БАҒДАРЛАМАЛАУ ТІЛДЕРІНЕ КІРІСПЕ

Арвинд Кумар Бансал

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

CRC Press баспасы
Taylor & Francis баспа тобы
6000 БрокенСаундПарквей NW, Сьют 300 БокаРэйтон FL 33487-2742

Қышқылсыз қағазға басып шығарылған 20140520 басылым
ISBN 13 978-1-4665-6514-2 (жұмсақ мұқаба)

http://www.taylorandfrancis.com және http://www.crcpress.com
сілтемелері арқылы Taylor&Francisжәне CRC Press баспаларының сайтына шолу жасаңыз

1. Кіріспе

Компьютерлер туралы ғылым өзінің пайда болған уақытынан бастап, яғни 1950 жылдардан бері аса қызығушылыққа ие.Өзінің пайда болған шағында тек күрделі ғылыми есептеу жұмыстарын жүргізуге арналып жасалған бұл ғылым қазіргі таңда өміріміздің барлық дерлік салаларында, соның ішінде медицинада, ғарышты меңгеруде, телекоммуникацияда, ақпарат алмасу мен алшақ қарым-қатынас орнатуда, үлгілеуде, автоматтандырылған үлгілеу мен навигацияда, автоматтандырылған өндірісте, операцияларда, жобалауда, өндірісті жоғарылату құралы ретінде, электронды мәміле жасау мен сатуда, көлік пен электростанцияларды басқаруда кеңінен қолданылады. Бүгінгі таңда біз өмірімізді қондырылатын немесе жеке компьютерсіз көзімізге елестете алмаймыз. Компьютерлер автокөліктер, ұялы телефондар, ұшақтар, ғарыш аппараттары, жоғары сапалы кір жуу және кептіру құрылғылары, пештер мен үй қауіпсіздігі жүйелері секілді заманға сай құралдардың ішіне орнатылып келеді. Қазіргі кезде Ақылды Үйлер салынып жатыр. Бұл үйлерде көп мәселелік режимде ақпаратты өңдеу мен үйдегі күнделікті міндеттерді орындауға арналған компьютерлер орнатылмақ.

Барлық аталған көмпьютерлік іс-әрекеттер бағдарламашылардың терең білімдерінің арқасында орындалып жатыр. Бағдарламашылар көптеген кешенді мәселелерге кезігіп, жоғарғы деңгейдегі нұсқауларға жүгіну арқылы оларды шешудің жолдарын ұсынады. Бұл нұсқаулар автоматтандырылған аудармашылар арқылы төмен деңгейдегі машиналық нұсқауларға аударылады. Осындай төмен деңгейдегі нұсқаулар компьютерлік бағдарламаларды іске қосады. Механизмдер деңгейіндегі нұсқаулар өздерінің мәнерлік күшінде шектелген және талаптарды өзгертумен қатар жоғарғы деңгейдегі шешімдерді үлгілеу мен түрлендіруге өз үлестерін қоспайды. *Күрделі мәселелерді шешу мен талаптардың өзгеруіне қарай осы шешімдерді өзгерту үшін шешімдерді жасау мақсатында жоғарғы деңгейдегі тілдер мен тілдік құрылымдарды әзірлеу қажет-*

тілігі туындайтыны анық. Бұл талаптар техникалық үдеріс әсерінен өзгерген қоғамды қайта ұйымдастыру нәтижесінде пайда бола бастайды. Бұл өз кезегінде бағдарламалық қамтамасыз етудің бір бөлігінің өзгеруі екіншісіне кері әсерін тигізбейтіндей етіп, бағдарламаны қосымша жетілдіру мен ретке келтіруді қажет етеді. Кері әсерлердің санын шектеу үшін бағдарламалық қамтамасыз ету модульді болуы тиіс: тоқтаусыз функционалдыққа ие және нақты анықталған түрлі модульдер. Бұл модульдер ішкі деректер мен операциялармен шектеулі алмасумен әлсіз түрде байланысты.

Қазіргі заманғы мәселелер жүйелі болып табылады және олардың шешімін табу үшін мыңдаған нұсқаулар тізбектері қажет. Осындай шешімдерді әзірлеу оңай тапсырма емес және көптеген адами жылдарды қамтиды: жылдар уақытына көбейтілген бағдарламашылар саны.

Бұндай күш салулар ұйымдастырушылық және қаржы ресурстарының ірі ауқымды міндеттемелерінің бар екенін білдіреді. Бұл күш салулар қайталана алмайды және эволюциялық түрлендіру мен бағдарламалық қамтамасыз етумен толықтырылуы тиіс. Ондай болмаған жағдайда бағдарламалық қамтамасыз етуді әзірлеу кідірісі мен жетілдірудің ақшалай құны үлкен болмақ.

1.1 Пәндер салаларының жиынтығы

Қазіргі заман дәуірінде компьютерлер көмегімен шешілетін мәселелер ғылыми есептеулер, мәтінді өңдеу, деректер қорын бағдарламалау, бизнес-қосымшалары, жүйелік бағдарламалау, технологиялық үдерістерді автоматтандыру, зияткерлік жүйелер, веб-қосымшалар және нақты уақыттағы деректерді өңдеу секілді пән салалары арқылы жүзеге асырылып жатыр. Аталған салалар бір бірінен түрлі талаптар бойынша ерекшеленеді.

Ғылыми есептеулердің мысалы ретінде миллардтаған жұлдыздардан тұратын бүкіл әлем туралы талқылауларды алуға болады. Олардың өзара қарым-қатынасы мен біздің күн жүйесімен байланысы туралы қорытынды жасау үшін біз триллиондаған байт ақпаратты қабылдайтын компьютерлендірілген телескоптардан алынған деректерді өңдеп, сараптауға тиіспіз. Ғылыми есептеулердің тағы бір мысалы жағалау бойындағы аймақтардаы қирататын торнадо секілді атмосфералық құбылыстарды үлгілеу және бақылау болып табылады. Ғылыми есептеулердің келесі мысалы – жер бетіндегі сейсмикалық белсенділікті өңдеу, сараптау және бақылау. Ғылыми мәселелерді шешу жолында ғылыми деректерді өңдеу мен пайдалы үлгілерді әзірлеу компьютерде үлкен матрицаларды ерекшелеп, өңдеуді қажет етеді. Сандық мәндерді дәлме-дәл өңдеуден

өткізген жөн. Мысалы, ғарыш кемесінің траекториясын есептеп шығару дәл есептеулер мен нақтылықты талап етеді.

Мәтінді өңдеу мысалдары болып біз күнделікті өмірімізде мәтіндер мен тұсаукесерлер дайындауда қолданылатын мәтіндік процессорлар мен аса жоғары өнімділік құралдары табылады. Бұндай мәселелер үлкен көлемдегі тармақтарды, суреттерді, кестелерді, бейнежазбаларды және тағы басқа мультимедиялық объектілерді ұсынып, өңдеуге қажетті тиімді мүмкіндіктерді талап етеді. Олар да кең көлемдегі есептеулерді қажет етеді, алайда есептеу қажеттілігі ғылыми есептеулер саласында секілді аса күрделі емес. Бұл үдеріс адамдарен тығыз байланысты болғандықтан, пайдаланушыға ыңғайлы болуы шарт.

Деректер қорын бағдарламалау логикалық тізбектегі кең көлемдегі деректерді, олар тез арада қол жетімді болу үшін, ұйымдастыруды, өңдеуді және іздестіруді қажет етеді. Нақты уақыт режиміндегі деректер қорын бағдарламалаудың мысаладры: 1) тіркеушінің кеңсесінде студенттің деректерін өңдеу және 2) жанармай құятын орында несиелік картаның оқылуы. Студент тіркеуші кеңсесіне келген бетте өз студенттік билетін ұсынады, ал есепші бұл студенттің барлық төлемдеру, адресы және оқитын пәндері жайлы толық ақпаратты көруі тиіс.

Бизнес-қосымшалар тұтынушылар мен жоғары санаттағы басқарушыларға ыңғайлы түрде ұсынылатын есептердің кең жиынтығын талап етеді. Бұдан басқа бизнес-қосымшалар пайдаланушыға ыңғайлы есеп жүйесі бар деректер қорларын біріктіруді қажет етеді. Алайда бизнес-қосымшаларды өңдеу үдерісі *пакеттік режимде* жүзеге асырылуы мүмкін; бұл жағдайда әр қашан нақты уақытта өңдеуді жүргізу қажет етілмейді.

Жүйелік бағдарламалау бір уақытта жүзеге асырылатын бірнеше үдерістерді өңдеуді талап етеді. Бұл бағдарламашылардың жұмыс өндірісін арттырып, тапсырманы орындау тиімділігін арттыру мақсатында төмен деңгейдегі бағдарламалаумен қарым-қатынасты қамтамасыз етеді

Жүйелік бағдарламалау үдерісте жаңылыс орын алған кезде компьютер жадының, қателерді өңдеудің, ескертулер жасаудың сатылы жұмысын, сондай-ақ пайдаланушы деңгейіндегі шақырулардың жоғары деңгейдегі шақырулармен байланысын талап етеді.

Нақты уақыт режиміндегі ақпаратты өңдеу деректердің осы шақта жиналып, өңделуін қажет етеді. Есептеуге байланысты қандай да жұмыс деректерді жинау мен өңдеу үдерісін кідіртпеу қажет, сәл кідіріс орын алған жағдайда нақты уақытта болып жатқан іс-әрекеттер жоғалып, кері нәтижелерге әкелуі мүмкін. Мысалы, компьютер АЭС-мен баяу байланыс орнатып, жүйені қыздырып жіберсе, онда АЭС өзіне зақым келуі

мүмкін. Егер жойғыш ұшақтың борттық компьютері баяу жұмыс істесе, бұл ұшақ жау ракеталары арқылы соққыға ұшырауы мүмкін. Бұл мысалдар оқиғаларды жинау мен өңдеу үдерісін жеңілдету мақсатында нақты уақытта тапсырманы орындау мен тез шешім қабылдаудың аса қажет екенін дәлелдейді.

Соңғы жылдары зияткерлік жүйелер өндіріс роботтарының жұмысы, цехтарды жобалау, әуежай қызметін жоспарлау, табиғи тілді немесе, тіпті, гроссмейстерлерді жеңіп, бейнелер анализін жасайтын, шахмат секілді ойындарді өңдеу тәрізді адам өмірінің көптеген салаларында кең қолданысқа ие. Бұл жүйелер өте үлкен кеңістік ішінде көптеген мәселелермен жұмыс жасап, мәселені логикалық түрде сараптау мен оны шешудің дұрыс жолдарын ұсынуға мүмкіндікке ие болулары тиіс. Әдетте, осындай мәселелерді шешуде эвристикалық бағдарламалау кең қолданылады, яғни мәселені тез арада шешуге бағытталған математикалық үлгілеуге негізделеген дыбыстық жорамал. Нақты уақыт феноменінің қылығындағы белгісіздік пен ақпараттың жеткіліксіз болуынан мәселе одан ары қиындай түседі.

Соңғы жылдары веб-бағдарламалау мультимедиялық жүйелермен бірігіп, бағдарламалау тілдерінде бірқатар талаптардың пайда болуына себепші болады. Бағдарламаларды қашықтықтағы сайттардан іздеп табуға (URLs) және жергілікті машина көмегімен орындауға мүмкін болуы тиіс. Ақпаратты қашықтықтағы машиналардан алу деректер құрылымдарын тармақтарға аударуды немесе керісінше үдерісті талап етсе, тиімді орындау төмен деңгейдегі нұсқауларға тиімді аударуды қажет етеді. Өкінішке орай аудару мен веб-бағдарламаны орындау бір уақытта жүзеге асырылады, бұл бағдарламаның орындалуын баяулатады және өңдеуді жеделдету мақсатында «бірден» компиляциялауды (JIT компиляция) қажет етеді. Алдағы уақытта біз түрлі домендерді өңдеуге арналған пән бойынша бағытталған тілдермен танысатын боламыз.

1.2 Мотивация

Пән саласындағы талаптарды ескере отырып, бағдарламалар үдерісті автоматтандыру үшін әзірленуі мүмкін. Осыған қарамастан, компьютерлерге арналған шешімдер байланысы күрделі, себебі адамдардың ниетін түсіну үшін компьютердің зияты жеткіліксіз, сондай-ақ компьютер адамдардың әдейі жасамаған қателіктері мен біздің хабарламаларымыдың астарында жасырын түрдегі қателіктер салдарын түзете алмайды. Барлық ақпарат компьютерлерге қол жетімді түрде әзірленіп, жүктелуі тиіс. Субъектілер абстрактылы және нақты түрде әзірленіп, шешім сатылы түрде, еш екі мәнділіксіз жасалуы тиіс.

Бағдарламалау тілі компьютермен қарым-қатынасқа түсудің ұйымдасқан тәсілі болып саналады, осылайша компьютер тапсырманы бағдарламашы арқылы берілген нақты нұсқауларға сәйкес орындайды. Нұсқауларда мәтіндік, визуалды, белгілер, ым-ишарат, аудио немесе шешімді табуға арналған тағы басқа тәсілдер секілді кез келген бұқаралық ақпарат көздері қолданылуы мүмкін. Алайда төменде аталған маңызды критерийлер міндетті түде қоладнылады:

1.Мәселенің шешімі қарапайым әрі толық түрде көрсетілуі тиіс.

2.Шешім спецификациясының даму потенциалы болғаны жөн.

3.Бағдарламашының ниеті мен компьютер орындайтын іс-әрекеттің арасында жеке бір мағыналы аударма болғаны жөн. Шешімдер жоғарғы деңгейде анықталғандықтан, шешімді көрсетуге арналған бір мағыналы құрылымдардың жеткіліксіз болуының себебінен бағдарламашы мен компьютер арасында түсініспеушіліктің пайда болуы бір талай.

Автоматтандыру деңгейі дамыған сайын қоғам автоматтандыру деңгейін меңгеріп, сондай-ақ жаңа пән салаларын дамыта отырып, өзін қайта құру үстінде. Мысалы, компьютердің ойлап табылған 1950 жылдары ең басты талаптардың бірі ғылыми есептеулердің жасалуы болып табылған. Мәтінді өңдеуге, өнімділікті арттыру құралдарына, графикалық дизайн үшін, бизнестің автоматандырылуына, зияткерлік жүйелерге, веб-транзакцияларына және веб-ынтымақтастыққа қойылатын талаптардың саны жедел түрде ұлғая түсті.

Технологиялырдың жетілуі, қоғамдық қайта құрылу мен салаларының дамуы өзара тығыз байланысты. Технологиялардың дамуына қарай талаптар да ұлғая түседі және мәселенің шешімін табу одан әрі күрделіленe түседі. Өзара қарым-қатынасқа түсіп, бұл шешімдер компьютерлік тілді емес, түсінікті адамдар тілін, яғни бағдарламалық қамтамасыз етудің одан әрі тиімді жұмысы үшін төмен деңгейдегі нұсқауларға автоматты түрде аударылатын бағдарламалау тілін қажет етеді. Ғаламтор негізіндегі тілдер мен веб-бағдарламалу 20 жылға жуық қолданылуда және дамуын жалғастырып келеді. Жаппай параллель сәулетке арналған жоғарғы деңгейдегі тілдер әлі де даму үстінде. Жаңа күрделі домендер де дамып келе жатыр. Бұл бағдарламалаудың түрлі стильдерімен байланыс орнатуды талап етеді.

Бағдарламалау тілдерін әзірлеу мен дизайнды жетілдіру барысында технологиялардың дамуы, компьютерлік сәулеттің дамуы, операциялық жүйелердің дамуы, ірі көлемдегі модульдік бағдарламалық қамтамасыз етуді әзірлеу қажеттілігі, сондай-ақ ұзақ мерзімде бағдарламалық қамтамасыз етуді сүйемелдеу тәрізді бірқатар аспектілер басшылыққа алына-

ды. Жаңа пән салалары дамыған сайын жаңа талаптар пайда болады, осылайша жаңа бағдарламалау тілдерін жетілдіруге деген қажеттілік туындай түседі.

1.3 Оқыту нәтижелері

Бұл курсты меңгерудің нәтижесі төмендегілер болып табылады:

1.*Жаңа тілдер үшін өнімділіктің арту қисығының қысқаруы:*

Талаптар автоматтандыру арқылы ұсынылатын қоғамдық инфрақұрылымды жақсарту жолымен жасалады. Болашақтағы бағдарламалау тілі жоғары деңгейде болады, өзінде көптеген бағдарламалау парадигмаларын қамтып, күрделі бағдарламалық қамтамасыз етуді әзірлеуде қолданылмақ. Бағдарламашылар бағдарламалау тілдерінің жаңа парадигмаларын меңгеруге тиіс. Бағдарламалау парадигмаларын, құрылымдарды бағдарламалау мен тағы басқа күтпеген жағдайларды терең және абстрактылы деңгейде түсінбей, қайта жабдықтауды жүзеге асыру мүмкін емес. Бағдарламалау парадигмаларын терең әрі абстрактылы деңгейде түсіну бағдарламашыларға жаңа тілдер арқылы бағдарламалау абстракциясына жаңа тілдер синтаксисін орнатуға мүмкіндік береді.

1.*Бағдарламашылар төмен деңгейдегі қылықтардың жүзеге асырылуын меңгереді*: Бұлкурс абстрактылы машиналардың төмен деңгейінде жоғарғы деңгейдегі конструкциялардың төмен деңгейдегі нұсқауларға аударылуы арқылы әсерін сипаттайды. Төмен деңгейдегі қылықтарды түсіну бағдарламалау барысында орын алатын бірқатар қателіктердің алдын алуға септігін тигізеді.

Бұл тиімді бағдарламалау мен жанама әсерлер курсымен таныстыра отырып, бағдарламалаудың студенттік стилін жақсартуға септігін тигізеді. Жанама әсерлерге есептеудің абстрактілі үлгілерінің нәтижесінде пайда болған қажетсіз есептеу әсерлері жатады. Олар бағдарламаның жұмысында жаңылыстардың пайда болуына әкеліп соғуы мүмкін.

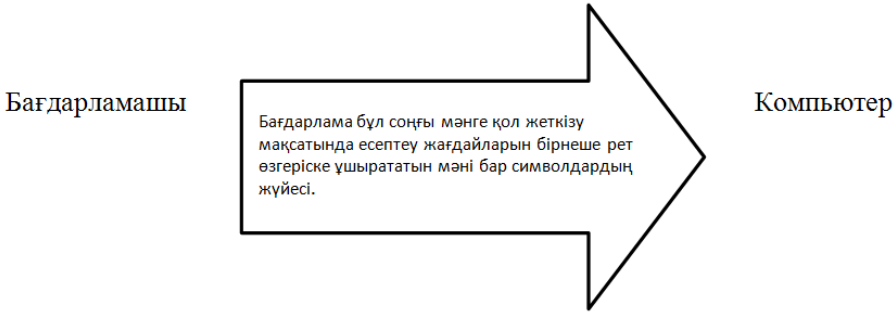
2.*Бағдарламашы компиляторды дамытуға үлесін қоса алады*: Жаңа пән бойынша бағытталған тілдердің дамуына орай жаңа компиляторларды жетілдіру қажеттігі туындап отыр. Бағдарламалау тілдерінің қылығын төменгі деңгейде түсіну тиімді орындауға қажетті кодты әзірлеудің негізі болып табылады.

3.*Бағдарламалау стилін жақсарту*: Студенттер бағдарламалау стилін қолдану аясын кеңейтетін түрлі бағдарламалау тілдері класстарында бірқатар құрылымдармен танысады. Бұған қоса сәкес деректер мен басқару абстракцияларын таңдау арқылы өз шешімдерін тиімді түрде білдіре алады. Әдетте бағдарламашылар C++, Java, PHP немесе C# секіл-

ді бағдарламалау тілдерінің шектелген санымен танысып, бағдарламалаудың белгілі бір стилдерінің әсеріне тап болады. Бағдарламалаудың түрлі парадигмаларында қолданылатын бағдарламалық құрылымдарды білу тікелей бағдарламалаудың өзін жақсартады.
4.*Бағдарламашы сәйкес бағдарламалау тілдері мен парадигмаларды таңдау мүмкіндігіне ие болады:* Студенттер нақты бағдарламалау парадигмаларын қолдану арқылы пән салаларын құрастырып, бағдарламаларды әзірлеуге қажетті сәйкес тілдерді таңдай алады.

1.4 Компоненттер мен бағдарламалар

Үдерісті автоматтандыру мен мәселені шешудің арасында біреуін таңдау сұрағы туындаған жағдайда, жүйелік талдаушы жүйенің үлгісін жасап, қосу мен өшірудің қылығын параметрлеуі тиіс, сондай-ақ блок-сызбаны қолдана отырып, түрлі модульдерді қосуы қажет. Бұл модульдер нақты болып жатқан үдерісті түйіндейді. Бағдарламалар деректерді өңдеуге және осы модульдер арасында деректер ағынын басқаруға бағытталған шешімдерге арналған техникалық талаптар болып табылады.
Нақты мәселерді шешуге арналған техникалық талаптар адам түсіне алатындай етіп, жоғарғы деңгейде сипатталады, ал компьютердің іс-әрекеттері төмен деңгейдегі машиналық командаларға негізделеді. Жоғарғы деңгейдегі нұсқауларды баламалы түрде төмен деңгейлерге (ешбір екі ұштылықсыз) тасымалдайтын аударманы жүзеге асыру қажеттігі туындайды. Екі ұштылықты болдырмау мақсатында нұсқаулар нақты бірегей мағынаға ие болулары тиіс.
Кешенді тапсырманың шешімін ресми түрде көрсету мақсатында бағдарлама мәнді таңбалар жүйесі болып табылады (1.1-сурет). Бағдарлама үш негізгі компоненттен тұрады: *логика+абстракция+* бақылау. *Логика* мәселені шешуге қажетті жоғарғы деңгейдегі техникалық талаптардың пайда болуын білдіреді. Бұл қарапайым тапсырмалардың құрылымданған әдісінде күрделі мәселені шешуді, сондай-ақ соңғы шешімді есептеп шығару мақсатында қатаң түрде белгіленген операцияларға жүгіне отырып, осы шешімдерді үйлестіруді қажет етеді.
Абстракция мәселені шешуге қажетті талап етілетін атрибуттармен үлгілеуді білдіреді. Объектінің бірнеше атрибуттары болуы мүмкін. Алайда барлық атрибуттар мәселені шешуге қажет емес. Абстракцияның артықшылығы бағдарламаларды оңай түсіну мен бағдарламаның қарапайым қызметіне жеңіл түрлендіру болып табылады.



1.1-сурет Бағдарламаларды абстрактілі түрде анықтау.

Бақылау фон Нейман машинасының жұмысына негізделген мәселенің шешімін көрсетуді білдіреді. Бұл жағдайда шешімді қамтитын соңғы есептеуді шығару үшін компьютер жады әрдайым өзгеріп отырады. Әр нұсқау есептеу жағдайын жаңа бір жағдайға өзгертіп отырады. Бағдарламада анық басқаруды қодану бағдарламашыға компьютер жадын нақты түрде өзгертуге мүмкіндік береді. Логиканы жүзеге асыру үшін компьютер жадының түрленуі бір бағдарламашыдан екінші бағдарламашыға өзгеріп отырады. Бұл өз кезегінде бағдарламаны түсіну үдерісін қиындатады.

1.1-мысал
Бұл мысал ретке келтірілмеген сандарды сұрыптаудан өткізетін (көпіршік арқылы сұрыптау) қарапайым алмасу арқылы сұрыптау арқылы үш компонентті көрсетеді. Қарапайым аламасулар арқылы сұрыптау ұсақ ауқымша элементтеріне арналған «қайталанатын максималды ауқым элементі» стратегиясын қолданады. Қарапайым алмасулар арқылы сұрыптау бағдарламасы үш негізгі компоненттен тұрады:

• **Абстракция:**Индекстелетін жүйе ретіндегі сандар жиынтығы болып табылады.
• **Логика:** Іргелес сандарды салыстырып, жүйе аяқталғанша, егер сан көршілес саннан кіші болса, олардың орындарын ауыстырады. Ағымдағы жүйедегі сандарды салыстыру максималды ағымдағы жүйелілікке қол жеткізуге мүмкіндік береді. Бұл элемент алдағы салыстыруларда кездеспейді және бір де бір элемент қалмайынша, қалған элементтер арасында қайталанады.

• **Бақылау:** Бақылау кезінде айнымалылармен байланысты түрлі жад торларында мәндермен алмасу және оларды жаңарту қолданылады.

Бағдарлама бірнеше блоктардың қолданылуымен ұйымдастырылған: (1) бағдарламаның атауы, (2) осыдан бұрын әзірленген, бағдарламалық қамтамасыз ету кітапханасынан немесе модульдерден импортталған бағдарламалар, (3) логиканы білдіруге қажетті тип және айнымалылар туралы ақпараттың мәлімдемесі, (4) түрлі бағдарлама модульдерінің арасында ақпарат алмасу параметрлері және (5) жарияланған айнымалыларды басқаруға арналған командалар жүйесі.

1.4.1 Бағдарламалардағы абстракциялар

Бағдарламада қолданылатын абстракциялардың екі түрі бар: *деректер абстракциясы* және *абстракцияны бақылау.* Деректер абстракциясы орыналған мәселені шешуге қажетті нақты атрибуттарды анықтау арқылы объектілерді үлгілеуде қолданылады. Мысалы, мақсатты классификациялау классын үлгілеу үшін, класс студенттер жиынтығы ретінде үлгіленеді. Бұл жерде әрбір студент үш формалардың бірігуі ретінде үлгіленеді (студент идентификаторы, студенттің есімі, әріптік бағалар). Студенттердің жынысы, бойы, салмағы, адресі, спорттық қызығушылықтары және тағы басқа сипаттамалары қолданылмайды, себебі осы секілді ақпарат орын алған мәселені шешуге көмектеспейді. Деректер абстракциясы деректерді ұсынудың түрлі тәсілдері арқылы жүзеге асырылуы мүмкін. Мысалы, студенттердің реттілігі студенттер массиві түрінде, ал үш форманың жиынтығы үш шегі бар «құрылым» ретінде берілуі мүмкін.

1.2-мысал

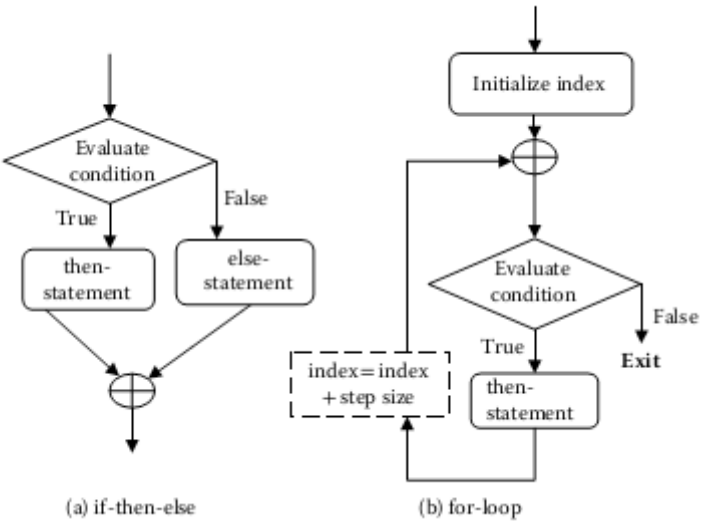
Мысалы, класстың бір осындай ұсынылуы төмендегідей көрсетілген:

```
constclass_size = 20;
struct student {string student-id;
string student-name;
charletter-grade;
}
student class[class_size];
```

Басқару абстракциясы жалпы қасиеттеріне қарай бағдарламалық құрылымдардың түрлі типтерін әр түрлі топтарға жіктейді. Мысалы, *мақұлдау* (түр операторы), *блок операторлары*, *таңдап алу конструк-*

циялары, соның ішінде *шартты операторлар* (*if-then-else конструкциясы)* немесе *көп мәнді таңдау операторлары (case конструкциясы), анықталмаған итерация* – конструкция реттілігінің шартты қайталануы, *анықталған итерация* – алдын ала анықталған итерациялардың нақты саны, *рәсімді бастау* және *функцияны бастау* – бұл басқарудың түрлі абстракциялары. Абстракциялардың басты артықшылығы оңай өзгертілетін және сақталатын жоғарғы деңгей бағдарламасының конструкцияларын сипаттау болып табылады.

Басқару абстракциясы ақпараттық диаграммаларды басқарумен байланысты. Мысалы, *егер (<предикат>) кейін<then-оператор> немесе <else-оператор>* - бұл<*предикат*>ақиқаттығының мәніне негізделе отырып, *<then-оператор>*немесе<*else-оператор*>таңдайтын басқару абстракциясы.Егер біз предикатты бағалауды ромб пішініндегі блок-сызба ретінде, ал операторларды тік төртбұрыш ретінде елестетсек, шартты операторлар 1.2а-суретте көрсетілгендей беріледі. Осыған ұқсас,



Evaluate condition – Бағалау шарты, True – Рас, False – Жалған, then-statement – онда-пікір, else-statement –басқа-пікір, if-then-else – егер-онда-басқа, index=index + step size – көрсеткіш=көрсеткіш + қадам өлшемі, Initialize index –Көрсеткіштің бастапқы мәнін белгілеу, for-loop – for-циклі, Exit – Шығу

1.2-сурет Шартты оператор (if-then-else сызбалары) мен анықталған итерацияның орындалу тәртібі.

Осы секілді біз операторлар блогының қайталануын *анықталған итерация, анықталаған итерация* немесе *деректер арқылы анықталатын итерация*, яғни итерациялар саны деректер элементтері арқылы анықталған итерация, ретінде жасай аламыз.

Анықталған итерация индекс айнымалысы тәсілін, төменгі шекті, жоғарғы шекті және қадамның белгіленген өлшемін қолдана отырып, операторлар блогын белгіленген рет қайталайды. Индекстік айнымалы бастапқы мәнді немесе төменгі (не жоғарғы) шекті қабылдап, әр қадамнан басқа қадамға қарай, *төменгі шек≤индекс айнымалысы≤төменгі шек* шегінен шыққанша өте береді.

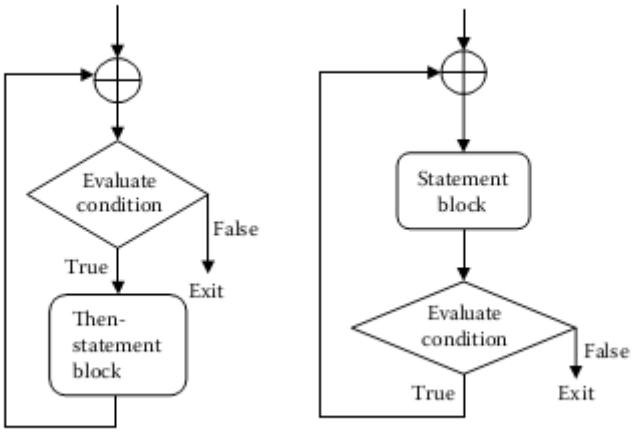
Бұл *анықталған итерация* деп аталады, себебі саны оператор блогы арқылы белгіленген түрде анықталады. Бұл үдеріс үш мәнге негізделеді: төменгі шек, жоғарғы шек және қадам өлшемі.

Бұл мәндер белгілі бір итерация ішінде операторлар блогының шеңберінде өзгертіле алмайды. "FOR" циклі үшін орындаудың жалпыланған блок-сызбасы 1.2b-суретте көрсетілген. Орындалу қалпын бағалау ромб пішініндегі блок-сызбада беріледі. Операторлар блогы тік төртбұрыш ішінде бейнеленген, ал үзік-үзік сызықтар жиегі индекстің орнатылған үстелуін білдіреді.

Анықталмаған итерацияның құрылымы келесі итерация циклін орындау үшін *деректердің логикалық типін (буль типі)* бағалайды, ал предикатты құрайтын компоненттер құрылым ішінде операторлар блогының тұлғасында өзгертіле алады. Осы қасиеттің себебінен анықталмаған итерацияда анықталмаған кідірістің пайда болуы мүмкіндігі туындайды.

Зерттелеген және опреторлар блогына ілесетін бірнеше шартты операторлар блогтарын қамтитын бірқатар циклдар типтері болуы мүмкін. Алайда, барлық түрлі циклдар типтерінің функционалдық күші анықталмаған екі итерацияның көмегімен енгізілгені дәлелденген: "WHILE" және "DO WHILE" циклдары. "WHILE" және "DO WHILE" циклдарының бір-ақ кіру және шығу нүктелері бар және циклдан тыс жерге шығуға жол берілмейді. Бұдан басқа олардың жалғыз ғана шартты опреторы бар. Бұл оператор итерацияның келесі циклына өту-өтпеу жағдайын немесе циклдан шығу қажеттігін анықтайды. "WHILE" және "DO WHILE" циклдары үшін орындаудың жалпыланған блок-сызбасы 1.3-суретте көрсетілген.

"WHILE" циклы блок-сызбаны құрудың алдында *деректердің логикалық типіне* сәйкестілігін тексереді. Pascal секілді тілдерде *"REPEAT-UNTIL"* деп аталатын *"DO WHILE"* циклы алдымен операторлар блогын орындайдын, содан кейін қалпын тексереді.



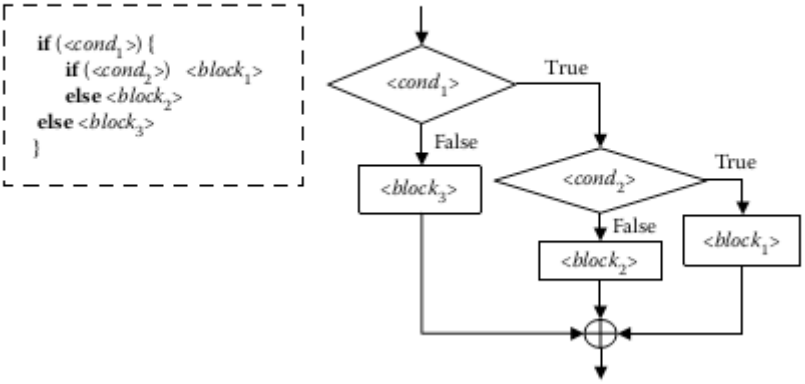
(a) While-loop (b) Do-while-loop

Evaluate condition – Бағалау шарты, True – Рас, False – Жалған, Then-statement block – Онда-пікір блогі, Statement block – Пікір блогі, While-loop – «While» циклі, Do-while loop – «Do-while» циклі, Exit – Шығу
1.3-сурет Анықталмаған итерация үшін орындалу тәртібінің жалпыланған блок-сызбасы.

