуы деп есептеледі. Сондықтан, программа сценарийін әзірлеу кезінде пайдаланушының болжамды іс-әрекеттерін ескеріп, әр жағдайда тиісті жауап беру қажет.

Компоненттер оқиғаларын **Events** парағының Object Inspector-да программалауға болады. Form компоненті үшін оқиғалардың өзгеруі 4.9-суретте көрсетілген.

Тышқанның сол жағын екі рет шерткенде сәйкес оқиғаны түзету пайда болады. Осы жағдайда әдіс шаблоны автоматты түрде құрылады.

Кейбір әрекеттерді орындағаннан кейін өңдеу барысында оқиғалар ары қарай кез-келген объект (компонент) пайдаланатын белгілі бір нәтиже алады.

Delphi-дегі программа. Delphi ортасындағы кез-келген программа жоба файлынан (.dpr кеңейтілімді файл) және бір немесе бірнеше модульдерден тұрады. Осындай файлдардың әрбіреуі Delphi-дің программалық бірлігін құрайды.

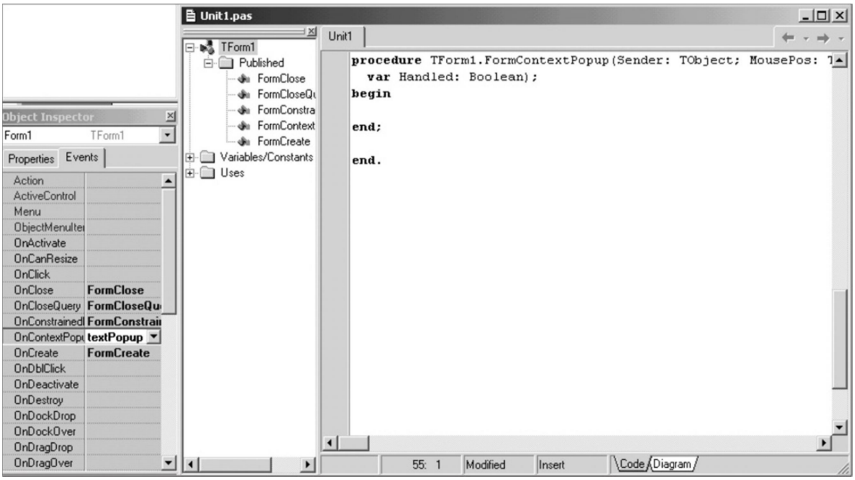
Әрбір программаға арналған жоба файлдарының түрлері ұқсас. Мысалы:

```
Program Projectl;
Uses Forms, Unitl In 'Unitl.pas' {Forml};
{$R *. res}
Begin
Application.Initialize; Application.
                CreateForm(TForm1, Form1);
                    Application.Run;
End.
```

Application объектісінде Windows-программалардың жалпы дұрыс жұмыс жасауы үшін мәліметтер мен ішпрограммалар жинақталған

Object: Inspecbor	
j Forml	-
Properties | Events |	
Action	ж
ActiveControl	
Align	aIN one
AlphaB lend	False
AlphaB lentf/sl	255
3 Anchor;	[akLeft.akT op]
AutoS croll	True
AutoS ize	False
BiDiM ode	bdLeftToR ighl
EDBorderIcons	[biSystemMenu,
BorderS Me	bsSizeable
BorderWidth	a
Caption	
ClientH eight	453
ClientWidth	S88
Color	**flcIBlnFace**
0 Constraints	(T Size Constrain!
CII3D	True
Cursor	crDefault
All shown	

4.8-сурет. Объектілер инспекторы терезесі



4.9-сурет. Оқиғаларды өңдеуге арналған программалық модуль терезесі

Delphi Application үшін объект-программаларды автоматты түрде құрады.

Программаға шақырылған әдістердің мазмұны мен тағайындалулары олардың атауына сәйкес келеді. Осылайша, Initialize әдісі программаны инициализациялайды. Application объектісінің CreateForm әдісі экрандағы негізгі пішін терезесін жасайды және көрсетеді, сонымен қатар Run әдісі пайдаланушы әрекеттері туралы Windows-тан түскен хабарларын қабылдау және өңдеу үшін шексіз циклды жүзеге асырады. Пайдаланушы Close түймешігін басқанда, Windows программа жұмысын тоқтатып, оған тағайындалған жүйелік ресурстарын босатуға мүмкіндік беретін арнайы хабар береді. Жоба файлы толықтай Delphi ортасы арқылы жасалады және ол көбінесе редакциялауға арналмаған (кем дегенде оқу тапсырмаларында).

Delphi программалау тілі мен Turbo Pascal тілінің арасындағы айырмашылық туралы бірнеше сөз: қарапайым деректер типтеріне мерзім-уақыт типі қосылады (осы типпен жұмыс істеуге арналған құралдар 3.1Қ-кестеде келтірілген), күрделі типтерге - процедуралық типтер, нұсқалар және класстар қосылады.

Delphi-де жолдар ерекше рөл атқарады, яғни енгізу-шығару (компоненттер арасындағы деректермен алмасу) үшін ең жиі қолданылатын деректердің жолдық типі. Тиісті құралдармен жұмыс істеуге арналған жиынтық қосымшаның 3.2Қ-кестесінде келтірілген.

Delphi-де программалық код модуль түрінде сақталады. Модульдің құрылымы Турбо Паскальдағы модуль құрылымына ұқсас, тек кейбір ерекшеліктері бар:

```
Unit <атау>;
Interface
        <интерфейсті бөлік>
Implementation
        <орындалатын бөлік>
Initialization
        <Инициализациялау бөлігі>
Finalization
        <аяқталу бөлігі>
End.
```

Мұндағы, **Unit** — модуль тақырыбын бастайтын тағайындалған сөз (бірлік); <атау> —модуль атауы (дұрыс идентификатор); **Interface** — модульдің интерфейсті бөлігін бастайтын тағайындалған сөз (интерфейс); **Implementation** — модульдің орындалатын бөлігін бастайтын тағайындалған сөз (орындау); **Initialization** — модульдің инициализациялау бөлігін бастайтын тағайындалған сөз (инициализация); **Finalization** — модульдің аяқталу бөлігін бастайтын тағайындалған сөз (аяқталу); **End** — модульдің аяғын білдіретін тағайындалған сөз.

Осылайша, модуль тақырыптан және кез-келгені бос болуы мүмкін төрт құрама бөліктен тұрады.

Бастапқы екі бөліктің қызметі Турбо Паскальдағыдай, сондықтан оларға толығырақ тоқталмай-ақ қояйық. Инициализациялау және аяқтау бөліктері көбінесе оларды бастайтын **Initialization** және **Finalization** сөздерімен бірге қолданылмайды. Инициализация бөлігінде негізгі программаны басқаруға дейін орындалатын және оның жұмысын дайындауға қолданылатын операторлар орналасады. Мысалы, мұнда айнымалылар инициализациялануы және қажетті файлдар ашылуы мүмкін. Аяқталу бөлігінде негізгі программаның аяқталуынан кейін орындалатын операторлар көрсетіледі. Егер инициализациялаушы бөлік бірнеше модульдерді қамтыса, бұл бөліктер бірінен соң бірі негізгі программадағы Uses-те модульдердің кезегімен орындалады.

Егер аяқтаушы бөлік бірнеше модульдерді қамтыса, бұл бөліктер бірінен соң бірі негізгі программадағы Uses-те модульдердің қарсы кезегімен орындалады.

Кейбір стандартты компоненттерді қолдануды қарастырайық.

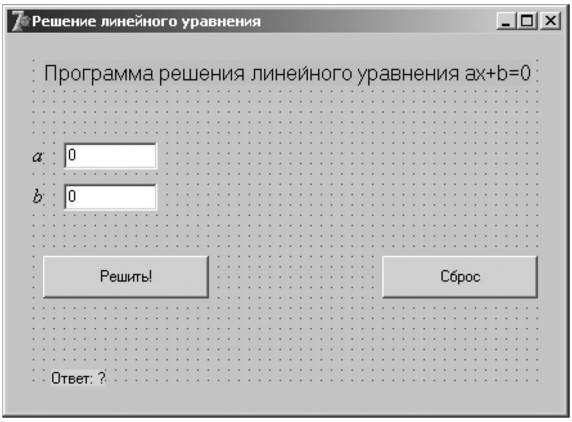
Мысал ретінде *ax + b = 0* түріндегі сызықтық теңдеуді шешетін программа құрайық.

Жобаны жүзеге асыруды жоспарлайық. Әлбетте, есепті шешудің пішіні шешілетін есептің мәтінімен тақырыпты қамтуы керек. Сонымен қатар, a, b мәндеріне арналған мәтін енгізу өрістері болуы керек. Нәтиже де нақты өрісте шығарылуы керек. Деректерді енгізгеннен кейін нәтижені алу үшін, пішінде сәйкес батырманы орналастыру қажет. Айнымалыларды инициализациялау және бұрын алынған нәтижелерден пішінді тазарту үшін ысыру батырмасы қажет.

Ескерту: Пайдаланушылардың дұрыс деректерді енгізуін болжаймыз. Сондай-ақ, Турбо Паскальға қарағанда, нақты санның бүтін және бөлшек бөліктерін Windows жүйесінің орыс тіліндегі ажыратушы белгі - үтір (егер басқа баптаулар таңдалмаса) болып табылады

Берілген есепті шешу программасының терезесі 4.10-суретте көрсетілген.

Пішінге қандай компоненттерді қою керектігін анықтайық. «ax + b = 0 сызықтық теңдеуді шешу программасы», «a», «b» және жауап алатын батырма ретінде Label компонентін аламыз. Ақпаратты енгізу үшін - Edit, және батырмалар үшін —Button компоненттерін таңдаймыз.



(*ax + b* = 0 сызықтық теңдеуді шешу программасы. Есептеу. Ысыру)

Сонымен, барлық қажет компоненттерді пішінге орналастырып. олардың өлшемдерін реттейміз. Компоненттер идентификаторларына үнсіз келісім бойынша мәндерді орнатамыз.

[Есептеу!] компоненті идентификаторын – Solution деп, [Ысыру] батырмасын — Clear, жауап шығару өрісін — Answer деп атайық. Қалған идентификаторлар

4.10-сурет. Қосымша интерфейсі пішіні

өзгеріссіз қалады.

Барлық компоненттердің Caption қасиетіне сәйкес мәтіндерді енгіземіз.

Ары қарай қолданушының әрекетіне жауап беруін қадағалаймыз. Біздің жағдайда, [Есептеу!] және [Ысыру] батырмаларын басқандағы әрекетті алдын ала ойластыру болып табылады.

[Ысыру] батырмасының коды:

```
Procedure TForml.ClearClick(Sender : TObject);
Begin
    Editl.Text := ' 0 ' ;
    Edit2. Text := ' 0 ' ;
    Answer.Caption := 'Жауап:?';
End;
```

[Есептеу!] батырмасының коды:

```
Procedure TForm1.SolutionClick(Sender : TObject);
Var a, b : real; x : real; s : String;
Begin
    a := StrToFloat(Edit1.Text);
    b :=StrToFloat(Edit2.Text);
    If a <> 0
    Then Begin
                    x := —b/a;
                    s := 'Жалғыз шешім:                    ';
        End
    Else If a = 0
            Then If b = 0
                    Then s := 'Есептеудің шексіз мәні' Else s := 'шешімі жоқ';
    If a<>0
    Then Answer.Caption := 'Жауап:                    ' + s +
FloatToStr(x)
    Else Answer.Caption := 'Жауап:                    ' + s;
End;
```

Соңғы кодтағы типтерге назар аударайық. Өңдеу өрісінде деректер мәтін форматында енгізіледі және әрі қарай өңдеу үшін оларды сандық форматқа аудару керек. Деректерді шығару үшін, керісінше, тек жолдық мәндері қажет.

Бұл программа кез-келген нақты деректер үшін берілген сызықты теңдеуді шешуге мүмкіндік береді. Оны тестілеу үшін бізге барлық тармақтарды тексеруді қамтамасыз ететін кемінде үш сынақ қажет.

Жүзеге асырылатын жобаның толық модуль коды:

```
Unit Unitl;
Interface
```

```
Uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;
Type
    TForm1 = Class(TForm)
                        Label1 : TLabel;
                        Label2 : TLabel;
                        Edit1 : TEdit;
                        Label3 : TLabel;
                        Edit2 : TEdit;
                        Solution : TButton;
                        Clear : TButton;
                        Answer : TLabel;
    Procedure SolutionClick(Sender : TObject); Procedure ClearClick(Sender :
    TObject);
    Private
            {Private declarations}
    Public
            {Public declarations}
    End;
Var Form1 : TForm1;
Implementation
    {$R *. dfm}
Procedure TForm1.SolutionClick(Sender : TObject);
Var a, b : Real; x : Real; s : String;
Begin
    a := StrToFloat(Edit1.Text); b := StrToFloat(Edit2.Text);
    If a <> 0
    Then Begin
                    x := -b/a;
```

```
                s          Единственное решение:
            End
    Else If a = 0
            Then If b = 0
                    Then s := 'Шексіз шешімдер' Else s := 'шешім жоқ';
    If a <> 0
    Then Answer.Caption := 'Жауап:                    ' + s +
    FloatToStr(x)
    Else Answer.Caption := 'Жауап:                    ' + s;
End;
Procedure TForm1.ClearClick(Sender: TObject);
Begin
            Edit1.Text := ' 0 ' ;
            Edit2. Text : = ' 0 ' ;
            Answer.Caption := ' Ж а у а п : ? ' ;
    End;
End.
```

Берілген программа жұмысқа дайын. Компиляция келесі түрде жүзеге асырылады:

- *Проект/Опции* (Project/Option) командасын орындау, одан кейін *Опции проекта* (Project Options) сұхбат терезесі ашылады;
- *Каталог/ Условия* (Directories/Conditionals) парағын таңдау;
- *Каталоги* бөлімінде және Результат жолында (Output directory) жоба файлының бумасына дейінгі жолды көрсету. Бұл файлда жобаның нәтижелі exe-файлы сақталады;
- *Проект/Компилировать* (Project/ Build All) командасын таңдау.

Exe-файл атауы жоба файлы атауымен бірдей болып табылады. Бұл файлды Delphi ортасынан тыс аймақтарда да жүктеуге болады. Егер exe-файл қажет болмаса, онда *Результат* (Output directory) жолын тазалап тастау қажет. Бұл жағдайда файл жадыға компиляцияланатын болады.

Түзету процесі кезінде программаны жүктеу үшін F9 пернесін басуға болады.

ЖАТТЫҒУЛАР

1. Сызықты теңдеуді шешу қосымшасы негізінде келесі есептерге қосымша құрыңдар:

а) квадрат теңдеуді шешу қосымшасы;

б) екі белгісізі бар екі сызықты теңдеулер жүйесін шешу қосымшасы.

2. Сызықты теңдеуді шешу қосымшасы мен бірінші жаттығудағы қосымшаларды мәліметтерді енгізуді тексерумен толықтырыңдар.

3. 4.2-бөлімдегі екінші жаттығуға қосымша құрыңдар (жай бөлшектермен жұмыс).

4.6.

DELPHI-ДЕ ҚОСЫМШАЛАРДЫ ҚҰРУ ТЕХНОЛОГИЯСЫ

Объектіге бағдарланған көзқарас кез-келген программалау парадигмасы программалық өнімдерді дамытудағы бірқатар кезеңдердің өтуін білдіреді. Бұған дейін біз программалаудың құрылымдық әдісін қолдана отырып, программаны әзірлеу кезеңдерін, сондай-ақ объектіге-бағытталған жобалау әдісін сипаттадық. Әрине, объектілерді пайдалану, қолданба жасау кезінде әрекеттердің кезектілігіндегі кейбір ерекшеліктерді анықтайды, бірақ бұрын талқыланған және қолданылғандардың көбі күшінде қалады. Оқу қосымшасын құру - бұл «ересек» программалық өнімдерді әзірлеу бойынша әрекеттерді модельдеу, бұл оның негізгі ерекшеліктерін сақтай отырып, процестің жеңілдетілуін білдіреді. Осы тұрғыдан алғанда, Delphi-дегі қосымшаларды әзірлеу сатыларын қарастырамыз. Сондай-ақ, программаны құру процесі көбінесе итеративті болып табылады, яғни алдымен талдау, интерфейсті немесе оның бөліктерін жобалау жүзеге асырылады, содан соң программа соңғы нұсқаны алғанға дейін кеңейтіледі.

4.5–бөліміндегі жасалған программада осы кезеңдердің көбісі қамтылған. Оған кеңінен тоқталайық

Қосымшаларды өңдеу кезеңдері. Өңдеу кезеңдерінің жеңілдетілген кезеңдері келесідей:

- есептің қойылымы, пәндік аймақты зерттеу, модель құру (математикалық, ақпараттық), құрылған модельді ОБП қосымшасында көрсету;
- ОБП-қосымшасын жобалау;
- қолданушы интерфейсін жасау;
- қосымшаны программалау, программалық кодты тестілеу және дұрыстау;
- құжаттамаларды әзірлеу.

4.11-сурет. «Калькулятор» қосымшасының интерфейсі

Бірінші кезеңге тоқталмаймыз, себебі оның мазмұны программалаудың құрылымдық тақырыбында қарастырылған болатын және оның ОБП-дан аса айырмашылығы жоқ.

Өңдеудің келесі кезеңдеріне толығырақ тоқталып өтеміз.

ОБП-қосымшаларын жобалау. Өңдеудің бірінші кезеңі нәтижесі зерттелетін процестің ақпараттық моделі, құбылысы болып табылады (бұл жағдайда маңызды есептерді шешу). Шын мәнінде, бұл модель қазірдің өзінде кем дегенде, олардың арасында объектілер мен қатынастардың жиынтығын (яғни бастапқы орындау жүйесі үшін декомпозиция орындалды) қамтиды. ОБП-қосымшаларын жобалау кезеңінде әр объектке нақты атау (идентификатор тағайындау) берілуі тиіс. Мұнда объектілердің қайсысы визуализацияланатынын, яғни Delphi терминдерінде, объектінің көрсетілетініне шешім қабылданады.

Әрбір таңдалған объект үшін сипаттамалардың (қасиеттерінің) тізімін көрсету керек. Объектідегі әрекеттер көптеген әдістер арқылы көрсетіледі. Көбінесе графикалық интерфейсті операциялық жүйеге арналған (мысалы, Windows) осындай әрекеттер ОБП-қосымшаларда ішкі және сыртқы реакция ретінде көрінеді.

Қолданушы интерфейсін өңдеу. Қолданушы интерфейсін өңдеу кезінде, қосымшаның функционалдылық тағайындауынан абай болу керек. Delphi-де интерфейсті өңдеу ерекшелігі - оның көптеген элементтері (пішін, компоненттер және т.б.) алдын ала программалау ортасында қамтылған. Осылайша, интерфейс қолданыстағы «кірпіштер» негізінде құрылады және әлеуетті түпкі пайдаланушының сұрауларына бейімделеді.

Пайдаланушы интерфейсін әзірлеу кезінде бірқатар ережелер сақталуы керек.

1. Мәліметтерді енгізудегі терезелерді (пішіндерді) жобалау кезінде:

- қолданушы тек тышқанды ғана пайдаланбайтындай командаларға әрқашан пернелер комбинациялары баламасын құру керек;
- негізгі пішенде элементтер қолданушының тапсырмаларымен бірлесіп орналасуы қажет;
- мәліметтерді енгізу барысында қолданушымен байланыс көрініп тұруы қажет (мысалы, синтаксистік бақылау, немесе қателерді түзеу);
- Әр түрлі енгізу пішіндеріне ерекшеленетін интерфейсті қолданудың қажеті жоқ.

2. Мәзірді құру барысында:

- Windows-та қабылданған мәзір пунктерінің орналасуының

стандартты келісімі негізінде құрған абзал;

- мәзірге пункттерді логикалық ретпен және мазмұны бойынша топтастырған дұрыс;
- ашылатын мәзірлердегі пункттерді топтастыру үшін, ажыратқыш сызықтарды пайдаланған жөн;
- тым көп артық мәзірлердің қажеті жоқ;
- командалардың пернетақталық эквиваленттерін және «ыстық пернелерді» қолдану;
- құрал-саймандар тақтасына жиі қолданылатын командаларды орналастыру.

3. Қосымшаның күту процесімен жұмыс істеу барысында қолданушыға жұмыс барысын хабарлап отырған жөн (мысалы, тапсырманы орындаудың индикатор жағдайы арқылы).

Жалпы дизайнерлік талаптар мынадай: қосымшаның фондық түсі, пайдаланылатын түстердің саны мен оған көрсетілген мәтіннің түстері және т.б. үйлесімді болғаны жөн.

Программалау, тестілеу, түзету. Жоғарыда айтылғандай, қосымшаны жобалау кезеңінде оның барлық элементтерінің функционалды мақсаты сыртқы және ішкі оқиғаларға ықтимал жауаптар береді.

Қосымшаның интерфейсіне қатысты программа кодының бөлігі тиісті элементтердің қасиеттерін қою және баптау кезінде автоматты түрде жасалады. Оқиғалардың өңделуіне қатысты кодтың бір бөлігін программалаушы жасайды. Сонымен қатар, программа коды жекелеген бөліктердің жұмыс істеуі немесе тұтастай қолданыс үшін қажетті қосалқы ішкі программаларды қамтуы мүмкін. Мұндай ішпрограммалардың коды кейінірек талқыланады.

Әлбетте, әрбір әзірленген әдіс пен қосалқы ішпрограмма және тұтас кешен тестілеу процедурасынан өтеді. Тестілеу әдістері құрылымдық программалардың кодын тексеру үшін қолданылатын әдістерден ерекшеленбейді. Әрбір әдіс немесе ішпрограмма үшін бұл әдісті өтуде мүмкіндігінше көп сынақтар бар, барлық шекаралас және қолайсыз жағдайлар тексеріледі. Бұл жағдайда тест жинақтарына қойылатын талаптар сақталады. Программа кодында деректерді енгізу барысында оларды алдын ала тексеріп алу қажет.

Ары қарай программаның әртүрлі элементтері мен жалпы қосымшаның өзара әрекеттестігі тестіленеді. Сынақ жинақтары тестілеу процедурасы басталғанға дейін әзірленетінін ескерейік. Жалпы, олар қосымшаның құжаттамасының бір бөлігі болып табылады. Қазіргі уақытта программалық қамтамасыз етуді әзірлеу кезінде шешілетін есептің моделі негізінде құрылған тест жиынтықтарының автоматты генерациясы жүйелері қолданылады.

Оқыту программалауында тест жинақтары «қолмен» жасалады.

Құжаттамаларды өңдеу. Кез-келген программалық өнім құжаттамамен бірге жүреді. ОБП қосымшасын әзірлеу кезінде құжаттаманың құрамы мен мақсаты басқа программалау парадигмалары үшін программалық жасақтама әзірлеуге қойылатын талаптарға сай болуы керек. Құжаттаманың ең аз жобасы программалаушы нұсқаулығы және пайдаланушы нұсқаулығы болып табылады. Құжаттаманың құрамы мен құрылымы әдетте стандартталған болып келеді.

Программа кодының өзін-өзі құжаттауы, атап айтқанда, программа мәтініндегі оның блоктары туралы түсініктемелерді атап өту маңызды.

Оқу бағдарламаларын әзірлеу кезінде есептік құжаттаманың келесі құрамы ұсынылады:

- есептің қойылымын сипаттау;
- модельді сипаттау;
- жобаны сипаттау (пішіндегі элементтер саны, олардың функциялары, өзара әрекеттесуі және басқа қосымшалармен әрекеттесуі) және объект қасиеттерін инициализациялау;
- әрбір әдіске, қосалқы ішпрограммаларға, интерфейстік бөлікке және жалпы қосымшаға тестілік жинақ;
- түсіндірмелермен бірге программалық код;
- программа жұмысының нәтижелері (осы нәтижелерді күтетін есептеулерді немесе басқа бір әрекеттерді орындау барысында).

4.7. DELPI ҚОСЫМШАЛАРЫН ӨҢДЕУ МЫСАЛДАРЫ

Программалық өнімнің өңдеу кезеңдерінен өту мысалдарын келтірейік.

4.4-Мысал. p ($2 < p < 9$) негіздегі санау жүйесіндегі сандармен әрекеттерді орындайтын калькулятордың программасын құру қажет.

Жүзеге асырылатын жобаны және қосымша интерфейсін жоспарлайық. Пішінде сандарды басатын батырмалар болу қажет. Егер қандайда бір сандар ағымдағы негіздегі санау жүйелеріне кірмесе, олар жасырынып қалулары қажет. Санау жүйесінің негізін таңдау үшін пішіндегі ашылатын тізімді пайдаланамыз. Нәтижелер (аралық және соңғы) белгілі бір өрісте пайда болады. Тағы төрт батырманы арифметикалық амалдарға арнаймыз. Сонымен қатар, [=] және [C] ысыру батырмалары қажет.