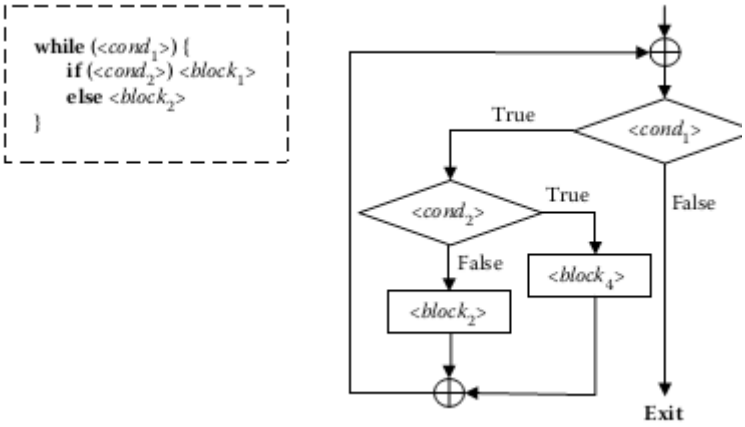
Бұл екі конструкциялардың арасындағы айырмашылық "DO WHILE" циклы операторлар блогын кемінде бір рет орындайды, ал "WHILE" циклы оператор блогын мүлдем қолданбауы мүмкін
Деректер қорларының көмегімен басқарылатын итерациялар, яғни итераторлар, әдетте мультисеттер мен мультисеттерді үлгілеудің жүйелі тізімдері секілді деректер элементтерінің көрсеткіштерін жалпыландыратын деректер абстракциясында қолданылады. Массивтерді қолданатын итерациялардың классикалық сызбасынан өзгеше деректердің ішкі құрылымы бағдарламашыға жасырын болып келеді. Бұл итераторлар әр элемент үшін жалпы деректер блогын орындайды. Итераторлар бұл қызықты абстракциялар, себебі итерациялар саны деректер өлшеміне байланысты және олар тізімдегі элементтерден автоматты түрде аттап өтеді. Lisp және Java-да осындай құрылым *"FOREACH"* циклы болып табылады. Ол итерацияны тізімдегі әр элемент сайын орындайды. Итерацияға арналған жалпы конструкция төменде көрсетілген::
foreachelement **in** <*multiset*>**{**
<*block of statements*>;**}**

<мульти-жиынтықтағы> {<элементтер блогіндегі>} әр элемент үшін
Қарапайым мысалдарға жүгіне отырып, блок-сызба мен тіркелеген блок-сызбаларды талдайық. 1.3 мысалынан тіркелген шартты операторлар үшін орындалу тәртібінің блок-сызбасын көруге болады. 1.4 мысалы анықталмаған итерацияға тіркелген анықталған итерация үшін орындалу тәртібінің блок-сызбасын көрсетеді. Сызбаның әдісі жоғарғы деңгей конструкциясын аударып, содан кейін біртіндеп конструкциялардың келесі деңгейлерін аудару болып табылады.

1.3-мысал
Бұл мысалда біз алдымен сыртқы шартты оператор үшін блок-сызбаны әзірлейміз; сыртқы блок-сызбаға арналған блок-сызбаны әзірлеу кезінде толық шартты оператор, оператор блогының бірі ретінде қарастырылады. Екінші қадам барысында біз шартты операторлардың тіркелген ішкі блогын кеңейтеміз. Нәтиже 1.4-суретте көрсетілген.



if (<cond$_1$>) – егер (<шарт$_1$>), if (<cond$_2$>) – егер (<шарт$_2$>), else (<block$_2$>) – басқа (<блок$_2$>), else (<block$_3$>) – басқа (<блок$_3$>), <block$_1$> – <блок$_1$> <cond$_1$> – <шарт$_1$>, True – Рас, False – Жалған, <cond$_2$> – <шарт$_2$>, <block$_2$> – <блок$_2$>, <block$_3$> – <блок$_3$>
1.4-сурет Тіркелген шартты операторлар үшін жалпыланған блок-сызба.



while (<cond$_1$>) – while (<шарт$_1$>), if (<cond$_2$>) – егер (<шарт$_2$>), <block$_1$> – <блок$_1$>, else (<block$_2$>) – басқа (<блок$_2$>), <cond$_1$> – <шарт$_1$>, <block$_2$> – <блок$_2$>, <cond$_2$> – <шарт$_2$>, <block$_4$> – <блок$_4$>, True – Рас, False – Жалған, Exit – Шығу

1.5-сурет. Тіркелген операторлар үшін блок-сызба.

1.4-мысал

Бұл мысал "WHILE" тіркелген циклына арналған блок-сызбаның дамуын көрсетеді. Алдымен "WHILE" сыртқы циклына арналған блок-сызба "WHILE"сыртқы циклының щеңберіндегі операторларда секілді ішкі шартты операторды өңдеу арқылы әзірленеді. Бұдан кейін енгізілген шартты оператор үшін блок-сызба жасалады. Дайын блок-сызба 1.5-суретте бейнеленген.

1.4.2 Қолсараптамасы (бағдарламаны түсіну) және Ауысулар

Бағдарламалық қамтамасыз ету циклына қатысты маңызды мәселелердің бірі бағдарламалық қамтамасыз етуді сүйемелдеу болып табылады, себебі

(1) даму мүмкіндігі бар, (2) бағдарламашылар өз білімдерін ары қарай жетілдіреді, (3) бағдарламашылар белгілі бір уақыт аралығында өз тәсілдерін ұмытады және (4) сәулет пен технологиялар айтарлықтай өзгереді. Бағдарламалық қамтамасыз етуді сүйемелдеу қол сараптамасына тікелей қатысты: егер басқару элементі күрделі болса, оны түсіну

қиынға соғады және бағдарламалық қамтамасыз етуді өзгерту әрекеттері қателіктерге әкеліп соғады. Бағдарламалауды түсіну мен стандарттарын қолдайтын қажетті тілдік функциялардың болмауының әсерінен БҚ пайдасыз болып қалады немесе ұйымдар бағдарламалық қамтамасыз етуді сүйемелдеу үшін едәуір уақыт пен ресурстарды қажет ететін болады.

Бағдарламашылар бөлімдерді белгілі абстрактілі конструкциялар, (2) абстрактылы аймақтарда орындалу тәртібін түсіну, (3) түрлі бағдарламалық бөлімшелер арасында ақпараттық ағымдарды үлгілеу және (4) бағдарламаны орындамай тұрып, жекелеген конструкцияларды ойша құрғаннан кейін біртіндеп болжамдау арқылы соңғы шартты болжамдау көмегімен түсінеді.

Бағдарламаның адамға түсінікті болуының көптеген факторлары бар. Кодты түсіну үшін бағдарламалау саласында білімнің болуы қажет, бірақ басқа адам арқылы жазылған кодты түсіну үшін бұл жеткіліксіз. Әзірленген кодты түсінуге әсер ететін бірнеше факторлар бар: (1) бағдарламалау тіліндегі абстракция деңгейі, (2) логиканы кодқа айналдыруға қажетті абстракцияның қарапайымдылығы, (3) кодтан шамадан тыс бақылауды жою және (4) басқалар түсіне алатын айнамалылар, блок, модуль және алгоритм деңгейлерінде айтарлықтай түсініктеме беру.

Бұдан басқа көптеген ұйымдар бағдарламалаудың белгілі бір стилі мен ең қолайлы тілді қолданады.

Адамдар құрылылмдалған, түсінікті түрде сыныпталған, модульдері функционалды және орындау тәртібіне сәйкес бағдарламаларды оңай түсінеді. Себебі модульде тармақатар саны ұлғайған сайын адамдарға модульдің жалпы функционалдығын түсініп, сыныптау қиынға соғады. Бұдан басқа ауысулардың тым көп болуы бағдарлама жұмысының құрылымдану деңгейін төмендетіп, адамдар бағдарламаны орындау барысында оның қылығын көзге елестете алмайды.

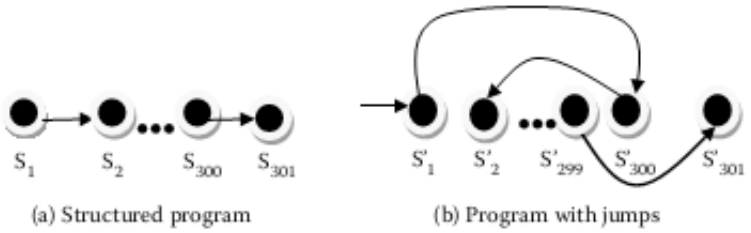
Шартсыз ауысулардың (*ауысу* операторы) функционалдық күші максималды екені дәлелденген болатын. Шартсыз ауысулар (1) *таңдау:* шартты оператор немесе таңдау операторы және (2) *итерация:* "WHILE DO"циклі, "DO WHILE" циклі, "FOR" циклі, сондай-ақ процедураларды шақыру мен функцияларды шақыру секілді басқару абстракцияларын жүзеге асыруда қолданылады. Модульде ауысуларда (1) бірнеше нұсқауларды қашықтықтан жергілікті түрде тасымалдау және (2) операторлар блогынан шығу секілді ауысулар қолданылады. 1970 жылдары компьютер саласындағы ғалымдар арасында қол сараптамасы мен ауысуларды қолдану жөнінде ашық дискуссия өткізілді. Шартсыз ауысулар мен қол сараптамасын қолдану бір-біріне кері пропорционал болып табыла-

ды. Адамдар ауысуларды бірнеше нұсқаулар үшін қашықтықта тікелей бағытта өңдей алады. Алайда артқа қарай ауысу, үлкен қашықтықтағы ауысу және жалпы тым көп ауысулар бағдарламаны сараптау үдерісін едәуір қысқартады. Бағдарламаны блоктарға, ішкі бағдарламаларға, "WHILE DO" және "DO WHILE" циклдары секілді анық функциялары бар объектілер мен абстракцияларға құрылымдау бағдарламаны сараптау үдерісін жақсартады.

1.6a және b-суреттері екі бірдей бағдарламаларды орындау тәртібін көрсетеді. 1.6a-суретіндегі бағдарлама бір нұсқауды бір бағытта бір рет ауыстырылып отырса, 1.6b –суретінде көптеген ауысулар тік және кері ауысулармен қатар қолданылады, ал операторлар тіркелеген ауысу операторларымен араласып кетеді ("GOTO" операторлары).

S1 мен S'1 S'300 –ге ілесе отырып, баламалы, S2 мен S'300кері бағытта S'2 –ге баламалы, S3 пен S'2, S300 эквивалентна S'301-ге кері ауыса отырып баламалы және S4-тен S299-ге дейінгі барлық операторлар сәйкес S'3 -тен S'298-ге дейінгі операторлармен баламалы деп көзге елестетейік. Егер біз функционалдық баламалылыққа, яғни бірдей тапсырманы орындау мүмкіндігіне, көз жүгіртсек, екі бағдарламаның функционалдығы баламалы екеніне көз жеткіземіз. Алайда құрылымданған бағдарламаны 1.6a-суретінен түсіну оңайырақ.

Ауысудың шартсыз операторларын еркін қолданудың себебінен туындаған түсінбеушілік түрлі басқару абстракцияларының жасалуына әкеліп соқты. Ауысу операторларын қолдану абстракцияларды бақылау мен тіркелген блоктардан шығумен шектеледі.



Structured program – Құрылымданған бағдарлама, Program with jumps – Қарғулары бар бағдарлама

1.6-сурет Ауысулары бар спагетти кодымен салыстырғандағы құрылымданған бағдарламаның блок-сызбасы

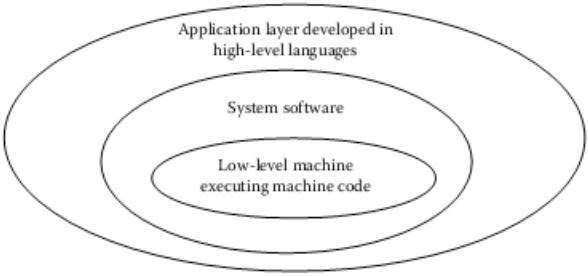
Бұл сурет аусу операторларын қолданатын бағдарламалар бульдік айнымалылар амалдары мен (IF-THEN-ELSE) шартты операторы, "WHILE" немесе "DO WHILE" циклдары секілді басқару абстракцияларын қолдану арқылы функционалдығы жағынан баламалы құрылымданған бағдарламаға аударылу мүмкіндігін дәлелдейді.

1.4.3 Бағдарламалардың орындалуы

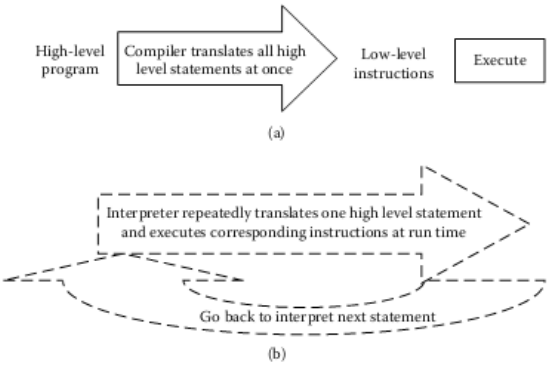
Бағдарламаның дыбыстық орындалуының негізгі белгілері:

1. Бағдарламалау тілінің тілдік конструкциялары нақты түрде анықталуы қажет.
2. Әрбір тілдік конструкция үшін бірегей мән болуы қажет, себебі компьютерлер екі мәнді сұранымдарды өңдей алмайды.
3. Әрбір жоғарғы деңгейдегі конструкция компьютерде әрдайым бірдей іс-ірекеттерді орындайтын төменгі деңгей конструкциясының жүйесіне аударылуы тиіс
4. Компьютер бірдей ақырғы нәтижені шығаратын төменгі деңгейдегі конструкциялардың іс-әрекеттер тәртібін орындауы қажет.

1.7-суретте көрсетілгендей бағдарламалау тілі қабатына дейін бағдарламаны сүйемелдеудің бірнеше қабаты бар. Ең төменгі деңгейде «бос» (БҚ жоқ) машина және машиналық код орналасады. Келесі деңгей – бұл операциялық жүйе, жүйелік утилит және тілдік аудармашылар. Сыртқы қабат – бұл жоғарғы деңгейдегі бағдарламалау және қолданбалы деңгейдің хаттамасы. Жоғарғы деңгейдегі бағдарлама төменгі деңгейдегі нұсқауларға аударылып, бағдарламаны орындауға қажетті компьютерлік жүйелік ресурстар мен утилиттреді пайдалану үшін, аралық қабаттың бірнеше интерфейсін қолданады.
Жоғарғы деңгей нұсқауларын төменгі деңгей нұсқауларына аударудың үш тәсілі бар: (1) бағдарламаны қолданар алдында нұсқауларды компиляциялау; (2) орнатылған абстрактілі машинаны қолдана отырып, жоғарғы деңгей бағдарламасын түсіндіру; және (3) жартылай компиляцияланған код пен жартылай түсіндірілген код нәтижесін көрсететін жоғарғы деңгей тілінің мерзімді компиляциялануы.
1.8-суретте көрсетілгендей компиляциялау үдерісі бағдарламаны орындамас бұрын жоғарғы деңгей нұсқауларын төменгі деңгей нұсқауларына аударады. Екінші жағынан, аудармашылар бұл нұсқауды бірден аударып, орындайды. Компиляцияланған кодпен салыстырғанда түсіндірілген код оператор үшін аудару және орындау циклдарынан өтеді.



Application layer developed in high-level languages – Жоғары деңгейлі тілдерде әзірленген қосымша қабаты
System software – Жүйелердің бағдарламалық жасақтамасы
Low-level machine executing machine code – Машиналық тілді орындайтый төмен деңгейлі машина
1.7-сурет. Компьютердегі бағдарламалық қамтамасыз етудің түрлі қабаттары.



High-level program – Жоғары деңгейлі бағдарлама, Compiler translates all high level statements at once – Құрастырушы бағдарлама барлық жоғары деңгейлі пікірлерді бір уақытта аударады, Low-level instructions – Төменгі деңгейлі нұсқаулар, Execute – Орындау, Interpreter repeatedly translates one high level statement and executes corresponding instructions at run time – Аудармашы бір жоғарғы деңгейлі пікірді бірнеше рет аударады және орындау кезінде тиісті нұсқауларды орындайды, Go back to interpret next statement – Келесі пікірді аударуға кері қайту
1.8-сурет. Компиляциялау мен түсіндіруді салыстыру. (a) схемалық компиляция (b) схемалық түсіндіру.

Компиляцияланған кодтың артықшылығы бағдарламаны орындау барысында ешбір қосымша шығындардың жоқтығы болып табылады. Компиляторларды қолданудың тағы басқа артықшылығы – бұл бағдарлама орындалмас бұрын, бағдарламалық қателіктердің көп пайызы анықталып, жадының таралуы оңтайландырылады.

Түсіндірілген кодтың тиімділігі компиляциялаумен салыстырғанда әлдеқайда төменірек, себебі аудару үдерісі орындау үдерісімен сапырылысып кетеді. Түсіндірудің тағы бір кемшілігі – бағдарламаны орындаудың алдында барлық қателіктер анықталмайды, сондай-ақ бағдарламаны түсіндіру көптеген нұсқауларды орындаудан кейін жаңылыс беруі мүмкін. Бұл аса маңызды бағдарламалар үшін қауіп төндіреді. Алайда аудармашыларды әзірлеу оңай және олар компиляциялау әдістері дамымаған тілдер үшін қолданылған болатын.

Ресми түрдегі бағдарламадағы бағдарламаны дерек ретінде жетілдіріп, кейін орындау барысында оларды бағдарламаға айналдыра алатын тілдер үшін аудармашылар таңдалған болатын. Аудармашылар айнымалылардың барлық мәндерін көрсете отырып, нұсқаулар деңгейінде ретке келтірудің одан да интерактивті мүмкіндіктердің пайда болуына себепші болады. Аудармашылар айнымалы орындау барысында кез келген деректер объектісімен байланыса алтын динамикалық түрде типтелген тілдердің орындалуын сүйемелдейді. Бұдан басқа аудармашылар машиналық тәуелсіз бағдарламаларды сүйемелдейді, себебі бағдарламалар аудармашының шеңберінде орындалады, ал бағдарлама түрлі машиналарға тасымалданбауы тиіс.

Ғаламтор технологияларына негізделген тілдерде код белгілі бір операциялық жүйеге арналып орнатылған абстрактілі машинаны қабылдай алмайтын клиенттік компьютерлерге ағын ретінде беріледі. *Абстрактілі машина* – бұл операциялық жүйе мен компьютер сәулетіне тәуелсіз нұсқауларды орындайтын және осы машинада жұмыс істейтін бағдарлама. Java тілдерінің виртуалды машинасы (JVM) – бұл Java бағдарламаларын бірнеше платформаларда орындайтын абстрактілі машиналардың бірі. Нұсқауларды JVM арқылы орындау машиналық командаларды өз машиналарында орындауға қарағанда әлдеқайда баяу, себебі JVM бірегей сәйкестілік үшін *нөлдік дербестендіруді* қолданады, ал қазіргі компьютерлер екі немесе үш дербестігі бар нұсқаулар жиынтығын қолданады.

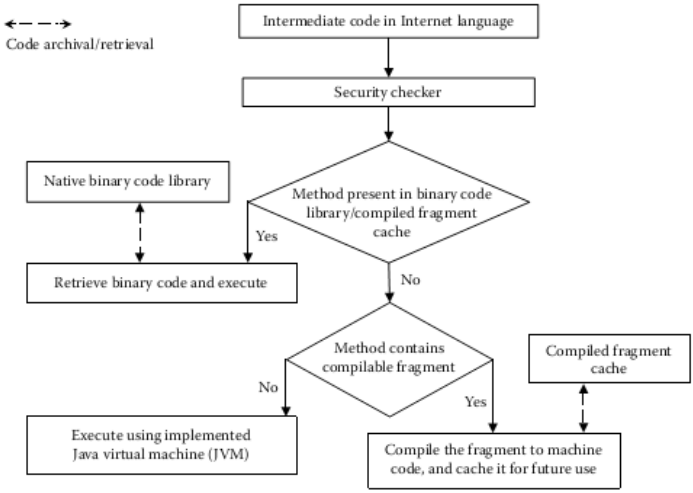
2.1-тарауда сипатталғандай дербестігі үлкен машиналарда кодты орындау нөлдік дербестігі бар машиналарға қарағанда әлдеқайда тезірек жүзеге асырылады.

Аса жоғары дербес машиналарында орындауды жеделдету үшін, JIT

компиляторлары Java машинасында баламалы нұсқауларды орындаудан әлдеқайда жылдам туған операциялық жүйенің командаларын қолдана отырып, орындалып отырған тәсілдердің қорын сақтайды. Қауіпсіздікті тексергеннен кейін аралық код үзінділері аралық код басталғаннан кейін машиналық кодқа аударылып, кітапханада алдағы іздеу үшін кэштеледі. Келесі үзінді басталған кезде, орындалатын машиналық код ізделіп, орындалады. Осыған ұқсас, егер жоғары деңгейлі Java тәсілі өз кітапханасында компиляцияланған кодқа ие болса, ол шығарылып, орындалады. Егер екілік код түсініксіз немесе үзінді шынайы уақыт аралығында компиляцияланбайтын болса, бұл жағдайда жоғарғы деңгей коды нұсқауларға сәйкес Java машинасына ауларылып, орындалады. JIT компиляциялары үшін жалпы сызба 1.9-суретте көрсетілген.

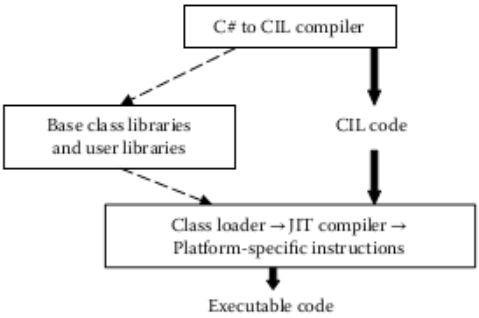
Кодтық үзінділер алғаш рет компиляцияға ұшыраған кезде аударудың қосымша шығындары пайда болады. Алайда келесі үзінділерде аударудың ешбір қосымша шығындары пайда болмайды. JIT компиляциясының көмегімен кодтың орындалуының тиімділігі түсіндірілетін код пен компиляцияланған кодтың орындалу тиімділігінің арасында жатыр.

Microsoft тәрізді кейбір өндірушілердің *Жалпы Аралық Тіл* (CIL) деп аталатын өз аралық тілдері бар. Жалпы аралық тілдің артықшылығы жоғарғы деңгейдің түрлі тілдерінің арасында сәйкестілікті қамьамасыз ету болып табылады. 1.10-суретте көрсетілгендей жоғарғы деңгей тілі алдымен жалпы аралық тілге аударылады, кейін Ғаламтор арқылы тасымалданып, JIT компиляциясының көмегімен түрлі сәулеттерде компиляцияға ұшырайды. C#, visual C++, F# және Visual Basic секілді тілдер жалпы аралық тілге аударылады.



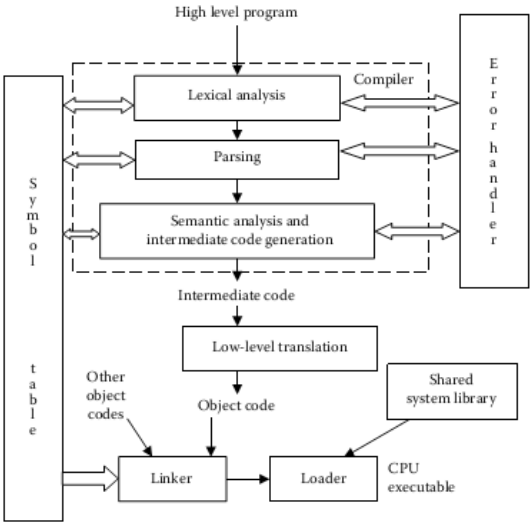
Intermediate code in Internet language – Ғаламтор тіліндегі аралы код, Code archival/retrieval – Кодты мұрахаттандыру/мұрағаттан шығару, Security checker – Қауіпсіздікті тексеруші, Native binary code library – Екілік машиналық кодтың кітапханасы, Retrieve binary code and execute – Екілік кодты мұрахаттан шығару және орындау, Method present… – Екілік код кітапханасында /құрастырылған фрагменттің кэшінде бар тәсіл, Method contains compatible fragment – Тәсілде бірлескен фрагмент бар, Execute using implemented… – Енгізілген.
1.9-сурет Схемалық JIT компиляциясы.

Қазіргі заманғы компиляторлар жоғарғы деңгей бағдарламаларын түрлі сәулеттерде орындау мақсатында аударудың екі сатылық нұсқасын қолданады. Әр түрлі компиляторлардың түрлі құрастыру тілдері болады, сондықтан бірнеше сәулеттер үшін бірыңғай компиляторды жазу мүмкін емес. Оның орнына бағдарламалау тілдеріне арналған аралық код жетілдірілді. Бірінші сатыда компилятор жоғарғы деңгей бағдарламаларын компьютерлік сәулеттерге тәуелсіз аралық деңгей кодына аударады. Екінші сатыда аралық код төменгі деңгейдегі машиналық кодқа аударылады.
1.11-суретте көрсетілгендей аударудың бірінші сатысы (1) *лексикалық сараптама, (2) синтаксистық сараптама, (3) семантикалық сараптама* және *код генерациясынан* тұрады.



C# to CIL compiler – C# CIL компиляциясына; Base class libraries and user libraries - Негізгі класс кітапханалар мен пайдаланушы кітапханалары; CIL code – CIL коды; Class loader->JIT compiler->Platform-specific instructions – Класс жүктеуі->JIT компиляциясы-> Платформа-нақты тапсырмалар; Executable code - Орындалатын код

1.10-сурет C# үшін JIT компиляциясы



High level program - Жоғары деңгейдегі бағдарлама; Lexical analysis - Лексикалық талдау; Parsing – Талдау; Semantic analysis and intermediate code generation - Семантикалық талдау және аралық кодты шығару; Symbol table – Рәміздік үстел; Compiler – Компилятор; Errror handler -

Қате өңдегіші; Intermediate code - Аралық коды; Low-level translation - Төмен деңгейдегі аударма; Object code - Нысан коды; Linker – байланыстырушы; Other object codes - Өзге нысан кодтары; Loader – Жүктеуші; Shared system library - Ортақ жүйелік кітапхана; CPU executable – Орындалатын CPU.
Орындалатын процессор
1.11-сурет. Компьютерде орындау үшін жоғарғы деңгей бағдарламаларын аудару.

Лексикалық сараптаушы кейінге сақталған сөздерді, яғни тілдің, идентификатордың, айнымалылардың және сандардың бөлігі болып табылатын сөздерді тексеріп, синтаксистік сараптаманы жеңілдету мақсатында оларды *маркерлер* деп аталатын ішкі түсінікке айналдырады. Лексикалық сараптаманың соңы – бұл маркерленген ағын және ол синтаксистік сараптаманың бастауы болып табылады. Синтаксистік сараптаушы бағдарламалық сөйлемнің құрылымын бағдарламалау тіліне сәйкес тексереді. Код генерациясы барысында ішкі өңдеуді жеңілдету мақсатында оның соңы ағаш (деректер құрылымы) түрінде беріледі. Семантикалық сараптаушы генерацияға ұшыраған сөйлемдердің мағыналы болуын және код генераторы дерекетер құрылымын сәйкес төменгі деңгей нұсқауларының *аралық код жүйесіне* желілендіруін тексереді. Оңтайландыру деңгейі кодтың артық үзінділерін жойып, орындалып отырған кодтың тиімділігін арттыру үшін процессор тіркелімдерін қолдануды жетілдіреді. *Белгілер кестесі* ақпаратты бұған дейінгі сатылардан қалған ақпаратты сақтау үшін қолданылады. Бұл ақпарат келесі сатыларда қажет болуы мүмкін. Белгілер кестесі қамтитын ақпаратқа айнымалылардың атаулары мен олардың орналасуы, үдерістердің тіркелу деңгейі, жергілікті емес айнымалылар, үдеріс қай жерде атау алды және тағы басқа туралы мәліметтер кіреді.
Екі сатылы компиляцияға қосымша құрастырушы көптеген объектілі файлдарды (компиляцияға ұшыраған код) бір үлкен орындалатын кодқа байланыстырады. Байланыстыру бағдарламашы арқылы құрастыру командасында анықталған тәртіп бойынша жүзеге асырылып, үдерістердің шақырушы үлгілеріне тәуелсіз болып табылады. Бағдарламаның компиляцияланған коды орындалатын кодта тек бір рет жүзеге асырылады және ішкі бағдарламаны шақыру жиілілігіне тәуелді емес. Байланыстырудан кейін үдерістердің өзара орналасуы туралы ақпарат бекітіледі. *Жүктеуші* байланыстырушы кодты жүктейді. Бұл код сегментте, яғни пайдаланушылық үдеріске арналған жад аясында орындалады. Жүктеуші ОЕҚ физикалық адресіне (оперативті есте сақтаушы

құрылғы) логикалық адресті алмастырады. Жүктелген, орындалу үстіндегі файл *процесс* деп аталады және операциялық жүйенің бағдарламалық қамтамасыз етуі мен аппараттық әдістердің қисындастырылуы арқылы орындалады. Бұл әдістермен сіздер операциялық жүйелер курсында танысасыздар. Динамикалық құрастыру кітапханалары – бұл түрлі бағдарламалық қосымшалар арқылы қайта-қайта қолданылатын жалпы жүйелік кітапханалар. Динамикалық құрастыру кітапханасының механизмдерімен сіздер операциялық жүйелер курсында танысасыздар.

1.5 Бағдарламалау тілдерінің сәйкестілігі

Бағдарламалау тілдерінің сәйкестілігі – бұл бір тілдегі код үзіндісінің бағдарламалаудың басқа тілдеріндегі код үзінділерімен өзара әрекетке түсу қабілеті.Бағдарламалаудың қазіргі заманға сай тілдері (1) күрделі тапсырманы орындау барысында пән салалырын алмастыруды қолдау үшін тілдің сәйкестілігін; (2) түрлі тілдерге тән абстракцияларды қолдануды; (3) төменгі деңгей тілдерінің кітапханаларымен әрекеттесу арқылы тиімділікті арттыруды және (4) бағдарламалық қамтамасыз етудің жетілдірілген кітапханаларының кодтарын қайта қолдануды барынша арттыруды, бағдарламалық қамтамасыз етуді қайта қолдануды жақсартуды қамтамасыз етеді.

Сәйкестілікті қамтамасыз ету мақсатында ақпаратпен алмасу үшін, басқа тілдерде жазылған компиляцияға ұшыраған модульды қосу мүмкін болып, *мета деректер*, яғни басқа тілдерде код арқылы қолданысқа түсе алатын экспортталушы деректер типтері мен модуль жайлы жеке жарияланған мәліметтер қамтылуы тиіс. Бұдан басқа ақпаратпен алмасу өз жеке абстракцияларына хост тілінде қайта түрлендірілуі тиіс.

Хост тілі мен түйіндес тілдердің арасында тілдің жалпы спецификациясы мен деректердің жалпы типтерінің жалпы спецификациясын қамтамасыз ету беталысы байқалады.

Жалпы тілдік спецификациялар (1) тілдер арасындағы келісілген типтерді анықтау және қолдану мен (2) корреспонденция типі туралы ақпаратты түрлі тілдерде шығару және сақтауды қамтиды. Кейбір компаниялар, айталық Microsoft компаниясы, әрекеттесуге қажетті жалпы белгілер мен ережелерді қамтамасыз ету мақсатында NET жобасында қолданатын тілдің жалпы спецификациясын жетілдірген болатын.

1.6 Бағдарламалық қамтамасыз етуді жетілдіру циклі

Енді біз сіздердің назарларыңызды бағдарламалау тілдері мен бағдарламалық қамтамасыз етуді жетілдіру арасындағы байланысқа аударамыз. Бағдарламалау тілдері кейбір нақты уақыттағы үдерістерді автомат-

тандыру мақсатында ірі бағдарламалық қамтамасыз етуді әзірлеу үшін қолданылады. Бағдарламалық қамтамасыз ету кодтардың жүздеген тармақтарынан тұруы мүмкін. Ірі бағдарламалық қамтамасыз етудің сәтті дамуы бағдарламашылардың, жүйелік сарапшылардың және ретке келтірушілердің көмегін қажет етеді.

Бағдарламалық қамтамасыз етудің даму циклі бірнеше сатыдан тұрады: *талаптар сараптамасы, жүйелік сараптама және жобалау, жетілдіру, енгізу, бастапқы тексеру, сынақтар, іске қосу* және *эфолюция.* Жаңа сатыға көшпес бұрын, алдағы үдерістерді айқындау үшін, әрбір орындалған сатыдан кейін кері байланыс орнатылады. Іске қосудың соңында жүйенің мүмкіндіктері тұтынушыға түсінікті болып, олардың қажеттіліктері одан әрі дамиды. Жаңа қажеттіліктерді қанағаттандыру үшін жүйелік анализ және жобалау салаларында нақтылау жұмыстарын жүргізу қажет. Инкремент эволюциясы өткен сараптамалар мен жіберілген жобалық қателіктер негізінде дамиды, сондай-ақ бағдарламалық қамтамасыз етуді

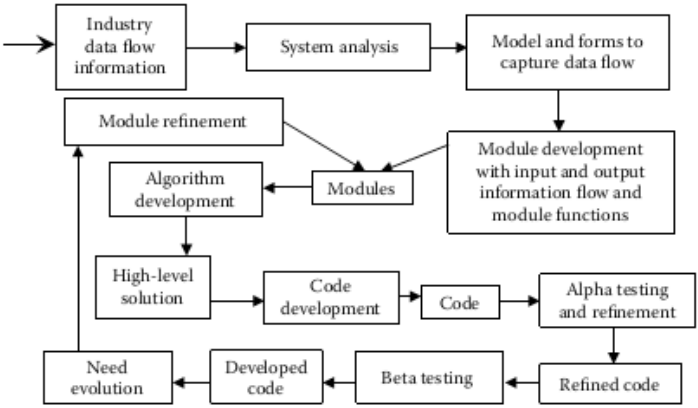
Талаптар сараптамасы сатысында жүйелік сарапшы тұтынушының толық емес және әдетте әлсіз анықталған ақпаратты автоматтандыру жайлы сұранысын, сондай-ақ басқа түрлі сұраныстарды тыңдайды. Автоматтандыру көлемі мен жоба мақсаттары бекітілген болып табылады. *Жүйелік сараптама* сатысында жүйелік сарапшылар тобы автоматтандырылатын жүйе мен оның көлемін зерттейді, содан кейін процесс үшін ақпараттық ағындар үлгісін әзірлейді. *Жетілдіру сатысында* алдыңғы қатарлы бағдарламашылар мен жобалаушылардың көмегімен ақпарат ағыныннң үлгісі түрлі өзара байланысқан модульдерге жіктеледі. Бұл модульдердің тәуелсіз функционалды мүмкіндіктері бар, сондықтан бір модульдің эволюциясы басқа модульдерге айтарлықтай әсер етпейді. Түрлі модульдерден ақпаратты енгізу-шығару құрылғылары бекітіледі. Жобалаушылар модульдерді даму потенциалымен жобалайды, себебі тұтынушылардың қажеттіліктері әрдайым өсіп отырады. Тұтынушылар қажеттіліктерінің эволюциясын болжамдау және елестету күрделі үдеріс болып табылады және тәжірибелі сарапшылар мен жүйені әзірлеушілерді қажет етеді.

Даму сатысында модульдер мен олардың байланыстары (модельдерді қосу-шығару) әзірленіп болған соң алдыңғы қатарлы бағдарламашылар техникалық қиындықтар мен конструкция шектеулерін есепке ала отырып, әрбір модуль үшін деректер абстракцияларын, интерфейстерін және алгоритмдерді әзірлейді. Егер автоматтандыру біреуден артық пән саласын қамтыса, бұл жағдайда модульдер функционалдық сәйкестілікті қолданатын бірнеше бағдарламалау тілдерін қолдана отырып әзірле-

неді.
Жүзеге асыру сатысында негізгі жоғарғы деңгейдегі алгоритмдерді әзірлеуден кейін бағдарламалық код толық жетілген болып есептеледі. Бұл бөлік бағдарламашылар тобы арқылы дайындалған және бұның алдындағы сатыларда әзірленген деректер ағымы интерфейсіне түрлендіріліп, сондай-ақ түрлі бағдарламалау модульдері арасында интерфейстің бағдарламалық модуліне таралады. Қателіктер санын азайту мақсатында бағдарламашылар арқылы басқа модульдердің жергілікті ерекшеліктерін қолдануды болдырмау үшін ақпараттың жасырын түрде сақталуы мақсатында түрлі іс-шаралар қолдануда.
Кодты әзірлеу бағдарламалық қамтамасыз етуді жүзеге асыру үдерісінің тек үштен бір бөлігі болып табылады. Негізгі күш салулар тексеру, тестілеу және іске қосу сатыларында байқалады. Бағдарламашылар бағдарламалық қамтамаыз етудің болашақ қажеттіліктері мен даму циклдеріне аса назар аударып, бағдарламалық қамтамасыз ету саласының дамуын жеңілдетуді қамтамасыз етулері тиіс. Басқа жағдайда тіпті кішігірім бағдарламалық немесе жобалық қателіктер бағдарламалық қамтамасыз етуді жарамсыз етіп, күнделікті негізгі функциялардың орындалуына кедергі келтіреді. Осындай қателіктердің бірі Y2K мәселесі болып табылады. Бұл қателікті түзету миллиардтаған доллар көлеміндегі қаражатты қажет етті. Бағдарламалық қамтамасыз ету аса жедел компьютелік сәулеттер және жақсартылған аппарттық қамтамасыз ету секілді төменгі деңгей технологияларының дамуына үлес қосуы тиіс.
Тексерудің бастапқысатысында бағдарламашылар бағдарламаны тұтынушылар ұсынған іріктелген деректер бойынша тексереді. Бұл саты сәтті аяқталғаннан кейін әзірленген бағдарламалық қамтамасыз ету альфа-нұсқа мәртебесіне ие болып, ретке келтірушілерге тұтынушының алдағы зерттеулеріне жіберіледі. Ретке келтірушілер өзгерістерді енгізіп, үлгіні түзету және бағдарламаға өзгертулерді егізу үшін тұтынушылармен кері байланыспен алмасады. Зерттеулер мен тексерулер барысында жаңа, түзетілген *бета-нұсқа* деп аталатын нұқса қандай да бір қателікті анықтау мақсатында, тұтынушының іс жүзіндегі мәліметтерін қолдана отырып, реттеуші арқылы барлық бағыттарда тексеріледі. Бұл үдеріс соңғы қателері жоқ нұсқа қосылмайынша жалғаса береді.
Іске қосу сатысында нақтыланған нұсқа көпшілік арасында пайдалануға беріледі (немесе тұтынушы арқылы қолдануға), сондай-ақ зерттеулер мен кері байланысты жинастыру жұмыстары басталады. Тұтынушы бағдарламалық қамтамасыз етудің мүмкіншіліктерін көріп, бағдарламалық қамтамасыз ету жұмыс орнында жалпы жүйемен біріктірілгеннен кейін жақсартуларды ұсынады. Ұсынылған өзгерістер *бағдарламалық*

қамтамасыз ету патчтері, яғни бағдарламалық қамтамасыз етудің кішігірім бөлімдері арқылы немесе жұмыс орындарында шағын қателіктер мен қауіпсіздік мәселелерін шешу үшін, бірден бағдарламалық қамтамасыз етуге енгізіледі.

Белгілі бір уақыт аралығында бағдарламалық қамтамасыз етудің күшті және әлсіз жақтары анықталып, тұтынушы бағдарламалық қамтамасыз етуге енгізілетін жаңа қажеттіліктерді анықтайды, сондай-ақ бағдарламалық қамтасамыз ету эволюциясының жаңа итерациялық циклі басталады. Бұл мерзімнің барысында аппараттық технологиялар да өзгеріске ұшырай алады, ал бағдарламалық қамтамасыз ету әртүрлі сәулеттерге тасымалдана алтындай етіліп, сүйемелденуі тиіс. Бағдарламалық қамтамасыз етуді жетілдіру циклінің сызбасы 1.12-суретте бейнеленген.



Industry data flow information - Өнеркәсіп деректер қозғалысы туралы ақпарат; System analysis - Жүйелік талдау; Model and forms to capture data flow - Деректер ағынын ұстау қалу үшін моделі және түрлері; Module development with input and output information flow and module functions - кіріс және шығыс ақпарат ағының модулін дамыту және модуль функциялары; Modules – Модульдер; Algorithm development - Алгоритм дамыту; High-level solution - Жоғары деңгейдегі шешім; Code development - Код дамыту; Code – Код; Alpha testing and refinement - Альфа тестілеу және нақтылау; Refined code – Сараптылған код; Beta testing - Бета тестілеу; Developed code - Дамыған код; Need evolution - Эволюция қажет; Module refinement - Модульды нақтылау.

1.12-сурет Бағдарламалық қамтамасыз етуді әзірлеудің циклі.

«(Жетілдіру үстіндегі жобаның) сарқырама үлгісі» және «спиральді үлгі» тәрізді бағдарламалық қамтамасыз етуді әзірлеуге арналған бағдарламалық циклдің көптеген үлгілері бар. Бұл үлгілер жоғарыдағы талқылауларға толықтай енгізілген жоқ. Алайда бұл үлгілерді салыстыру мен оларды егжей-тегжейлі зерттеу бағдарламалық инженерия курсының шеңберінде қарастырылады.

Бағдарламалық қамтамасыз ету уақыт өте келе дамиды және жаңа технологиялар ескі технологияларды ығыстырғанша әрі қарай дамуын жалғастырады. Бұл жағдайда бағдарламалық қамтамасыз ету үздік автматтандырылу үшін әзірленуі тиіс. Өнеркәсіптік автоматтандыру саласында төрт негізгі өзгеріс орын алған:

(1) жоғарғы деңгейлі, объектіге бағытталған бағдарламалау тілдерінің дамуы; (2)жақсы визуализация үшін визуалды және мультимедиялық технологиялардың дамуы; (3) тілдердің жоғарғы деңгейлі деректер қорларының дамуы және тасымалдауды жеңілдету мен Ғаламтор ресурстары арқылы бірігіп пайдалануға арналған XML секілді веб-тілдердің дамуы.

Бағдарламалық қамтамасыз етуді әзірлеу топтық жұмысты, бағдарламалық қамтамасыз етуге қызмет көрсетуді, ықшамдылық пен үздіксіз эволюцияны қажет ететіні анық. Үлкен топты қажет ететін кез келген бағдарлама ұзақ мерзім арасында бағдарламалық қамтамасыз етуді стандарттауды, модульділікті, тұтынушы үшін қолайлылықты және түсінікті болуды қажет етеді.

1.7 Дұрыс бағдарламалау тілінің критерийлері

Дұрыс бағдарламалау тілінің критерийлері доменнің қажеттіліктерін; бағдарламалық қамтамасыз етуді жетілдіруді, қызмет көрсету мен эфолюцияны, өзара әрекеттесу мен орындалу тиімділігін басшылыққа алады. Критерийлердің кейбіреулері: (1) *абстракция, (2) модульділік, (3) ортогоналдық, (4) ерекшеліктерді өңдеу, (5) тұтынушыға қолайлы болу, (6) дискреттік, (7) түсінудің* және *техникалық қызмет көрсетудің ыңғайлылығы.* Бұның алдында біз абстракциялар, модульдік, мобильдік және түсіну ыңғайлылығы жайлы талқылаған болатынбыз. *Ортогоналдық* бағдарламалау тілдеріндегі конструкциялар бір біріне тәуелсіз болып, шамадан артық болмауы тиіс дегенді білдіреді. *Ерекшеліктерді өңдеу мүмкіндігі* - бұл бағдарлама жаңылыстарын болдырмау мақсатында қателікті түзету үшін тұтынушылық үдерісті бастау арқылы қателіктердің пайда болу талаптарын бекіту мүмкіндігі. Атомдық электр станциясын бақылау, сондай-ақ ұшақ немес ғарыш кемесін басқару тәрізді ірі әрі сыни бағдарламаларда бұл маңызды критерий

болып табылады. *Дискреттік* тілдің бағдарламаны түсінікті әрі оқуға ыңғайлы ететін конструкциялар мен мүмкіндіктерді сүйемелдеуге тиіс екенін білдіреді. Мысалы, айнымалылардың атаулары бағдарламашыларға ыңғайлы етіліп ықшамдалуы қажет. Түсіну ыңғайлылығы мен жалпы қарапайымдылық бағдарламалық қамтамасыз етуді жетілдірумен, эволюциямен, техникалық қызмет көрсетумен және тілде бар абстракциялар санымен тығыз байланысты.

Сала басшылары қызықты сұрақ қойды: менің саламды автоматтандыру үшін қай тіл лайықты? Бұл сұрақ стандарттаудың базалық қажеттілігінен туындайды, осылайша компаниялар белгілі бір уақыт аралығында бғдарламалық қамтамасыз етуді сүйемелдеу үшін аз ресурстарды жұмсайды. Өкінішке орай ешқандай ғажайып амал жоқ; барлық пән салаларын өңдей алатын тіл болмайды. Әрбір саланың өзіне тән түрлі талаптары бар.

Мысалы, жүйелік бағдарламалау тілі құрастыру тілдеріне ұқсас болып, жад модульдері мен қосу/шығу құрылғыларына тиімді қол жетімділікті қамтамасыз етуге қажетті нұқсауларды қамтуы тиіс; жоғарғы деңгейдегі шамадан тыс абстракцияларды қосу тиімділіктің кепілі болып саналмайды.

Ғылыми есептеулер тілі көшірмесіз екі үлкен матрица болуы тиіс, себебі көшірмелеу үлкен көлемдегі жад кеңістігін жұмсап, ірі матрицаларды бір кеңістіктен екінші кеңістікке көшіру барысында қосымша есептеу шығындарының пайда болуына себепші болуы мүмкін. Шынайы уақыттағы тіл шынайы уақытта болып жатқан оқиғаны жоғалтып алмас үшін тиімді болуы тиіс. Ынтымақты тіл түрлі жариялауларды болдырмауға үлесін қосады, алайда бұл жағдайда ол жадты оңтайландыру, тиімді орындаудың болмауы тәрізді қиындықтарға тап болып, кей кездері бағдарламалар орындалу үстінде жаңылыс берулері мүмкін.

Лайықты тілдерге қойылатын осы талаптардың көбі бір-біріне қайшы келеді. Біз бірқатар жоғарғы деңгей абстракциялары бар дұрыс тілді жетілдіргіміз келсе, аударма үдерісі аудармашыларды қолданылатын жалпыланған механизмдер әсерінен көп артық нұсқауларды түрлендіреді.Дәл осылайша егер біз тілді құрастыру тілдеріне ұқсас етіп жетілдірсек, бұл жағдайда портативтілік, сондай-ақ бағдарламалық қамтамасыз етуге қызмет көрсетуге зиян тиеді, себебі түрлі сәулеттерде түрлі тілдер қарастырылады. Біз 16 разрядтық нұсқаулардан 32 разрядтық нұсқауларға көштік, ал соңғы кездері 64 разрядты нұсқаулар қолданылады. Егер бағдарлама нұсқаулармен құрастыру деңгейінде тығыз байланысты болса, бұл жағдайда порт сәулетін әрдайым өзгерту қиынға соғуы мүмкін. Технологиялар жедел өзгеріп келеді, ал бұл кідіріссіз және ең

төмен көлемдегі инвестициялары бар бағдарламалық шешімдерді қажет етеді. Осылайша, портативтілікті, түрленуді және бағдарламалық қамтамасыз етуді сақта мақсатында жаңа тілдер төмен тілдік конструкциялардан бөлінуі тиіс. Олай болмаған жағдайда біз бағдарламалық қамтамасыз етуді жаңа сәулеттерге көшірудің өзіне миллиондаған және миллиардтаған қаражат жұмсаймыз.

1.8 Парадигмалар мен бағдарламалау тілдерінің тарихы

Бағдарламалау парадигмасы –бұл бағдарламалау стилі. Бағдарламалау саласы дамыған сайын тұтынушылар мен компьютерлік ғалымдар бағдарламалаудың алдағы стильдерінде артықшылықтар мен кемшіліктерді анықтап, бағдарламалаудың жаңа стильдерін жетілдіре бастады. І-қосымшада көрсетілгендей қазіргі заманға сай тілдер екі немесе одан да көп парадигмаларды қамтиды.

Біз бағдарламаларды бағдарламалау парадигмаларының бір немесе бірнеше комбинацияларына жіктей аламыз:

(1) императивті бағдарламалау, (2) декларативті бағдарламалау, (3) объектіге бағытталған бағдарламалау, (4) параллель және үлестіруші бағдарламалау, (5) визуалды бағдарламалау, (6) веб-бағдарламалау, (7) оқиғаға бағытталған бағдарламалау, (8) мультимедиялық бағдарламалау, (9) мультиагенттік бағдарламалау және (10) синхронды бағдарламалау. Осымен тізім аяқталмайды. Алайда бұл кітаптың көлемі бағдарламалаудың негізгі парадигмаларын сүйемелдейтін бағдарламалау тілдерінің сыныптарын оқытумен шектеледі.

1. 1.8.1 Императивті бағдарламалау .парадигмасы

Императивті бағдарламалаудың негізі фон-нейман сәулетіне негізделетін *пайымдау* немесе иелену болып табылады. Бағдарламашы компьютерге не істеу керектігін айту үшін, логиканы өзі аударады. Императивті бағдарламалардағы айнымалы компьютер жадындағы ұяшықта сақталады, ал жад ұяшығы иелену операторлары арқылы бірнеше рет өзгертілуі мүмкін. Иелену операторының іс-әрекеті барысында жаңа мән жад ұяшығына жазылып, ескі мән жойылады. Иелену операторының артықшылығы жадты қайта қолдану болып табылады, себебі бір жад ұяшығы бірнеше мәнді сақтай алады.

Алайда бір жад ұяшығына бірнеше мәнді жазудың бірқатар кемшіліктері бар:

1.Жад ұяшығы қайта жазылған жағдайда ескі мән жойылып кетеді, сондықтан ескі мәнді болашақта қолдану мүмкін емес. Жасанды зият

арқылы бағдарламалау барысында шешім іздеудің ауқымды кеңістігінде ізделеді, қайтару-артқа қайту, іздеу бөлігін тоқтату және іздеудің басқа жолдарын қабылдау мүмкін болмайды.
2.Басқа үдеріске немесе функцияға жататын жад көлемі нақты уақыттағы үдеріс арқылы жазылса, ескі мән жойылады. Егер шақырылып отырған үдеріс немесе үдерістер модификациядан хабарсыз болып немесе ескі мәнді қажет еткен жағдайда жаңарту жад ұяшықтарын зақымдайды. Бұның нәтижесінде қате шешімдер шығарылады. Бұл мәселе *жанама әсер* деп аталады және 4.8-тарауда тереңірек қарастырылады.
3.Иелену операторы мен логиканың кезектесуі бағдарламаны түсінуде қиындықтар тудырады, себебі бақылау жекелеген бағдарламашының ойлау стилі үшін аса субъективті болып табылады және ұзақ мерзімді келешекте бағдарламалауға қызмет көрсету саласында қиындықтарды тудырады.

Императивті бағдарламалау парадигмасы 1950 жылдардың соңында және 1960 жылдардың басында *FORTRAN* әр түрлі нұсқаларында және 1960 жылдардың басында *ALGOL* секілді блок түрінде құрастырылған тілдерді жетілдіруде алғашқы жетілдірілген бағдарлама болды. Төмендегі төрт алғашқы тілдер императивті бағдарламалау парадигмалары арасында кең таралған тілдер болған еді: FORTRAN (Фортран), ALGOL (Алгол), COBOL (Кобол), C (Си), ALGOL (Алгол) тілінің ұрпағы.
Фортранның алғашқы нұсқалары ғылыми есептеулерді жүргізуде кең қолданыс тапты, бұл ауысу операторларын қолдануды либералды етті, сондай-ақ оларда сілтемелер болған жоқ. Алгол тілінде бағдарлама *WHILE DO* және *DO WHILE* циклдары тәрізді басқару конструкцияларын сүйемелдейтін блок құрылымы ретінде қарастырылды. Бұл бағдарламаны түсіну үдерісін одан әрі жеңілдетті. Алгол тілі «байланыс тізімі» және «ағаштар» секілді рекурсивті үдерістерді ұйымдастыру үшін қолданылған сілтемелер мен «құрылымдарды» қамтыды. Бұған қоса Алгол қазіргі заманғы бағдарламалаудың императивті тілдерінің негізін қалаушы болып саналады: Алгол арқылы жазылған деркетер абстракциялары мен басқару абстракцияларының көп бөлігі қазіргі тілдерде әлі де қолданысқа ие.
Си тілі бастапқы кезде Bell Labs компаниясының қызметкері арқылы UNIX операциялық жүйесін жүзеге асыру мақсатында әзірленген, жалпы мақсаттағы тіл болған еді. Бұл Алгол 68 тілінің қосалқы жиынтығы болды және ол UNIX жүйесі мен оның вариацияларының арқасында танымал болды. SDL (Алгол 68 жетілуін қолдаған Burroughs компаниясы арқылы әзірленген) және BLISS 32 (Digital Equipment Corporation әзір-

леген) тәрізді басқа да заманға сай тілдер болды, алайда олар өндіріс пен ғылым аясында кең қолданыс тапқан Unix жүйесінің ресми нұсқаларынан артта қалды. Бұл санаттағы басқа есте қалар тілдерге ADA (Ада), Pascal (Паскаль), Модула тілдері, соның ішінде Модула-3 пен Оберон кірді. Бұл тілдер жоғарғы деңгейдегі блокты құрылымданған бағдарламалау арқылы сүйемелденеді. Модула тілдерінің тобы модуль ұғымы мен функциялар импорты және экспорты тұжырымдамасын қолдайды. Барлық аталған тілдер Алгол 68 тілінен бастау алады. Алайда Си тілінен басқа барлық аталып өкен тілдер коммерциялық даңққа ие болмай, тек академиялық мақсатта қолданылған болатын.

Кобол – бұл бизнес ұсыныстарды әзірлеуге арналған тіл. Ол тұтынушыға ыңғайлы болуға, есептерді жасауға және қаржылық деректерді өңдеуге бағытталған. Фортран, Алгол және Кобол тілдерінде құрылымдық бағдарламалау үшін конструкцияларды қосу мақсатында IBM (Ай-Би-Эм) PL-I тілін жетілдіруге тырысты. PL-I бағдарламалаудың бірнеше домендерінде қолданыла алатын тіл ретінде жобаланған болатын. Алайда өзінің ауыр құрылымының әсерінен өл бұл жарыста жеңіліске тап болды.

Ада, Фортран және Кобол тәрізді коптеген бағдарламалаудың императивті тілдері тексерілген конструкцияларды қосудың арқасында дамуды жалғастырды. Фортран және Кобол тілдерінің жаңа нұсқалары блокты құрылымдық бағдарламалау, рекурсивті бағдарламалау, тізімдер негізінде енгізу (стектер), тармақтық деректерді өңдеу, сілтемелер, құрылымдар және объектіге бағытталған бағдарламалау секілді бірқатар функцияларды қамтиды. Уақыт өте келе бағдарламашылар қазіргі заманғы тілдердің артықшылықтары мен кемшіліктерін анықтап, пайдалы конструкцияларды енгізді. Тіл дамыған сайын оның ескі нұсқалармен сәйкестілігін сақтау мақсатында сақ болу қажет, ал эволюцияға ұшыраған бағдарламалар тілдің ескі нұсқаларын қолдану арқылы және жаңа нұсқаларға арналған компляторларды қолдана отырып жазылған бағдарламаларды қолдана алады.

1.8.2 Декларативті бағдарламалау парадигмасы

Декларативті бағдарламалауда бақылау бағдарламаға негізделген. Бақылаудың қамын жейтін абстрактылы машина бар. Декларативті бағдарлама бағдарламалау деңгейінде *логика+абстракциядан* тұрады. Декларативті бағдарламалардағы айнымалылар ұғымының императивті бағдарламалардағы айнымалылардан едәуір айырмашылығы бар. Декларативті бағдарламадағы айнымалы мағынаны ұстаушы болып табылады; айнамалыға мән берілгеннен кейін оны бағдарламашы өзгерте алмайды.

Артықшылықтарына төмендегілер кіреді: (1) шақырылып отырған үдеріс (немесе функция) жадқа шақырылып отырған үдерісті (немесе функцияны) жаза алмайтын жағдайлар секілді жанама әсерлердің аз болуы және (2) айнымалылардың ескі мәндері қажет болған жағдайда сақталып, қолданысқа түсе алады. Бұдан басқа бірқатар кемшіліктері бар, солардың ішінде:

1.Жад ұяшығы FOR циклы үшін индекстік айнымалы ретінде қажет етілмеген жағдайда да қайта қолданысқа түсе алмайды. Итеративті есептеу мен деректер элементтерінің ірі массивтерін басу секілді қосу-шығу үдерісін өңдеу жадының жарылуына әкеліп соғады.
2.Рекурсивті бағдарламалау жиі қолданылады, себебі *индекстің өзгермелі айнымалысын* қолданатын дәстүрлі итерацияларға өзгеріс айнымалысына қойылған шектеулердің әсерінен жол берілмейді. Итеративті бағдарламалаумен салыстырғанда рекурсивті бағдарламалау жадқа ие, сондай-ақ қосымша шығындарды орындау мен бағдарламалаудың аса күрделі үдерісін қамтиды.
3.Жартылай есептеулерді сүйемелдейтін және орындау тиімділігін арттыру мақсатында бағдарламаның баса бөліктері арқылы қолданыс табатын ғаламдық өзгеретін айнымалыларға жол берілмейді. Алайда бірқатар декларативті бағдарламалау тілдері бұл шектеулерді жеңуге талпыныс жасады: Лисп (Lisp) ғаламдық айнымалыларды қолдануды шектей отырып, жад ұяшығына жартылай жазуды мүмкін етеді; ал Пролог (Prolog) жүзеге асыру түріне қарай және жартылай есептеулер нәтижесін сақтау мақсатында "assert" (еркін кеңістікте еркін деректердің мәні туралы болжамдарды тексеруге арналған конструкция), «хабарландырулар тақтасы» (жүйенің барлық модульдері үшін қол жетімді жад аясы) немесе ғаламдық өзгермелілер секілді конструкцияларды қарастырады.
Бағдарламалаудың декларативті тілдерінің екі негізгі типі бар: бағдарламалаудың функционалды тілдері және логикалық бағдарламалау тілдері. 2-тарауда сипатталғандай бағдарламалаудың функционалды тілдері математикалық функцияларды қолдануға негізделген, ал логикалық бағдарламалау тілдері предикаттар логикасын, яғни сандық бағалау ұғымымен бірге буль логикасына негізделеді.
Лисп кейбір императивті конструкциялары бар функционаллды бағдарламамен бірге 1960 жылдардың басында бағдарламалаудың жасанды зиятын, яғни есептеу тәсілдері арқылы адам зиятына еліктеуге тырысатын бағдарламалауды жүзеге асыру мақсатында ойлап табылған танымал тіл. Логикалық бағдарламалаудың әйгілі Пролог тілі 1970 жылы теоремалар мен жасанды зиятты дәлелдеу мақсатында ойлап табылған.

Бұл екі тіл 1980 жылдың басында компиляцияның ауқымды әдістері пайда болған шақта одан әрі жетіле түсті. Бұл тілдерді құрастырушылар тиімімді орындалуда туындайтын қиындықтары мен белгілі бір уақыт аралығында жадты қайта қолдануды болдырмау үшін бірқатар инновацияларға жүгінді. Кейінгі жылдары Лисп тілінің әсеріне ұшыраған басқа да тілдер жасалып, танымал болды. Олардың ішінде: Scheme (Ским), ML (ЭмЭль), Miranda (Миранда) және Haskell (Хэскелл). Қазіргі кезде Ruby (Руби) және Scala (Скала) тәрізді көп парадигмалы тілдер бағдарламалаудың функционалды парадигмасына негізделеді. Толық тізім І-қосымшада берілген.
Бағдарламалаудың декларативті тілдері әдетте жасанды зият саласында кең қолданылады. Декларативті тілдердің жасанды зият саласына қажетті негізгі ерекшеліктері төменде сипатталады:

1.Жасанды зият жүйелері серпінді түрде дамып, білімдерін толықтырулары қажет. АІ тілдері білімдерді қосу үшін абстракцияларға негізделеді.
2.Декларативті тілдердің көбі бағдарламаны бірінші тап ретінде қарастырады. Бұл бағдарламалар орындалу барысында жасалатынын біледі, себебі бұдан кейін деректер орындауға болатын бағдарламаға айналады. Жасанды зият жаңа білімдерді компиляциялау үшін осы қасиетті қажет етеді.
3.Декларативті тілдер метабағдарламалау мүмкіндігін қамтамасыз етеді. Бұл басқа бағдарлама жайлы абстракт түрінде пайымдайтын бағдарлама болып табылады. Мета-бағдарламалаудың бұл қасиеті жасанды зият жүйелерінде пайымдауды жетілдіру мен мүмкіндіктерді түсіндіру барысында аса пайдалы.

1.8.3 Объектіге бағытталған бағдарламалау парадигмасы

Жадтың компьютерлік көлемі ұлғайған сайын ірі бағдарламаларды орындау мүмкін болды. 1960 жылы ең алғашқы объектіге бағытталған тілдердің бірі Симула болып есептелді. Алайда объектіге бағытталған бағдарламалау ұғымы бағдарламалауды қамтамасыз етуді құрастырушылар арасында 1980 жылдардың басында ғана кең тарала бастады. Бағдарлама көлемі өскен сайын адамдар одан да күрделі бағдарламалық қамтамасыз етуді әзірлеуге қажетті модульдік және бағдарламалық қамтамасыз етуді қайта қолдану ұғымдарының құнын түсіне бастады.
Кітапхананы толықтыруды және ақпаратты жасыруды қажет етпейтін, модульдік пен бағдарламалық қамтамасыз етуді қайта қолдану секілді өзара байланысқан ұғымдар бағдарламалық қамтамасыз етуді әзір-

леу үдерісін, техникалық қызмет көрсетуді және оның эволюциясын жеңілдету мақсатында ойлап табылған болатын. Модульдік дегеніміз – бұл ірі компьютерлік бағдарламалардың өзара байланысқа нмодульдер жиынтығына жіктеу. Бұл модульдер басқа модульдердің қызметімен қиылыспайтын нақты функционалдыққа ие болады.

Модульдік дамудың негізгі артықшылығы бір модульдегі модификация басқа модульдердің қызметіне зиян келтірмейтінінде. *Бағдарламалық қамтамасыз етуді қайта қолдану* бұған дейін әзірленген бағдарламалық қамтамасыз ету оны файлда сақтау және қажетті модульдерді немесе ішкі бағдарламаларды мұрағаттан импорттау арқылы қажет кезде қайта қолданыла алатынын білдіреді. Ақпаратты жасыру, егер ақпарат басқа модульдермен әрекеттесуге қажет болмаса, жүзеге асырылған бағдарламалық модульдің бір бөлігін басқа модульдерге қол жетімсіз етіп жасауды білдіреді.

Модульдік пен ақпаратты жасыруды қамтамасыз ету мақсатында объектілер мен сыныптар ұғымдары ойлап табылды. Объект деректер мен тәсілдер абстракциясын, сондай-ақ объектіде деректер абстракцияларын басқаруға қажетті үдерістерді қамтиды. Объект ішіндегі ақпарат *жеке, көпшілікке қолжетімді* немесе *қорғаулы* болуы мүмкін. *Көпшілікке қолжетімді* тәсілдер сыртқы әлемге ашық; *жеке* тәсілдер ақпаратты жасыруға қатысатын объектіге ерекше; ал *қорғаулы* тәсілдер объектілерді ағымдағы сыныптың сынып тармағында көре алады. Ақпаратты жасыру модульдік ұғымымен тығыз байланысты, себебі жасырылған ақпарат басқа объектілер немесе сыныптар арқылы қолданыла алмайды. Бағдарламалық қамтамасыз етуді қайта қолдану ұғымы мұралануды қолданудың арқасында мүмкін болды. Бұл жағдайда сынып тармағы ата сыныбынан тәсілдерді мұрагерлікке алады, сондай-ақ бағдарламалық қамтамасыз етуге енгізілетін кітапханаларды жетілдіруді өзінде қалдырады.

Объектіге бағытталған бағдарламалауға алғаш енгізілген тіл Симула (SIMULA) деп аталды. Содан бері көптеген тілдер объектіге бағытталған парадигмамен толықтырылды. Xerox Parc және Эйфель компаниясы арқылы жетілдірілген Smalltalk (Смоллток) тілдері объектілер концепциясына негізделді Мысалы C ++ - бұл C (Си) тілінің интеграциясы және объектіге бағытталған парадигма, Клос тілі(Common Lisp Object System — «Common Lisp'a объектілік жүйесі») Лисп тілі мен объектіге бағытталған парадигманың интеграциясы болып табылады. Пролог тілінің көптеген түрлері объектіге бағытталған бағдарламалауға қажетті кітапхананы қамтиды. Python (Питон), Ruby (Руби) және PHP тәрізді қазіргі заманғы скрипттік тілдер өздерінің императивті ата-бабаларынан қалған орнатылған объектіге бағытталған бағдарламалауға ие.

Фортран 2008 және Кобол 2002 — ерте императивті нұсқалары бар интеграцияланған объектіге бағытталған парадигманы қамтитын Фортран және Кобол тілдерінің соңғы эволюциясы.
Java ғаламторлық бағдарламалау парадигмалары мен объектіге бағытталған парадигмаларды интеграциялайтын C ++ тілінің ізін басушы болып табылады. Соңғы жылдары жаппай параллель сәулеті бар компбютерлерде жоғарғы деңгейдегі бағдарламалық қамтамасыз етуді жетілдіру мақсатында IBM зерттеушілері арқылы ойлап табылған X10 тілі бағдарламалаудың параллель объектіге бағытталған тілі болып табылады.

1.8.4 Параллель бағдарламалау парадигмасы

1980 жылдардың онжылдықтары барысында бағдарламалық қамтамасыз ету саласы ұлғайған сайын, аппарттық технологияда жедел дами бастады. Компьютерде көптеген жедел қызмет атқаратын процессорла рқолжетімді болып, компьютерлік желілер саласындағы зерттеулер бірнеше компьютерлер арасындағы ақпаратпен алмасу мәселесін шешті.
Осындай жаңа зерттеулердің негізінде тәуелсіз ішкі мәселелер бағдарламаның орындалу тиімділігін арттыру мақсатында жекелеген процессорларда көрініс табулары мүмкін. Даму екі бағыт бойынша жүрді: (1) Жүйелі бағдарламаны шығу деректері ретінде қабылдай алатын және оларды параллель нұсқада параллель орындалу үшін түрлендіретін параллель дейтін компиляторлар және (2) орындалу ағыны, белсенді күту циклі, сондай-ақ үдерістердің қашықтықтағы шақырулары секілді жоғарғы деңгей конструкцияларының бағдарламалық конструкцияларын қамту.
Аталған конструкциялар да үдеріс CPU орындалатын бағдарламаның белсенді бөлігі болып табылады және орындалу тиімділігін арттыру мақсатында бір немесе бірнеше бір уақытта жұмыс атқаратын ішкі мәселелерді қоса алады.
Параллель бағдарламалау парадигмасы бар көптеген тілдердің параллель баламалары "Concurrent X" немесе "Parallel X" атауларына ие болды. Бұл жерде X – бар тілдің атауы. Мысалы, Си тілінде параллель конструкцияларының интеграциясы "Concurrent C" деп аталды; Паскаль тіліндегі параллель конструкциялардың интеграциясы "Concurrent Pascal" деп аталды; Пролог тіліндегі параллель конструкциялардың интеграциясы "Concurrent Prolog"; ал Фортран тіліндегі бағдарламалаудың параллель конструкцияларының интеграциясы "Parallel FORTRAN" деген атауға ие болды.
Параллель бағдарламалық конструкциялар қазіргі заманға сай компью-

терлердің негізіндегі көп ядролық процессорлардың дамуына орай кең қолданыс табуда. Қазіргі заманғы бағдарламалар мен тілдер бағдарламаның орындалу тиімділігін арттыру мақсатында аталған конструкциялардың көп бөлігін қамтиды. Мысалы, орындау ағындарын қолдану орындау ағындарының қолданыстағы кітапханалары немесе орнатылған кітапхана ретінде қазіргі заманғы тілдерді кең таралған құбылыс Мысалы, ауқымды кітапхана C, C+ және Java үшін құрастырылды.