Қосымшаның болжамды жұмысын сипаттайық:

- сандық батырманы басқан уақытта кезекті сан ағымдағы санның жазбасы ретінде нәтиже өрісінде көрінетін болады;
- [C] батырмасын басқанда нәтиже өрісінде нөл пайда болады;
- амал белгілері бар түймелерді басқан кезде, ағымдағы мән бұрын таңдалған амалда екінші операнд (бірінші еске немесе кем дегенде нөлге тең деп есептеледі) ретінде пайдаланылады, нәтиже және ағымдағы таңдалған әрекет өрісте сақталады нәтиже нөлмен көрсетіледі (өріс келесі нөмірді енгізуге дайын);
- [=] батырмасын басқанда соңғы таңдалған амалдың нақты жауабы шығады;
- санау жүйесінің негізін таңдағанда нәтиже өрісіне сол санау жүйесіне алмастырылған жауап шығады.

Берілген есепті шешу терезесі үлгісі 4.11-суретте көрсетілген.

(Калькулятор. Негіз)

Бірнеше листингтер келтірейік (басқаларын ұқсас түрде өз беттеріңше келтіріңдер).

«Санау жүйесінің негізін таңдау» оқиғасының процедуралық коды келесідей (ComboBox компоненті қолданылды):

```
Procedure TCalculator.OsnovanieSelect(Sender: TObject);
Var a, p : Integer; s : String;
Begin p := pOld;
      PTo10(Rezalt.Text, p, a); pOld :=
    StrToInt(Osnovanie.Text);
```

```
  10toP(a, pOld, s);
Rezalt.Text := s;
Case StrToInt(Osnovanie.Text) Of
2: Begin
              2. Visible      = False;
              3. Visible      = False;
              4. Visible      = False;
              5. Visible      = False;
              6. Visible      = False;
              7. Visible      = False;
              8. Visible      = False;
           End;
3          Begin
              2. Visible      = True;
              3. Visible      = False;
              4. Visible      = False;
              5. Visible      = False;
              6. Visible      = False;
              7. Visible      = False;
              8. Visible      = False;
           End;
4          Begin
              2. Visible      = True;
              3. Visible      = True;
              4. Visible      = False;
              5. Visible      = False;
              6. Visible      = False;
              7. Visible      = False;
              8. Visible      = False;
          End;
```

```
5: Begin
                2. Visible  = True;
                3. Visible  = True;
                4. Visible  = True;
                5. Visible  = False
                6. Visible  = False
                7. Visible  = False
                8. Visible  = False
End;
6:  Begin
                2. Visible  =  True;
                3. Visible  =  True;
                4. Visible  =  True;
                5. Visible  =  True;
                6. Visible  =  False
                7. Visible  =  False
                8. Visible  =  False
    End;
7:  Begin
                2. Visible  =  True;
                3. Visible  =  True;
                4. Visible  =  True;
                5. Visible  =  True;
                6. Visible  =  True;
                7. Visible  =  False
                8. Visible  =  False
    End;
8:  Begin
                2. Visible  =  True;
                3. Visible  =  True;
                4. Visible  =  True;
                5. Visible  =  True;
                6. Visible  =  True;
                7. Visible  =  True;
                8. Visible  =  False
    End;
9:  Begin
                2. Visible  =  True;
                3. Visible  =  True;
                4. Visible  =  True;
                5. Visible  =  True;
                6. Visible  =  True;
                7. Visible  =  True;
_8.Visible := True;
```

```
            End;
        End;
    End;
```

[0] батырмасының мысалында батырманы тышқанмен шерту:

```
Procedure TCalculator. 0Click(Sender : TObject);
Begin
    If Rezalt.Text = '0' Then Rezalt.Text :=
  0.Caption
    Else Rezalt.Text := Rezalt.Text +                    0.Caption
End;
```

[+] амалы белгісін тышқанмен шерту:

```
Procedure TCalculator.PlusClick(Sender : TObject);
Begin
    Operation;
    Znak := '+';
End;
```

[=] батырмасын шерту:

```
Procedure TCalculator.Button1Click(Sender : TObject);
Var s : String;
Begin
    Operation;
    Znak : = '          ';
    _10ToP(a, POld, s);
    Rezalt.Text := s; a : = 0;
End;
```

Сондай-ақ ондық жүйеден p негізіндегі жүйеге және керісінше жүйеге аударуды, және арифметикалық амалдардың нақты орындалуын жүзеге асыратын көмекші процедуралардың орындалуын ұсынамыз:

```
Procedure 10ToP(a, p : Integer; Var s : String); Begin
    If a = 0 Then s := '0' Else s := '';
    While a <> 0 Do Begin
            s := IntToStr(a mod p) + s; a := a div p
    End;
End;
Procedure PTo10(s : String; p : Integer; Var a : Integer);
Var i : Integer;
Begin
    a : = 0;
    For i := 1 To length(s) Do
            a := a * p + StrToInt(s[i])
```

```
End;
Procedure Operation;
Var r : Integer;
Begin
    pTo10(Calculator. Rezalt.Text, POld, r);
  If a = 0 Then a := r Else Case Znak Of
                          + ':    a : = a   + r;
                          –':     a : = a   _ r;
                          *':     a : = a   * r;
                          /'      a : = a   div r
                          End;
                          Calculator. Rezalt.Text :=
                          '0';
End;
```

Көрсетілген процедуралар негізгілердің құрамына кірмейді және жоба модулін жүзеге асыру бөлімінде (Implementation) ғана болады.

Қосымшаны жүктеген уақытта, айнымалылар инициализацияланады (ауқымды және компоненттердің кейбір өрістері):

```
Procedure TCalculator.FormCreate(Sender :
TObject);
Begin
        2. Visible  = False;
        3. Visible  = False;
        4. Visible  = False;
        5. Visible  = False;
        6. Visible  = False;
        7. Visible  = False;
        8. Visible  = False;
        pOld := 2; a    := 0;
End;
```

4.5-Мысал. Бастауыш сынып оқушыларына арналған келесі ойынды модельдеуге арналған программаны құру керек. Өрісте қара және ақ шарларға арналған екі қорап, сондай-ақ қара және ақ шарлар бар (2-ден 10-ға дейін). Бұл шарларды қораптарға салып, олардың қайсысының саны көп - қара немесе ақ екенін көрсету қажет.

Талап етілетін қосымшаны іске асыру кезінде программаларды Windows жүйесіндегі Drag & Drop (апарып тастау) деп аталатын деректермен байланыстырудың арнайы әдісін қолданамыз. жүзеге асыру барысында Delphi объектілеріне қатысты осы механизмнің тиісті оқиғалары мен қасиеттерін пайдалану сипатталады.

Жобаны және қосымша интерфейсін шамамен жоспарлайық. Пішінде екі қорап (TLabel компоненттері) және он шар (TShape компоненттері) болады. Сонымен қатар программаны іске қосқан сайын, шарлар саны мен олардың түсі кездейсоқ орнатылады.

Қосымшаны жасауды сипаттайық:

- қосымшаны іске қосқанда компоненттердің қажет қасиеттері инициализацияланады және шарлардың саны, түсі тағайындалып, артық шарлар жасырылады;
- шарды «өзінің» қорабына апарғанда және тышқан шертпесін жібергенде ол жоғалып кетеді, ал қораптағы шарларды есептеуіш 1-ге артады;
- шарды «өзге» қорапқа сүйреп апару барысында ол қорапқа салынбайды, және шар өз орнында қалады.

Осы есептің пішін үлгісі 4.12-суретте көрсетілген.

Бұл программаның нәтижесі 4.13-суретте келтірілген.

Жасалып отырған қосымшаның жұмыс логикасын программалық код негізінде сипаттайық.

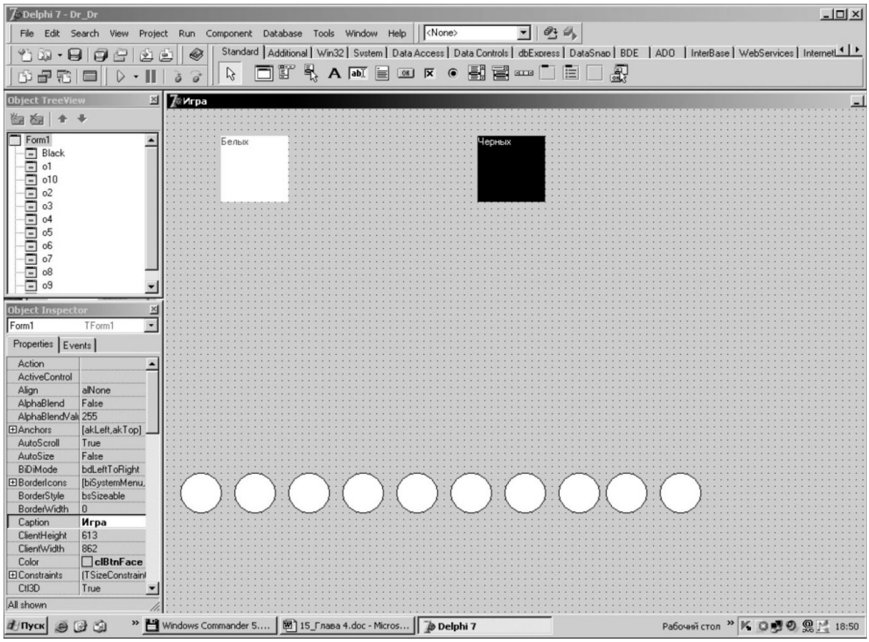
Қосымшаны жүктегенде барлық қасиеттерді инициализациялап, шарлардың кездейсоқ сандарын пішінге орналастыру керек.

Берілген процедураның программалық коды келесідей:

```
Procedure TForml.FormCreate(Sender : TObject Var i : Integer;
```

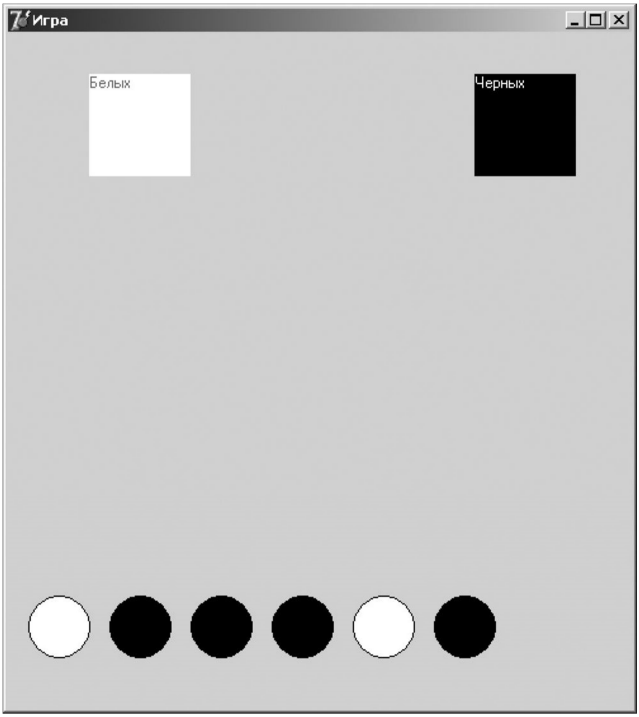
```
    b : = 0; w : = 0; Black.Height :=    Black.Width := 80;
    80; Black.Hint :='b';
    White.Height := 80;                  White.Width := 80;
    White.Hint := 'w';
    a[1]:= o1; a[2]:=                    o2; a[3]          := o3;
    a[4]:= o4; a[5]:=                    o5;

Begin
```



4.12-сурет. «Ойын» қосымшасы интерфейсін құру пішіні

```
    a[6]:= o6; a[7]:= o7; a[8]:= o8;
    a[9]:= o9; a[10]:= o10;
    randomize; n := random(9) + 2;
    {артық шарларды жасырамыз}
    For i := n + 1 To 10 Do With a[i] Do Visible :=
    False;
{Қалған шарларды ақ немесе қара түске бояймыз}
    For i : = 1 To n Do With a[i] Do
    Begin
            If random(2) = 0
            Then Begin
                        Brush.Color := clBlack;
                        Hint :='b'
                End
            Else Begin
                        Brush.Color := clwhite;
```



4.13-сурет. «Ойын» қосымшасының терезесі

Ақ. Қара

```
                                    Hint := 'w'
                            End;
        DragMode := dmAutomatic
        End;
    End;
```

Мұндағы *b, w* — қорапқа түскен сәйкес ақ және қара шарлардың санауышы; Black, White — компоненттер атауы, яғни ақ және қара шарларды сақтайтын қораптардың атауы.

Hint қасиетін орнату қорапқа салынатын шардың қасиеті мәнімен бірдей болу мақсатында қойылады.

Әрбір шарлар туралы ақпарат (әрі қарай пайдалану ыңғайлылығы үшін) жиымда орналастырылады.