1.8.5 Визуалды бағдарламалау парадигмасы

Мәтіндік бағдарламалау парадигмасы тек бір өлшемді қолданады. Алайда біз тек жүйелі бір өлшемді бағдарламалауды ғана емес, жақындық ұғымын да жақсы түсінеміз. Бірқатар бағдарламалау тілдері мәтіндік болып табылады және бір өлшемді болумен байланысты секвенциалдылықтың болмауынан зардап шегеді.

1980 жылдардың соңында Smalltalk тілімен жұмыс істеуден бастап, ынтымақты интерфейсті қамтамасыз ету мақсатында визуалды бағдарламалау парадигмасы бірнеше бағытта дами бастады. Кейбір ұсыныстар төмен деңгейлі бағдарламалауда түрлі абстракция деректерімен абстракцияларды басқаруға арналған символикалық көрініс қолдана отырып, бағдарламалау тілдері үшін еркін болды. Алайда ыңғайлы интерфейс пен анимацияны қамтамасыз ету мақсатында визуалды бағдарламаны қосу талпынысы Drag-and-drop (сөзбе-сөзаударғандасүйреплақтыру; алдалақтыр) операциясы арқылы шектелді. Соңғы кездері визуалды бағдарламалау оқиғаға абағытталған бағдарламалау үшін C # секілді тілдерде қолданылды: өзара әрекеттесетін объектілердің күрделі сценарийін жасау үшін объект атрибуты мен оқиғаларына сәйкес келетін символдар тасымалданады. Төменгі деңгейде бұл оқиғалар мен өзара әрекеттесетін объектілер тілдің төмен деңгейлі нұсқасына аударылады. Осыған ұқсас Alice (Элис) тәрізді визуалды тілдер SMIL, VRML және Java3D секілді анимациялар мен тілдер кодтарын әзірлеу үшін мультимедиялық презентациялар мен анимациялардың ғаламтор технологияларын анегізделген визуалды бағдарламалауды қолданады.

Төменгі деңгейдің ірі көлемдегі жалпы мақсаттары үшін визуалды бағдарламалау парадигмасын қолданудың әрекетсіз болуы төмендегі себептерге байланысты: (1) екі өлшемді жазықтықтағы синтаксистік сараптаманың күрделі болуы, (2) екі өлшемді жазықтықтарда ірі көлемдегі бағдарламалардың түсініксіз болуы, (3) бағдарламашыларға ірі визуалды бағдарламаларды ұсынудың қиындығы. Бұған қоса визуалды бағдарламалауда қолданылатын символдардың стандартталуы қарастырылмаған.

1.8.6 Мультимедиялық бағдарламалау парадигмасы

Мультимедиялық бағдарламалау парадигмасы деп визуализацияның түрлі режимдерінің интеграциясын түсінеміз. Оларға мәтін, сурет, бейне жазба және ымдар жатады. Адамдар аталған белгілер арқылы бірбірімен қарым-қатынасқа түседі. Бұл белгілерсіз біздің қарым-қатынасымыз және өмірді қабылдауымыз толық емес болар еді.

1990 жылдардың басында веб-бағдарламалаудың дамуы мен дыбыстық және бейне жазбаларға арналған кешенді үлгісін жетілдірудің арқасында визуализация мен қабылдауды жақсарту мақсатында мультимедиялық объектілер мен бейне жазбаларды бағдарламаларға кіріктіру мүмкіндігі пайда болды. Бейне жазба реттік кадрлар ретінде ұсынылуы мүмкін. Бұл жерде әр кадр өзара әрекеттесетін объектілер жиынтығы болып табылады. 3D-анимацияланған объектілер мен виртуалды ақиқатты үлгілеуге қажетті іс-әрекеттерді орындайтын Alice (Элис) және виртуалды ақиқатты үлгілеу тілідері (VRML) бар. Соңғы жылдары X3D және Java3D тәрізді 3D үлгілеу тілдері ғаламтор арқылы нақты уақытта анимация үшін есептеулер мен 3D үлгілеуді біріктірген даму сатысына көшті.

1.8.7 Веб-бағдарламалау парадигмасы

1990 жылдардың басында Ғаламтордың пайда болуы бізге қашықтықтағы веб-бағдарламаларда орналасқан деректермен, бейнелермен, бейнежазба және аудио материалдарымен, деректер қорларымен және ұялы кодтармен алмасуға үлкен мүмкіндік берді. Бұған қоса біз кодпен деректер мобильділігіне қол жеткіздік. Егер қашықтықтағы ресурс кодты бірігіп қолданбаса, ол деректерді дереккөздерде анықтап, алынған деректерді тасымалдай алады. Екінші тұрғыдан, серверге артық күш түспеуі үшін, ол есептеуді ең соңында орындау мақсатында тұтынушыға кодты жібереді. Веб-бағдарламалау мультимедиялық визуализацияның қозғалтқышы болып, қор нарығымен банк ісі тәрізді қаржылық есептеулерге едәуір әсерін тигізді. Java және SMIL секілді ғаламтор негізінде жасалған және PHP, Javascript пен XML (белгілеудің кеңейтілген тілі) секілді веб-жетілдіру тілдері бар. XML ғаламтор арқылы деректер қорын, есептеулерді және үлгілеуді ұсынуда кең қолданылатын аралық тілге айналды. Java *Java виртуалды машинасы (JVM)* деп аталатын орта деңгейлі әйгілі абстракт машинасына ие. Java бағдарламалары нөлдік адресі бар ассемблер тіліне ұқсас Java виртуалды машинасын қолданумен түсіндіріледі. Нөлдік адресі бар машиналар тізіммен нұсқаудағы жад адресін қолданбайды. Бұл операндтар бағалау тізіміне енгізіліп, ығыстырылып, бағаланатын және қайта стекке жіберілетін стекті есеп-

теу машинасы. Нөлдік адресі бар машиналардың қолданылу себебі Java тілінің микротолқынды пеш, ақылды үйлер мен тоңазытқыштар секілді күнделікті құрылғыларда қолданылу мүмкіндігі болып табылады. Нөлдік адресі бар машиналар барлық дерлік орнатылған компьютерлерде қосыла алатын кең таралған абстракт машиналары болып табылады.

Нөлдік адресі бар машиналардың кемшілігі жоғарғы деңгей нұсқауы басқа нұсқауларға процессордағы синхронизациялау циклын бос жұмсай отырып аударылуы болып табылады. Орындалу үдерісін жеделдету мақсатында JIT компиляциясының жаңа типі жетілдірілді. Бұл типте Java және C # секілді жоғарғы деңгей тілдерінің кітапханалары өздерінің екілік кодтарында компиляцияланады. Жоғарғы деңгей тілінің бағдарламалық үзіндісі төменгі деңгей баламасына аударылғанда екі мүмкіндік туындайды: (1) жедел орындалуды қамтамасыз ету үшін кодты компиляцияға ұшыраған өзекілік кодына аудару немесе (2) Java немесе NET (Microsoft Windows операциялық жүйесі үшін) виртуалды машиналары секілді машинаның абстракт түсіндірушісін қолдану. JIT компиляцияланған бағдарламалары Java виртуалды машинасында түсіндірілген кодтарға қарағанда тезірек орындалады.

1.8.8 Оқиға-бағдарланған бағдарламалау парадигмасы

Оқиға – бұл кейбір әрекеттер немесе есептеулер туындататын шарт тудыратын жағдайлар. Мысалы, тышқанның тетігін басу оқиға болады; компьютер тышқанын суреттің бетімен қозғалту оқиға болады; және соңғы мәнді алу құралы оқиға болады.

Дәстүрлі енгізу-бағдарланған бағдарламалау мен оқиға-бағдарланған бағдарламалау арасындағы айырмашылық енгізу-басқару бағдарламалау енгізудің кейбір мәліметтерін сұрайды, енгізуді күтеді, содан кейін ғана енгізу дабылының мәніне негізделген әрекетке көшеді. Бұған қарама қарсы оқиға-бағдарлану бағдарламалау ешқашан да сыртқы ортадан ешқандай енгізу дабылын күтпейді. Соған қарамастан, ол орын алған сәтте-ақ бір немесе бірнеше оқиғаға әсерлеседі. Оқиға-бағдарланған моделдеу шынайы құбылысты моделдеу үшін пайдаланылуы мүмкін. Өйткені, оқиға алдағы уақытта қосымша әрекетке алып келетін каскадты оқиғаға себеп болуы мүмкін. Оқиғалы бағдарламалау өз негізін 1970-жылдың басында жасалған Симула (SIMULA) бағдарламасынан алған. Соған қарамастан, соңғы жылдары ол графикалық және веб-әсерлесу салдарынан кеңінен тарала бастады. Қазіргі C # сияқты тілдерге оқиғалы бағдарламалау моделдеу және графикалық қолданушы интерфейстер үшін қолданылады.

1.8.9 Бағдарламалау парадигмаларын интерграциялау

Қазіргі тілдер бағдарламалаудың тек бір парадигмасына жазылмаған: бағдарламалаудың бірнеше парадигмасы тілге орнатылған. Мысалы, C++ императивті бағдарламалау парадигмасын және нысанды-бағдарлау бағдарламалау парадигмасын қолданады. Visual C++ визуалды бағдарламалау парадигмасын қолданады. C# императивті, нысанды-бағдарлы, оқиғалы бағдарламалаудың парадигмасын және визуалдаудың үлкен мүмкіндігі – мультимедиялық бағдарламалау парадигмасының функцияларын қолданады. Javada императивті, нысанды-бағдарлы, веб-бағдарламалау, параллель және оқиғалық бағдарламалау парадигмасының көптеген сипаттары бар. Scala және Ruby функционалды және нысанды-бағдарлы бағдарламалау парадигмаларын біріктіреді. IBM компаниясы жасаған X10 бағдарламалау тілі императивті, нысанды-бағдарлы және параллель бағдарламалау парадигмаларын біріктіреді. Қазіргі PHP және Python скриптілідері императивті, құрылымдық және нысанды-бағдарлы бағадарламалау парадигмаларын біртұтас етіп біріктіреді. Кейбір әйгілі бағдарламалау тілдерінде қолданылатын бағдарламалау парадигмаларының тізімі 1-қосымшада келтірілген.

Бағдарламалаудың әрбір парадигмасының артықшылығымен қоса кемшілігі де бар. Мысалы, нысанды-бағдарлы бағдарламалаудағы нысан түсінігі ақпаратты жасыру және модуль түсінігімен қатар бағдарламамен қамтамасыз етудің ірі масштабты дамуы үшін жарайды және қазіргі таңда Фортан (соңғы нұсқасы Фортан 2008) сияқты ескі тілдерді де қабылдайды. Бұл тілдер тізім негізінде қолдану, белгілі бір уақыт ішіндегі рекурсия мен нысандар сияқты жақсы функцияларды қоса отырып, уақыт өте келе дамыған. Сәйкесінше, COBOL тіліне (соңғы нұсқасы, COBOL 2002) нысандар қосыла бастады. Егер тіл уақыт өте келе дамуын тоқтатса, онда бағдарламашылар мүмкіндігі шектелгендіктен ол тілді қолдануды доғарып, ескі тілдерді бағдарламалаудың белгілі парадигмалары бар жаңа тілдермен алмастырады.

1.9 Тілдердің жіктелуі

Тілдер бағдарламалау тілінде қолданылатын бағдарламалау парадигмасы, қолдану моделі, түрлердің түсінігі және олардың мүмкін болатын пәндік салалары сияқты көптеген белгілеріне қарай жіктелуі мүмкін. Соған қарамастан, қазіргі көптеген тілдерді нақты бөліну жоқ. Өйткені олар негізгі артықшылықтарын пайдалану мақсатында белгілер мен парадигмалардың жиынтығын біріктіреді.

1.9.1 Бағдарламалау парадигмаларды негізіндегі жіктеу

Біз бағдарламалаудың әртүрлі парадигмаларын қарастырдық. Мысалы, бұрынғы Фортран, Си, Алгол 60, Паскаль, АДА және PL/I тілдері императивті тілдер болды; Лисп, ML және Scheme көне нұсқалары - функционалды бағдарламалаудың әйгілі тілдері болды; Пролог – логикалық бағдарламалаудың кең тараған тілі; ал C++, Java, Scala, Ruby, Python және X10 – бағдарламалаудың көптеген парадигмаларын қамтыған тіл өкілдері. C++ - бұл императивті және нысанды-бағдарлы бағдарламалау парадигмасын біріктіреді; CLOS («Common Lisp'a нысанды жүйесі») – функционалды және нысанды-бағдарлы бағдарламалау парадигмасының интерграциясы; Concurrent C, Concurrent Pascal, және Parallel Fortran – бұлар параллель және императивті бағдарламалау интеграциясының мысалдары және т.с.с. Javaны орындау ағыны параллель бағдарламалауға мүмкіндік береді. Осылайша, Java императивті, нысанды-бағдарлы, параллель, оқиға-бағдарлы және веб бағдарламалау парадигмаларының интеграциясы болып табылады. Microsoft C# - бұл императивті, нысанды-бағдарлы және оқиға-бағдарлы бағдарламалау парадигмасының интеграциясы. .NET-пен үйлесіп, Microsoft C# веб-бағдарламалау үшін де жарай береді. Scala және Ruby сияқты соңғы тілдер көптеген бағдарламалау парадигмаларын қолданады. Болашақта бағдарламалаудың бірнеше парадигмаларын біріктіретін осындай тілдер дами береді. Бұдан да көне тілдер ішінде параллель бағдарламалау және нысанды-бағдарлы бағдарламалау бар еді.

1.9.2 Қолдану негізінде жіктеу

Тілдер сондай-ақ қолдану үлгісі негізінде де жіктелуі мүмкін. Кеңінен қолданудың төрт үлгісі бар: (1) статистикалық қолдану, (2) тізім негізінде қолдану, (3) хипті (heap - үйме) толтыру негізінде қолдану және (4) қолданудың атқару тиімділігін арттыру үшін жадыны статистикалық, ағынды және үймелі таралуын интегралдайтын интегралданған үлгісі. 5-тарауда айтылғандай статистикалық қолдану компиляция кезінде берілгендер мен айнымалылардың әрбір жарияланған құрылымы үшін жадыны таратады және атқару кезінде жадының өсуіне ешқандай мүміндік болмайды. *Тізім негізінде қолдану «басқарушы стек»* деп атайтын тізімді қолданады. Бұл шақырылған рәсімді іске қосу үшін қажетті әртүрлі көрсеткіштер сияқты бағдарламаны іске асыру кезіндегі жергілікті айнымалылар мен басқа да талаптарды бөлу үшін керек. Стек қолданудың артықшылығы стек ішкі бағдарлама деп аталатын тізбекке қажетті жады негізінде атқару кезінде кеңеюі және азаюы мүмкін. Бұл кеңею жұмыс кезеңінде жадының динамикалық өсуіне, сондай-ақ ре-

курсивті процедураларды қолдануды қолдауға және жадыны қайтадан пайдалануға мүмкіндік береді. Хипті толтыруға негізделген қолдану үлгісі барлық процедураларға қол жетімді жадының жалпы орталық облысын пайдаланады. Хипті толтыруға негізделген қолдану үлгісі жадының барлық процедураларға қолжетімді орталық жалпы облысын пайдаланады. Жадыға керек кезінде хиптан алынатын мәліметтер құрылымы керек, және оны атқарып жатқан кезде динамикалық түрде шығару керек немесе бағдарламашының нақты әрекеттерінің көмегімен қолмен немесе мәліметтер құрылымы енді қолданбайын кезде автоматты түрде іске қосылады. Хипті толтыруға негізделген қолдану мәліметтердің динамикалық құрылымын нысандарды нысанды-бағдарлы бағдарламалауға таратуға, сондай-ақ рекурсивті мәліметтер құрылымын байланысқан тізімдер мен ағашты таратуға мүмкіндік береді.

Статистикалық қолдану байланысқан тізімдер және ағаштар және динамикалық нысандар сияқты мәліметтердің рекурсивті құрылымдарын, рекурсивті процедураларды сақтамайды. Соған қарамастан, статистикалық қолдану мәліметтер элементіне рұхсат алу үшін бір жадыға рұхсатты қолдана отырып жадының абсолют адресін пайдаланады. Осыдан, статистикалық қолдану жадыға бірден көп рұхсатқа бара-бар көрсекіштерге негізделген стек негізіндегі қолдану мен хипті толтыруға негізделген қолдануға қарағанда мәліметтер элементтеріне тезірек рұхсат ала алады.

Стек негізіндегі қолданудағы стек өлшемі операциялық жүйелер рұхсат еткен жадымен ғана шектелген. Процедураларға қажетті ұяшықтар сәйкес процедуралар шақырылған кезде жұмыс атқарылып жатқан уақытта бөлінуі мүмкін. Жады ұяшықтары процедураның шақырғыш шаблоны негізінде стекың жоғарғы бөлігінде орналасады және шақырылған процедура аяқталған сәтте қалпына келеді. Бұл *жадыны қайта пайдалануға* мүмкіндік береді. Өйткені, жадының дәл сондай көлемі басқа процедураны шақырған кезде пайдаланылуы мүмкін. Қорытындылай келе, стек негізінде қолданудың екі негізгі артықшылығы бар деп айтуға болады: рекурсивті процедураны атқару және жадыны қайтадан пайдалану. Соған қарамастан, стек негізінде қолдану кезінде дербестендіру механизмі 5 тарауда көрсетілгендей көрсеткіштер мен жылжытуды қолданады. Сәйкесінше, статистикалық қолданумен салыстырғанда стек негізіндегі қолдану айнымалылар мен мәліметтер құрылымына рұхсат алу үшін қосымша шығындар қажет етеді.

Бұл қолданудың негізгі кемшіліктерінің бірі мәліметтердің байланысқан тізімдер мен ағаштар сияқты рекурсивті құрылымдары, сондай-ақ динамикалық жасалған нысандар сияқты динамикалық мәліметтер әлі күнге

дейін қолданылуы мүмкін. Рекурсивті мәліметтер және динамикалық нысандар құрылымында стек негізінде қолдану кезінде олардың жұмыс істеуіне кедергі келтіретін екі негізгі мәселе бар. Бірінші мәселе компиляция кезінде мәліметтер құрылымын жасау кезіндегі белгісіздікте, ал екіншісі мәліметтер құрылымының өлшемі. Рекурсивті мәліметтер құрылымы жағдайында компиляция кезінде тапсырманы орындау барысында қанша жады керек екені анық емес. Мәліметтерді шақыру және енгізуге байланысты рекурсивті мәліметтер құрылымында ұяшықтардың әртүрлі санында бірнеше рет шақырылуы мүмкін. Сонымен қатар компиляция кезінде атқару барысында қандай нысанның жасалатыны белгісіз. Әрі рекурсивті мәліметтер және динамикалық нысандар құрылымдары жасалған процедураның уақыты біткен кезде, оның да мерзімі аяқталады. Сол себепті *хип* деп аталатын жадының жалпы облысы пайдаланылады. Хип – үдерістер керек кезінде қосымша жады ала алатын және қажеттілік болмаған жағдайда жадыны қайтара алатын жады банкі. Логикалық мәліметтердің бір құрылымы көрсеткіштер көмегімен байлынсқан жадының әртүрлі блоктарында хиптің өн бойында таралуы мүмкін. Жадының бұл блоктары әртүрлі уақытта бағдарламаны орындау кезінде жіберілген сұранымға жауап ретінде бөлінеді. Рекурсивті мәліметер немесе динамикалық нысандар құрылымын жою хипте жадыны босатумен бірдей. Жады босағаннан кейін қоқыс жинағыш жадыны тазалайды. Енді оны басқа үдерістерде пайдалануға болады. *Бұл хиптің мәліметтер құрылымында білген сұрыпталған кескінден өзгеше* екеніне назар аударыңыз.

Хип негізінде қолдану стек негізінде қолдануға қарағанда баяулау. Өйткені, (1) нысандарды атқаруға уақыт бөлген кезде қосымша шығындар болады, (2) үймедегі мәліметтер құрылымын айналып өту үшін көрсеткіштерді артық падалану кезіндегі қосымша шығындар және (3) қайтадан пайдалану үшін жадыны пайдалану үшін қосымша шығындар болады.

Қазіргі тілдер атқару тиімділігінің максималды мүмкіндігімен қатар барлық мүмкіндіктерді беру үшін жадыны бөлудің барлық үш түрін пайдаланады. Мысалы, C++ сияқты қазіргі тілдер статистикалық таратуды қолданатын статистикалық айнымалыларды, рекурсивті үдерістерді өңдеу үшін хип негізіндегі қолдануды және жадыны қайта пайдалануды, сондай-ақ мәліметтердің динамикалық нысандары мен рекурсивті мәліметтер құрылымы үшін динамикалық жадыны пайдаланады.

1.9.3 Басқа жіктеу

Бағдарламалау тілдері олардың жүйелер түріне сәйкес жіктелуі мүмкін.

Түрлердің негізгі екі жіктелуі бар: мономорфты түр және полиморфты түр. Атаулары айтып тұрғандай, мономорфты түрдегі тілде мәліметтер құрылымының тек тір түрі ғана өңделеді. Мысалы, егер біз байланысқан тізімдегі элементтер санын есептеу үшін функция жазсақ, онда көптеген осыған ұқсас функциялар біз бүтін сандар тізімін есептейміз бе, әлде мәліметтері басқа тізімдегі құбылмалы нүктелі сандар тізімін есептейміз бе, соған байланысты мономорфты тілде жазылу керек. Соған қарамастан, байланысқан тізімдердің әр түріне бейімделетін полиморфты тілде тек бір ғана жалпы функция жазылу керек. Фортан, Алгол, Паскаль және Си сияқты тілдер-мономорфты тілдер: олар 7-тарауда сипатталғандай, мәжбүрлі және қайта жүктеу формасындағы полиморфизмнің тек шектеулі санын ғана жалғастырады. C++, Лисп, ML, Хэскелл, Java және Пролог сияқты көптеген заманауи тілдер – полиморфты тілдер.

Бағдарламалау тілдерін жіктеудің тағы бір қызық әдісі тақырыптық салаға қажетті тілдер функциясына байланысты ең жақсы жұмыс істейтін тақырыптар классы бойынша анықтау болып табылады. Тақырыптық саланың барлық талаптарына жауап беретін тіл осы домен бойынша лайықты тіл деп жіктелуі мүмкін. Мысалы, C++ үлкен бағдарламалық қамтамасыз етуді жасауға жақсы тіл болып табылады. Соған қарамастан, бұл шынайы уақытта бағдарламалау режимі үшін жақсы емес. Дәл солай, Фортан ғылыми бағдарламалау үшін жақсы тіл болып табылады.

1.10 Қысқаша қорытындылар

Бағдарламаның негізгі үш құраушысы бар: *логика*, *абстракция* және *бақылау.* Әр тіл әртүрлі тақырыптық облыс үшін жақсы. Бағдарламалаудың тілдері үшін қойылатын талаптар кейде бір-бірімен үйлеспей жатады. Бағдарламалаудың императивті бағдарламалау парадигмасы, функционалды бағдарламалау парадигмасы, логикалық бағдарламалау парадигмасы, нысанды-бағдарлы бағдарламалау парадигмасы, визуалды бағдарламалау парадигмасы, параллель бағдарламалау парадигмасы, оқиға-бағдарлы бағдарламалау парадигмасы, мульти медиалық бағдарламалау парадигмасы және веб-бағдарламалау парадигмасы сияқты әртүрлі парадигмалар әртүрлі функциялармен сипатталуы мүмкін. Бағдарламалаудың әртүрлі парадигмаларының дамуымен бірге тіл ғалымдары бағдарламалаудың әртүрлі парадигмаларының жақсы жақтарын талқылап және анықтады. Қазіргі тілдер осы қасиеттерді біріктіріп, көптеген бағдарламаларға парадигмалар қосты. Мысалы, Java тілі – императивті, нысанды-бағдарлы, оқиғалы, параллель және веб-бағдарламалау парадигмаларын біріктіреді.

Жоғары деңгейлі тілдер бағдарламалық қамтамасыз етуді жақсы түсіну, басқару және дамыту үшін мәліметтер мен басқару абстракциясын қолдану керек. Тілдер сонымен қоса бір модульдің өзгеріс тиімділігі осы модульде шектелетіндей модулдікті қабылдау керек. Біз аударудың үш түрлі механизмін талқыладық: аудармашы жасайтын аударма, компилятор қамтамасыз ететін компиляция және JIT компиляторлар қамтамасыз ететін JIT компилятор. Толықтырылған кодтар аудару үдерісі атқарудың бір бөлігі болмағандықтан, неғұрлым тиімді орындалады. Аудару және атқару атқару кезінде бір-бірімен сәйкестеніп қалатындықтан аудармашылар бағдарламаны баяу атқарады.

Веб-тілдер виртуалды машиналарды және JIT компиляцияны атқару үшін төменгі деңгейдегі кодты аударуға пайдаланылады. JIT компиляцияны атқарудың тиімділігі толықтырылған код пен түсіндірілген кодтың арасында жатады.

Бағдарламалау тілін қолдану үшін төменгі деңгейлі әртүрлі абстрактілі үлгілерді қолданады. Статистикалық қолдану жадыға тікелей рұхсатты қолданады. Алайда жадының өсуіне қолдау болмағандықтан статистикалық қолдану рекурсивті үдерістерді, жадыны қайта пайдалануды, байланысқан тізімдер мен ағаштар сияқты рекурсивті мәліметтер құрылымын, сондай-ақ мәліметтердің динамикалық нысанын өңдей алмайды. Стек негізінде қолдану рекурсивті үдерістер мен жадыны қайтадан пайдалануға мүмкіндік береді. Алайда, стек негізінде қолдану мәліметтердің рекурсивті құрылымдарын және мәліметтердің динамикалық нысандарын өңдей алмайды. Мәліметтердің рекурсивті құрылымдарын және бағдарламаның қажетілігіне байланысты бөліктерге бөлінуі мүмкін мәліметтердің динамикалық нысандарын құруға кеткен уақытты өңдеу үшін бізге барлық қолдануылған *ішкі бағдарламалармен* қатар *хипті-жадының жалпы облысын* қолдану керек.

Тілдерді (1) жіктеу негізіндегі бағдарламалау парадигмасы, (2) жіктеулер негізіндегі қолдану және (3) түр негізіндегі жіктеулер сияқты көптеген әртүрлі өлшемдерді пайдаланып жіктеуге болады. Алайда, қазіргі тілдерді жіктеу өте қиын. Қазіргі бағдарламалау тілдері белгілердің жиынтығынан пайдалы көреді және бір категорияға жіктеле алмайды.

1.11 Бағалау

1.11.1 Тұжырымдамасы мен анықтамалар

Абстракция; алгоритм; альфа-тестілеу; жинау коды; бета- тестілеу; бинарлы код; кодты оңтайландыру; құрастырушы; параллелизм; параллель бағдарламалау; шартты ауысу; бақылау; абстракцияны бақылау; ағынды басқару; басқарудың блок-сызбасы; мәліметтер абстракциясы;

мәліметтер негізінде итрерация; ресми түрде бағдарламалау парадигмасы; белгілі итерация; жойғыш жаңарту; оқиға-бағдарлы бағдарламалау парадигмасы, функционалды баламалық; функционалды бағдарламалау парадигмасы; шартсыз ауысу операторы (GOTO операторы); хип негізіндегі қолдану; императивті тіл; анықталмаған итерация; мәліметтерді жасыру; аралық код; ғаламтор-бағдарламалау парадигмасы; аудармашы; JIT компиляция; лексикалық сараптау; линкер; жүктегіш; логикалық бағдарламалау парадигмасы; модульдік; мономорфті түр; мультимедиалық бағдарламалау парадигмасы; нысанды-бағдарлы бағдарламалау парадигмасы; ортогоналдық; синтаксистік сараптау; полиморфты түр; ықшамдылық; тақырыптық облыс; бағдарламаны түсіну; оқу; бағдарламалық қамтамасыз етуді жасау кезеңі; бағдарламалық қамтамасыз етуді жасау; бағдарламалық қамтамасыз ету эволюциясы; бағдарламалық қамтамасыз етуді қызмет ету; бағдарламалық қамтамасыз етуді қайта пайдалану; стек негізінде қолдану; статистикалық қолдану; таңбасыдар кестесі; шартсыз ауысу; визуалды бағдарламалау жазба мүмкіндігі.

1.11.2 Тапсырма

1. Төменде көрсетілгендей "FOR EACH" кезеңін құру үшін "FOR" және "WHILE" кезеңдерін пайдаланып бағдарлама жазыңдар. Мұндағы "a" мәліметтер құрылымының жиынтығы және "size(a)" функциясы элементтер санын "a"-ға қайтарады, ал келесі (a) "a" мәліметтер құрылымының келесі элементін қайтарады.

```
foreach x in a {
y = f(x); print (x, y);
}
```

2. Келесі код үшін басқару блок-сызбасын жазыңдар.

```
i = 0; j = 10;
for (k = 0; k = < 9; k++) a[k] = 0;
while (i < 5) { j = j – 1;
for (k = 0; k <  = j; k++) {
a[k] = a[k] + 7;
if (a[k] mod 2 == 0} print (‘жұп сан’);} i = i + 1
                                                    }
```

1.11.3 Толық жауап

3. Бағдарламадағы негізгі үш құраушыны түсіндіріңдер: логика, бақылау және абстракция.

4. Мәліметтерді бақылаудың және абстракцияның бағдарламалау тіл-

деріндегі артықшылығын түсіндіріңдер.
5. Бағдаламалаудың жақсы тілдерінің әртүрлі белгілерін салыстырыңдар және сәйкестендіріңдер.
6. Бағдарламалаудың жақсы тілі үшін әртүрлі тақырыптық облыстар мен қарама-қарсы белгілерінен мысалдар келтіре отырып бағдарламалаудың әмбебап жақсы тілінің не үшін болмайтынын түсіндіріңдер.
7. "GOTO" бағдарламалау түсінігімен қалай байланысты? Суретті пайдалана отырып түсіндіріңдер.
8. Бағдарламалық қамтамасыз етуінің даму кезеңін және оның бағдарламалау тілдерінің дамуына тиімділігін түсіндіріңдер.
9. Бағдарламалаудың әртүрлі парадигмаларының сипаттарын түсіндіріңдер.
10. Бағдарламалау тілін қолдану кезінде жадыны таратудың қандай әртүрлі категориялары бар? Әр категорияның артықшылығы мен кемшілігін түсіндіріңдер.
11. ЖТ компиляция дегенде нені түсінесіңдер? Оның дәстүрлі интерпретация мен компиляциядан қандай айырмашылығы бар? Суретті пайдалана отырып, түсіндіріңдер.
12. Веб-тілдер не үшін ЖТ-компиляцияны пайдаланады? Түсіндіріңдер.
13. Бағдарламалау тілдерін категориялаудың қандай сызбалары бар? Әр жіктелуді қысқаша сипаттаңдар.
14. Атқарылатын командалардың блок-сызбасы деген не? Осындай блок-сызбалардың негізгі құраушыларын түсіндіріңдер.

2. Алғысөз және Негізгі түсініктер

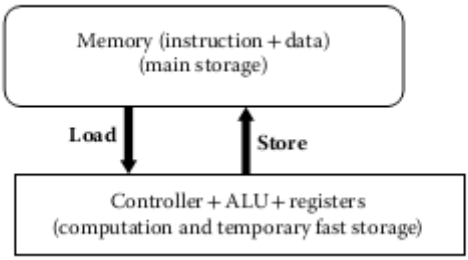
Негізгі тұжырымдамалар
Бағдарламалау бойынша ерте білімдер; мәліметтер құрылымдар курсы; дискретті құрылымдар курсы
Бұл тарауда компьютерлік құрылымның, математикалық түсініктердің тұжырымдамалары, мәліметтер құрылымы тұжырымдамалары, сондай-ақ абстракция түсінігі және бағдарламалау тілдерін қолдануға қажетті абстрактылы есептеу тұжырымдары сипатталады. Бұл түсініктер белгілі бір ретпен оқытылуы міндетті емес. Мұны әртүрлі тараулардағы анықтамалық материалдар ретінде қажетті кезде үйренуге болады.

2.1. Фон нейман машинасы

Қазіргі компьютерлер фон Нейман машинасы деп аталатын төменгі деңгейлі абстрактылы машинаға негізделген. Бұл Джон фон Нейман бағдарламаны сақтайтын компьютерлік үлгі ұсынған сәттен басталды. Фон Нейман машинасын түсіну біз үшін өте маңызды. Өйткені, төменгі деңгейлі машина нұсқаулықтары фон Нейман машинасында атқару үшін жасалған. Бағдарламалық аудармашы жоғары деңгейлі құрылымдарды төменгі деңгейлі ұқсас нұсқаулықтарға аударады. Шартты операторлар, "WHILE DO" және"FOR" кезеңдері сияқты жоғары деңгейлі құрылымдар фон Нейманның машинасында төменгі деңгейлі командалар ретімен аударылады. Фон Нейманның машинасын және төменгі деңгейлі командалар жинағын қолдану 3-тарауда жоғары деңгейлі құрылымның мәнін төменгі деңгейдің абстрактылы нұсқаулық тұрғысында талқылап және 5-тарауда абстракция басқаруын төменгі деңгейлі абстарктылы нұсқаулық ретіне аударады талқылаған кезде түсінікті болады

Фон Нейман машинасының негізгі екі құраушысы бар: (1) бағдарлама мен мәліметтерді сақтайтын жады және (2) нұсқаулықтарға негіздел-

ген мәліметтерді өңдейтін және нұсқаулықтарды шығаратын орталық процессор. Нұсқаулықтар шиналарды – жады мен орталық процессор арасындағы жоғары жылдамдықты байланысты қосу арқылы орталық процессордың бірінен шығады. Процессор сол шинаны пайдаланып команданы түсіндіреді және жадыдан мәліметтерді жүктейді.



Memory (instruction + data) (main storage) - Жад (нұсқаулық + деректер) (негізгі сақтау); Load – Жүктеу; Controller + ALU + registers (computation and temporary fast storage) - Контроллер + ALU + регистрлер (есептеу және уақытша жылдам сақтау); Store – Сақтау
2.1-сурет. Фон Нейман машинасы

Мәліметтер орталық процессор шегіндегі арифметика-логикалық құрылғыда (АЛҚ) өңделіп, нәтижесінде алынған мәліметтер 2.1-суретте көрсетілгендей жадыда қайтадан сақталады.Әдетте мәліметтердің пайдаланылған элементтері немесе олардың сілтемелері тез қолжетімді болу үшін құрылымдық тіркелмелерінде сақталады. Фон Нейман машинасында келесі орындалатын нұсқаулықтың жады адресі бар бағдарламалық есептегішке ие Командалар есептеуіші ағымдағы нұсқаулықты алғаннан кейін бір бірлікке артады. Соған қарамастан, оның мәні шартты және шартсыз ауысу операторлары арқылы өзгеруі мүмкін.
Нұсқаулықтар жадыдан жүктеу (жүктеу); жадыда сақтау (сақтауыш); қосу, алу, көбейту және бөлу арифметикалық есептеулерді атқару; "AND" логикалық операторы, "OR" логикалық операторы, "OR"шығарғыш операторы және терістеу сияқты логикалық операторлар; екі мәнді салыстыру; бағдарлама есептеуішінің мәнін өзгерту және жүйелік деңгейдің әртүрлі жалаулар күйін тексеру арқылы басқа сыбайлас емес нұсқаулықтарға шартты немесе шартсыз өту сияқты жіктелуі мүмкін. Әртүрлі жалаулар *бағдарлама күйі сөзі* (PSW) деп аталатын тіркеуіште сақталады. Нұсқаулықтың осы категориясына қосымша нұсқаулықтар жиын-

тығы компьютер құрылысына байланысты операндыны түзету үшін мекендеудің әртүрлі механизмдерін қолданады.

2.1.1 Адрес механизмдері

Фон Нейман машинасында мәліметтер немесе жады ұяшығының адреслары тіркеуіштерде(RAM) немесе жадыда уақытша сақталуы мүмкін. Осыған ұқсас жады облысында мәліметтер де, сонымен қатар жадыда басқа орынның адресі да болуы мүмкін. Егер жадыда мәліметтер болса оған бір реттік рұхсатты пайдаланып арнайы нұсқаулықпен кіруді *тікелей кіру* деп атайды. Алайда, егер жады ұяшығында жадының басқа ұяшығының адресі болса, онда мәліметтерді орталық процессорға мәліметтерді жүктеу үшін жадыны екі рет пайдалану керек. Бұл әдіс *жанама* кіру деп аталады. Тікелей және жанама кіруден бөлек жылжыту тіркеуіште сақталған адрес немесе жаңа адресті есептеуге арналған басқа адрес бойынша қосылуы немесе алып тасталынуы мүмкін. Жылжыту негізіндегі бұл әдіс мәліметтер құрылымының бірінші жады ұяшығының негізгі мекен-жайын сақтап және ішкі массивтің жылжытуын негізгі адреске қосу әдісімен ішкі массивтердің адресін есептеу қажет болатын мәліметтердің күрделі құрылымының ішкі массивтеріне кіру үшін қолданылады.

Компьютерде компютер құрылысыны байланысты 0-адрес, 1-адрес, 2-адрес немесе 3-адрес болуы мүмкін. 2.1-кестеде келтірілгендей жады немесе тіркеуіштер ұяшықтарын дербестендірген кезде адреслар саны төменгі деңгейлі машинада жинау деңгейінде нұсқаулықтар аргументтерінің жиынтығының максималды санымен беріледі.

2.1- кесте. Адрес механизмдерінің әртүрлі типтері

Нұсқаулық типі	**Операциялар типі**
Үш адреслы машиналық командалар: *<instruction-name> <src1>*, *<src2>*, *<dst>*	Үш адреслы машинадағы нұсқаулық. Нұсқаулық кез келген арифметикалық немесе логикалық диадалық операция болуы мүмкін.
Екі адреслы маиналық командалар: *<instruction-name> <src1>*, *<src2>*	Екі адреслы машинадағы нұсқаулық Нұсқаулық кез келген арифметикалық немесе логикалық диадалық операция болуы мүмкін. Белгілеу екінші аргументтегідей болады.
Бір адреслы командалар: *<instruction-name> <src>*	Нұсқаулық кез келген арифметикалық немесе логикалық диадалық операция болуы мүмкін. Тіркеуіштердің бірі белгілеу тармағы ретінде әрекет ететін үнсіз келісіммен *аккумулятор* болады.

Нөл адресты машиналық командалар: <*instruction-name*>	Жүктеме, сақтауыш, қосу, алу, көбейту, бөлу және т.с.с. Стек негізінде бағалау қолданылады. Операцияға керек аргументтер стектың жоғарғы бөлігінен алынады.

Диадалық операциялар бар өрнек үшін екі кіріс аргумент және бір шығыс аргумент бар. Мысалы, *add_integer* механизмі үшін үш адрес керек: екеуі кіріс аргументтер үшін және біреуі шығыс аргументтер үшін. Кіріс мағынасы бар аргументтер *көз д*еп, ал шығыс мағынасы бар аргументтер *белгілеу* деп аталады.

Көтеген заманауи процессорларда үш адресты құрылым жинағы бар. Соған қарамастан, көптеген адреслар сөздің 16-биттік өлшеміне сыймай жатады. *Екі адресты машиналардың аккумулятор* деп аталатын арнайы тіркеуіштері бар. Оны әрқашан белгілеу пункті ретінде пайдаланады және механизмді дербестендіруге керекті битер санын сақтау үшін командалар аргументтерінің бірі ретінде пайдаланбайды. Оны шығыс тіркеуіштердің бірі ретінде де пайдалануға болады. Битер санын сақтау үшін аккумуляторлар нұсқаулықта жазылмайды. Аккумулыторды көз ретінде және нұсқаулық бойымен әрі қарай қысқартуды белгілеу үшін анық пайдаланады.

Нөлдік адресты машиналадың қандай да бір нақты аргументтері жоқ және аргументтерді сақтау үшін орнатылған стек қолданылады. Нөлдік адресты машиналардың нұсқаулығы бұл: жүктемелер, сақтауыш, *push, pop*, қосу, алу операциялары және т.с.с. Мысалы, қосу командасы берілген кезде мәліметтердің екі жоғарғы элементтері стектан АЛҚ-да есептелу үшін *шығарылады,* бағаланады және нәтижесі есептеу стектің жоғарғы жағына қайтадан *лақтырылады*. *Жүктеу және сақтауыш* нұсқаулықтары мәліметтерді жергілікті айнымалылардан есептеу стектің жоғарғы жағына шығарады немесе есептеу стектің жоғарғы бөлігіндегі мәліметтерді жергілікті айнымалыларға сақтайды. Нөлдік адресті машиналар одан да жоғары адресті машиналардың көмегімен қосылуы да мүмкін. Жоғары адресті машиналардағы нұсқаулықтар төменгі адрестердің екі немесе одан да жоғары командаларына ауыстырылуы мүмкін. Нөлдік адресті машиналар Java (JVM) виртуалды машинасында Java бағдарламасын орындау үшін пайдаланылған.

2.1 Мысал

Бұл мысал $X = A + B$ өрнегінің ауыстырылуын көрсетеді. Мұндағы, *X, A*, және *B* - бүтін айнымалылар. Бұл айнымалылар жады ұяшығында көрінеді деп есептеледі.

Пайыз белгісі "%" шолуды бастайды. Үш адреслы машинадағы нұсқаулық келесі түрде болады:

```
integer_add A, B, X % Add data loaded from memory locations A
% and B, and store back in X.
```

Екі адреслы машиналар үш адреске ие юлда алмағандықтан, 3 адреслы машиналардың нұсқаулықтары 2-адреслы нұсқаулықтың командаларының ретіне бөлінген. Екінші аргумент те белгілеу пункті ретінде пайдаланылады. 2-адреслы машинадағы төменгі деңгейдің сәйкес нұсқаулығы келесі түрде болады:

```
load A, R0 % load the content of memory location A into the
% register R0
integer_add B, R0 % add the contents of the memory location B
% and the register R0
store R0, X % store the content of R0 into the memory location X
```

Бір адресті машиналар аккумуляторды кіріс аргументтері мен белгілеу пунктінің бірі ретінде айқын емес пайдаланады. Төменгі деңгейлі бір адресті нұсқаулықтардың сәйкес жиынтығы келесі түрде болады:

```
load A % load the content of memory location A into the
% accumulator
integer_add B % add the contents of the memory location B with
% the content of the accumulator
store X % store the content of accumulator in the memory
% location X
```

Нөлдік адресті машина (Java виртуалды машинасының командалар жинағында көрсетілгендей) A және B жады ұяшықтарын құрылған стекқа жүктейді, стектың екі мәнін қосады және сумманы стек басына қалдырып, содан кейін стектың жоғарғы бөлігіндегі X жады ұяшығына сақтайды. Жүктеу операциясы дегеніміз екі операцияның ретінен тұрады: әуелі ол жады ұяшығының адресін "load_literal» нұсқаулығы арқылы стектың басына жүктеп, содан кейін жады ұяшығының мәнін жүктеу командасының көмегімен жүктейді.

```
load_literal 3 % Push the address of X on top of the
% evaluation stack.
load_literal 1 % Push the address of A on top of the
```

```
% evaluation stack.
load % Pop the address of A and push the content of A on top
% of the evaluation stack.
load_literal 2 % Push the address of B on top of the evaluation
% stack.
load % Pop the address of B and push the content of B on top
% of the evaluation stack.
integer_add % Pop top two elements from the stack, add them,
% and push the result on the stack.
store % Pop the result, pop the address X, and store result
% into the memory location X.
```

2.1 мысалда көрсетілгендей дәл сол тапсырманы орындау үшін төменгі адрес нұсқаулығын қажетті көбірек нұсқаулық орнату керек. Өйткені әрбір команда жады таңдауды білдіреді. Төменгі адресті машиналарға жоғарғы адреслы машиналарда таңдау жасаған кезде көбірек жады керек болады. Осылайша, нөлдік адресті виртуалды машиналар жоғары адресті машиналардағы бастапқы құрастырылған кодқа қарағанда баяуырақ жұмыс істейді.

2.2 Үзілісті құрылымдардың тұжырымдамасы

Бұл бөлімде біз мәліметтердің абстрактылы көріністеріне қатысы бар дискретті құрылымдардың таңдап алынған тұжырымдамаларын жаңартамыз. Олардың өзектілігі келесі тарауларда бағдарламалау тілдерінің тұжырымдамаларын талқылауды жалғастырған кезде анық болады.

2.2.1 Жиындармен операциялар

Бағдарламалау тілдеріндегі жиын және жиындармен операциялар түсінігі әртүрлі деңгейде қолданылады. Жиын (1) 7-тарауда айтылғандай типтер теориясындағы типтерді анықтау үшін; (2) Паскаль, Сетл (SETL), Модула, Клэр, Руби және Питон сияқты тілдердегі жиын бағдарламалау үшін; және (3) Java, C ++ және Пролог сияқты көптеген кең тараған тілдерде жиын бағдарламалау үшін кітапханаларда қолданылады.

Жиын дегенміз мәліметтердің бірегей элементтерінің жиынтығы. Жиындағы элементтердің орнын ауыстырғанмен жиынның өзі өзгермейді. *Мультижиын* – мәліметтердің мағынасы бір бірнеше элементтері бар нысандар жиынтығы. Мысалы, {*1, 2, 3, 2, 4, 5, 7*} 2 элементі үш рет қайталанатын мультижиын. Соған қарамастан, мультижиындағы элементтердің орнын ауыстырғанмен, мультижиын өзгеріссіз қалды.

Мультижиынның қызықты ішкі классы бар. Ол – *реттелген мульт-*

жиын. Реттелген мультижиынның әрбір элементі сәйкес позицияға байланысты. {*Миира, Джон, Ли*} реттелген мультжиында *Миира* 1-позициямен байланысты, *Джон* 2-позициямен байланысты, ал *Ли* 3-позициямен байланысты. Реттелген мультжиында элементтердің орнын ауыстырған кезде мультижиынның реті өзгереді. Өйткені сәйкес позициясы да өзгереді. Мысалы, {*Меера, Джон, Ли*} реттелген мультжиынындағы {*Джон, Меера*, *Ли*} мультжиынының дәл өзі емес. Өйткені, *Меера* бірінші реттелген мультжиында 1-позициямен, ал екінші мультжиында 2-позициямен байланысып тұр. Реттелген мультжиын бағдарламалау тілдерінің көптеген құрылымдарында үлкен рөл атқарады. *Тізбектер, қатарлар, файлдар* және басқа да құрылымдар 7-тарауда айтылғандай реттелген мультижиын сияқты моделденеді.

Ол ішкі жиындармен, жиындардың бірігуімен және қиылысуымен жақсы қарым-қатынаста. Осы жиынның барлық мүмкін болатын ішкі жиындарындағы жиындар берілген жиынның *көрсеткіш жиыны* деп аталады. Мысалы, {*X, Y, Z*} жиынының көрсеткіш жиыны былай беріледі {{} (бос жиын), {*X*}, {*Y*}, {*Z*}, {*X, Y*}, {*Y, Z*}, {*X, Z*}, or {*X, Y, Z*}}. Ішкі жиын алу үшін элемент не таңдап алынуы мүмкін, не барлық мүмкіндіктерді беріп еленбей қалуы мүмкін. Осылайша, берілген жиынның көрсеткіш жиынтығы 2*N*, мұндағы *N* соңғы жиынтағы элементтер саны.

2.2.1.1 Декарттың шығармасы

Декарттық көбейтінді дегеніміз ол екі немесе одан да көп жиын және жаңа *N*-қосалқы жиын алатын жиын. Мұндағы, *N* – декарттық көбейтіндідегі жиын саны. A1 × A2 ×, …, ×A*N* (*N* ≥ 2) декарттық көбейтіндіні есепке ала отырып, *N*-қосалқы жиынның бірінші қатары A1 және *Ith* жиынының элементі болады, (*I* ≤ *N*) *N*-қосалқы жиын *Ith* элементі болады. Алынған жиындағы элементтер саны былай беріледі: |A1| × |A2| × |A3| × |A4| … |A*I*| × … × |A*N*| , мұндағы |Ai| *Ith* жиындағы элементтер саны.

2.2 Мысал

Үш жиынды қарастырайық: A = {*a, b, c*}, B = {*x, y, z*}, және C = {*1, 2, 3*}. A × B × C декарттық көбейтіндінің 3 × 3 × 3 = 27 үш мүмкіндігі бар: {(*a, x, 1*), (*a, x, 2*), …, (*c, z, 1*), (*c, z, 2*), (*c, z, 3*)}.

2.2.1.2 Бейнелеу

Бейнелеу дегеніміз жиынның бір жиынтығының элементтерін басқа элементтермен байланыстыру, "one-to-one", немесе "many-to-one" байланыстары арқылы байланыстыру. 2.2 суретте D домендері мен R мәндерінің

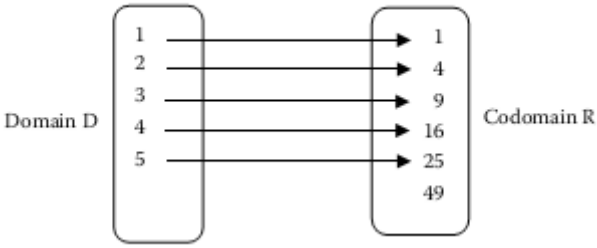
облыстарының сәйкестігі келтірілген. D жиынында бес элемент және R жиынында алты элемент бар. Егер біз бейнелеуге «.D-дағы элементтер квадраты» деп анықтама берсек, онда D доменнің әрбір элементі R элементінде бейнеленеді. D облысындағы біреуден артық элемент R-дегі тура сол элементті бейнелейді. Алайда, D облысындағы бір элемент R мәні облысында бір элементтен артық бейнелей алмайды.

2.2.1.3 Изоморфизм

S1 мен S2 :арасында бейнелеу болатындай F және G биективті функциялар жұбы болса, онда S1 және S2 екі жиыны *изоморфты*. F функциясы S1 және S2 элементтерін "one-to-one" байланыс арқылы, G функциясы S2 және S1 элементтерін "one-to-one" байланысы арқылы бейнелейді. Ал G функциясы "F" функциясының кері функциясы болып табылады.

2.3 Мысал

Мысалы, S1 = {*1, 2, 3*} и S2 = {*4, 5, 6*} екі жиынын алайық. S1 жиынының барлық элементтерін "one-to-one" байланысы арқылы S1 облысына 3 элемент қосу арқылы S2 жиынының барлық элементтерін бейнелейтін *add_3* функциясы бар.



Domain D - Домен D; Codomain R – Кодомен R.

2.2-сурет. Квадраттық функцияның көмегімен бейнелеу.

Мұның сәйкес кері функциясы *subtract*_3, ол S2 әрбір элементін S1 жиынының элементінде бір-бірлеп алу әдісінің негізінде бейнелейді. Екі функция да add_3 және subtract_3 - бір-біріне кері функциялар.

2.2.1.4 Функциялар

F функциясы D облысындағы кез-келген x ∈ элементтің C облысындағы F(x) ∈ бейнесінде бейнеленуі. F(x ∈ D) ⊆ C бейнесінің жиыны

функция *диапазоны* деп аталады. *Тең функция* элементті өзі бейнелесе, *тұрақты функция* жиынның әрбір элементін тіркелген бірегей элементке бейнелейді. Функция *"one-to-one"* немесе *инъективті деп аталады*. Егер домендегі барлық элементтер үшін мәні облысында айтарлықтай бейне болса, яғни егер y1 = F (x1∈ D) и y2= F(x2∈ D), және x1 ≠ x2 и y1, y2 ∈ C онда y1 ≠ y2. Егер y ∈ C кез келген элементі үшін f(x) = y функциясы үшін x ∈ D элементі болса онда бұл функция *бейнелеу немесе сюръективті функция* деп аталады. Егер функция бейнелеу және "one-to-one" болса, онда ол *биективті* функция деп аталады. Бұл дегеніміз, мән облысындағы элемент үшін доменде элементтің бірегей бейнесі бар. F биективті функциясын ескере отырып,
F-1 кері функциясы мән облысында облыстағы F-1 (F (x ∈ D)) = x болып табылатын элементке сәйкес келетін бірегей бейнені бейнелейтін функция ретінде анықталуы мүмкін.
Бағдарламалауда сюръективті функциялар өте маңызды. Өйткені анықталған функцияның болмауы қате шартқа пара-пар. Сюръективті емес функция диапазонда ⊥ белгісімен белгіленген *нөлдік таңбасы* енгізу арқылы сюръективті болады. Нөлдік таңбасы мән облысында кез келген нөлдік емес таңбасымен сәйкес келмейтін облыстағы барлық таңбасыдардың бейнесі болып табылады. Нөлдік таңбасы идеясы 3-тараудағы бағдарламалау тілінің денотациялық семантикасын және 9-тараудағы функционалды бағдарламалауды түсіндіру үшін пайдаланылған.
Функцияның бағдарламалық көрінісі үш бөлімнен тұрады: *айнымалы, өрнек және кіріс параметрлер.* Егер кіріс мәндер берілсе, онда кіріс мәндер айнымалымен байланысты болады. Айнымалысы бар өрнекте алмастыру орын алады. Өрнек ықшамдалғаннан кейін шығыс мәндеп қайтып келеді. Функцияны бірнеше бағдарламалық модулдегі әртүрлі орыннан шақыру үшін функцияға ат беру мүмкін.

2.4 Мысал

Мәліметтер құрылымы мен бағдарламалау курсында оқыған белгілі *факториал(n)* функциясын қарастырайық. Функцияның берілген параметрге байланған *n* айнымалысы бар. Функция денесі анықтаманың оң жақ бөлігінде берілген.

```
factorial(n) =
if (n == 0) return(1) % base case
else return(n *   factorial(n – 1))% recursive
% definition
```

Факториала (n) функциясының денесі екі өрнектен тұрады: "*if (n == 0) then return (1); if (n > 0) return (n * factorial(n – 1))*." Екі өрнек те (if-then-else) абстракция шартты операторымен байланысқан. Ал (*n*> 0) тест дегеніміз шартты оператор негізінде алынады.
Біз ф*акториалды* (3) шақырған кезде 3 *n* айнымалымен байланыста. Содан кейін, ол функция денесінің өн бойында ауысады да, функцияның өзі "*if (3 == 0*), *then return 1 else return (3 * factorial(3 - 1))*." көшіріледі. Шарт бағаланады, ал функция қосымша *return(3 * factorial(3 – 1))* үшін ықшамдалады. (*3 – 1*) ықшамдалады, ал функция *return(3 * factorial(2))* қосымша ықшамдалады. Қазіргі уақытта *function(3)* –ті бағалау *factorial(2)* есептелмейінше тоқтатылады.

2.2.2 Буль логикасы және предикаттарды есептеу.
Буль логикасы кейбір пайымдаулардың ақиқат мағынасын байланыстыруға негізделген. Аксиома не «ақиқат» (true) не «жалған» (false) болатын пайымдау немесе болжам болады. Ақиқаттылықтың бұл мәндері logical-AND ("∧" белгісімен белгіленеді), logical-OR ("∨" белгісімен белгіленеді), жанама ("→" белгісімен белгіленеді) және терістеу ("¬" белгісімен белгіленеді) сияқты әртүрлі логикалық операторларды қолданумен құрастырылған болуы мүмкін. logical-AND және logical-OR операторлары А және В екі логикалық аксиоманы біріктіреді. 2.2-кестеде көрсетілгендей А және В ақиқат болу үшін А ∧ В ақиқат болу керек және екеуі де А және В жалған болу үшін А ∨ В жалған болуы керек. логикалық операцияның қорытындысы А → В егер А ақиқат болса, онда В да ақиқат дегенді білдіреді. Егер А жалған болса В-ның ақиқат екенін анықтау мүмкін емес. Осылайша, егер А мәні жалған болса, онда В ақиқат та, жалған да болады. Терістеу егер А аксиомасы ақиқат болса, онда А жалған, ал егер А жалған болса, онда А ақиқат дегенді білдіреді.

2.5 мысал
А аксиомасына «адамдар жаңашыл», ал В аксиомасына «адамдар тәкаппар» дегенді алайық. А аксиомасы және В аксиомасы ақиқат деп есептейік: онда А ∧ В да ақиқат. Осылайша «Адамдар жаңашыл және тәкаппар» болып шығады. Егер біз logical-OR операторын қарастырсақ, және егер А аксиомасы ақиқат, немесе В аксиомасы ақиқат болса немесе екеуінің біреуі ақиқат болса, онда «Адамдар жаңашыл немесе тәкаппар болады» аксиомасы ақиқат болады.
Logical-AND және logical-OR операторлары 2.3-кестеде көрсетілгендей бір-бірімен байланысты. Бұл тұжырым бағдарламалар жасау кезінде және бағдарламаларды параллель орындау кезінде кеңінен қолданыла-

ды. Бір-біріне ұқсас бульдік операциялардың көптеген амалдары бар. Мысалы, Де Морган теоремасы терістеу операциясы арқылы logical-AND және logical-OR операцияларына жатады. Де Моргана заңы: (1) конъюкцияны терістеу дизъюнкцияны терістеу сияқты және (2) дизъюнкцияны терістеу конъюнкцияны терістеу болып табылады

2.2-кесте. Бағдарламалау тілдерінде пайдаланылатын логикалық операторлар үшін ақиқат кестесі.

A	**B**	**¬(A)**	**A ∧ B**	**A ∨ B**	**A → B**
Ақиқат	Жалған	Жалған	Жалған	Ақиқат	Жалған
Жалған	Жалған	Ақиқат	Жалған	Жалған	Ақиқат
Ақиқат	Ақиқат	Жалған	Ақиқат	Ақиқат	Ақиқат
Жалған	Ақиқат	Ақиқат	Жалған	Ақиқат	Ақиқат

2.3-кесте. Бульдік операциялардағы эквиваленттілігі

Операция типі	**Эквиваленттілік**
Терістеу	$\neg(\neg P_1) \equiv P_1$
Ассоциативтілік	$P_1 \wedge (P_2 \wedge P_3) \equiv (P_1 \wedge P_2) \wedge P_3$ $P_1 \vee (P_2 \vee P_3) \equiv (P_1 \vee P_2) \vee P_3$
Коммутативтік	$P_1 \wedge P_2 \equiv P_2 \wedge P_1$ $P_1 \vee P_2 \equiv P_2 \vee P_1$
Дистрибуттік	$P_1 \wedge (P_2 \vee P_3) \equiv (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$ $P_1 \vee (P_2 \wedge P_3) \equiv (P_1 \vee P_2) \wedge (P_1 \vee P_3)$
де Морган ережесі	$\neg(P_1 \wedge P_2) \equiv (\neg P_1) \vee (\neg P_2)$ $\neg (P_1 \vee P_2) \equiv (\neg P_1) \wedge (\neg P_2)$

2.2.2.1 Бірінші қатардың предиктін есептеу

Күрделі пікірлерді зерттейтін а*йтылымдық логика* қарапайым пікірлерден логикалық оператордың көмегімен пайда болған. Егер біз айтылымдық логиканы кванторлардың екі типімен – жалпылық кванторы (∀ белгісімен белгіленеді) және болу кванторымен (∃ белгісімен белгіленеді) толықтырсақ – онда ондай логика «бірінші қатардың логикасы» деп аталады. Жалпылық кванторы бұл барлық белгіленген элементтер үшін дұрыс шарт. Әр бір ерекшелікті жиынның әрбір элементімен байланыстырғаннан кейін егер нысан ерекшелік байланысқан жиынның бір бөлшегі болса, онда нысан да қасиетпен байланысатын болады. Мысалы, егер «Ер адамдар ұзақ өмір сүргенді қалайды, ал Джон ер адам» десек, бұл жерден шығаратын жалғыз қорытынды «Джон ұзақ өмір сүруді қалайды» болмақ. Енді бұл пайымдауды бірінші қатардың логикасының ережесімен (FOPC). сәйкестендіріп көрейік. Бірінші қатардағы логика ережесі бойынша, ∀x (*ер адам*(x) → *ұзақ өмір сүргенді қалайды*(x)).

Егер біз осы ережені оқысақ, онда ол жерде барлық "x" үшін, егер "x" ер адам, онда "x" ұзақ өмір сүргенді қалайды дегенді білдіреді. Джон-ер адам. Импликация ережесі қолданып, Джон ұзақ өмір сүруді қалайды. Болу кванторы ерекше қасиеттерін қанағаттандыратын элементті сұрап тұр. Мысалы, ∀x ∀y ережесі (*бірдеңгейлі элемент*(*x, y*) → ∃z (*parent*(*x, z*), *parent*(*y, z*), *not* (*x == y*) барлық *x* және барлық *y* үшін *x y-тың* бірдеңгейлі элементі болып табылады, егер осындай *z* болса, онда *z X* – тың аталығы болады, ал *Z y-тің* аталығы болады, *x y-ке* тең болмайды деп пайымдайды. *z* айнмалысы - *x* және *y* екеуі үшін де аталық болып табылатын жиындағы элемент үшін болу кванторы бо-лып табылады. 2.4-кестеде көрсетілгендей екі кванторға қатысты көптеген балама қаси-еттер бар. Болу кванторы да, жалпылама коммутативті де. Алайда, жал-пылама кванторы болу кванторымен араласқан кезде онда жағдай күр-деленеді. Мысалы, егер ерекшелік жиынның барлық элементтері үшін дұрыс болса (∀*x P*(*x*), онда бұл бұл қасиет ақиқат болмайтын (¬∃x ¬P(x)) элемент бар дейтін жағдай емес.

2.4 кесте. Квантордағы эквивалеттілік

Операциялар	**эквиваленттілік**
Коммутативтік	∀x ∀y P(x, y) ≡ ∀y ∀x P(x, y)
∃x ∃y P(x, y) ≡ ∃y ∃x P(x, y)	
Екі жақтылық	∀x P(x) ≡ ¬ ∃x (¬ P(x))
∃x P(x) ≡ ¬∀x (¬ P(x))	

Дәл осылай (∃x P (x)) ¬ ∀x (¬ P (x))-ға эквивалентті. Егер біз жалпылама кванторды болу кванторымен араластырсақ, онда коммутация заңы қол-данылмайды: ∀x ∃y мен ∃y ∀x эквивалентті емес. «Әрбір X адам үшін, X жақсы көретін Y адам бар» деген мәлімет (∀x ∃y жақсы көреді (x, y)) «Әрбір X адам жақсы көретін Y адам бар» (∃y ∀x жақсы көреді x, y)) деген мәліметпен бірдей емес.

Бірінші қатардың логикасы 10-тарауда сипатталған логикалық бағдарла-малау парадигмасы негізінде жатыр. Бірінші қатардың логикасының бір шектеуі, ол қарым-қатынасқа байланысты қатынасты көрсете алмайды. Жоғарғы қатарды есептеу қарым-қатынасқа қатысты қатынасты көрсе-туі мүмкін. Соған қарамастан, бұл тақырыпты талқылау кітаптан тыс.

2.2.2.2 Қатынастар

Функциялар сияқты қатынас та бағдарламалауда және бағдарламалауға қажетті типтер көрінісінде кеңінен қолданылады. Бұл бөлімде біз қаты-настың негізгі қасиеттерін қарастырамыз.

Бинарлық қатынас R екі жиынның декарттық көбейтіндінің жеке ішкі жиыны болып табылады: S1 домен мен S2 облысы. Математикалық тілмен айтсақ, $R \subseteq S1 \times S2$ қатынас және әрбір $x \in S1$ элемент $y \in S2$ элементімен R қатынас арқылы байланыста. Бұл реттелген $(x, y) \in R$ жұп үшін. Екі субъект арасындағы байланыс xRy немесе R (x, y) түрінде көрініс табуы мүмкін.

Қатынас *рефлексивті, симметриялы, антисимметриялы* немесе *транзитивитті* болу мүмкін. Егер xRx доменнің әрбір элементі үшін дұрыс болса R қатынасы рефлексивті. Егер әрбір$(x, y) \in R$ реттелген жұп үшін басқа $(y, x) \in R$ реттелген жұп болса, онда R *симметриялы*. Басқаша айтқанда, xRy yRx-ге эквивалентті. Егер xRy ешқашан x-пен yRx қатынас арқылы байланыспаған болса, онда қатынас *антисимметриялы*. Егер xRy және yRz реттелген $(x, z) \in R$ жұп бар десе, онда бұл қатынас *транзитивті*. Қатынас түсінігі бағдарламалаудың негізгі қасиеттерін дамытуда кеңінен қолданылады. R қатынасы егер R рефлексивті, симметриялы және транзитивті болса, онда *эквиваленттілік қатынасы* болады.

2.6-мысал

Мысалы, «көбірек» қатынасы (немесе «азырақ») *антисимметриялы және транзитивті*. Дәл осылай «теңдік» қатынасы рефлексивті, симметриялы жіне транзитті болып келеді. x айнымалының мәні (xRx)-ке тең; егер x айнымалының мәні y айнымалының мәніне тең болса, онда y айнымалының мәні x айнымалының мәніне тең (xRy дегеніміз yRx); егер x айнымалының мәні y айнымалының мәніне тең болса, ал y айнымалының мәні z айнмалының мәніне тең болса, онда x айнымалының мәні z айнымалының мәніне тең (x == y және y == z дегеніміз x == z).

Рефлексивтіліктің, симметрияның және Транзитивтіліктің қасиеті сандар қатарын сұрыптау сияқты қарапайым бағдарламалардағы көптеген операторларда болымсыз қолданылған. Транзитивтілік қасиеті 8-тарауда мәліметтер тәуелділігінің графигінде және айнымалылар псевдонимдерінің эквиваленттілігін сараптау кезінде қолданылады.

2.2.3 Рекурсия

Рекурсия – функцияның өзінен тікелей немесе басқа функциялар арқылы функцияны (үдерісті) шақыру. Рекурсия факториал және Фибоначчи сияқты рекурсивті функцияларды анықтау үшін немесе байланысқан тізім немесе ағаш сияқты мәліметтердің рекурсивті құрылымы сияқты рекурсивті функцияларды анықтау үшін пайдаланылуы мүмкін.

Рекурситі анықтаманың кем дегенде бір базалық жағдайы және кем дегенде бір рекурсивті анықтамасы бар. Рекурсивті анықтама біртіндеп

жайылып, негізгі нұсқаға(ларға) жақындайды. Жайылудың соңында рекурсия негізгі жағдаймен аяқталып, есептеу нәтижесі шақырту жіберген рекурсивті функцияға кері ретпен кері қайтады. Рекурсивті функцияларды шақырту саны кіріс дабылдардың белгісімен анықталады. Алдыңғы шақырту келесі жайылумен шақырылған шақырту бағаланғанша тоқтай тұрады. Шақырылған функцияның тізбегінде есептеу жүргізу үшін қажетті жады ұяшықтарын сақтау үшін стекты үздіксіз пайдалануды қажет ететін тоқтаған үдерісті сақтау керек.