Көрсетілген шарлардың саны, қосымша жасырынуы, кездейсоқ түске (қара немесе ақ) боялуы, белгілі бір түстің шарына арналған Hint қасиетінің тиісті мәнін орнатады.

DragMode қасиеті Drag&Drop-пен байланысқан барлық әрекеттер кешенінің орындалуын анықтайды. Егер DragMode мәні dmManual болса, онда сүйреу әрекеттерінің барлығы қолмен жасалады (яғни программалаушы программа орындалу барысында бұл әрекеттерді өзі орындайды), және арнайы әдістерді шақырғаннан кейін ғана сүйреу әрекеті орындалады. Егер DragMode мәні dmAutomatic болса, онда сүйреу әрекеттерінің оқиғалары автоматты түрде анықталып, олар қолданушы тышқанмен шерткенде бірден жүзеге асады. Біздің жағдайда dmAutomatic мәні орнатылады.

Ары қарай объектілерді сүйреу әрекетімен байланысатын оқиғаларды өңдеу қажет.

Сүйретілетін объектті қабылдау дайындығын тексеру OnDragOver қасиетімен байланысты.

Тексеру программасы мынадай:

```
Procedure TForml.BlackDragOver(Sender, Source :
TObject; X, Y : Integer; State: TDragState; Var
Accept : Boolean);
Begin
    If (Sender Is TLabel) And (Source Is TShape) Then Accept := ((Sender As
    TLabel).Hint = (Source As
TShape).Hint)
End;
```

Sender параметрі объектінің жылжытатын компонентін көрсетеді. Source параметрі жіберуші құрамдасы туралы ақпаратты қамтиды. X және Y параметрлері Sender компонентіне қатысты пиксельдермен көрсетілген тышқан көрсеткішінің координаталарын қамтиды. State параметрі Sender компонентіне қатысты жылжытылған объект жағдайын анықтайды. Accept параметрі Sender апарылған деректерді қабылдауға дайын немесе дайын емес екенін көрсетеді. Егер параметр True болса, онда Sender апарылған нысанды қабылдауға дайын деген сөз.

Осы мысалда көрсетілген оқиғаны өңдеу жіберушінің Hint қасиетінің және қабылдағыштың мәндерін салыстырудан тұрады.

Оқиғалар Black объектісі үшін орындалады. White объектісінде сәйкес оқиғаға сілтеме сол оқиғалар үшін жасалған.

Сүйреу әрекеті соңында (алынған деректердің қабылданғанына немесе қабылданбағанына қарамастан) объектіні жылжыту үшін OnDragEnd оқиғасы туындайды. Бұл оқиға, сондай-ақ, апарып тастаудан бас тартқанда да орын алады және оны орындау міндетті емес. Сүйреу әрекеті осы оқиғаны өңдемей-ақ орындалуы мүмкін.

Алғашқы (басқа шарлар үшін осы оқиғаны өңдеуге сілтеме жасалған) шар үшін өңдеуді жүзеге асырайық:

```
Procedure TForml.o1EndDrag(Sender, Target:
TObject; X, Y: Integer);
```

```
Var s: String;
Begin
    If Target <> Nil Then Begin
        If (Target As TLabel).Name = 'Black'
        Then Begin b := b + 1; str(b, s);
                        (Target As TLabel).Caption := 'Қара' +#10 +#13 + s
                End
        Else Begin w := w + 1; str(w, s);
                        (Target As TLabel).Caption := 'Ақ' +#10 +#13 + s
                End;
        (Sender As TShape).Visible := False;
    End;
End;
```

Қабылданған шарлардың сәйкес санағышын ұлғайтамыз және қорапқа лақтырылған шарды жасырамыз.

Программалық модульдің толық нұсқасы:

```
Unit Ex 3;
Interface
Uses Windows, Messages, SysUtils, Variants,
Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
TYPe
    TForm1 = Class(TForm)
            White : TLabel;
            Black : TLabel;
            1     : TShape;
            2     : TShape;
            3     : TShape;
            4     : TShape;
                    o5 :  TShape;
                    o6 :  TShape;
                    o7 :  TShape;
                    o8 :  TShape;
                    o9 :  TShape;
                    o10   : TShape;
        Procedure FormCreate(Sender : TObject); Procedure
BlackDragOver(Sender, Source : TObject; X, Y : Integer; State : TDragState; Var
Accept : Boolean);
        Procedure o1EndDrag(Sender, Target : TObject; X, Y :
Integer);
    End;
Var Form1 : TForm1; n : Integer; a : Array[1..10] Of TShape; b, w: byte;
Implementation
```

```
        {$R *. dfm}
Procedure TForm1.FormCreate(Sender: TObject);
Var i : Integer;
Begin b := 0; w : = 0;
    Black.Height := 80;
    Black.Width := 80;
    Black.Hint := 'b';
    White.Height := 80;
                    White   . Width   =
                    White   . Hint    := 'w';
                    a[1]    := o1;    a[2]    :=  o2;
                    a[3]    : = o3;   a[4]    :=  o4;
                    a[5]    : = o5;   a[6]    :=  o6;
                    a[7]    := o7;    a[8]    : = o8;
                    a[9]    := o9;    a[10]     :  = o1
                    randomize;
                    n : =     random(9) + 2;
    {Артық шарларды жасырамыз}
    For i := n + 1 To 10 Do
        With a[i] Do Visible := False;
        {Қалған шарларды ақ немесе қара түске бояймыз}
    For i : = 1 To n Do With a[i] Do
```

```
        Begin
            If random(2) =              0
            Then
            Begin Brush.Color := clBlack; Hint := 'b' End
            Else
            Begin Brush.Color := clwhite; Hint := 'w' End; DragMode := dmAutomatic
        End;
End;
Procedure TForm1.BlackDragOver(Sender, Source : TObject; X, Y : Integer; State :
TDragState;
Var Accept: Boolean);
Begin
    If (Sender Is TLabel) And (Source Is TShape)
    Then
        Accept := ((Sender As TLabel).Hint = (Source As TShape).Hint)
End;
Procedure TForm1.o1EndDrag(Sender, Target :
TObject; X, Y : Integer);
Var s : String;
Begin
    If Target <> Nil Then Begin
        If (Target As TLabel).Name = 'Black'
        Then Begin b := b + 1; str(b, s);
                        (Target As TLabel).Caption:= 'Қара' + #10 + #13 + s
                End
        Else Begin w := w + 1; str(w, s);
                        (Target As TLabel).Caption := 'Ақ' + #10 + #13 + s
            End;
        (Sender As TShape).Visible := False;
    End;
End;
End.
```

ЖАТТЫҒУЛАР

1. Басқа сандық батырмалар және амалдар белгісі бар батырмалар үшін 4.4-мысалының программасын жалғастырыңдар, нәтижесінде жұмыс істеп тұрған калькулятор алыну керек.

2. 4.5 мысалының пішінін (4.12-сурет) [Жабу] батырмасымен толықтырыңдар. Бұл батырма барлық шарлар қорапқа орналасқан соң ғана пайда болу керек. Батырманың басылуының нәтижесінде қосымша жабылатындай етіп өңдеңдер.

4.8. КЛАССТАР ИЕРАРХИЯСЫ

Delphi-де қабылданған объектіге-бағытталған программалауға қатысты терминология Турбо Паскаль терминологиясынан біршама ерекшеленеді. Осы терминологияны, сондай-ақ Delphi программасында қабылданған программалауға объектілі көзқарасты және оның негізгі қағидаттарын іске асыру жолдарын қарастырайық.

Delphi-дегі класстар. Класстарды, қасиеттерді және әдістерді жариялау. Турбо Паскальдағы объектілерді Delphi-де класс деп атауға болады. Delphi-дегі *класстар* келесі элементтерді қамтитын арнайы типтер болып табылады: өрістер, әдістер және қасиеттер. Кез-келген басқа типтегідей, класс тек *объектілер* деп аталатын белгілі бір іске асыру даналарын жасау үшін үлгі ретінде қызмет етеді. Бұл термин C ++ тілінде пайдаланылады.

Басқа типтердегі класстар арасындағы маңызды айырмашылық, олардың объектілері үнемі реттелмеген құрылымға бөлінген (Variable), сондықтан «айнымалы» объектісі бұл шын мәнінде динамикалық жад аймағына нұсқағышты ғана ұсынады. Дегенмен, басқа нұсқағыштардан оның айырмашылығы, егер объектінің мазмұнына сілтеме жасайтын болсақ, объектінің атының артындағы «^» белгісіне тыйым салынады.

Delphi-де қолданылатын барлық класстар (кітапханалық және программалаушылар әзірлеген) TObject класынан мұраланған. Жариялау кезінде бұларды «ата-ана» ретінде көрсету міндетті емес.

Жаңа классты сипаттайтын пішім келесідей:

```
Type <Класс типінің идентификаторы> = Class
 Private <класстың жасырын элементтері>;
 Protected <класстың қорғалған элементтері>;
Public <класстың жалпыға рұқсат элементтері>;
Published <класстың жарияланған элементтері>;
End;
```

Классты жариялау барысында берілген шаблондағы секциялардың барлығы қолданыла бермейді. Олардың мағынасын түсіндірейік.

Private секциясында ішкі элементтері бар, оларды ішінде класстың жариялануы бар модульдер шегінде ғана шақыра аламыз.

Protected секциясында қорғалған элементтер бар. Оларға ішінде класстың жариялануы және класс «ұрпақтары» бар модульдер шегінде ғана қол жеткізе аламыз.

Public секциясында программаның кез-келген бөлігінде шақыра алатын жалпыға рұқсат элементтер сақталған.

Published секциясында Public секциясындағы сияқты программаның кез-келген бөлігінде шақыра алатын жалпыға рұқсат жарияланған элементтер сақталған.

Класстың «ұрпақтары» Private бөлімінде жарияланғандардан басқа «ата-ана» класының барлық элементтерінің қол жетімділігін өзгерте алады.

Егер класстың элементтерін жариялаған кезде секцияның қандай бөлігі екенін көрсетпесе, онда бұл элементтерге қол жеткізу Public бөлімінде жарияланғандай орындалады. Delphi-де кез-келген секцияны бірнеше рет жариялауға болады, ал бөлімдердің реті маңызды емес және кез-келген бөлім бос болуы мүмкін.

Мұнда класс жариялануының мысалы келтірілген. Біз «Жай бөлшек» деп аталатын классты «алым мен бөлімнің ЕҮОБ», «Қысқарту», «Натурал дәреже» әдістерімен сипаттай аламыз (4.1-мысалды қара):

```
Type Natur = 1..32767;

     Frac = Record P : Integer; Q : Natur End;
     Drob = Class
         Procedure Vvod; {Бөлшекті таңдау}
         Procedure Print; {Бөлшекті шығару}
         Procedure NOD(Var C : Natur);
         Procedure Sokr;
         Procedure Stepen(N : Natur; Var C : Frac);
         Property A : Frac Read Print Write Vvod;
     End;
```

Мұнда өрістерді, әдістерді және қасиеттерді біртұтас біріктіру, Турбо Паскальдағыдай инкапсуляция деп аталады. Өрістер кез-келген типте болуы мүмкін, соның ішінде класс та бола алады. Класстағы процедуралар мен функциялары бойынша инкапсуляциялау әдіс болып табылады. Қасиеттер - өрістерге қатынасуды реттейтін арнайы класс механизмі. Қасиеттер резервтелген Property, Read және Write сөздері (Read және Write сөздері қасиеттерді жариялау мәтін мәнінде ғана резервтелген бола алады) арқылы жарияланады. Әдетте, қасиет кейбір өрістермен

байланыстырылады және осы өріске жазу немесе оқу кезінде пайдаланылуы керек класс әдістерін көрсетеді.

Турбо Паскальдағыдай әдістерді енгізу класс декларациясынан тыс жүзеге асырылады, алайда класстың іске асырылу әдісін көрсету қажет. Мысалы, 4.1-мысалдан Vvod әдісін жүзеге асыру барысында тақырып келесідей болады:

```
Procedure Drob.Vvod;
```

Мұрагерлік. Кез-келген класс басқа бір класстан пайда болуы мүмкін. сол үшін оның жариялануында «ата-ана» класының атауы көрсетіледі:

```
<еншілес класс> = Class (<ата-аналық класс>)
```

Білімді класс автоматты түрде өрістерін, әдістерін, сондай-ақ оның «ата-ананың» қасиеттерін мұраға алады және осыған жаңаларын қоса алады. Мұрагерлік принципі күрделі классты біртіндеп жасауға мүмкіндік береді және өзіндік класс кітапханаларын дамытады.