2.7-мысал
factorial(0) = 1. % base case
factorial(n) = n * factorial(n – 1) % recursive definition

Факториал(*n*) функциясы рекурсивті анықталады *n * факториал*(*n – 1*), ал факториалдың негізгі жағдайы - *факториал*(*0*) *= 1*. Ф*акториал*(*4*) функциясын шақыру *4 * факториал*(3) сияқты жайылады; *факториал*(*3*) *3 * факториал*(*2*) сияқты жайылады; *факториал*(*2*) *2 * факториал*(*1*) сияқты жайылады; және *факториал*(*1*) как *1 * факториал*(*0*) сияқты жайылады; *факториал*(*0*) негізгі жағдай болып табылады. Ф*акториал* (*0*) бағаланғаннан кейін, 1 мәні *факториал* (*1*)-ге беріледі; *факториал* (*1*) *факториала* (*2*) үшін *1 * 1 = 1*-ге өтеді. *факториал* (*2*) *факториала* (*3*) үшін *2 * 1 = 2-ге өтеді; факториал* (*3*) *факториала* (*4*) үшін *3 * 2-ге өтеді.* Соңғы мән 4 * 6 = 24 соңында оралады.

2.8 мысал
Рекурсивті функцияның басқа мысалы «Фибоначчи санын» табу болып табылады. Фибоначчи санын анықтаудың төменде көрсетілгендей екі базалық жағдайы бар.

Фибоначчи(0) = 1 % base case Фибоначчи(1) = 1 % base case
Фибоначчи(n) = Фибоначчи(n - 1) + Фибоначчи(n - 2) % recursive
% definition

2.2.3.1 Құйрықтық Рекурсия және Итерация
Соңғы рекурсия бағдарламалау тілдерін оқу кезінде ерекше назар аударуды талап ететін рекурсивті анықтаманың маңызды класс тармағы болып табылады. Соңғы рекурсияда рекурсивті шақырту функциядағы соңғы орындалатын операция болып табылады. Мысалы, *GCD* функциясы соңғы рекурсия көмегімен келесі түрде жазылады:

GCD(x, y) = gcd((bigger(x, y) modulo smaller(x, y)), smaller(x, y)). % tail recursive definition
GCD(0, x) = x % base case
Жоғарыда келтірілген анықтамада егер бірінші аргумент 0-ге тең болса негізгі нұсқа екінші аргументің мәнін қайтарады. Бұл соңғы рекурсияның сол жағы аз аргументтердің жиыны ретінде көп аргумент алған кезде болады. Соңғы рекурсивтілікті анықтау бірінші аргументтен үлкен және кіші аргументтерді бөлуден қалған рекурсивті *GCD* функцияны шақырады. Ал екінші аргумент соңғы аз аргументтің соңғы аз аргумент ретінде *GCD* функциясының жаңа шақыртуында үлкен аргумент болар еді.
Соңғы рекурсияның бір қасиеті анықталмаған итерацияны пайдаланып моделденеді және рекурсивті анықтаманың жаңа үдерістерін шақыртумен байланысты жадының қосымша шығындарын болдырмайды. Назар аударыңыздар, факториалды анықтау да, Фибоначчиді анықтау да соңғы рекурсия болып табылмайды. Факториалда «көбейту» соңғы операция ретінде болады. Ал Фибоначчидің рекурсивті анықтамасы бар екі шақыртуы бар. С*оңғы рекурсияны оңтайландыру* – бұл кодты оңтайландыру әдісі уақытты қысқартуға мүмкіндік беретін жадыны қайта пайдалануға арналған және атқаруды итерацияның белгісіз мерзімге соңғы рекурсия бағдарламасын түрлендіру арқылы тежейді. Итерациялық бағдарламалар бағдарламаны баяу жүзеге асырып және жады кеңістігін шығындайтын стектағы жады мен уақыттың қосымша шығындарын төмендетеді.

2.2.3.2 Сызықтық рекурсия және итерация

Сызықтық рекурсивті функцияның үлкен классы, рекурсивті анықтамадағы өзіне тек бір реет қана шақыратын функциялар *мерзімсіз итерация және жинақтағышты* пайдаланып итерациялық бағдарламаға түрлендіреді. *Жинақтағыш*- соңғы ішінара есептелген нәтижені сақтайтын абстракция. Итерациялық нұсқада айнымалының сол жинағы қайтадан қолданылады. Ал рекурсивті үдерістің шақыртулары негізгі жағдайдан басталып, әрбір циклдан кейін ішінара шығыс мәндерін жинап, сақтайтын қайталанатын кодпен алмастырады.

2.9-мысал

Факториал функциясының итераривті нұсқасы 2.3-суретте келтірілген. Жинақтағыш *факториал* (*0*) = 1-ге факториал функциясының негізгі мәніне жүктелген. Ал рекурсивті шақыртулар итерациямен алмастырылған. Әрбір қадамнан кейін жинақтағыштың мәні жаңарып тұрады. Бұл рекурсияның негізгі жағдайының басына және құрылуға балама

болады; қазіргі жағдайда оны бұрынғыдан да тиімді ететін рекурсивті үдеріс ашылмайды.

Algorithm iterative_factorial
Input: value of n;
Output: accumulator value;
{ accumulator = 1;
for (i = 1; i =< n; i++) accumulator = i * accumulator;
return(accumulator);
}

2.3-сурет. Факториал функциясының итеративті нұсқасы.

2.2.4 Соңғы автоматтар

Соңғы автомат -шынайы әлемнің құбылыстарын әртүрлі жағдайларды (күйлерді) моделдеу арқылы моделдеуге арналған абстрактылы машина. Графалардағы түйіндер мен түйіндер арасындағы доға белгісі түріндегі күйлер арасындағы өткелдер бейнесінде болады. Автомат бір күйден екінші күйге кіріс мәндердің негізінде өтеді. Бұл күйлердің барлығы барлық күйлерге ашық болмауы мүмкін. Бір немесе бірнеше бастапқы күй болады және бір немесе бірнеше соңғы күй болады. автоматтар бастапқы күйдің біреуінен бастап, соңғы күймен аяқталады.

2.10 мысал

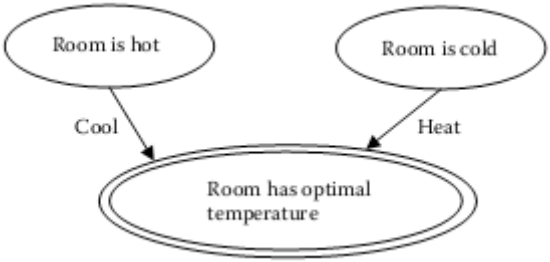
Қыздыру және салқындату жүйесі 2.4-суретте көрсетілгендей соңғы автомат сияқты моделденуі мүмкін. Үш күй келтірілген: «бөлмеде ыстық», «бөлмеде суық», «температура оңтайлы». Осы күйдің барлығы бастапқы күй болуы мүмкін. Соған қарамастан, тек «оңтайлы температура» күйі ғана соңғы күй болып табылады.

«Бөлмеде ыстық» күйінде термостат салқындай бастайды. Мұның нәтижесі «оңтайлы температура» күйіне өту болады. «Бөлмеде суық» күйінде термостат қыза бастайды. Мұның нәтижесі «оңтайлы температура» күйіне өту болады. Егер берілген температура мен бөлмедегі температура бастапқы мәннің шегінде болса, онда ешқандай дабыл берілмейді.

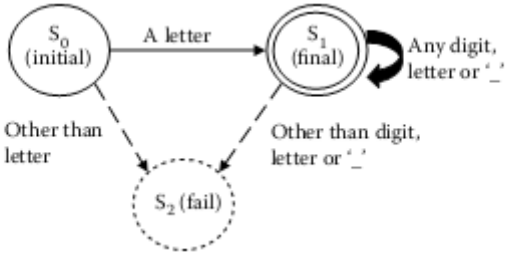
2.11-мысал

2.5-суреттегі соңғы автомат бағдарламадағы айнымалылар аттарын анықтайтын болады. Айнымалы ағылшын әрібі сияқты анықталады.келесілері латын әріптері немесе сандарының кез-келген санымне анықта-

лады. Автоматтың бастапқы күйі S0 соңғы күйі S1, ал S2 күйі барлық қателіктердің шарттарын өңдеу үшін. Автомат S0 шартында басталады. S1шартына өту үшін тек ағылшын әріптері қолданылады; кез келген басқа сипаттар автоматты S0 шартынан S2 шартына алып келеді. Мұнда қате күй туралы дабыл алынып, атқару дұрыс аяқталатын болады.



Room is hot - Бөлме ыстық; Cool – Салқындату; Room has optimal temperature - Бөлменің температурасы оңтайлы; Room is cold – Бөлме салқын; Heat – Жылыту.
2.4-сурет. Термостаты моделдеудің соңғы автоматы.



S_o (initial) – S_o (бастапқы); A letter – Әріп; S_1 (final) – S_1 (соңғы); Any digit, letter or '_' - Кез келген сан, әріп немесе '_'; Other than digit, letter or '_' – Саннан, әріптен немесе '_' басқа; S_2 (fail) – S_2 (қате); Other than letter – Әріптен басқа.
2.5-сурет. Айнымалылардың аттарын қабылдау үшін соңғы автомат

Бірінші әріпті қабылдағаннан кейін автомат S1 күйге көшеді. Ол жерде басқа әріп я болмаса басқа сандар да қабылданады. Автомат әріп, сан немесе "_" сияқты басқа да арнайы таңбаларды алу-алмуына қарамастанмастан а мА S1 -ден S1 –ге тәуелсіз ауысады.
Басқа жағдайда ол S2 – қате шартқа өтеді.

2.3 Деректер құрылымының тұжырымдамасы

Бағдарламалау тілдерін қолдану үшін түсінуге тиісті мәліметтер құрылымында көптеген тұжырымдамалар кіріді. Кейбір маңызды тұжырымдар мыналар: өңдеуде қолданылатын ағаш; бағдарламалау тілдері механизмдерінің рекурсияларында және жүзеге асыруда болымсыз пайдаланылатын стектар; кейбір тиімді қоқыс жинау сызбаларында қолданылатын тізімдер мен графиктер; және бағдарламалаудың әртүрлі парадигмаларын жүзеге асыратын механизмдер мен нысандарға болымсыз рұхсат алу үшін қолданылатын хэштегтеу.

2.3.1 Әрекеттер тізбегі

Әрекеттер тізбегі – тікелей алдыңғы элемент тірек элементтің позициясынан тұп-тура бір элементке кем қалыппен тізбектей байланысқан реттелген мультижиын элементтері. Мысалы, *тармақ* дегеніміз таңбасыдар тізбегі, ал *файл* мәліметтер нысанының тізбегі. Тіпті мәліметтер типінің абстарктылы стектары мына тізімдері келесі тармақта көрсетілгендей әрекеттер тізбегін пайдалана отырып моделденуі мүмкін.

Тізбекті құрушы элементтермен бұрыштық жақшалармен белгілейміз '<' және '>'. Мысалы, *<x, y, z>* x мәліметтер элементі 1 қалыппен байланысқан, y элементі 2-қалыппен, *z* элементі 3-қалыппен байланысқан тізбек. Егер біз екі функцияны –тізбектегі мәліметтер элементімен *алдыңғы және қабылдағыш* функциялар деп белгілесек, онда *алдыңғы(y) = x* және *алдыңғы (z) = y*, өйткені y мәліметтер элементінің қалыбы x мәліметтер элементінің қалыбынан көп. Ал *z* мәліметтер элементінің позициясы y мәліметтер элементінің қалыбынан көп. Сәйкесінше, *қабылдағыш(x) = y және қабылдағыш(y) = z*. Тізбекпен операция элементті алу, элементті қою, элементті басқа элементпен алмастыру, екі тізбекті жалғастыру, тізбек басқа тізбектің тізбегі екендігін тексеру, тізбекті алу және тізбекті басқа тізбекпен алмастыру категориясына жатады. Тізбекпен жасалатын негізгі операциялар 2.5-кестеде келтірілген.

Элемент басынан бастап, соңынан немесе элемент индексін ұсыну арқылы қолжетімді болуы мүмкін. *first* операторы тізбектің бірінші элементін қайтарады, *second* операторы тізбектің екінші элементін, ал *last* операторы тізбектің соңғы элементін қайтарады. Қосымша операция элемент түскеннен кейін тізбек ішінде қалғандарын анықтау болып табылады. *rest* операторы соңғы тізбектің қалған ішкі тізбегіне минус бірінші элементті береді. Ал оператор *butlast* соңғы тізбектің қалған ішкі тізбегіне минус соңғы элементті береді. *select* операторы элемент индексінің және тізбектін екі кірісін қабылдап, тізбектің сәйкес элементін қайтарады. *Cons* (construct сөзінің қысқарған түрі) берілген эле-

ментті соңғы тізбектің басына қойып, жаңа тізбек құрады.

2.5 кесте. Әрекет тізбегін қосатын негізгі операциялар

Операция	Шығыс	Түсіндірме
first(<x_1…x_n>)	x_1	Бірінші элемент тізбек
last(<x_1…x_n>)	x_n	Соңғы элемент тізбек
rest(<x_1…x_n>)	<x_2,…,x_n>	Қалған бөлік тізбек
butlast(<x_1…x_n>)	<x_1,…,x_n-1>	Соңғы элементті қоспағандағы тізбек
select(i, (<x_1…x_n>)	x_i	"i"- элемент тізбек
cons(a, <x_1…x_n>)	<a, x_1,…,x_n>	Ескі тізбектің басына "a" қосу арқылы жаңа тізбек жасау
insert(i, a, <x_1…x_n>)	<x_1…x_{i-1},a,x_{i+1}… x_n>	
append(<x_1…x_n>,<y_1…y_m>)	(<x_1…x_n, y_1…y_m>	тізбектің жиыны реттелген
subsequence(<x_1…x_n>,i, m)	<x_i,…,x_{i+m-1}>	Тізбектің іші i орыннан *m ұзындықтан басталады*
is_subseq(<x_1…x_n>, <y_1…y_m>)	Boolean	Қайтару "ақиқат", егер <x_1…x_n> кірсе<y_1,…y_m> , басқаша жағдайда қайтару «Жалған»

insert операциясы берілген элементті берілген күйге қойып, жаңа тізбек құрады. *cons* операторы – бұл қоюдың арнайы жағдайы. *append* операторы келесідей екі тізбек болады *<x1…xn>* и *<y1…ym>* және екінші тізбектің элементтерін бірінші тізбектің элменттерінен кейін қойып *<x1… xn, y1…ym>* жаңа тізбек жасайды. *subseq* операторы үш кіріс аргументті қабылдайды: (1) соңғы тізбек, (2) соңғы тізбектегі бастапқы қалып және (3) тізбек ұзындығы мен тізбекті алу. Мысалы, *subseq*(*<4, 5, 6, 7>*, *3, 2*)-тің тізбегі <6, 7>. *is_subseq* предикаты кіріс ретінде екі тізбекті қабылдап, тізбек екінші тізбектің тізбегі бола мА, соны тексереді. Мысалы, *is_subseq*(*<6, 7>*, *<4, 5, 6, 7>*) қайтуы *"ақиқат"*.

Тізбек байланысқан тізбек, массив немесе векторды пайдаланып жүзеге асуы мүмкін. Мәліметтердің барлық құрылымдарының сипаттары әртүрлі. Байланысқан тізімде элементтерге ереже бойынша жоғарыдан аяғына дейін барады; массивте элементтер бейберекет түрде қолжетімді, ал вектор кездейсоқ қолжетімді және атқару кезінде кеңейеді.

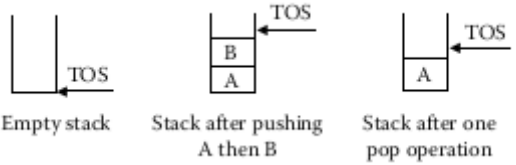
2.3.2 Стектар мен Тізімдер

Стектар да, тізімдер де бағдарламалау тілдерін жүзеге асыру үшін маңызды. Стекар да, тізім де ағаш немесе график түрінде моделденген іздеудің күрделі кеңістігін іздеуде қолданылады. Стектер (1) бағдараламалу тілдерін жүзеге асыру, (2) рекурсияны өңдеу, (3) және 5,6 тарауларда айтылғандай хиптегі мәліметтердің динамикалық құрылымын басқару

және қайта өңдеу кезінде қолданылады. Тізімдер 6-тарауда айтылғандай хиптен бөлінген мәліметтердің динамикалық құрылымын болымсыз қайта жасауа қолданылады.

2.3.2.1Стек

Стек – бұл мәліметтер тек бір шеттен ғана енгізілетін абстрактылы мәліметтер типі. Енгізілген мәліметтер қашан да стектің ағымдағы басына орналасады, ал мәліметтер элементі стектің ағымдағы басынан жоғалады; келесі шеті бекітіледі. Оның үстіне, біз стектің жоғарғы элементін стектегі элементті жоймай-ақ оқи аламыз және біз оны бос стек үшін де тексере аламыз. Стектің негізгі артықшылығы - оның мәліметтердің ең соңғы элементтерімен жұмыс істеу мүмкіндігі. Стектер сондай-ақ «соңынан келіп – бірінші кетті» (LIFO, акроним Last In, First Out).



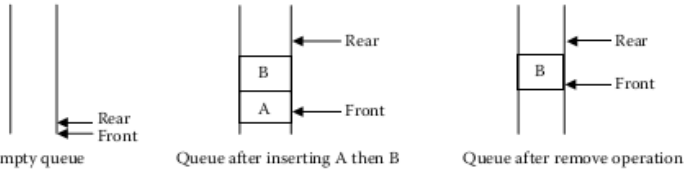
TOS – TOS; Empty stack - Бос стек; Stack after pushing A then B – A мен B басқаннан кейінгі стек; Stack after one pop operation – Бір операциядан кейінгі стек.
2.6.-сурет. Стек операциясы.

Стектің төрт абстрактылы операциясы бар: (1) *push* (*стек, мәліметтер элементі*) – мәліметтер элементінің стектің жоғарғы жағына жылжуы; (2) *pop (стек)* – стектің жоғарғы элементін оқу және жою; (3) *top (стек)* – стектің жоғарғы элементін стекті өзгертусіз оқу; және (4) *is_empty (стек)* – стек бос па, сонны тексеру. *push (стек, мәліметтер элементі)* нұсқаулықтары және *pop (стек)* стекті өзгертеді. Push операциясы кезінде қойлыған элемент стектің жоғарғы элементі болады. *pop(стек)* жағдайында жоғарғы элемент стектен жойылады. *top(стек)* нұсқаулығы стектің жоғарғы элементін стектің мазмұнын модификациялаусыз оқиды. Стек негізіндегі операциялар 2.6 суретте келтірілген.
Стек мәліметтер соңынан бастап қойылатын және алынатын тізбек ретінде моделденуі мүмкін. *x* элементін стеке жіберу S = <*a1, a2, ..., aN*> жаңа <*x, a1, a2, ..., aN*> сапасындағы S’ стек береді және стек өлшемі 1-ге артады. Элементті S = <*a1, a2, ..., aN*> стектен итеріп шығару жаңа S’ = <*a2, ..., aN*> стек береді және стек өлшемі 1-ге кемиді. Бос стек бос

тізбек< > ретінде моделденеді.
Стекті компьютер жадысында қолдану екі көрсеткішті талап етеді: бастапқы көрсеткіш және соңғы көрсеткіш. Бос стектің жоғарғы жағын көрсеткіш бастапқы көрсеткішке тең және толы стектің жоғарғы жағын көрсеткіш соңғы көрсеткішке тең. Егер стектің жоғарғы жағын көрсеткіш көрсеткіштің соңынан шығуға тырысса, онда бұл «жады толып кетті» деген қателікті көрсетеді.

2.3.2.2 Тізім

Тізім – мәліметтер бір шетінен шығарылып, ал мәліметтер екінші шетіне қойылатын мәліметтердің абстрактылы түрі. Әдетте, мәліметтер элементі тізімге келген рет бойынша шығарылады. Тізімке екі көрсеткіш керек: алдыңғы және артқы. Алдыңғы көрсеткіш бірінші мәліметтер элементін алуға қолданылса, артқы көрсеткіш жаңа мәліметтер элементін қоюға арналған. Егер алдыңғы көрсеткіш артқы көрсеткішті қуып жетсе, тізім бос. Элементті тізімге қою үшін мәліметтерді артқы көрсеткіш көрсеткен жерге орналастырады. Ал артқы көрсеткіш бір бірлікке артады. Мәліметтер элементін екінші шетінен алу үшін алдыңғы көрсеткіш көрсеткен мәліметтер элементін алады ал алдыңғы көрсеткіш бір бірлікке артады. Егер «артқы» және «алдыңғы» көрсеткіштер 2.7-суретте көрсетілген бағытта бірге қозғалатын болса, онда тізім сызықтық деп аталады. Тізім «алдыңғы» және «артқы» көрсеткіштер тізімдегі соңғы элемент өткеннен кейін бірінші элемент өткен кезде модул бойынша арифметикалық операция арқылы шеңберлі болады.



Rear – Артқы; Front – Алды; Empty queue - Бос кезек; Queue after inserting A then B – A мен B қосқаннан кейінгі кезек; Queue after remove operation - Жою операциядан кейінгі кезек.
2.7-сурет. Тізімдермен операциялар
Жою операциясынан кейінгі тізім

Шеңберлі тізім мәліметтер элементін жойғаннан кейін босаған жады кеңістігін тиімді түрде қайта пайдаланады.
Гипотезалық тұрғыдан тізімдер мәліметтердің жаңа элементтері артқы бөлікке орналасып, ал мәліметтердің бірінші элементі соңғы бөліктен

жойылатын элементтер тізімі сияқты моделденуі мүмкін. Q = <a1, a2, …, aN> тізіміне жаңа x элементін қосу жаңа Q' <a1, a2, …, aN, x>тізім береді, ал тізімнің өлшемі 1-ге артады. Q = <a1, a2, …, aN> тізімінен элементті алып тастау, жаңа Q’ = <a2, …, aN> тізімін береді, ал тізім өлшемі 1-ге кемиді.

2.3.3 Референттік механизмдер

Жады ұяшықтары және тіркеуіштер ақпараттың екі түрін сақтайды: (1) есептеу жүргізілетін мәліметтер мәні және (2) басқа жады ұяшықтарына көрсететін жады ұяшықтары. *Көрсеткіштер* – бұл тіркеуіште немесе жады ұяшығында сақталатын басқа жады ұяшықтарының адресі. Көрсеткіштерді пайдаланудың көптеген артықшылықтары бар:

1. Жадының басқа бір бөлігінде сақталатын мәліметтердің күрделі құрылымын референттейді.
2. Физикалық жағынан көп мәліметтер құрылымының орнын ауыстыруға кететін қосымша шығындардан қашу.
3. Атқаруға дейін айнымалы үшін жадының бөлінуін кері шегеріу. Бұл қасиет байланысқан тізімдер, ағаштар, векторлар және сол сияқты кеңейген мәліметтер құрылымы үшін жадыны тиімді тарату үшін пайдалануы мүмкін.
4. Физикалық жадының бірнеше блоктары әртүрлі уақытта таралатын және көрсеткіштер арқылы байланған мәліметтер құрылымының күрделі логикалық іргелестер үшін жады блогын бөлу.
5. Физикалық жадыда мәліметтердің орын ауыстыруынан бағдарламаның тәуелсіздігін қамтамасыз етіп, қайта адрестеу сілтемесі ретінде пайдалану.
6. Күрделі мәліметтер құрылымының бөліктерін басқа мәліметтер құрылымымен бірге пайдалану.

Көрсеткіштер рекурсивті мәліметер құрылымы үшін жадының нақты өлшемі компиляция кезінде белгісіз болғандықтан, байланысқан тізімер, ағаштар сияқты рекурсивті мәліметтер құрылымын жүзеге асыру кезінде пайдаланылды. Сонымен қатар, кіріс мәліметтерге байланысты мәліметтердің рекурсивті құрылымдарының өлшемдері өзгеруі мүмкін және компиляция кезінде бағаланбауы мүмкін емес.
Жүйелік бағдарламалауға жады ұяшықтары арқылы әр адам сайын рұхсат алу талап етіледі. Бұл талап көрсеткіш жүйелік бағдарламалар бірөлшемді массив сияқты моделденетін жады облысы арқылы өте алатын көбейту және алу мүмкіндіктерімен қамтамасыз етілу қажет

болғандықтан қойылады. Бұл пайдаланушы анықтаған көрсеткіш *сегмент*- белгілі бір бағдарламаны орындауға бөлінген жедел жады облысы, шекарасын қиып өтпеу үшін керек.
Көрсеткіштердің артқышылығына қарамастан, көрсеткіштерді жоғары деңгейлі тілдерде қолданудың негізгі үш кемшілігі бар:
1. Көрсеткіштермен арифметикалық операция жадының секіртпелі түрде орналасуына мүмкіндік береді. Жады ұяшығына сегменттен тыс рұхсат алудың арифметикалық операциясы «сегменттің бұзылуы» қатесін тудырады.
2. Программаны компиляциялағаннан кейін әртүрлі мәліметтер түрі бар әртүрлі айнымалылар сызықтық жадының бір кеңістігіне түрленеді; мәліметтердің осы түрлерінің арасындағы барлық ақпарат жойылады. Егер біз көрсеткіштерге арифметиканы жіберсек, онда көрсеткіштер оқу кезінде берілген нысандардың әртүрі арқылы өтуі мүмкін немесе мәннің өзгерісі дұрыс болмайды.
3. Мәліметтердің бірнеше құрылымымен пайдалнатын жады ұяшықтары жады ұяшықтарын көрсететін көрсеткіштер босамайынша қайтадан пайдаланылмайды.
Көрсеткіштердің осы қиындықтары себепті бағдарламаларда қателіктер кетіп, жұмыс барысында ақау орын алады. Көрсеткіштерге байланысты мұндай қателіктерді анықтау қиын.

2.3.4 Рекурсивті мәліметтер құрылымы.

Рекурсивті анықталатын байланысқан тізімдер, ағаштар және векторлар сияқты мәліметтер құрылымының класстары бар. Байланысқан тізімдер тізімнің қалған бөлігінің ақпарат түйіні сияқты рекурсивті анықталады, ал жалпы жағдайда бос тізім болып қалады.

<*linked-list*> ::= <*information-node*> <*linked-list*>
<*linked-list*> ::= **null**

Бинарлы ағаш сол жақ ағаш және оң жақ ағаш болатын ақпараттық түйін сияқты рекурсивті анықталады. Жалпы жағдайда бос ағаш болады.

<*binary-tree*> ::= <*binary-tree*><*information-node*><*binary-tree*>
<*binary-tree*> ::= **null**

Мәліметтер құрылымы негізіндегі көрсеткіштерінде тиімді қою және жою операциялары бар. Алайда, мәліметтердің рекурсивті құрылымы мәліметтердің рекурсивті құрылымының орындалу уақытының өл-

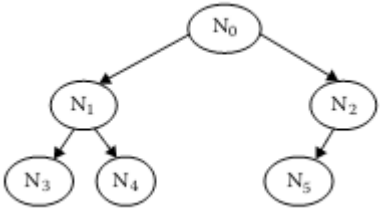
шемін білмегендіктен компиляция кезінде бөлінбейді. Мәліметтердің рекурсивті құрылымы бағдарламалардың орындалуына және кіріс мәліметтеріне байланысты атқару кезінде өседі. Рекурсивті мәліметтер құрылымы жады бағдарламалаушының талабына сүйеніп, атқару кезінде бөлшектеніп бөлінеді.
Бізге рекурсивті мәліметтер құрылымын жүзеге асыру үшін *хип* деп аталатын жалпы жадының арнайы облысы керек. Бұл хип біздің мәліметтер құрылымында оқыған хиптен өзгеше. Рекурсивті мәліметтер құрылымы хипке физикалық тұрғыдан атқару кезінде жадыны бөлу үшін бағдарлама талап ететін жады мен рет слотының болуына негізделіп таратылады. Сол рекурсивті құрылымның әртүрлі физикалық фрагменттері көрсеткіштің тізбегі арқылы байланысқан.

2.3.5 Ағаш

Ағаш бір немесе бірнеше мәліметтер элементі бар және ақпараттық түйіннің түйін-тармақтарына қатысты бекітілген бұтақтары бар жалпы ақпараттық түйін түрінде ұйымдасқан бірнеше ақпараттық түйіннен тұратын рекурсивті мәліметтер құрылымы. Негізгі жағдай «нөлдік» ағаш деп аталады. Ағаштың бірнеше тармақтары болуы мүмкін. *Бинарлы ағаштың* әрбір ақпараттық түйін үшін бір ұяшығы болады және екіден көп емес бұтағы болады: сол жақ бұтақ және оң жақ бұтақ. Егер түйіннен таралатын бұтақтардың максималды саны "*N*" болса, онда ол ағаш "*n-жұп ағаш*" деп аталады. ағаштар екі әдісті қолдану арқылы жүзеге асады: (1) массивтер немесе векторлар сияқты индекстелген мәліметтер құрылымы және (2) негізгі түйіндерді таралған түйіндерге байланыстыратын көрсеткіштер. Индекстелген мәліметтер құрылымы толық бинарлы ағаштарды немесе толық дерлік бинарлы ағаштарды көрсетуге арналған. *Толық дерлік ағаштың* екі тармағы бар барлығы жапырақты емес түйіні болады. Тек, соңғы оң жағында 2.8 суретте көрсетілгендей бір сол жақ тармағы бар екі жапырақты түйіні бар.
Көрсеткіш негізінде жүзеге асыру кез келген ағашқа жарай береді. Соған қарамастан, кері қайту үшін негізге түйіннің индексін есептеу мүмкіндігі жоқ. Көрсеткішті пайдаланып жүзеге асқан ағаш табиғатынан бағытталған болып табылады: ағаштар тек негізгі түйіннен таралған түйінге ғана алмаса алады. Көрсеткіштің осы бағытталған қасиетінен өтіп кету үшін бізге негізгі мекеннен ұрпаққа өткенге дейінгі мекежайды сақтайтын стек керек немесе, көрсеткішке қосымша негізгі түйіннен ұрпақ түйінге өткен ұрпақ түйінде сақталған кері көрсеткішті пайдалану керек. Екі сызбаның да қосымша шығын жадылары бар, сондай –ақ атқару уақытының да қосымша шығыны бар. Соған қарамастан, стек

жағдайында жадының бос шығындалуы ағаштың тереңдігімен шектелген.

Ағаш бағдарламалау тілдерінде және компиляторларда синтаксистік сараптау фазасы кезінде қолданылады. «ЖӘНЕ-НЕМЕСЕ» деп аталатын ағаштың арнайы түрі логикалық бағдарламалау парадигмасының негізгі жүзеге асырылуы болып табылады және 10 тарауда кеңінен қаралады. Ағаш сондай-ақ, бағдарламадағы үдерістің немесе үдерістегі блоктардың тіркеме құрылымын көрсету үшін, пайдаланылады және солайша үдерісте немесе блокта белгіленген айнымалы көлемінің дәрежесін түсінуге көмектеседі. N-ағаш 10 тарауда көрсетілгендей логикалық бағдарламалау парадигмасындағы мәліметтер құрылымын жүзеге асыру үшін пайдаланылады.



2.8-сурет. Толық дерлік бинарлы ағаш.

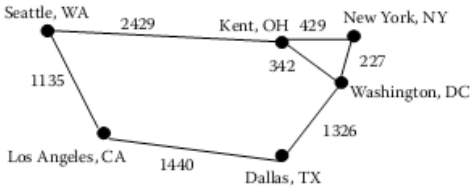
2.3.6 Бағаналар

Бағаналар негізінен (V, E) жұбы ретінде анықталады. Мұндағы V төбелер жиыны, ал E қабырғалардың мультижиыны. Бағдарланбаған бағанада қабырғалар (vi, vj) жұбы ретінде моделденген, мұндағы vi, vj $\in$ V төбелер болып табылады. (vi, vj) $\in$ E жұбы vi және vj төбелер арасындағы байланысты көрсетеді. Екі түйіннің арасында біреуден артық қабырға болуы мүмкін. Жол – бұл соңғы түйінді белгілеу түйінімен жалғастыратын қабырғалар тізбегі. Екі түйіннің арасында біреуден артық жолдар болады.

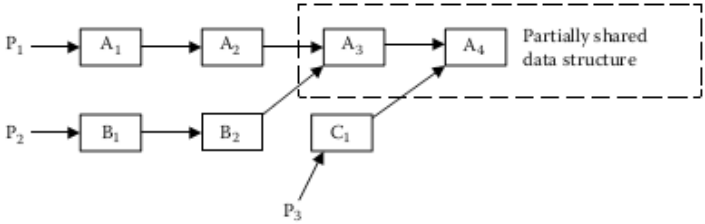
Бағдарланбаған бағанада (vi, vj) қабырғаның болуы симметриялы қабырға (vj, vi) бар екенін білдіреді. Қабырғалардың бағыты бар бағдарланған бағанада қабырға реттелген (vi, vj) жұп ретінде моделденеді. Ал (vi, vj) болуы (vj, vi) қабырғаның бар екенін білдірмейді. Өлшенген бағананың қабырғалармен байланысқан салмақтық коэффициенттері бар. Өлшенген бағанадағы қабырғалар (vi, vj, весij) түріндегі үштік ретінде моделденеді. Мұндағы ij қабырғаның салмағы. *Өлшенген бағдарланбаған бағанада* өлшенген (*vi, vj, вес ij*) қабырғаның болуы *(vj,*

vi,весij) бар екенін білдіреді; симметриялық қабырғалардың салмақтары бұрынғыдай қалады. *Өлшенген бағдарланған бағанада* (*vi, vj, весij*) формалы өлшенген қабырғаның болуы дәл сондай салмақты, бағыты қарама-қарсы симметриялы қабырғаның бар екенін білдіреді. Байланысқан бағана – бұл транзитивті қатынас. Егер *vi* түйіні *vj* –мен байланысты болса, ал *vj* түйіні *vk*-мен байланысты болса, онда (*vi, vk*) анық қабырғасы болмаса да, *vi* түйіні *vk* түйінімен байланысты. Бұл қасиет мәліметтер элементінің рекурсивті мәліметтер құрылымындағы қолжетімділігі тұрғысынан маңызды мәнге ие.

2.12-мысал.
2.9 суретте бағдарланбаған өлшенген бағананы пайдаланып, Құрама Штаттардағы қалалар арасындағы жолдар мен қашықтықтардың байланысы бейнеленген. Қалалар түйіндер арасында бірнеше қабырға алу үшін темір жол хабарламалары немесе әуе хабарламалары арқылы байланысуы мүмкін. Соған қарамастан, біз бұл бағанадағы тек сол сілтемелерін ғана пайдаланамыз. Бағаналар бағана түйіні (немесе төбелер) көрсетілген алты қаланың бар екенін көрсетеді. Ал олардың арасын байланыстыратын жолдар төбелер арасындағы өлшенген қабырғаларды пайдаланып көрсетілген. Мұндағы салмақ қалалар арасындағы жолды миль есебімен көрсетеді.
Бағана *кезеңденбеген* – оның кезеңі жоқ; егер түйіннен бастаса, онда әртүрлі екі қабырғалар тізбегін пайдаланып ешкім сол түйінге жете алмайды. Ағаш – кезеңденбеген бағана. Бағана кезеңденген. Егер ең аз дегенде бір түйіннен бастасақ, біз қабырға қайталанбайтын жолды пайдаланып дәл сол түйінге жете аламыз. Мысалы, 2.9-суретте байланысқан бағанадағы көптеген кезеңдер бар: <(*Сиэтл, Кент*), (*Кент, Вашингтон Колумбия аймағы*).



Seattle, WA – Сиэтл, штат Вашингтон; Kent, OH – Кент, штат Огайо; New York, NY – Нью-Йорк, штат Нью-Йорк; Washington, DC – Вашингтон, Дистрикт Коламбия; Dallas, TX – Даллас, штат Техас; Los Angeles, CA – Лос-Анджелес, штат Калифорния.
2.9-сурет. Өлшенген бағдарланбаған бағана мысалы.



Partially shared data structure - Ішінара ортақ деректер құрылымы.

2.10-сурет. Жалпы ақпараттық түйінді бағдарланған кезеңденбеген бағананы моделдеу.

(*Вашингтон Колумбия аймағы, Даллас*), (*Даллас, Лос-Анджелес*), (*Лос-Анджелес, Сиэтл*)>. Дәл осындай басқа кезең<(*Кент, Нью-Йорк*), (*Нью-Йорк, Вашингтон, Колумбия аймағы*), (*Вашингтон, Колумбия аймағы, Кент*)>.

Бағаналардағы кезеңдерді анықтау маңызды мәселе. Кезеңді анықтау үшін бұрын барған түйіндерді сақтау керек. Сондықтан қатыстылықты тексеру бұрын барған түйіндер қарастырылды ма, соны тексеруге пайдалануылуы мүмкін. Қатыстылықты тексеру 2.3.9 бөлімде сипатталғандай хештеуді пайдаланып, анық емес атқарылуы мүмкін. Бағдарламалау тілдерінде бағаналар 2.13-мысалда келтірілгендей мәліметтердің жалпы құрылымын көрсету үшін пайдаланылады. Бағаналар сондай-ақ (1) бағдараламалар операторларының арасындағы ақпараттық ағын моделі; (2) жадыны пайдалану кезінде хипте мәліметтердің қандай ұяшығы пайдаланылғанын анықтау үшін; (3) тапсырманы шешу кеңістігін моделдеу үшін; және (4) бағдарламалаудың функционалды тілдерін жүзеге асыру үшін пайдаланылады.

2.13-мысал.

2.10-суретте өз мәліметтер құрылымының бөліктерін айырбастаудың үш байланысқан тізімі келтірілген: бірінші байланысқан тізім *A*1, *A*2, *A*3, *A*4төрт ақпараттық түйіннен тұрады; екінші байланысқан тізім *B*1, *B*2, *A*3, *A*4 ақпараттық түйіндерден тұрады;ал үшінші байланысқан *C*1, *A*4 ақпараттық түйіндерден тұрады. Бірінші және екінші мәліметтер құрылымы екі ақпараттық түйінді бөледі: *A*3 және *A*4; бірінші, екінші және үшінші мәліметтер құрылымы жалпы *A*4 ақпараттық түйінді бөледі.

2.3.7 Іріктеп алу әдісі

Есептеу күй кеңістігінде іздеу мәселесі сияқты моделденуі мүмкін. Күй кеңістігіндегі мәселенің бірнеше есептеу күйі бар және әрбір есептеу нұсқаулығы машинаны бір есептеу күйінен келесі есептеу күйіне ауыстырады. Есептеу күй кеңістігіндегі бағана ретінде моделденуі мүмкін. Іздеудің әртүрлі әдістері негізгі екі категорияға жіктелуі мүмкін: (1) Есептеу күй кеңістігінде бағана түрінде моделденуі мүмкін. Іздеудің әртүрлі әдістері негізгі екі категорияға жіктеледі: (1) іріктеп алу әдісі және (2) эвристикалық іздеу. *Іріктеп алу* әдісінің сызбасы мақсат табылмайынша бағана күйінің кеңістігіндегі барлық түйіндерді іздейді. Мақсатты түйін – бұл қалаулы соңғы шартқа ие күй. Іріктеп алу әдісі егер іздеу кезеңге кірмеген болса, шешім алуға кепіл болады. Ағаштардағы немесе бағаналардағы іріктеп алу әдісіне екі негізгі әдіс бар: *тереңдігін іздеу және енін іздеу.*

Эвристикалық іздеу ағымдағы күйдің соңғы күйге қарай қозғалысының күйін және бағытының қашықтығы мен жақындығын бағалау үшін математикалық функцияларды пайдаланады. Эвристикалық іздеу іріктеп алу әдісіне қарағанда тиімдірек, бірақ ол шешімге кепілдік бере алмайды. Бағдарламалау тілдерінің тапсырмаларының көпшілігі іріктеп алу әдісін қолданады. Ал жасанды интеллект эвристикалық іздеуді қолданады.

2.3.7.1 Тереңінен іздеу

Тереңдігін іздеу берілген рет бойынша ағашты түйінге айналып өтеді. Әдетте солдан оңға қарай. Мұндай іздеу стекті оң жақ бұтақтарға ауыстыруды жеңілдету үшін ақпаратты сақтау үшін пайдаланады. Іздеу түпкі түйіннен басталады және ойналып өту үшін соңғы сол жақ тармақты таңдайды. Зерттелмеген тармақтың түйінінде тамыры бар бұтақ зерттеліп жатқанда егер көрсеткіштер ағымдағы түйінге тағы қайтып келмесе көрсеткіштер үшін онда оң жақ бұтаққа қайтып оралу мүмкіндігі жоқ. Керекті бұтаққа өтуді жеңілдету үшін ағымдағы түйіннің адресі стекте сақталады. Стек LIFO режиміндегі ақпаратты сақтағандықтан, ағымдағы түйіннің аналығы айналып өткен кезде бірінші болып алынады. Жапырақ түйінді айналып өткеннен кейін (бұтағы жоқ түйін) келесі түйін стектің жоғарғы бөлігінен алынады. Бұл стектен алу үдерісі және стектен зерттеу және келесі оң жақ бұтақты зерттеу алынған түйін үшін стек босап қалғанша және айналып өтетін тармақ қалмағанша жалғаса береді. Айналып өтетін бұтақ қалмаған кезде іздеу тоқтайды.

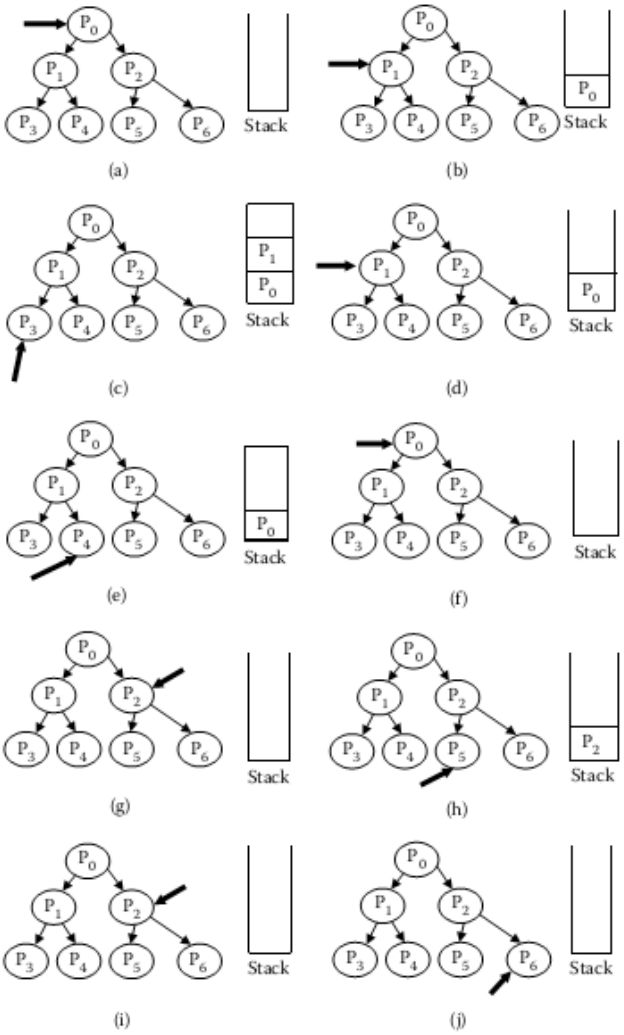
2.14-мысал

2.11-суреттегі мысалды қарастырайық. Көрсеткіш әуелі Р0 түйінін айналып өтеді. Бұл кезде стек бос. Алайда Р0 түйінінде сол жақ және оң жақ тармақ бар. содан кейін Р1 түйіні сол жақ еншілес түйін Р0-дан айналып өтеді. Ал Р0 түйіннің адресі стекте сақталады. Бұл кейін Р2 түйіннің айналып өтуі үшін жеңіл болады. Келесі түйін Р3- Р1-түйіннің сол жақ бұтағы айналып өтеді. Ал Р1 түйіннің адресі стекте сақталады. Ал стек былай болады: <адрес (Р1), адрес (Р0)>. Р3 түйіні жапырақты түйін болғандықтан, Р1 түйіні стектен Р3 түйіннен айналып өткен соң алынады және оң жақ тармақ - Р4түйіні айналып өтеді. Р1 түйінінен адресті алғаннан кейін стек келесі түрде болады: <адрес (Р0)>.

Р1 түйінінің басқа бұтақтары болмағандықтан Р4 түйінін айналып өткен кезде ешқандай ақпарат стеке орналаспайды. Р4 түйінін айналып өткен соң Р0 түйінінің адресі стектен алынып, стек бос болып қалады. Соған қарамастан, Р0 түйінінің оң жақ бұтағы айналып өту үшін қалады. Стек бос болып қалады. Өйткені Р2 түйінінің оң жақ бір деңгейлі элементі жоқ Р2 түйінін айналып өткен соң Р2 түйінінің адресі стекте сақталады ал Р5 түйіні айналп өтеді. Бұл кезеңде стек мынадай болады: <адрес Р2)>.

Р5 түйінін айналып өткен соң стек алынады. Р6 түйіні, Р2 түйінінің оң жақ бұтағы айналып өтеді де, стек бос болып қалады. Р6 түйінін айналып өткен соң стек босап қалады. Бір де бір қосымша түйін айналып өтуді қажет етпегендіктен іздеу тоқтатылады.

Тереңінен іздеудің негізгі артықшылығы стектің өлшемі ағаштың тереңдігімен шектелген, бұл уақытта ағаштың фокусталған бөлігінде ғана өтеді. Соған қарамастан, тереңнен егжей-тегжейлі іздеу цикл кезінде білгілі бір уақытқа тұрып қалуы мүмкін. Циклді анықтау үшін біраз уақыт кетеді және барлық барған түйіндерді сақтау үшін біраз жадының шығыны болады және барған түйіндердің арасында түйінге тиістілік тексеріледі. Хэш-кесте циклдердің тиімді идентификациялау үшін пайдаланылады. Соған қарамастан, мәліметтердің көп құрылымы үшін айналып өткен түйіндерді сақтауға кеткен жадының қосымша шығындары өте көп.



Stack – Стек.

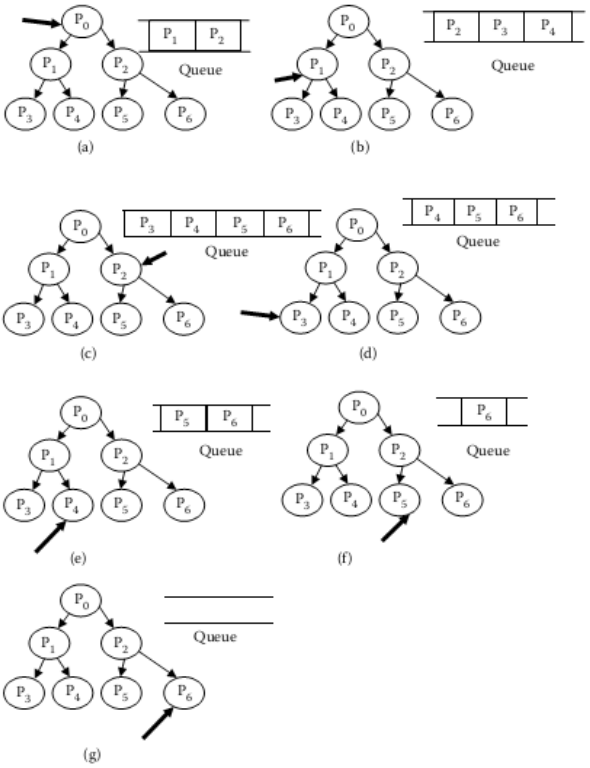
2.3.7.2 Енінен іздеу

Енінен іздеу (2.12. суретті қараңыз) ағашты сол жақтан оң жаққа қарай деңгейден деңгейге өту арқылы жүзеге асады және баруға арналған бұтақ түйіндерін сақтау үшін тізімді пайдаланады. Түпкі түйін (соңғы

түйін) бастапқы кезеңге кезекке тұрады. Содан кейін кезектен бір элемент жойылған сайын жойылған түйіннің барлық бұтақтары артқы бөлікке кезекке тұрады. Бұл үдеріс кезек босап қалғанша жалғаса береді.

2.15-мысал.

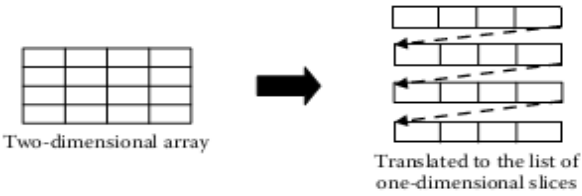
2.12-суретте енінен іздеу көрсетілген. Көрсеткіш әуелі P0 түпкі түйінге барады және P1 және P2 түйіндерінің адресларын кезекке қояды. Содан кейін кезектен P1 түйіннің адресі жойылады және P3 және P4 бұтақтарының адреслары кезекке қойылады және P1 түйініне барады. Содан кейін ол P2 түйінін кезектен жойып, P5 және P6 бұтақтардың адреслары кезекке қояды.



Queue – Кезек.

2.12-сурет. Енінен іздеу мысалы.

P3, P4, P5, және P6 түйіндері жапырақ түрінде саналғандықтан, олар айналып өткенде ешнәрсе қойылмайды. Соңғы P6 жапырақ алынғаннан кейін барлық түйіндер айналып өткен соң, кесте бос болып қалады.
Енінен іздеудің артықшылығы ол ағашты бір деңгейді бір рет қана айналып өтеді. Енінен іздеудің көптеген кемшіліктері де бар. Еніне іздеу кездегі қосымша жады өте үлкен. Ағашты айналып өту кезінде ол бұрынғы деңгейдегі терминалдық емес түйіндердегі барлық түйіндерді қамтып алады. Бұл санның көп болғаны соншалық, ол ағаштың теңгерімі кезіндегі жапырақтарыдң жартысындай болады. Алайда, егер тізім жадысының қосымша шығындарын тиімді басқару мүмкін болса, еніне іздеуді бағананы тиімді айналып өту үшін пайдалануға және жадыны хипте пайдаланудың тиімді сызбаларында пайдалануға болады.



Two-dimensional array - Екі өлшемді массив; Translated to the list of one-dimensional slices - Бір өлшемді кесектерінің тізіміне аударылған.
2.13-сурет. Екіөлшемді матрицаны бір өлшемді жадыға көшіру.

2.3.8 Бірөлшемді жадыдағы мәліметтер құрылымының бейнесі.
Бағдарлама құрылымдарды (жолдар жиынымен кортеж) мәліметтердің күрделі нысандарын моделдеу үшін немесе көпөлшемді массивтерді мәліметтер нысандарының жиынын моделдеу үшін пайдалану мүмкін. Массивтер *статистикалық және динамикалық* болады. *Статистикалық массивтер* бұл шақырту үдерісі кезінде бір рет, содан кейін көп жадының бөлінуін талап етпейтін бөлінген жады. *Динамикалық массивтердің* өлшемі бағдарламаның орындалуына негізделіп, атқару кезінде өзгереді. Массивтер мен құрылымдарға қосымша C++, Java немесе C # сияқты нысанды-бағдарлы тілдерде және бағдарламалаудың басқа заманауи тілдерінде динамикалық нысандар болады. Осы құрылымдардың барлығы ОЖ-да бейнеленген. ОЖ жады адресі бойынша индекстелген бір өлшемді массив ретінде абстракциялануы мүмкін.
ОЖ-да көп өлшемді массивтерді бейнелеу үшін көпөлшемді массивтер бірөлшемді массивке аударылуы керек, ал ОЖ-дағы адреслар мәліметт-

тер құрылымының жалпы адресіне (бастапқы адреске) қосылуы мүмкін салыстырмалы адреске элементтің индекс мерзімін аударатын теңдеудің көмегімен есептелуі мүмкін. Екі өлшемді массивті аудару үдерісін түсіну үшін біз әрбір қатарды (бағананы) жеке-жеке бірінен кейін бірін алып, оларды бірінен кейін бірін 2.13-суретте көрсетілгендей бір өлшемдегі жолаққа орналастырамыз.

Өлшемдері $M \times N$ екі өлшемді массив үшін a(*i, j*) элементінің адресін білу тапсырмасы 2.1 теңдеуде берілген. Мұнда массив индексі 0-ден басталады. Оң жақтағы бірінші қосылғыш негізгі адрес - [0, 0] адрес болып табылады. Ал екінші қосылғыш орнын ауыстырады. Екінші қосылғыштың бірінші элементі *i*- қатар ағымдағы қатарға дейін болады, ал екінші қосылғыштың екінші элементі ағымдағы қатардың элементтерінің орын ауыстыруын көрсетеді.

$$\text{Address} = \text{Address}\left(a[0,\ 0]\right) + \left(i \times N + j\right) \times \text{Bytes in one element} \qquad (2.1)$$

Address- Мекен-жай; Bytes in one element-Бір бөлшектегі бит.

2.16 мысал

M бүтін санның екі өлшемді матрицасын көрсетеміз. Оның өлшемі 5×6 (5 қатар және 6 бағана). Біз m[2, 4] облысында, №2 қатардағы, №4 баға-надағы элементтің адресін табуымыз керек. Индекс 0-ден басталады деп есептейміз, ал негізгі адрес m [0, 0] - 1000, және 32-биттік бүтін санға төрт байт керек. 2.1 теңдеуді пайдаланып, m[2, 4] орналасқан жері 1000 + (2 × 6 + 4) * 4 = 1064. m[2, 4] элементе сақталған бүтін сан 1064 адресдан басталып, 4 байтқа тең болады. Бұл тұжырымдама бір өлшемді матрицаға келгенше аз екі өлшемді матрицалардың тізбегі түрінде көп өлшемді матрицалар көрінісі ретінде біртіндеп кез келген туынды екі өлшемді матрицалардың

$$\text{Address} = \text{Address}\left(m[0,\ldots,0]\right) + \left(\left(i_1 \times D_2 \times \ldots \times D_N\right) + \ldots + \left(i_{N-1} \times D_N\right) + i_N\right) \times \text{Bytes in one element} \qquad (2.2)$$

Мекен-жай=мекен-жай(m[0,…,0])+((i_1*D_2*…*D_N)+…+(i_{N-1}*D_N)+i_N)*
*бір элементтегі байттар

Бірнеше жолағы бар (жолақ 1, жолақ 2,…, жолақ*N)* «құрылым» сияқты композициялық құрылым алу үшін әр жолақтың орын ауысуы компиляция кезінде есептеледі. Бұл жылжыту "*i*"-жолаққа қолжетімді болу үшін негізгі адреске қосылады.

2.3.9 Хэш-кестелер

Компиляция кезінде және бағдарламаны орындау барысында мәліметтерді алу өте маңызды. Статистикалық массивтер мәліметтер элементтерін алу үшін тұрақты әрекет етіп тұрғандықтан іздеу үшін кіріс мәліметтер берілген кезде мәліметтер элементін алуға жарамайды. Мәліметтер элементі индекс болып табылмайды бірақ жазба кілті болады. Хэш-таблицалар тұрақты уақытта кілт негізіндегі іздеуді қолданады. Ол кезде индекс мәніне <*кілт*> кілтті көрсететін *f* хэш-функциясын пайдаланады: *f*(<*кілт*>) мәліметтер элементі сақталатын сақтау индексіне тең. Хэш-кестелер бағдарламалау тілдерін моделдеу кезінде іздеуге тиімді болғандықтан пайдаланылады.

Хэштеуге негізделген іздеу кезіндегі ең басты шектеу *кілттердің соқтығысу үдерісі* - екі кілттің индекстің бір мәнінде көрініс табуы болып табылады. Соқтығысу индексін өңдеу үшін кейбір әйгілі әдістер:

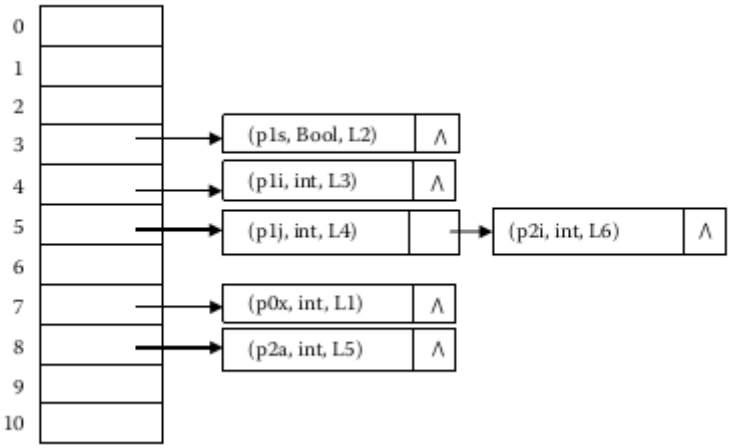
(1) кесте өлшемін жай сан ретінде алу және осы жай санды индексті алу үшін хэш-функцияда қолдану. Хэш-функцияларды жай сандарға бөлу кілттерді хэш-кестелерге соқтығысу ықтималдығын төмендете отырып біркелкі таратады; (2), бірнеше соқтығысатын кілттерді өңдеу үшін белгілі бір индекске бекітілген байланысқан тізімді пайдалану; және (3) соқтығысу кезінде хэш-кестелерді кеңейту және қайта хэштеу немесе хэш-кестеде элементтердің жинақталуы шектік мәннен асып кетеді.

2.17 мысал

2.14 сурет 6 айнымалыдан және олардың белгілерінен тұратын жиынды сақтау үшін пайдаланылатын өлшемі 11 хэш-кестенің мысалын көрсетеді. Үш процедура бар: *p0, p1*, және *p2*. *p0* процедурасының *x* айнымалысы бар; *p1* процедурасының *s*, *i*, және *j* айнымалылары бар; *p2* процедурасының *a және i* айнымалылары бар. Атаулардың сәйкеспей қалуын болдырмау үшін прцедураның атауы префикс ретінде айнымалымен бірге жазылады. Мысалы, *p0* –дегі *x* айнымалысының кілті *p0x*; *p1* –дегі *s*, *i*, және *j* айнымалылары үшін кілт *p1s, p1i,* және *p1j* сәйкесінше; *p2*-дегі *a* және *i* айнымалылар үшін кілт *p2a* және *p2i*. Енді барлық кілттер сәйкеспейтін аттары бар айнымалылар үшін де бірегей болды. Мәліметтердің әрбір элементі осы үштікте көрініп тұр (*идентификатор, типі, жады ұяшығы*). Алты үштік мыналар: {(*p0x, int, L1*), (*p1s, Bool, L2*), (*p1i, int, L3*), (*p1j, int, L4*), (*p2a, int, L5*), және (*p2i, int, L6*)}.

Кілтті бейнелейтін функция бізге (1) кілттегі '*a*' -дан '*z*' –ке дейінгі таңбасыдардың реттік мәндерін солдан оңға дейін суммалау үшін керек; (2) кілтке цифрлар мәнін қосу; (3) жай сан болып табылатын кестенің 11 өлшеміне суммаларды бөлу үшін керек. Байланысқан кілттер кілттердің

соқтығысуын болдыру үшін пайдаланылады.



2.14-сурет. Сәйкессіздіктерді шешу үшін байланысқан тізімді хэш-кестенің мысалы.

Мысалы, *p0x* атауында үш таңбасы бар. Ол реттік мәні 16 болатын 'p', цифры, мәні 0 болатын '0' цифры және реттік саны 24 болатын 'x' таңбасы. Осы мәндерді қоссақ 40 мәнін аламыз, ал өлшемі бойынша 11 кестеге бөлсек, 7 индексінің мәнін аламыз. *p1s* кілті 3 индексінің мәнін бейнелейді. *p1i* кілті 4 индексінің мәнін бейнелейді. *p1j* кілті 5 индексінің мәнін бейнелейді. *p2a* кілті 8 индексінің мәнін бейнелейді. Ал *p2i* кілті 5 индексінің кілтін бейнелейді. *p1j* және *p2i* кілттерінің соқтығысуы байланысқан тізімдерді пайдаланып шешіледі.

2.4 ЕСЕПТЕУ БАРЫСЫНДА АБСТРАКТІЛІ ҰҒЫМДАР ҚОЛДАНУ

Бұл тарауда бағдарламалау тілдерін жасауда пайдаланылатын кейбір анықтамалық түсініктер сипатталады. Бұл түсініктердің көп бөлігі мәліметтер құрылымы және дискретті құрылымдар сияқты негіздер мен бағдарламалау курсының алдыңғы әртүрлі мәнмәтіндерінде кездескен. Осы негізгі түсініктер курс бойында көп рет қолданылды және бағдарламалау тілдерін жасауда және болашақта пайдалануда нақты түсіндірмелерді қажет етеді.

Бағдарлама – бұл *мәндік нұсқаулықтардың тізбегі.* Әрбір осындай нұсқаулық *сөйлем* немесе *өрнек* деп аталады. Әрбір өрнек көп тілдерге бөлушімен немесе жолдарды кейбір тілдерге аударумен аяқталады. *Жадыға сақталған сөз* кәдімгі ауызекі телге ұқсас бағдарламала тілінің бір бөлігі болып табылады. Жадыға сақталған өзіндік мәні бар екі сөз немесе екі айнымалы немесе кез келген екі нысан бір-бірінен бос орын, үтір, нүктелі үтір, екі нүкте, бір қатарда тұру немесе жадыға сақталған сөздермен алшақ тұрады. Бұл сепараторлар *бөлгіштер* деп аталады.

Бағдарламаларда *литералдар, r-мән, l-мән, идентификаторлар, анықтауыштар, айнымалылар, меншіктеу операторлары, нұсқаулықтар, командалар, өрнектер, таңбалар қатары, типтерді мәлімдеу, операторлар, шақырту үдерістері, параметрлер, белгілер және секвенсорлар болады. Литерал* – бұл өте кіші бөліктерге қосымша бөлінене алмайтын және қайта қаралмайтын қарапайым жазба. Литералдар *есептелген константалар, константалар және дескрипторлар* ретінде де белгілі. Мысалы, сан литерал, таңба литерал, тырнақшадағы «Арвинд» есімі литерал. Алайда, қатарлар литерал емес: қатарлар таңбалардың тізбегі болады; арнайы ақиқат және жалған мәндері литерал емес. Өйткені, оларды қайта қарастыруға болады.

R-мән – бұл меншіктеу нұсқаулығының оң жағына пайдалануға рұхсат берілген мән. L-мән – бұл айнымалымен немесе массив элементімен немесе құрылым ішіндегі жолақпен байланысқан жады облысы. *Идентификатор* басқа нысанмен байланыста болуы мүмкін бағдарламада қолданылатын таңбалы атау. Мысалы, бағдараламалар атауы, функциялар атауы, айнымалылар атауы, константа атауы, қолданушы типі – бұлар идентификаторлар. *Анықтама* блок шегіндегі немесе үдерістегі сәйкес мәнді білдіреді. Анықтама компиляция кезінде сәйкес мәнмен алмастырылады.

Айнымалы ақпараттық блокпен байланысты. *Ақпараттық блок облыстық тип сияқты нақты облыстағы жай мәнмен, күрделі нысанмен немесе мәнммен абстрактылы облыста* болуы мүмкін. *Нақты мән* бағдарлама жұмыс істейтін негізгі мән. Айнымалы тип типі туралы ақпаратты қамтиды. Мысалы, айнымалы тип абстрактылы доменнің *толық есептелген айнымалымен* байланысуы мүмкін. Айнымалы типтің түсінігі полиморфты –тарауда түсіндіріледі. Осы сәттен бастап, және әрі қарай біз айнымалы ретінде нақты мәндердің мәнін алып жүруші және ақпараттық типті алып жүруші тілдердегі айнымалы типін түсінеміз. Ішкі айнымалы компиляция кезінде жады ұяшығында бейнеленеді. Сәйкес жады ұяшықтары нақты мәнді немесе хип нысанындағы көрсеткішті алып жүре алады.

Меншіктеу операторы *x = y + 4* өрнектегі пайымдаудың оң жағында айнымалының жады ұяшығындағы сақталған мәнді есептеген, алгебралық өрнекті бағалаған және алынған мәнді пайымдаудың сол жағындағы айнымалының жады ұяшығына сақталған мәнді жазатын бағдарламалаушының пайымы. Жоғарыда келтірілген операторда айнымалының мәні сақтауыштан саналады, *value-of*(*y*) + 4 өрнегі есептеледі және өрнекті есептеуді мәні x идентификаторына сай келетін жады ұяшығына сақталады.

Команда дегеніміз кем дегенде соған құрылған бір меншіктеу операторы бар басқару абстракциясы. Меншіктеу операторын пайдалану жаңа мән кем дегенде командадағы бір жады ұяшығына жазылады дегенді білдіреді. Командадан айырмашылығы көрініс өзіне тапсырма алмайды; көрініс жады ұяшығындағы мәнді есептеп, оны бағалайды.

Қатар дегеніміз таңбалар тізбегі. Бос қатарда бір де бір таңба болмайды. *Типті жариялау* нысандарды шынайы мәселелерде моделдеу үшін мәліметтер қолданушы көрсеткен әдіспен сақтала алатындай қолданушы анықтаған мәліметтер абстракциясын құру үшін пайдаланады. Соған қарамастан типті жариялау өздігінен жады ұяшығын құрмайды. Жады ұяшықтары тек айнымалы жарияланғанда ғана құрылады.

Оператор арифметикалық немесе логикалық оператор болады. Оператор *унарлы (бір орынды)* немесе *бинарлы (екі орынды)* болады. Унарлы операторларда бір операнды бар, ал бинарлы операторда екі операнды бар. Арифметикалық операторлар '<' (кем), '>' (артық), '= <' (кем немесе тең), '> =' (артық немесе тең), '==' (тең), '<>' (тең емес), '= / =' (Прологтағыдай тең) болуы мүмкін. Сонымен қатар, қатардағы белгіні қамтамасыз ету үшін "+" және "-" унарлы операторлары бар. Логикалық ЖӘНЕ(∧), логикалық НЕМЕСЕ (∨), шығарушы НЕМЕСЕ (⊕) және шығыс операторлары сияқты бинарлы логикалық операторлар екі Буль өрнектерін жоғарыда айтылғандай ақиқат мәнін пайдаланып біріктіреді, ал унарлы оператор аргумент ретінде бір Буль өрнегін қабылдамай, ақиқат немесе жалған деген пікірді қайтарады.

Арифметикалық және Буль операторларының *оператор артықшылығы* бар: көріністі алғаннан кейін кейбір операторлар бірінші болып оңайланады. Унарлы операторлардың бинарлы операторларға қарағанда артықшылығы көп; бинарлы операторда көбейту мен бөлу, қосу мен алуға қарағанда жоғары артықшылыққа ие. Дәл солай, «жоқ» (-) Буль унарлы операторның артықшылығы жоғары, одан кейін логикалық «және» (∧), содан кейін логикалық НЕМЕСЕ (∨), содан кейін салдар (→).

2.4.1 Өзгергіш айнымалылар өзгермейтін айнымалыларға қарсы

Императивті бағдарламалау парадигмасында айнымалылар өзгермелі болады: қолданушы *меншіктеу операциясын* пайдаланып, өз қалауымен айнымалыға сәйкес келетін жады ұяшығындағы мәнді бұза отырып жаңарта алады. Мысалы, $x = y + 4$. Императивті бағдарламалау парадигмасынан айырмашылығы декларативті бағдарламалау парадигмасы жады ұяшықтарын пайдалану деңгейінде мутациялауға мүмкіндік бермейді. Айнымалы мәнмен тек бір рет қана байланыса алатын *өзгермейтін* мән болып қалады.

Айнымалыларды өңдеудің екі әдісінің де артықшылығы мен кемшілігі бар. Жады ұяшығының бүлініп жаңаруының негізгі артықшылығы *жадыны қайтадан пайдалану* болып табылады. Мысалы, қайталап қолдануда FOR циклының индексті айнымалысы қолданылады. Жады ұяшығының бүлініп жаңаруының негізгі кемшілігі (1) өткен мәндердің жоғалуы және (2) 4-тарауда сипатталғандай бағдарламаның негізгі математикалық қасиетінің бұзылуымен пайда болған бағдарламаның қажетсіз әрекетінің нәтижесінде пайда болған *жанама әдістер*. Егер бұрынғы мәндерді қайта қалпына келтіру мүмкін болмаса, онда шешім бар болған күннің өзінде шешім алудың басқа мүмкіндіктерін байқап көру үшін мәселені шешудің кез келген әдісі оны қайтара алмайды.