Delphi-дің барлық класстары бір TObject «ата-анадан» қалыптастырылған. Оның ешқандай өрістері және қасиеттері жоқ, бірақ ол кез-келген объектілердің жаратылысынан бастап жойылуына дейінгі толық өмірлік циклын қамтамасыз ету үшін жалпы пайдалану әдістерін қамтиды.

Мұрагерлік принципі TObject класының «ұрпақтарына» жылжытылған кезде біртіндеп созылып, бір тармақталған ағаштың класстарының құрылуына әкеледі. Әрбір «ұрпақ» өзінің жаңа «ата-анасын» толықтырады және оларды өздерінің «ұрпақтарына» жібереді.

Delphi-де жиындалған мұрагерліктер жүзеге асырылмаған, яғни «еншілік» класс тек бір ғана «ата-ананы» қамтиды.

Мысал ретінде жоғарыда қарастырылған «Есептеуіш» (4.2-мысалды қара) класын қарастырайық:

```
        Type BaseType = Double;
        Vichislitel = Class
    A, B, C : BaseType;
    Procedure Init; {Өрістерді енгізу немесе инициализациялау}
    Procedure Slozh;
    Procedure Vich;
    Procedure Umn;
    Procedure Delen;
End;
```

```
        NovijVichislitel = Class(Vichislitel)
        N : Integer;
        Procedure Stepen;
        Procedure Koren;
End;
```

Полиморфизм, қайта жүктеу әдістері. Delphi-дегі полиморфизмнің анықтамасы Турбо Паскальдағы анықтамаға ұқсас: полиморфизм - әртүрлі тәсілдермен бірдей мағыналы әрекеттерді орындауға арналған класстардың қасиеті. Класстың «ұрпақтары» әдісінің алгоритмін өзгерту арқылы, оларға «ата-анада» жоқ ерекше қасиеттерді таратамыз.

Ол әдісті өзгерту үшін, «ұрпақты» жауып тастау керек, яғни «ұрпақта» біратаулы әдіс жариялап, оған қажетті әрекеттерді жүзеге асыру керек. Нәтижесінде «ата-ана» және «ұрпақ» объектісі әр түрлі алгоритмдік негізде бірдей атаудың екі әдісіне ие болады, сондықтан олар объектілерге әр түрлі қасиеттер береді.

Delphi-де полиморфизмге тек ата-аналық әдісті мұра етумен ғана емес, сонымен қатар «ата-ана» әдістері «ұрпақтар» әдістеріне қол жеткізуге мүмкіндік беретін виртуализациямен де қол жеткізіледі.

«Ұрпақ» арқылы «ата-ана» әдісін жабатын механизмінде, соңғысы жабылған «ата-ана» әдісін «көрмей қалуы» мүмкін. Сондықтан оны резервтелген Inherited сөзі арқылы шақыра аламыз. Delphi-де резервтелген «Overload» (қалпына келтіру) сөзі енгізілген, оның көмегімен «ата-ана» біратаулы әдісі мен «ұрпақ» сияқты әдістерді көре аламыз.

Объектіге-бағытталған жобалау ұғымы. Объектілі тәсілмен дамыған программалық жасақтама модельдері нақты әлемнің объектілері мен құбылыстарына, сондай-ақ, белгілі бір пәндік облыстың элементтерінің (объектілерінің) жай-күйімен байланысты дамыған программалық жасақтаманың талап етілетін мінез-құлық сипаттамасына негізделеді.

Программалық жасақтама әзірлеуге объектілі көзқарас *объектінің декомпозициясына* негізделген, яғни өңделген программалық қамтамасыз етуді өзара әрекеттесу барысында хабарламалар мен қажетті функцияларды орындау арқылы жүзеге асыратын объектілер жиынтығы түрінде ұсыну.

Дегенмен, объектілі көзқараспен, құрылымдық сияқты, бірден қарапайым программалық жасақтамаға декомпозиция жасай аламыз. Сондықтан, алдымен, әр түрлі модельдер мен белгілерді қолдана отырып, объектіге-бағытталған программалау үшін

программалық жасақтаманы талдау және жобалаудың тиісті әдістері жасалды.

1994 жылы UML тілі (Unified Modeling Language Unified Modeling Language) әзірленді. Қазіргі уақытта ол объектіге-бағытталған тәсілді қолдану арқылы жасалған жобаларды сипаттайтын стандартты құралы ретінде танылады (Бұл тілді толық сипаттау үшін әдебиеттерді қара).

ЖАТТЫҒУЛАР

1. Келесі класстарды әдістерін көрсете отырып сипаттаңдар:
а) «Телефон қоңырауы» — телефон нөмірі, сөйлесу мерзімі, ұзақтығы, қала коды және т.с.с.;
ә) «Жол жүру» — пойыз нөмірі, белгіленген пункті, жүру күндері, келу уақыты, тұру уақыты және т.с.с.;
б) «Кинотеатр» — кинофильм атауы, сеанс, билет бағасы, көрермендер саны, және т.б.

Класстардағы жарияланған әдістерді жүзеге асырыңдар. Көрсетілген класстардағы «ұрпақ»-объектілерді сипаттаңдар. Қосымшаны түзету мен тестілеуді орындаңдар.

2. Класстар жиынын «Тіктөртбұрыш» классымен, сонымен қатар параллелограмм болып табылмайтын (трапеция және т.б.) басқа да дөңес төртбұрыштармен толықтырыңдар және сәйкес әдістермен оларды жүзеге асырыңдар.

3. 4.2-бөлімдегі 6-шы жаттығуды Delphi ортасында орындаңдар.

БАҚЫЛАУ СҰРАҚТАРЫ

1. Объектіге-бағытталған программалау парадигмасының процедуралық программалаудан қандай айырмашылығы бар?
2. Object Pascal-да объект ұғымы қалай анықталады?
3. Инкапсуляция, мұрагерлік, полиморфизм дегеніміз не?
4. Delphi программалау жүйесі қашан және қандай мақсатта пайда болды?
5. Визуалды программалау технологиясы дегеніміз не?
6. Delphi ортасы интерфейсінің негізгі құрамдас бөліктері қандай және олардың мақсаты не?
7. Қосымша интерфейсінің негізгі компоненттері (объектілері) қандай?
8. Оқиғалы-басқарушы программалау дегеніміз не?
9. Әр түрлі графикалық интерфейс объектілері үшін «оқиға» дегеніміз не?
10. Оқиғаға компьютер қалай жауап береді?
11. Delphi-де қосымшаларды өңдеудің қандай кезеңдерін

білесіңдер? Сипаттап беріңдер.

Қосымша

1-қосымша

Турбо Паскаль. CRT модулі

1.1Қ-кесте. Жұмыс режимінің тұрақтылары

Тұрақты атауы	Режим нөмірі	Режим
BW40	0	Ақ-қара, 40 символ, 25 жол
CO40	1	Түрлі-түсті, 40 x 25
BW80	2	Ақ-қара, 80 x 25
CO80	3	Түрлі-түсті, 80 x 25
Mono	7	Монохромды, 80 x 25, монохромды дисплейлер үшін

1.2Қ-кесте. Түстер тұрақтылары

Тұрақты атауы	Түс нөмірі	Түс
Black	0	Қара
Blue	1	Қою көк
Green	2	Қою жасыл
Cyan	3	Көгілдір
Red	4	Қызыл
Magenta	5	Күлгін
Brown	6	Қоңыр
LightGray	7	Ашық сұр
DarkGray	8	Қою сұр
LightBlue	9	Көк
LightGreen	10	Ашық жасыл
LightCyan	11	Ашық көгілдір

Тұрақты атауы	Түс нөмірі	Түс
LightRed	12	Алқызыл
LightMagenta	13	Қызғылт
Yellow	14	Сары
White	15	Ақ
Blink	128	Символдың жылтырауы

1.3Қ-кесте. Режимдерді басқаратын процедуралар мен функциялар

Интерфейс	Қызметі
Режимдер мен терезелерді орнату	
Procedure AssignCrt (File : Text);	Экранға шығаруды тездететін дисплей терезесін мәтіндік файлмен байланыстырады.
Procedure ClrScr;	Экранды тазалап, меңзерді жоғарғы сол жақ бұрышқа орналастырады
Procedure TextMode (Mode : Integer); Mode — мәтіндік режим нөмірі немесе сәйкес тұрақты	Мәтіндік режимді таңдайды.
Procedure Window *(x1,* y1, x2, *y2* : Byte); (x1, y1) и (x2, y2) — жоғарғы сол жақ және төменгі оң жақ координаталар	Мәтіндік режимде шығару терезесін анықтайды
Фонның түсі мен мәтінін басқару	
Procedure HighVideo;	Шығарылатын символдарға жоғарғы жарықтықты орнатады
Procedure LowVideo;	Шығарылатын символдарға төменгі жарықтықты орнатады
Procedure NormVideo;	Берілген графикалық режимге сәйкес үнсіз келісім бойынша символдар мен фон түсін қайтарады
Procedure TextBackground (Color : Byte); Color — код түсі немесе сәйкес тұрақты	Символдар түсін таңдайды

Интерфейс	Қызметі
Мәтінді шығаруды басқару	
Procedure ClrEol;	Меңзердің ағымдағы орнынан жолдың соңына дейінгі символдарды өшіреді.
Procedure DelLine;	Меңзер тұрған орыннан сызықты өшіреді.
Procedure InsLine;	Меңзер тұрған орынның алдына жаңа жол қояды
Пернетақтамен жұмыс	
Function KeyPressed : Boolean; мәні True, егер кез-келген перне басылған болса, және қарсы жағдайда - False	Пернетақтада қандайда бір перне басылғанын анықтайды
Function ReadKey : Char; Функция мәні — пернетақтада басылған перне символының коды	Пернетақтаның буферінен символдарды оқиды
Меңзерді басқару	
Procedure GotoXY (X,Y : Integer); *X, Y* — меңзер координаталары	Меңзерді көрсетілген координатаға апарады
Function WhereX : Integer; функция мәні — меңзердің X координатасы	Меңзердің *X* координатасын қайтарады
Function WhereY : Integer; Функция мәні — меңзердің *Y* координатасы	Меңзердің *Y* координатасын қайтарады
Дыбысты басқару	
Procedure NoSound;	Динамикті қосады
Procedure Sound (Hz : Word); Hz — герцпен берілген дыбыс жиілігі	Берілген жиілікте динамиктің дыбысын қосады
Уақытты басқару	
Procedure Delay (Ms : Word); *Ms* — миллисекундпен берілген кідіріс мәні	Программаның орындалуын берілген миллисекунд санына байланысты кідіртеді

Турбо Паскаль. GRAPH модулі

2.1Қ-кесте. Графикалық құрылғылар драйверлерінің коды

Атауы	Мәні	Бағыты
Detect	0	Драйверді автоматты түрде таңдау
CGA	1	
MCGA	2	
EGA	3	
EGA64	4	
EGAMono	5	
IBM8514	6	
HercMono	7	
ATT400	8	
VGA	9	
PC3270	10	
CurrentDriver	-128	Ағымдағы драйвер

2.2қ-кесте. Графикалық режим тұрақтылары

Атауы	Мәні	Өріс өлшемі	Палитра	Бет саны
ATT400C0	0	320 x 200	C0	1
ATT400C1	1	320 x 200	C1	1
ATT400C2	2	320 x 200	C2	1
ATT400C3	3	320 x 200	C3	1
ATT400Med	4	640 x 200	2 түс	1
ATT400Hi	5	640 x 400	2 түс	1
CGAC0	0	320 x 200	C0	1
CGAC1	1	320 x 200	C1	1
CGAC2	2	320 x 200	C2	1

Атауы	Мәні	Өріс өлшемі	Палитра	Бет саны
CGAC3	3	320 x 200	C3	1
CGACHi	4	640 x 200	2 түс	1
EGALo	0	640 x 200	16 түс	4
EGAHi	1	640 x 350	16 түс	2
EGA64Lo	0	640 x 200	16 түс	1
EGA64Hi	1	640 x 350	4 түс	1
EGAMonoHi	0	640 x 350	2 түс	1 немесе 2
HercMonoHi	0	720 x 348	2 түс	2
IBM8514Lo	0	640 x 480	256 түс	1
IBM8514Hi	0	1024 x 768	256 түс	1
MCGAC0	0	320 x 200	C0	1
MCGAC1	1	320 x 200	C1	1
MCGAC2	2	320 x 200	C2	1
MCGAC3	3	320 x 200	C3	1
MCGAMed	4	640 x 200	2 түс	1
MCGAHi	5	640 x 480	2 түс	1
PC3270Hi	0	720 x 350	2 түс	1
VGALo	0	640 x 200	16 түс	4
VGAMed	1	640 x 350	16 түс	2
VGAHi	2	640 x 480	16 түс	1

Ескерту. С0 палитрасында - ашық жасыл, қызғылт және сары түстер, С1 палитрасында— ашық көк, ашық күлгін және ақ, С2 палитрасында — жасыл, қызыл және қоңыр, С3 палитрасында — көк, күлгін және ашық сұр түстері бар.

2.3қ-кесте. Түстер тұрақтылары

Тұрақты атауы	Түс нөмірі	Түс
Black	0	Қара
Blue	1	Қою көк

Тұрақты атауы	Түс нөмірі	Түс
Green	2	Қою жасыл
Cyan	3	Көгілдір
Red	4	Қызыл
Magenta	5	Күлгін
Brown	6	Қоңыр
LightGray	7	Ашық сұр
DarkGray	8	Қою сұр
LightBlue	9	Көк
LightGreen	10	Ашық жасыл
LightCyan	11	Ашық көгілдір
LightRed	12	Алқызыл
LightMagenta	13	Қызғылт
Yellow	14	Сары
White	15	Ақ

2.4Қ-кесте. Сызықтар кодтары

Атауы	Мәні	Қызметі
Сызықтар типтерінің кодтары(SetLineStyle процедурасы үшін)		
SolidLn	0	Біртұтас
DottedLn	1	Пунктирлі
CenterLn	2	Штрихпунктирлі
DashedLn	3	Штрихтелген
UserBitLn	4	Қолданушы белгілеген
Сызық жуандығының кодтары		
NormWidth	1	Қалыпты
ThickWidth	3	Қалың

2.5Қ-кесте. Толтыру типтерінің тұрақтылары (SetFillStyle процедурасы үшін)

Тұрақты атауы	Мәні	Қызметі
EmptyFill	0	Фон түсімен толтыру
SolidFill	1	Түспен біртұтас толтыру
LineFill	2	— түрде толтыру
LtSlashFill	3	/// түрде толтыру
SlashFill	4	Қалың сызықпен /// түрде толтыру
BkSlashFill	5	Қалың сызықпен \ түрде толтыру
LtBkSlashFill	6	\ түрде толтыру
HatchFill	7	Торкөзбен толтыру
XHatchFill	8	Қисық торкөзбен толтыру
InterleaveFill	9	Жиі торкөзбен толтыру
WideDotFill	10	Сирек торкөзбен толтыру
CloseDotFill	11	Жиі нүктелермен толтыру
UserFill	12	Қолданушы анықтайды

2.6Қ-кесте. Процедуралар мен функциялар

Интерфейс	Қызметі
Графикалық режимді басқару	
Procedure CloseGraph;	Графикалық режимді жабады
Procedure DetectGraph (Var GrDriver, GrMode: Integer); GrDriver —драйвер коды, GrMode — графикалық режимнің коды	Берілген компьютер үшін ұсынылған графикалық драйвер мен режимді анықтайды
Function GetDriverName : String; Функция мәні — қолданылатын драйвер атауы	Пайдаланылатын графикалық драйверімен файл атауын көрсетеді
Function GetGraphMode : Integer; функция мәні — графикалық режим коды	Пайдаланылатын графикалық режимнің кодын анықтайды

Интерфейс	Қызметі
Function GetMaxName : Integer; Функция мәні — қолданылатын графикалық режим атауы	Қолданылатын графикалық режим атауын анықтайды
Function GetMaxMode : Integer; Функция мәні — режим кодының максималды мәні	Қолданылатын драйверге графикалық режим кодының максималды мәнін анықтайды.
Procedure GetModeRange (GrDriver : Integer; Var LoMode, HiMode : Integer); GrDriver — графикалық режим коды; LoMode, HiMode — берілген драйвер үшін графикалық режимнің ең кіші және ең үлкен мәні	Берілген драйвер үшін оны шақыру барысында графикалық режимнің ең кіші және ең үлкен мәнін анықтайды
Procedure GraphDefaults;	Графикалық нұсқағышты координаталар басына орналастырады және графикалық жүйені қайта орнатады
Function GraphErrorMsg (ErrorCode : Integer) : String; ErrorCode — графикалық қатенің коды. Функция мәні — қате туралы мәтіндік хабарлама.	Графикалық режимнің кодының қатесі туралы мәтіндік хабарлама береді
Function GraphResult : Integer; Функция мәні — қате коды	Модульдің процедураларының орындалу барысында қателерді анықтайды
Procedure InitGraph (var GrDriver, GrMode : Integer; PathToDriver : String); GrDriver — драйвер коды, GrMode — графикалық режим коды; PathToDriver — қолданылатын драйвер файлына жол	Берілген графикалық режимді орнатады.
Function InstallUserDriver (Name : String; AutoDetectPtr : Pointer) : Integer; Name — графикалық драйвер атауы, AutoDetectPtr — драйвердің сәтті қосылуын анықтайтын процедураға көрсеткіш. Функция мәні -	Графикалық режимнің қолданушы драйверін орнатады.

Интерфейс	Қызметі
орнатылған драйвердің сандық коды	
Procedure RegisterBGIDriver (Driver : Pointer) : Integer; Driver — драйверге нұсқағыш	Графикалық жүйенің драйверін тіркейді
Procedure RestoreCrtMode;	Графикалық режимді жабады, бұрын орнатылған мәтіндік режимді қалпына келтіреді.
Procedure SetGraphMode(GrMode : Integer);GrMode — графикалық режим коды	Драйверді өзгертпей басқа графикалық режим орнатады

Экран және терезені басқару

Procedure ClearDevice;	Экранды тазалайды, графикалық орнатулардың барлығын үнсіз келісімге ысырады, графикалық нұсқағышты (0, 0) нүктеге әкеледі.
Procedure ClearViewPort;	Экранды тазалайды, SetBkColor бойынша фонды орнатады
Function GetMaxX : Integer; Функция мәні— *X* –тің максималды координатасы Function GetMaxY : Integer; Функция мәні— *Y* –тің максималды координатасы	Экранның берілген графикалық режимдегі максималды координаталарын анықтайды
Procedure GetAspectRatio (Var Xasp, Yasp: Word); Xasp, Yasp — *X* және *Y* осьтері бойынша коэффициенттер	X және Y осі бойынша пикселдер арасындағы сызықтық қашықтықтың біркелкі емес екенін сипаттайтын коэффициенттерді анықтайды
Function GetBkColor : Word; Функция мәні— фон түсінің коды	Фонның орнатылған түсін анықтайды

Экран және терезелер

Procedure GetViewSettings (Var ViewPort : ViewPortType); ViewPort — ағымдағы терезенің параметрлері	Ағымдағы терезенің параметрлерін және қию опцияларын сұрайды

Интерфейс	Қызметі
Procedure SetBkColor (Colorv: Word); Функция мәні —фон түсінің коды	Фон түсін орнатады
Procedure SetActivePage (Page : Word); Page — ағымдағы бет нөмірі	Ағымдағы графикалық бетті орнатады
Procedure SetAspectRatio (Var Xasp, Yasp : Word); Xasp, Yasp — *X* және *Y* осьтері бойынша коэффициенттер	Экранның қабырғалық ара-қатынасының масштабты коэффициентін өзгертеді
Procedure SetVisualPage (Page : Word); Page — бет нөмірі	Көрініп тұрған графикалық беттің нөмірін орнатады
Procedure SetViewPort (x1, y1, *x2, y2* : Integer; Clip : Boolean); (x1, y1) және (x2, y2) — терезенің қарсы бұрыштарының координаталары; Clip — терезеден тыс кеткен суреттерді қиюды анықтайды	Графикалық суретті шығару үшін визуалды порт (терезе) орнатады
Function GetColor : Word; Функция мәні — түс коды	Ағымдағы түсті қайтарады (SetColor орнатқан)
Procedure GetDefaultPalette (Var Palette : PaletteType); Palette — палитра (PaletteType типті жиым)	Үнсіз келісім режимінде қандай палитра орнатылғанын анықтайды
Function GetMaxColor : Word; Функция мәні — максималды түс коды	Орнатылған режимде максималды түс кодын анықтайды
Procedure GetPalette (Var Palette : PaletteType); Palette — палитра (PaletteType типті жиым)	Қандай палитра орнатылғанын анықтайды
Function GetPaletteSize : Word; Функция мәні —палитрадағы түстер саны	Палитра кестесінің өлшемін қайтарады
Procedure SetAUPalette (Var Palette : PaletteType); Palette — палитра (PaletteType типті жиым)	Палитраның барлық түстерін орнатады
Procedure SetColor (Color : Word); Color — түс коды	Сызық үшін түс орнатады

Интерфейс	Қызметі
Procedure SetPalette(ColorNum, Color : Word); ColorNum — ауыстырылатын түс коды; Color — жаңа түстің коды	Палитрада жаңа түс орнатады
Procedure SetRGBPalette (ColorNum, RedValue, GreenValue, BlueValue : Word); ColorNum — палитрадағы орнатылатын түс коды; RedValue, GreenValue, BlueValue — қызыл, жасыл және көк түстердің қанықтылығы	Бояғышты үш құрамдас бөліктермен анықталған түске орнатады: қызыл, жасыл және көк

Нұсқағышты басқару

Function GetX : Integer; GetX —*X* нұсқағыштың координатасы	Ағымдағы нұсқағыштың *X* координатасын қайтарады
Function GetY : Integer; GetY — *Y* нұсқағыштың координатасы	Ағымдағы нұсқағыштың *Y* координатасын қайтарады
Procedure MoveTo (X : Integer; Y : Integer); *(X, Y)* — экран нүктелерінің координаталары	Нұсқағышты *X, Y* координаталары көрсеткен нүктеге ауыстырады
Procedure MoveRel (dx : Integer; dy : Integer); dx, dy — экран нүктелерінің координаталық өсімшесінің векторы	Нұсқағышты ағымдағы координаттардан dx, dy мәндерімен ерекшеленетін нүктеге жылжытады.

Шаблондар; аймақтарды бояу

Procedure FloodFill (X, Y : Integer; ColorBorder : Word); (X, Y) — маңайы боялған нүкте координаталары; ColorBorder — бояу аймағының шекарасын білдіретін түс	Берілген шаблон мен түске байланысты еркін аймақты берілген нүктенің айналасынан белгілі бір түспен белгіленген шекараға дейін бояйды
Procedure GetFillPattern (Var FillPattern : FillPatternType); FillPattern —шаблон туралы ақпаратты қамтитын FillPatternType типті жиым	Шаблонның орнатылған типін анықтайды
Procedure GetFillSettings (Var FillInfo : FillSettingsType);	Шаблонның орнатылған типін анықтайды

Интерфейс	Қызметі
FillInfo — шаблон туралы ақпаратты қамтитын FillSettingsType типті жазба	
Procedure GetLineSettings (Var LineInfo : LineSettingsType); LineInfo — сызық типі туралы ақпаратты қамтитын LineSettingsType типті жазба	Сызықтың орнатылған типін анықтайды
Procedure GetGraphBufSize (Size: Word); Size — буфер өлшемі	Толтыру функциясы үшін буфер өлшемін өзгертеді
Procedure SetFillPattern (Pattern : FillPatternType; Color : Word); Pattern —шаблон туралы ақпаратты қамтитын FillPatternType типті жиым; Color — бояу түсі	Шаблон мен бояу түсін орнатады
Procedure SetFillStyle (Pattern, Color : Word); Pattern — стандартты шаблон коды; Color — бояу түсі	Стандартты шаблондардың біреуін және бояу түсін орнатады
Procedure SetLineStyle (LineStyle, Pattern, Thickness : Word); LineStyle — сызық стилінің коды; Pattern — сызықтың өзіндік шаблоны; Thickness — сызық қалыңдығы	Сызық стилін стандартты немесе өзіндік шаблон ретінде орнатады.

Геометриялық фигураларды бейнелеу

Procedure Arc (X, Y : Integer; StartAngle, EndAngle, R : Word); (*X*, *Y*) — шеңбер центрінің координаталары; StartAngle, EndAngle — шеңбер доғасының градустпен берілген бастапқы және соңғы бұрыштары (0,359); *R* — *X* бағыты бойынша пиксель бойынша шеңбер радиусы	Шеңбер доғасын бейнелейді. Процедура масштабтың осьтер бойынша бірдей еместігін санайды
Procedure Bar (x1, y1, x2, y2 : Integer); *(x1,* y1) и (x2, y2) — тіктөртбұрыштың сол жақ жоғарғы және оң жақ төменгі бұрыштарының координаталары	Контурсыз боялған тіктөртбұрышты бейнелейді

Интерфейс	Қызметі
Procedure Bar3D (x1, y1, x2, y2 : Integer; Depth : Word; Top : boolean); (x1, y1) и *(x2,* y2) — параллелепипедтің сол жақ жоғарғы және оң жақ төменгі бұрыштарының координаталары, Depth — оның тереңдігі; Top — фигураның үстіңгі бетін бейнелеуді білдіретінін анықтайды (true — бейнелеу)	Үшөлшемді тікбұрышты алдыңғы беті боялған параллелепипедті бейнелейді. Үстіңгі беті бейнеленбеуі де мүмкін
Procedure Circle (x, y : Integer; R : Word); (x, y) — шеңбер центрінің координаталары; *R* — пиксельмен берілген *X* бағытындағы шеңбер радиусы	Шеңберді бейнелейді. Процедура масштабтың осьтер бойынша бірдей еместігін санайды
Procedure DrawPoly (NumPoints : Word; Var PolyPoints); NumPoints — сынық сызықты сызатын нүктелер саны; PolyPoints — сынық сызықты беретін нүктелер жиымы (Point типті элементтер)	Берілген нүктелер жиымы арқылы өтетін сынық сызықты бейнелейді
Procedure Ellipse (X, Y : Integer; StartAngle, EndAngle, Rx, Ry : Word); (X, *Y)* — эллипс центрінің координаталары; StartAngle, EndAngle — эллипс доғасының градустпен берілген бастапқы және соңғы бұрыштары (0,359); *Rx, Ry —X* және *Y* бағытындағы пиксельмен берілген эллипстің осьтерінің жартысы	X және Y бағытындағы пиксельмен берілген эллипстің доға осьтерінің жартысын бейнелейді
Procedure GetArcCoords (Var ArcCoords : ArcCoordsType); — доға координаталары бар жазба	Arc және Ellipse процедураларымен бейнеленген доға координаталарын қайтарады.
Procedure FillEllipse (X, Y : Integer; Rx, Ry : Word); (X, Y) — эллипс центрінің координаталары; Rx, *Ry* — X және Y бағытындағы пиксельмен берілген эллипстің осьтерінің жартысы	X және Y бағытындағы осьтерінің жартысы пиксельмен берілген боялған эллипстің сызады
Procedure FillPoly (NumPoints : Word; Var PolyPoints); NumPoints — сынық сызықты беретін нүктелер саны	Төбелер жиымы берілген боялған көпбұрышты бейнелейді.

Интерфейс	Қызметі
PolyPoints — сынық сызықты беретін нүктелер жиымы (Point типті элементтер)	
Function GetPixel (x, y : Integer) : Word; (x, y) — нүкте координаталары, нәтиже — нүкте түсі	Берілген нүктенің түсін қайтарады
Procedure Line (x1, y1, x2, y2 : Integer); *(x1,* y1), *(x2,* y2) — түзу кесіндісінің шеткі координаталары	Түзу кесіндісінің шеткі координаталары арқылы сызады
Procedure LineRel (dx, dy : Integer); (dx, dy) — нұсқағыштың бастапқы орнына қатысты соңғы нүктені ауыстыру	Нұсқағыш орнынан берілген мәнге ауысқан нүктеге дейінгі түзу кесіндісін сызады
Procedure LineTo (x, y : Integer); (x, y) — кесінді орналасатын нүктелер	Нұсқағыш орнынан берілген нүктеге дейінгі түзу кесіндісін сызады
Procedure Rectangle (x1, y1, x2, y2 : Integer); (x1, y1), (x2, y2) — тіктөртбұрыштың қарсы бұрыштарының координаталары	Тіктөртбұрыш контурын сызады
Procedure Sector (X, Y : Integer; StartAngle, EndAngle, Rx, Ry : Word); (X, *Y)* — эллипс центрінің координаталары; StartAngle, EndAngle — эллипс доғасының градуспен берілген бастапқы және соңғы бұрыштары (0,359); *Rx, Ry* — X және Y бағытындағы пиксельмен берілген эллипстің осьтерінің жартысы	StartAngle-ден EndAngle-ге дейінгі бұрыштармен шектелген эллипстің боялған секторын көрсетеді. Сектор бұрыштардың ең минималды мәндерінен ең жоғары мәндеріне дейін толтырылады
Procedure PieSlice (X, Y : Integer; StartAngle, EndAngle, Rx : word); (X, Y) — эллипс центрінің координаталары; StartAngle, EndAngle — эллипс доғасының градуспен берілген бастапқы және соңғы бұрыштары (0, 359); *Rx* — *X* бағытындағы пиксельмен берілген шеңбер радиусы	Дөңгелектің боялған секторын StartAngle-ден EndAngle-ге дейінгі бұрыштармен шектелген, X бағытындағы пикселдермен көрсетілген шеңбердің радиусын бейнелейді және ось бойымен бейне масштабын ескереді. Сектор бұрыштардың минималды мәндерінен, максимумға дейін

Интерфейс	Қызметі
Procedure PutPixel (X, Y : Integer; Color : word); (X, *Y*) — нүкте координаталары; Color — нүкте түсі	Экран нүктесін берілген түске бояйды

Мәтінді шығару

Procedure GetTextSettings (Var TextInfo : TextSettingsType); TextInfo — мәтін шығаруды орнататын жазба	Мәтінді шығарылуының орнатылуын анықтайды
Procedure OutText (Text : String); Text — мәтіннің шығарылатын жолы	Мәтінді графикалық экранда графикалық нұсқағыштың орнынан бастап, шығарылатын мәтіннің ені бойынша жылжытады.
Procedure OutTextXY (X, Y : Integer; Text : String); (X, Y) — шығарылатын мәтін байланысатын нүкте координаталары; Text — мәтіннің шығарылатын жолы	Көрсетілген нүктеге қатысты графикалық экранда мәтінді пайдаланылатын теңестірудің типін ескере отырып көрсетеді
Function InstallUserFont (FontFileName : String): Integer; FontFileName — қарпі бар файл атауы. Нәтиже — орнатылған қаріп нөмірі (коды)	Жаңа қаріп орнатады
Procedure RegisterBGIFont (Font : Pointer); Font — қаріпке нұсқағыш	Графикалық жүйе үшін қаріп орнатады
Procedure SetTextJustify (Horiz, Vert : Word); Horiz, Vert — горизонталь және вертикаль бағытты мәтінді байланыстыру	Горизонталь және вертикаль бағытты нүктеге мәтінді байланыстыру типін орнатады
Procedure SetTextStyle (Font, Direction, CharSize : Word); Font — қолданылатын қаріп типі; Direction — мәтінді шығару бағыты; CharSize —символдар өлшемі	Шығарылатын мәтіннің стилін орнатады

Интерфейс	Қызметі
Procedure SetUserCharSize (MultX, DivX, MultY, DivY : Word); MultX, DivX, MultY, DivY — *X* және *Y* осьтері бойынша көбейту (Mult) және бөлу (Div) коэффициенттері	Еркін бөлшекті масштабты қаріптерді масштабтауды орындайды
Function TextHeight (Text : String) : Integer; Text — мәтіннің шығарылатын жолы; Нәтиже — мәтіннің пиксель бойынша биіктігі	Орнатылған стиль бойынша мәтіндік жолдың биіктігін анықтайды
Function TextWidth (Text : String) : Integer; Text — мәтіннің шығарылатын жолы; Нәтиже — пиксель бойынша мәтін ұзындығы	Орнатылған стиль бойынша мәтіндік жолдың ұзындығын анықтайды

Экран бөліктерін көшірмелеу

Procedure GetImage (x1, y1, x2, y2 : Integer; Var BitMap: Pointer); (*x*1, *y*1) и (*x*2, *y*2) — көшірмеленетін экран аймағын шектейтін тіктөртбұрыштың қарсы бұрыштарының координаталары; BitMap — берілген суретті сақтауға арналған жады аймағына нұсқағышы	Экранның тіктөртбұрышты аймағын ЖЕСҚ-на көшірмелейді
Function ImageSize (x1, y1, x2, y2 : Integer) : Word; (*x*1, *y*1) и (*x*2, *y*2) — экран аймағын шектейтін тіктөртбұрыштың қарсы бұрыштарының координаталары; Нәтиже — байтпен берілген ақпарат көлемі	Экранның тіктөртбұрышты аймағын сақтау үшін жадының өлшемін байтпен анықтайды
Procedure PutImage (x, y : Integer; Var BitMap : Pointer; BitBlt : Word); (x, y) — шығатын бейне үшін сол жақ жоғарғы бұрыштың координаталары; BitMap — жады аймағының нұсқағышы; BitBlt — логикалық амалдың коды	Экранның көрсетілген аумағына компьютердің жад аймағынан кескінді шығарып, ескі суреттегі жаңа кескінді үстіңгі жағына қою

3-қосымша

3.1Қ-кесте. Мерзім және уақытпен жұмыс істеуге арналған ішпрограммалар

Интерфейс	Қызметі
Function Date : TDateTime;	Ағымдағы уақытты қайтарады
Function DateToStr (D : TDateTime) : String;	Мерзімді символдар жолына түрлендіреді
Function DateTimeToStr (D : TDateTime) : String;	Мерзім мен уақытты символдар жолына түрлендіреді
Function FormatDateTime (Format : String; Value : TDateTime) : String;	Value параметрінен мерзім мен уақытты Format параметріне сәйкес символдар жолына түрлендіреді
Function Now : TDateTime;	Ағымдағы күн мен уақытты қайтарады
Function Time : TDateTime;	Ағымдағы уақытты қайтарады
Function TimeToStr (T : TDateTime) : String;	Уақытты жолға түрлендіреді

3.2Қ-кесте. Жолдармен жұмыс істеуге арналған ішпрограммалар

Интерфейс	Қызметі
Function AnsiLowerCase (const S : String) : String;	Ұлттық Windows кодтауға сәйкес (мысалы, кириллица әріптерін ескере отырып) бас әріптер кіші әріптерге ауыстырылған бастапқы S жолын қайтарады
Function AnsiUpperCase (const S : String) : String;	Ұлттық Windows кодтауға сәйкес (мысалы, кириллица әріптерін ескере отырып) кіші әріптер бас әріптерге ауыстырылған бастапқы S жолын қайтарады
Function Concat (S1 [, S2,..., SN] : String) : String;	S1, S2, ..., SN параметр жолдарының байланыстарын білдіретін жолды береді

Продолжение табл. П3.2

Интерфейс	Қызметі
Function Copy (S : String; N, K : Integer) : String;	*N* нөмірлі символдан бастап *S* жолынан *K* символдарын көшіреді
Procedure Delete (Var S : String; N, K : Integer);	S жолынан нөмірлі символдан бастап K символдарын өшіреді
Procedure Insert (SubSt : String; Var S : String; N : Integer);	*N* нөмірлі символдан бастап *S* жолына SubSt ішкі жолын кірістіреді
Function Length (St : String) : Integer;	*St* жолының ағымдағы ұзындығын қайтарады
Function LowerCase (const S : String) : String;	Латынның барлық бас әріптері кіші әріптеріне алмасатын S бастапқы жолын қайтарады,
Function Pos (SubSt, S : String) : Integer;	S жолында бірінші SubSt ішкі жолдың пайда болуын іздейді және ол басталатын позиция санын қайтарады. Егер ешқандай ішкі жол табылмаса, 0
Procedure SetLength (S : String; NewLength : Integer);	S жолының жаңа (ең кіші) ұзындығын NewLength орнатады. Егер NewLength жолдың ағымдағы ұзындығынан үлкен болса, SetLength шақыруы еленбейді
Function StringOfChar (Ch : Char; K : Integer) : String;	Ch символының *K* рет қайталануынан тұратын жолды жасайды
Function StringToOleStr (const Source : String) : PWideChar;	Қарапайым жолды екі байттық жолға көшіреді
Function StringToWideChar (const Source : String; Dest : PWideChar; DestSize : Integer) : PWideChar;	Қарапайым жолды Unicode символды жолға түрлендіреді
Function Uppercase (const S : String) : String;	Латынның барлық кіші әріптері бас әріптеріне алмасатын S бастапқы жолын қайтарады

Басқа типтерге жолдарды түрлендіру ішпрограммалары

Function StrToCurr (S : String) : Currency;	*S* жолының символдарын бүтін типті Currency сандарына түрлендіреді. Сонымен қатар, жолда бос орын болмау керек

Продолжение табл. П3.2

Интерфейс	Қызметі
Function StrToDate (S : String) : TDateTime;	*S* жолының символдарын мерзімге түрлендіреді. Бұл жағдайда жолда Windows үшін дұрыс ажыратқышпен бөлінген екі немесе үш сан болуы керек. Алғашқы сан - дұрыс күн, екіншісі - дұрыс ай. Егер үшінші нөмір көрсетілсе, ол XX немесе XXXX пішімінде бір жылды көрсетуі керек. Жыл символдары болмаса, күн ағымдағы жылмен толықтырылады
Function StrToDateTime (S : String) : TDateTime;	S жолының символдарын күн мен уақытқа түрлендіреді. Жолда бос орынмен бөлінген дұрыс күн мен уақыт болуы керек
Function StrToFloat (S : String) : Extended;	*S* жолының символдарын нақты сандарға түрлендіреді
Function StrToInt (S : String) : Integer;	*S* жолының символдарын бүтін сандарға түрлендіреді
Function StrToIntDef (S : String; D : Integer) : Integer;	*S* жолының символдарын бүтін сандарға түрлендіреді. Егер жолда бүтін сан дұрыс сипатталмаса, *D* мәні қайтарылады
Function StrToIntRange (S : String; Min, Max : LongInt) : LongInt;	*S* жолының символдарын бүтін сандарға түрлендіреді және егер сан берілген Min..Max диапазоннан шығып кетсе ERangeError болмауын қадағалайды
Function StrToTime (S : String) : TDateTime;	S жолының символдарын күн мен уақытқа түрлендіреді. Бұл жағдайда жолда Windows үшін дұрыс ажыратқышпен бөлінген екі немесе үш сан болуы керек. Сандарда сағат, минут және секунд көрсетіледі. Соңғы саннан кейін бос орын арқылы 12-сағаттық форматты көрсететін «am» немесе «pm» символдары жазылады.

Продолжение табл. П3.2

Интерфейс	Қызметі
Procedure Val (S : String; Var X; C : Integer);	*S* символды жолды осы айнымалының типімен анықталатын *X* айнымалысының бүтін немесе нақты ішкі көрінісіне түрлендіреді. Егер түрлендіру сәтті өтсе, онда C параметрі нөлді қамтиды да, *X* –ке түрлендіру нәтижесі орналасады. Қарсы жағдайда параметр қате символ табылған S жолындағы позиция нөмірін қамтиды. Бұл жағдайда *X* мазмұны өзгеріссіз қалады. *S* жолында жетекші бос орындар болуы мүмкін

Жолдық типке арналған ішпрограммалар

Function DateTimeToStr (Value : TDateTime) : String;	Value параметрінен күн мен уақытты жолдық символдарға түрлендіреді
Procedure DateTimeToString (Var S : String; Format : String; Value : TDataTime);	Value параметрінен күн мен уақытты Format параметріне сәйкес *S* жолдық символдарға түрлендіреді
Function DateToStr (Value : TDateTime) : String;	Value параметрінен мерзімді символдық жолға түрлендіреді
Function FloatToStr (Value : Extended) : String;	Value нақты мәнін символдық жолға түрлендіреді
Function FloatToStrF (Value : Extended; Format : TFloatFormat; Precision, Digits : Integer) : String;	Format, Precision және Digits параметрлеріне сәйкес Value нақты мәнін символдық жолға түрлендіреді
Function Format (const Format : String; const Args : array of const) : String;	Format параметріне сәйкес ашық Args жиымының аргументтерінің еркін санын жолға түрлендіреді
Function FormatDateTime (Format : String; Value : TDateTime) : String;	Format параметріне сәйкес Value параметрінен күн мен уақытты символдық жолға түрлендіреді

Интерфейс	Қызметі
Function FormatFloat (Format : String; Value : Extended) : String;	Format параметріне сәйкес Value нақты мәнін символдық жолға түрлендіреді
Function IntToHex (Value : Integer; Digits : Integer) : String;	Value бүтін мәнін оналтылық форматтағы символға түрлендіреді: Digits — жолдағы
Function IntToStr (Value : Integer) : String;	Value бүтін мәнін символдық жолға түрлендіреді
Procedure Str (X [Width [:Decimals]]; Var S : String);	*X* санын S символдық жолындағы кез-келген нақты немесе бүтін типке түрлендіреді; Width және Decimals параметрлері бар болса, түрлендіру форматын белгілейді: Width X санынан бөлінген сәйкес символдық, ал Decimals — бөлшек бөліктегі символдар саны(бұл параметр егер *X* — нақты сан болса ғана жүзеге асады)
Function TimeToStr (Value : TDateTime) : String;	Value параметрінен уақытты символдық жолға түрлендіреді

Интерфейс	Қызметі
Procedure AssignFile (Var F; FileName : String);	*F* файлдық айнымалыны FileName файл атауымен байланыстырады
Procedure CloseFile (Var F);	Файлды жабады, *F* файлдық айнымалы мен AssignFile процедурасы орнатқан файл атауы арасындағы байланыс сақталады.
Function EOF (Var F) : Boolean;	Файл соңын тестілейді. Егер файлдық нұсқағыш файлдың соңында тұрса, онда True мәнін

Продолжение табл. П3.2

Интерфейс	Қызметі
	Бұл жазу барысында кезектегі компонент файл соңына тіркеледі, ал оқу барысында файл аяқталғанын білдіреді
Procedure Erase (Var F);	F файлын жояды, процедураны орындаудан бұрын оны міндетті түрде жабу қажет. Erase процедурасының орнына DeleteFile-ды қолданған ыңғайлы, себебі ол файл атауы мен файлдық айнымалының алдын-ала байланысын талап етпейді
Function FileExists (const FileName : String) : Boolean;	Егер FileName атаулы файл бар болса (рұқсат етілген маршруты бар болса), True мәнін қайтарады
Function FindFirst (const Path : String; Attr : Integer; Var F : TSearchRec) : Integer;	Көрсетілген каталогта тіркелген алғашқы файл атрибуттарын қайтарады: Path —іздеу маршруты және файлдарды таңдау маскасы; Attr — таңдалатын файлдар атрибуттары; *F* —бірінші таңдалған файл атауы қайтарылатын TSesrchRec типті айнымалы.
Procedure FindClose (Var F : TSearchRec);	FindFirst және FindNext функцияларымен файлдарды іздеуге бөлінген жадыны босатады
Function FindNext (Var F : TSearchRec) : Integer;	*F* айнымалысына каталогтағы келесі файл атауын қайтарады, *F* айнымалысы алдын ала FindFirst процедурасының шақырылуында инициализациялануы қажет. Сәтті іздеу нәтижесі 0-ді қайтарады
Procedure Flush (Var F);	Файлдың ішкі буферін тазалайды, бұл уақытта дисктегі файлдағы соңғы өзгертулер сақталады.

Интерфейс	Қызметі
Procedure GetDir (D : Byte; Var S : String);	Ағымдағы каталог атауын қайтарады (үнсіз келісім бойынша): *D* — құрылғы нөмірі (0 — үнсіз келісім құрылғысы, 1 — A дискісі, 2 — *B* дискісі және т.б.); *S* — дисктегі ағымдағы каталогты көрсету жолы қайтарылатын String типті айнымалы.
Procedure MkDir (Dir : String);	Көрсетілген дискте жаңа каталог құрады: Dir — каталогты іздеу маршруты. Маршруттағы соңғы атау, яғни бұрынғы каталог атауымен бірдей болмау керек
Procedure Rename (Var F; S : String);	*F* файлының атауын өзгертеді *(S* — жаңа файл атауы бар жолдық өрнек), процедураны орындамас бұрын, файлды жабу қажет
Procedure Reset (Var F : File; RecSize : Word]);	Бар файлды ашады (RecSize тек типтелмеген файлдарға ғана әсер етеді) және берілгендер блогының өлшемін анықтайды
Procedure Rewrite (Var F : File [; Recsize: Word]);	Жаңа файл құрады (RecSize тек типтелмеген файлдарға ғана әсер етеді) және берілгендер блогының өлшемін анықтайды
Procedure RmDir (Dir : String);	Dir каталогын жояды, жоятын каталог бос болу керек, яғни онда файлдар немесе төменгі деңгей каталогтары болмау керек.

1. *Абрамов В. Г.* Введение в язык Паскаль / В.Г. Абрамов, Н. П. Трифонов, Г. Н. Трифонова. — М. : Наука, 1988.

2. *Ван Тассел* Д. Стиль, разработка, эффективность, отладка и испытание программ / Д. Ван Тассел. — М. : Мир, 1981.

3. *Вирт Н.* Алгоритмы и структуры данных / Н. Вирт. — М. : Мир, 1989.

4. *Гладков В. П.* Задачи по информатике на вступительном экзамене в ВУЗ и их решения / В. П. Гладков. — Пермь : Перм. техн. ун-т Баспасы, 1994.

5. *Грогоно П.* Программирование на языке Паскаль / П. Грогоно. — М. : Мир, 1982.

6. *Дагене В. А.* 100 задач по программированию / В.А. Дагене, Г. К. Григас, К.Ф.Аугутис. — М. : Просвещение, 1993.

7. Delphi: книга рецептов. Практические примеры, трюки и секреты / пер. с чеш. ; под ред. М. В. Финикова, О. И. Березкиной. — (Серия «Просто о сложном»). — СПб. : Наука и техника, 2006.

8. *Епашников А.М.* Программирование в среде Турбо Паскаль 7.0 / А. М.Епашников, В. А.Епашников. — М. : МИФИ, 1994.

9. Задачи по программированию / [С. А. Абрамов, Г. Г. Гнездилова, Е. Н. Капустина және т.б.]. — М. : Наука, 1988.

10. *Зубов В.С.* Программирование на языке Turbo Pascal (версии 6.0 и 7.0) / В.С.Зубов. — М. : Информационно-издательский дом «Филинъ», 1997.

11. *Иванова* Г *С.* Объектно-ориентированное программирование : учебник для вузов / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев ; под ред. Г. С. Ивановой. — М. : Н.Э. Бауман ат. ММТУ. баспасы, 2001.

12. Информатика. Задачник-практикум : в 2 т. / под ред. И. Семакина, Е.Хеннера. — М. : Лаборатория Базовых Знаний, 1999.

13. *Истомин Е. П.* Программирование на алгоритмических языках высокого уровня / Е. П. Истомин, С. Ю. Неклюдов. — СПб. : Михайлова В. А. баспасы, 2003.

14. *Йенсен К.* Паскаль — руководство для пользователей и описание языка / К.Йенсен, Н. Вирт. — М. : Мир, 1982.

15. *Касаткин В.Н.* Информация. Алгоритмы. ЭВМ / В.Н. Касаткин. — М. : Просвещение, 1991.

16. *Культин Н. Б.* Delphi в задачах и примерах / Н. Б. Культин. — СПб. : БХВ-Петербург, 2005.

17. *Ляхович В. Ф.* Руководство к решению задач по основам информатики и вычислительной техники / В. Ф.Ляхович. — М. : Жоғ. мек.., 1994.

18. *Марченко А.И.* Программирование в среде Turbo Pascal 7.0 / А. И. Марченко, Л. А.Марченко ; под ред. В. П.Тарасенко. — Киев : ВЕК+, М. : Бином

Универсал, 1998.

19. *Миков А. И.* Информатика. Введение в компьютерные науки / А. И.Миков. — Пермь : Изд-во ПГУ, 1998.

20. *Пильщиков В.Н.* Сборник упражнений по языку Паскаль / В. Н. Пильщиков. — М. : Наука, 1989.

21. *Попов Б. В.* TURBO PASCAL для школьников. Версия 7.0 / Б. В.Попов. — М. : Финансы и статистика, 1996.

22. *Фаронов В. В.* Delphi. Программирование на языке высокого уровня : учебник для вузов / В. В. Фаронов. — СПб. : Питер, 2003.

23. *Хонсбергер Р.* Математические изюминки / Р. Хонсбергер. — М. : Наука, 1992.

24. *Шень А.* Программирование: теоремы и задачи / А. Шень. — М. : МЦНМО, 1995.

25. *Шупруга В.В.* Delphi 2006 на примерах / В.В.Шупруга. — СПб. : БХВ-Петербург, 2006.

Мазмұны

Оқу басылымы

Семакин Игорь Геннадьевич,
Шестаков Александр Петрович

Алгоритмдеу және программалау негіздері

Оқулық

Редактор *Толочкова Л. В., Камшыгер С.*
Техникалық редактор *Коржуева Е. Ф.*
Компьютерлік енгізу: *Волкова Р. Ю.*
Корректор *Ермакова И.А.*

№ 103116273 бас. Мөрге қол қойылған 25.02.2016. Формат 60x90/16.
Гарнитура «Балтика». № 1 офсет қағазы. Офсет баспасы. 19,0 шарт. мөр. л.
Тираж 1 000 дана. Тапсырыс №

ООО «Академия» баспа орталығы. www.academia-moscow.ru 129085, Мәскеу, Мир даңғ., 101В, құр. 1.
Тел./факс: (495) 648-0507, 616-00-29.
Санитарлы-эпидемиологиялық сараптама № РОСС RU. АЕ51. Н 16679 25.05.2015.

Идел-Прессте басылып шығарылды.