Бір рет қана белгіленген *өзгермейтін айнымалылардың* артықшылығы: (1) жанама әдістер туындататын бағдарламаның қажетсіз әрекеттерін қысқарту мүмкіндігі және (2) 9-тарауда қарастырылған логикалық бағдарламалау сияқты балама шешімдерді пайдалану үшін соңғы мәнді пайдалану. Негізгі кемшілік, мәліметтер элементтерін мәліметердің үлкен құрылымының қатысуымен өңдеген кезде, әсіресе итеративті әдіспен,*жады жарылысы* нәтижесінде жады қайта пайдалану мүмкін болмайды. Бірнеше парадигманы қолданатын қазіргі көптеген тілдер екі қадамның да артықшылықтарын артығымен пайдалану үшін өзгеретін және өзгермейтін айнымалыларды қатар пайдаланады.

2.4.2 Көріну облысын байланыстыру және ережесі

Байланыстыру екі элементті немесе белгілері сәйкес келетін элементті байланыстыру. Мысалы, айнымалының аты жады ұяшығына байланысуы мүмкін, жады ұяшығы мәнмен, идентификатор мәнмен, ал айнымалы типппен байланысты болуы мүмкін. осылайша, операторлар тобы идентификатормен байланысып, *функция немесе үдеріс* деп аталады. *Көріну облысының ережесі* айнымалыны пайдалануды және оның көріну дәрежесін анықтайды. Айнымалы көрінеді және өз облсының шегінде пайдаланылады: оның облыстан тыс ешқандай мәні жоқ. Об-

лысқа байланысты байланысу уақытша немесе тұрақты болады.
Көріну облысы ережесінің екі түрі бар: *статистикалық және динамикалық.* Аттары айтып тұрғандай, *көрінудің статисткалық облысы* блок шегі, ішкі бағдарлама немесе функция (немесе бағдарламалау нысанды-бағдарлы тілдердені әдіс) шегі, класс шектері сияқты бағдарлама құрылымдарына негізделеді. Ол ішкі бағдарламаның шақырту шаблонымен өзгермейді. *Көрінудің динамикалық облысы* ішкі бағдарламаның шақырту шаблонына негізделген айнымлының масштабын өзгертеді. Көрінудің статистикалық облысының мысалы 2.18-мысалда, көрінудің динамикалық облысының мысалы 2.19- мысалды келтірілген.

```
Бүтін сандар x, y, z;
main ( )
{x = 4; y = 10; z = 12;
{бүтін сандар temp, z;
temp = x; x = y; y = temp; z = 5;}
print(x, y, z);
}
```

2.15-сурет. Айнымалылардың көріну аймағындағы статистикалық облыс мысалы.

2.18-мысал

2.15-суреттегі бағдарламаның ішкі блокта үш *x, y*, және *z* жаһандық айнымалысы және ішкі блокта екі жергілікті *temp* және *z* айнымалылары бар. Жаһандық айнымалылар бүкіл бағдарламаны алып жатыр, ал жергілікті айнымалылардың көлемі олар жарияланған блокпен шектелген. *z* айнымалысының атау қайшылығы пайлануға жақын айнымалыға басымдық беру арқылы шешіледі. Басып шығару операторындағы *z* айнымалысы *z* жаһандық айнымалысына жатады, ал ішкі блоктағы *z* айнымалысы *z* жергілікті айнымалыға жатады. Осы екі айнымалы да мәліметтердің бөлек нысандары болып табылады және жадының әртүрлі жерінде көрініс тауып, қызмет ету мерзімері де әртүрлі болады. *z* айнымалысы ішкі блоктың облысынан тыс жерде басылып жатқанда *z* = 12 жаһандық айнымалының мәні де басылып жатады. Осы айнымалылардың масштабтары туралы ақпарат бағдарламаның құрылымына қарау арқылы анықталады және оның бағдарламаның шақырту шаблонымен ортақ ешнәрсесі жоқ.

2.19-мысал

2.16- суреттегі бағдарлама динамикалық облыстың ережесін қолдана-

ды. Бағдармада екі айнымалысы бар *sum* функциясы бар: *x* айнымалысы *sum* функциясының шегінде жарияланған, ал *y а*йнымалысы – еркін кіру, яғни у айнымалысы *sum* функциясында жарияланбаған. Х айнымалысы мәнін параметрлерді беру механизмінен алады. Соған қарамастан, у айнымалысы байланыстырудың сәйкес мәнін анықтау үшін шақырту үдерісінің тізбегі үшін бөлінген жады облысында дәл сондай атты іздейді.

Негізгі бағдарлама екі блоктан тұрады. Бірінші блоктың *y* және *z*, екі жергілікті айнымалылары бар және у айнымалысын пайдаланып, *sum* функциясын дәлелмен шақырады. Екінші блок *w, y*, және *z* үш жергілікті айнымалылардан тұрады және *sum* функциясын *z* нақты параметрімен шақырады.

```
integer sum(integer x);
return (x + y);
main ( )
{ {integer y, z; y = 4; z = 5; sum(y);}
  {integer w, y, z; w = 4, y = 5; z = 6; sum(z);}
}
```

2.16-сурет. Айнымалылардың көрінуінің динамикалық облысына мысал.

sum(y) функциясының бірінші шақыртылымы 8 мәнін қайтарады. Өйткені х формалды параметрі у нақты параметрінің мәніне ие болады, ал у еркін айнымалысы *sum* функциясында негізгі бағдарламаның бірінші блогында жарияланған у айнымалысымен динамикалық байланыста. *sum* функциясын екінші рет шақырту 11 мәнін қайтарады. Өйткені, *sum* функциясындағы х формалды параметрі негізгі бағдарламадағы негізгі *z* параметрінің мәнін алады. Ал у еркін айнымалысы негізгі бағдарламаның екінші блогында жарияланған у айнымалысымен байланысты *sum* функциясында 5 мәнін алады. Еркін айнымалыларды осылай байланыстыру және оны ішкі бағдарламаның шақырылған шаблонындағы тізбектегі соңғы жарияланымға байланыстыру айнымалылыардың көрінуаймағын динамикалық етеді.

2.4.3 Айнымалылар типі

Айнымалылар көрінуережесі мен өмір сүру уақытының негізінде олардың көрінуіне байланысты жіктелген. Қызмет ету мерзімі бар айнымалы *жаһандық айнымалы* деп аталады; қызмет ету мерзімі және үдеріс

әрекетінің облысы бар айнымалы *жергілікті айнымалы* деп аталады; ал салыныған үдерістік құрылымның сыртқы көріну деңгейінде жарияланған, бірақ салынған үдерістердің бірінде пайдаланылатын айнымалы *жергілікті емес айнымалы* деп аталады. Блок операторлар түрінде ұйымдастырылған бағдарлама талаптарын орындайтын блок-құрылымдық тілдерде айнымалылар блокқа қатысты жергілікті болуы мүмкін. Айнымалылалырдың әрекет ету облысы олар жарияланған блок шегінде шектелген.

Жергілікті және жергілікті емес айнымалылар немесе жаһандық және жергілікті айнымалылардың аттары арасында біраз қарама-қайшылықтар болуы мүмкін. Бұл қайшылық егер жергілікті айнымалы дәл сол атаумен аталған болса, жергілікті немесе жаһандық айнымалыны *көлеңкелеу* (көрсетпеу) әдісімен шешіледі. Көлеңкелеу тұжырымы егер олар салымның әртүрлі деңгейінде жарияланған болса, жергілікті емес айнымалылардың көріну аймағын өңдеуге жатады. Мұндай жағдайда салым деңгейінде жарияланған айнымалы айнымалының дәл сола атауымен аталған жақын ағымдағы үдерісте көрінеді, ал бірдей атаумен аталған айнымалылар көлеңкеленеді.

Айнымалылар *статистикалық немесе динамикалық* болады. *Статистикалық айнымалылар* компиляция кезінде тіркелген жадымен бөлінеді және бұл жады ұяшығы бағдарламаны орындау барысында өзгермейді, 5-тарауда сипатталғандай, статистикалық таралудың артықшылығы ол қандай да бір көрсеткішсіз жадыға тез қолжетімді. *Статистикалық* айнымалылар- статистикалық айнымалылар облысы барлық бағдарлама болған кезде *жаһандық айнымалылар статистикасы* немесе бағдарлама облысы ол жарияланған облыс шегімен шектелген болса, *статистикалық жергілікті айнымалы* болады. Соған қарамастан, айнымалылардың көріну шегіндегі жергілікті облысқа байланысты мән үдеріс шегінен тыс жерде қолжетімді емес және үдерістің келесі шақыртылымында ғана қолжетімді болуы мүмкін. *Динамикалық айнымалылар* – орны бағдарлама орындалып жатқанда анықталатын және басқару стекына орналасатын айнымалылар. Жергілікті айнымалылардың көптеген жарияланымдары динамикалық жергілікті айнымалылар болып табылады және 5-тарауда сипатталғандай соның негзініде тілдерді қолдану үшін стеке орналасады.

Жаһандық айнымалылар да статистикалық айнымалылар сияқты жадыға тиімді қолжетімді орналасқан. Қалған айнымалылардың барлығы бағдарламалық блоктардың массивтеріне салынған, сондай-ақ блоктарынан алынған жағдайға байланысты шекті көріну аймағы бар. Жергілікті айнымалылар олар жарияланған блок шегінде ғана көрінеді. Жер-

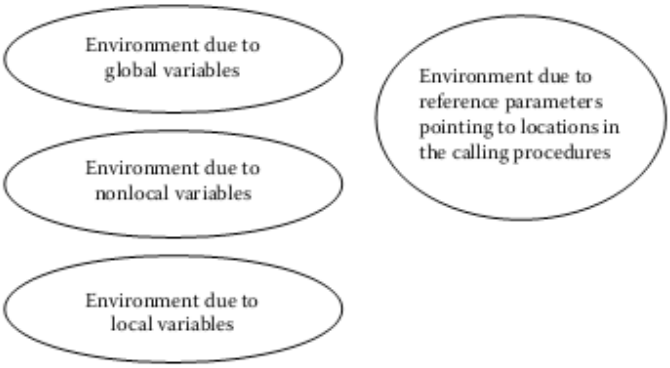
гілікті айнымалылар үдеріс деңгейінде үдеріс шегінде ғана көрінеді. Нысанды-бағдарлы тілдерде айнымалылар кластарға бөліне алмайды, олар класстың барлық түрінің арасында таралады. Мұндай айнымалылар *айнымалы класстар* деп аталады. Айнымалы кластардан өзге жасалған нысан үшін мүмкіндігі шектелген *нұсқа айнымалылары* бар. Айнмалылардың басқа жіктелуі алдында айтылғандай *өзгергіштікке* негізделген. *Өзгергіш айнымалы* қанша рет болса да қайта құрылып, жаңара береді. Ал *өзгермейтін айнымалының тек бір рет белгіленген* қасиеті бар. ол мәнмен байланған екен, ешқашан өзгермейді және бағдарламалау жолымен шешіп алынбайды. Соған қарамастан, өзгермейтін айнымалылар қайтарумен іздеу, логикалық бағдарламалау парадигмасын қолдану сияқты қолдану механизмдерімен шешіліп алынуы мүмкін.

2.4.4 Қабықша және Сақтаушы

Бағдарламаны орындауда екі абстарктылы құраушы пайдаланылады: *қабықша және сақтаушы. Қабықша* – бұл есептеу жүретін аймақ. Қабықша түрлер жұбының жиынтығы (жады ұяшықтарындағы айнымалылар аты ↦айнымалылар атрибуты) немесе константаны жариялау (аттар идентификаторы↦мән) жағдайында анықталады. Жаңа жарияланымдар қабықшаны (1) түр идентификаторының жаңа байланыстырғышын құру↦ жады ұяшығы немесе мән идентификаторы және (2) жергілікті айнымалылармен қайшылықты атауы бар жергілікті емес немесе жаһандық айнымалылардың байланысын көлеңкелеу арқылы өзгертеді. Ішкі бағдарламаны шақырғаннан кейін қабықша өзгереді. 2.17-суретте көрсетілгендей ішкі бағдарламаның ағымдағы қабықшасы (1)жергілікті айнымалыларды жады ұяшықтарына байланыстырудан (2) жергілікті емес айнымалыларды жады ұяшықтарына байланыстырудан (3) жаһандық айнымалыларды жады ұяшықтарына байланыстырудан және (4) сілтеме параметрлерін пайдалана отырып қолжетімді шақырғыш бағдарламалармен тізбектегі жады ұяшығынан тұрады. Шақырылған ішкі бағдарлама аяқталғаннан кейін қабықша оларды құрған шақырғыш ішкі бағдарламаның шегінен тыс өмір сүре алатын динамикалық нысандар мен рекурсивті мәліметтердің құрылымынан жасалған қосымша қабықшалы ішкі бағдарламаны шақырылған ішкі бағдарлама аяқталған соң, қабықша оларды құрған ішкі бағдарлааның өмір сүру уақытының шегінен артық мерзімі бар динамикалық нысандар мен мәліметтердің рекурсивті құрылымдарымен жасалған қабықшамен толтырылған ішкі бағдарламаны шақыруға болатын ортаға қайтып келеді.

Сақтаушы – түрлердің байланған жиыны (жады ұяшығы ↦ мән) және

ол меншіктеу операторы мәнді жады ұяшығына жазып, мән жүктелген кезде немесе мән параметрлер берілісінен өзгерген кезде өзгереді.



Environment due to global variables – Қоршаған орта жаһандық айнымалыларға байланысты; Environment due to reference parameters pointing to locations in the calling procedures – Қоршаған орта шақырушы рәсімдердегі орындарға көрсететін анықтамалық параметрлерге байланысты; Environment due to nonlocal variables – Қоршаған орта жергілікті емес айнымалыларға байланысты; Environment due to local variables - Қоршаған орта жергілікті айнымалыларға байланысты.

2.17-сурет. Ішкі бағдарламада атқаруға арналған қабықша.
Иелену операциясы келесіде сақталған мәндерді жаңартуы мүмкін:

1. Жергілікті, жергілікті емес немесе жаһандық айнымалылармен байланысты жады ұяшықтары.
2. Параметр сілтеме параметрі ретінде берілген кездегі шақырылған ішкі бағдарламалардағы айнамалылардың жады ұяшықтары.
3. Шақырылған бағдарламадағы жүктемедегі формалды параметрлердің жады ұяшықтары.
4. Шақырылған ішкі бағдарламаларда есептелген нәтижелерді беру жолымен өзекті параметрлердің жады ұяшықтары.
5. Мәліметтердің рекурсивті құрылымдары немесе мәліметтердің динамикалық нысандарының жады ұяшықтары.

Үдеріс шақырылғаннан кейін, қабықша да, сақтауыш да шақырылған

үдерістің ішінен өзгеріс алады. Сақтауыш (1) параметрлер мәнінің өтуінен немесе (2) жергілікті айнымалылардың жүктелуі себепті өзгереді. Шақырылған үдеріс аяқталған соң жергілікті айнымалылар (шақырылған ішкі бағдарламада) құрған орта жоғалады; шақырыған ішкі бағдарламаның мұрағат ортасы қалпына келіп және шақырылған ішкі бағдарламаның қызмет ету мерзімінің шегінен тыс мерзімі бар мәліметтердің рекурсивті құрылымдары мен динамикалық нысандар үшін құрылған жалпы айнымалылардың қабықшаларымен және жаңа қабықшалармен толықтырылады.

Сақтаушы сілтеме параметрі ретінде берілу кезінде жаһандық айнымалылардың, жергілікті емес айнымалылардың және шақырту үдерісіндегі айнымалылардың жойылып жаңаруы себепті осы айнымалылардың жаңа мәнді меншіктеуі шақырылған үдерістің сақтаушысын жаңартып отырады және бұл өзгерістер шақырылған үдеріс аяқталғаннан кейін де сақталып қалады. Әрі қарай біз сақтаушыдағы қандай өзгеріс бағдараламаның қарастырылмаған бұзу әрекеттерін тудыратынын қарастырамыз.

2.4.5 Функциялар мен Үдерістер

Әрі қарай біз өрнек пен командатар, сондай-ақ функциялар мен үдерістердің арасындағы айырмашылықтарды қарастырамыз. Өрнек жадыдағы бір немесе бірнеше мәнді оқып, оны бағалайды. Соған қарамастан, ол есептеу мәнін кері жадыға жазбайды. Бұл дегеніміз, өрнекті есептеу кезде сақтаушы өзгеріссіз қалады деген сөз. Сақтаушыдан айырмашылығы команда мәліметтерді сақтаушыны өзгерте отырып жады ұяшығына жазады. Мысалы, *x+y+* 4 өрнек болып табылады, ал меншіктеу нұсқаулығы *z= x+ y+* 4 команда болып табылады. *x+y+ 4* өрнегінде процессор жадыдағы *x* және *y* айнымалыларының мәнін есептейді. Содан кейін өрнекті есептейді. Салыстыру үшін *z=x+y+4* меншіктеу нұсқаулығы *x+ y+ 4* өрнегінің есептеу нәтижесін *z* айнымалысына сәйкес келетін жады ұяшығына жазады.

Функцияның төрт құраушысы бар: (1) функцияның аты, (2) кіріс параметрлер, (3) айнымалыларды жариялау және (4) шығыс дабыллдарының мәнін кіріс мәндерде көрсету үшін байланысқан өрнектер жиыны. Функцияның бағдарламаның сақтаушысына кері жазба жазатын «меншіктеу» нұсқаулығы жоқ. Салыстыру үшін үдерістің бір немесе бірнеше меншіктеу нұсқаулығы болады.

Императивті бағдарламалау парадигмасында бағдараламалау үдерісін меншіктеу операторларын пайдаланбай жүргізу мүмкін емес. Меншіктеу операторы бар функция – бұл функция әсерін беретін нақты үдеріс.

2.4.6 Бағдарламаны орындауды дерексіздендіру.

Бағдарламаны атқару бағдарлама соңғы шартты қанағаттандыратын соңғы күйге жеткенше бір есептеу күйінен екінші есептеу күйіне өткен сияқты дерексізденуі мүмкін. Есептеу күйінің моделіне екі әдіс бар. Есептеу күйінің моделінің бірінші әдісі σ мына белгілердің үштігі сияқты көрінеді (σE, σS, σD), мұндағы σE таңбасы *қабықша,* σS таңбасы *сақтаушы* ал σD таңбасы шақыртудың түсу ретіне кері шақырту ішкі бағдарламасы тізбегінің жұбындағы стек (шақырған үдерісте шақырған үдеріс талап етілмейтін (немесе созылатын) қабықша бөлігі, шақырушы ішкі бағдарламадағы сақтау бөлігі шақырған үдерісті бөлмейді). Есептеу күйінің екінші әдісі логикалық НЕМЕСЕ, логикалық ЖӘНЕ, терістеу және импликация сияқты логикалық операторларды пайдаланып логикалық аксиомаларды біріктіретін бульдік өрнек ретінде көрінеді. Бульдік өрнек меншіктеу операторы орындалған сайын өзгеріп тұрады. Екінші әдіс фон Нейман машинасына тәуелді емес және 3-тарауда келтірілгендей бағдарламаның дұрыстығын талқылау және 4-тарауда көрсетілгендей соңғы күйден бастап бағдарламаны сатылап қайта құруда пайдаланылады.

2.4.6.1 Үштік сияқты есептеу күйі (Қабықша, Сақтаушы, Дамп)

Есептеу күйінің бірінші әдісінде σ нұсқаулықты орындаған соң өзгереді: (1) шақырылған мәлімдеудегі σE, қабықша өзгереді; (2) меншіктеу операторымен немесе инициализациямен шақыртылған σSбелгіленген сақтаушының өзгеруімен; немесе (3) функцияны немесе үдерісті шақырту себепті σD белгіленген дамптың өзгеруімен. Мәлімдеу операторынан кейін ағымдағы σE қабықшасы σ'Еболады, ал есептеу күйі өзгереді. Меншіктеудің жаңа операторынан кейін σS сақтаушысы жаңа σ'S сақтаушыға өзгереді және есептеу күйі де өзгереді.

Үдерісті шақырғаннан кейін түр жұбы жаңа (σ'E, σ'S, σ'D) есептеу күйін беру үшін (шақыртушы үдерістегі қабықшаның бөлігі және шақыртушы ішкі бағдарламадағы сақтаушы бөлігі шақырылған үдеріспен бөлінбейді) σD дампқа итеріледі. Функциядан немесе үдерістен қайтқаннан кейін қабықша шақырылған үдерістегі жергілікті айнымалылармен жаңару жолымен лақтырылып, ал қабықшаның басқа және итерілген бөлігін біріктіру σEnew қабықшаны береді. Шақырту үдерісінің жаңа қабықшасы σEnew егер шақырылған ішкі бағдарламаның өмір сүру уақытынан артық өмір сүру уақыты бар құрылым болмаса ескі σEтүрінде қала береді. Сақтаушымен де дәл осылай болады. Шақыртылған үдерістегі жергілікті айнымалымен байланысқан сақтаушы лақтырады, ал шақыртылған үдерістегі сақтаушының қалған бөлігі және дамптан алынған

шақырылған үдерістің сақтаушысының бөлігі шақырту үдерісіндегі жаңа σSnew сақтаушы болады. Назар аударыңыздар, σSnew σSсақтаушысы сияқты бола алмайды. Әрине, бұл күй бағдаралама сәтті орындалғанда алынады.

2.20-мысал

2.18-суреттегі мысалды қарастырайық. Бағдарламаның үш айнымалысы бар: *x,y,z*. *x* және *y* айнымалыларының жаһандық көріну облысы бар, ал *z* айнымалысы негізгі бөлікке қатысты жергілікті болады. негізгі бөлік барлық үш айнымалының мәнін иемденеді және *x* және *y* айнымалыларының сілтемелерін беру арқылы Swap (айырбас) функциясын шақырады. Айырбас денесіндегі '*' дескриптор адресті қайта атап және сәйкес *x* және *y* жаһандық айнымалысын қамтамасыз ететін жады ұяшығында оқиды (немесе жазады). Айырбас үдерісі *x* және *y* жаһандық айнымалылардың мәндерінің орнын ауыстырады.

Бастапқы есептеу күйі {<>, <>, <>}: қабықша, сақтаушы және бос дамптар. x және y жаһандық айнымалыларын жариялағаннан кейін x идентификаторы л-мән1 адресда көрінеді, ал y идентификаторы л-мән2 адресінда көрініс табады. Ал қабықша <x↦л-мән1, y↦л-мән2>. Жаңа есептелген күй:

{<x↦л-мән1, y↦л-мән2>, <>, <>}. Негізгі бөліктің ішіндегі z бүтін саны айнымалысын жариялау қабықшаны өзгертеді:

<(x↦л-мән1,y↦л-мән2,z↦л-мән3> Жаңа есетелген күй: {<(x↦л-мән1,y↦л-мән2,z↦л-мән3>, <>, <>}. Меншіктеудің әрбір үш операторын орындағаннан кейін сақтаушы есептеу күйінің өзгерісін өзгертеді. x = 4 меншіктеу операторы орындалғаннан кейін жаңа сақтаушы <л-мән1↦4>, ал жаңа есептеу күйі: {<(x↦л-мән1, y↦л-мән2, z↦л-мән3>, <л-мән1 ↦4>, <>}. y = 10 және z = 12 операторларын орындағаннан кейін есептеу күйі: {<(x↦л-мән1,y↦л-мән2,z↦л-мән3>,<л-мән1↦4,л-мән2↦10,

л-мән 3 ↦12>, <>}.

Айырбас үдерісін шақырту негізгі бағдарламаның жергілікті бөлігін дампқа итереді, ал m және n идентификаторлары сәйкесінше л-мән4 және л-мән5 адрестерінде көрініс табады. л-мән4 орны сілтемеге сәйкес л-мән1 орнында көрініс табады: л-мән5 орны сілтемеге сәйкес л-мән2 орнында көрініс табады. x және y жаһандық айнымалылар үшін қабықша және сақтаушы сақталады. Ал негізгі бөліктің жұптары (сақтаушыға сәйкес жергілікті айнымалыларға арналған қабықша) дампқа орналасады. Айырбас үдерісіндегі жариялаудың алдындағы есептеу күйі: {<x↦л-мән1,

```
integer x, y;
procedure swap( integer x, y);
main ( )
{ integer z;
x = 4; y = 10; z = 12;
swap(ref x, ref y)
print( x, y, z);
}
procedure swap(integer *m, *n)
{ integer temp;
temp = *m; *m = *n; *n = temp;
}
```

2.18 сурет. Есептеу күйінің бейнесінің бағдарламасы.

y↦ l -мәні2,m↦ l -мәні4,n↦ l -мәні5>, < l -мәні1↦4, l -мәні2↦10, l -мәні4↦л-мәні1, l -мәні5↦ l -мәні2>, <(<z↦ l -мәні3>, < l -мәні3↦12>)>}. Үлгі "**integer temp**" хабарламасы түрін өзгертеді, және оның жаңа түрі мынадай: *<x→ l -з мәні1,y→ l -мәні2,m→ l -мәні4,n→ l -мәні5, үлгі→ l -мәні6>* Есептеу жағдайы тиісті үлгіде өзгереді. *temp= *m* мәнін беретін оператор мекенжай бойынша тұрақсыз мәнді қосады: *l -мәні6* есте сақтау ұяшығы *l -мәні1* есте сақтау ұяшығында сақталған мәнмен салыстырылады және жаңа қойма *< l - мәні1 → 4, l - мәні2 → 10, l - мәні4 → l - мәні 1, l - мәні 5→ l - мәні2, l - мәні6 →4>* түріне айналады. *Тұрақсыз мән* 4-мәніне қолжеткізу үшін *l* -мәнімен 4→ *l -мәні1* және *l -мәні1*→ 4 арасындағы функцияның транзитивтігін пайдаланады. **m= *n* мәнін беретін оператор *l* -мәнінің1, орналасқан жеріне *n* тұрақсыз мәнін жазады және жаңа қойманың түрі мынадай *< l- мәні1 → 10, l - мәні2 → 10, l - мәні4 → l - мәні1, l - мәні5 → l - мәні 2, l - мәні6 → 4>*. **n= temp* мәнін беретін оператор *l -мәнінің2* тұрақсыз орналасқан жеріндегі үлгі мәнін жазып отырады. Жаңа қойма *< l - мәні1 → 10, l - мәні2 → 4, l - мәні4 → l - мәні1, l - мәні5 → l - мәні2, l - мәні6 → 4>* мәніне айналады. Мәндермен алмасу ішкі бағдарламасына кері қайтарылғанға дейінгі есептеу жағдайы: *{<x→ l- мәні1, y→ l- мәні2,m→ l- мәні4,n→ l- мәні5, үлгі → l- мәні6>, < l-з мәні1 → 10, l - мәні2 → 4, l - мәні 4 → l- мәні1, l- мәні 5 → l - мәні2, l- мәні6→4>, < (<z→ l- мәні3>, < l- мәні3→12>)>}*. *Мәндермен алмасу* рәсімінен кері қайтарылғаннан кейін, жергілікті қабықшасы және мәндермен алмасу рәсімнен арналған тиісті қойма қабықшадан және қоймадан жойылады, ал негізгі бөлігінің жергілікті қабықшасы

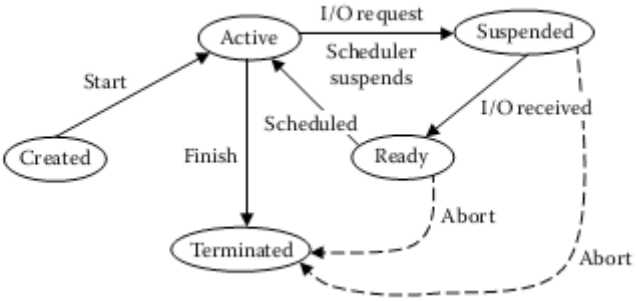
мен қоймасы дамптан алынып таслады, яғни дамп қайтадан бос күйінде қалады. Мәндермен алмасу рәсімінен қайтарылғаннан кейінгі жаңа есептеу жағдайы мынадай {<*x→ l- мәні1,y→ l- мәні2, z→ l- мәні3*>, < *l- мәні1→10, l- мәні2→4, l - мәні3→12*>, <>}.

2.4.6.2 Булев предикат ретінде есептеу жағдайы

Екінші әдісте есептеу жағдайын білдіретін логикалық мәні мән беру операторы орындағаннан кейін өзгереді. x= 5 мәнін беретін оператор x== 5 баяндауышын ақиқат мәнге айналдырады. y= 6 операторын орындағаннан кейін конъюнктивті баяндауыш x==5^y==6 ақиқат мәніне айналады. Мұндай әдіс оны қандай да бір нақты құрылымнан босата отырып, бағдарламаның орындалуын көру үшін қажет, және олардың дұрыстығына арналған бағдарламаларды талдау үшін қолданылады.

2.4.7 Процестер және Ағындар

Тіл қойылымдары мен параллельді бағдарламалау парадигмасын қолдайтын заманауи бағдарламалау тілдері процестер және ағындар ұғымын қолданады. Бағдарламаның немесе ішкі бағдарламаның белсенді бөлігі процес деп аталады. Процесті орындау жағдайын сақтау үшін бульдік жалаушалар мен басқа да процестермен қарым-қатынас жасау үшін процестің жеке меншік идентификаторы, хип, стек және жады саласы болады. Процес жады блогында сақталады. Процеске тиісті жады блоктары процесті орындау үшін қатқыл дисктен ЖСҚ-на жүктеледі. 2.19-суретте көрсетілгендей, процесс мынадай бес жағдайда болуы мүмкін: құрылды, дайын, белсенді, берілді және тоқтатылды. Белсенді процесс (1) жоспарлау стратегиясына сүйене отырып, процеске мүмкіндік беру үшін; (2) кіру/шығу аяқталғанға дейін күту үшін; немесе (3) орындау кезінде командаға сүйене отырып белгілі бір уақытта процеске тоқтау мүмкіндігін беру үшін тоқтатылады.. Тоқтатылған процесс кіру/шығу операцияларынан деректер қол жетімді болғаннан кейін немесе жоспарлаушы процеске қатысты тағы бір айналымды берген кезде іске қосылады.



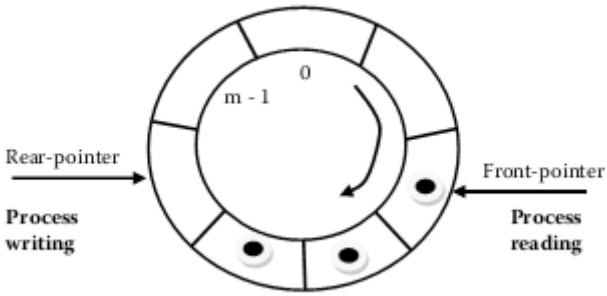
Created – Құрылды; Start – Басталды; Active – Белсенді; I/O request – I/O сұрау; Scheduler suspends – Жоспарлаушының тоқтауы; Suspended - Тоқтата тұру; I/O received – I/O алынды; Ready – Дайын; Scheduled – Жоспарланды; Abort – Тоқтату; Terminated – Аяқталды; Finish – Аяқтау.
2.19-сурет Белгіленген ауысуларды қамтитын процес жағдайының диаграммасы.

Ағын әрекеттердің реттілігін білдіреді. Процес пен ағын арасындағы айырма мынада, яғни ағын орындауға және қайта белсендіруге арналған үстеме шығындардың аз мөлшерін қамтиды, себебі ол осы ағындарды тудыратын қолданбалы процес деректерінің ауқымды көлемін қамтиды. Процес бірнеше ағындарды құруы мүмкін, олар ішкі міндеттер орындалғаннан кейін құрылған процеске қайтадан қайтарылады. Ағынды орындау кезінде, процес іске қосылуы мүмкін немесе ағын тоқтатылды деген сигналды күте отырып тоқтатылуы мүмкін. Ағын ішкі міндеттер аяқталғаннан кейін тоқтатылады не болмаса операциялық жүйе әрекеттерін тоқтатылады.

2.4.8 Буферлік салалар

Екі процес немесе ағын өзара бір-бірімен әрекеттескен кезде немесе бір-біріне деректерді берген кезде олар мұны әр түрлі жылдамдықта асинхронды режимде жасай алады. Жадының көп реттік көлемі деректерді асинхронды түрде беруді жеңілдету үшін қажет. Жадының мұндай көп реттік көлемін буфер деп атайды. Буферде екі маңызды операция бар: (1)деректерді енгізу және (2) деректерді алу.
Буферлер айналма кезекті пайдалана отырып іске асырылады. 2.20-суретте көрсетілгендей айналма кезектер деректердің жойылған элементтерінен босатылған жадыны үздіксіз көп мәрте пайдаланады. Айналма

кезектер ауқымдағы жадының соңғы элементін анықтағаннан кейін, ауқымның басталуына қайтып келу үшін сызықты ауқымға арналған модуль бойынша арифметикалық операцияларды пайдаланады.
Буферде төрт операция бар: (1), буфердің бос екендігін тексеру үшін, (2), буфердің толтырылғанын тексеру үшін, (3),бос емес буферден элементті жою үшін, және (4) бос емес буферге элементті ендіру үшін,



Rear-pointer - Артқы меңзер; Process writing - Процессті жазу; Front-pointer – Алдыңғы меңзер; Process reading - Процессті оқу.
2.20-Сурет Айналма кезек секілді буфердің схемалық жоспары.

Мысалы, буфер m мөлшеріндегі ауқым түрінде іске асырылды делік; онда буфердегі операциялар былайша айқындалады:

Бос буфер(буфер) **::** **егер**(алдыңғы көрсеткіш == артқы көрсеткіш)

```
is_empty_buffer(buffer) ::    if (front-pointer == rear-pointer)
                              return(true)
                              else return(false);
is_full_buffer(buffer) ::     if (((rear-pointer + 1) modulo m)
                              == front-pointer) return(true)
                              else return(false);
insert(buffer, element) ::    if not (is_full_buffer(buffer)) {
                                 buffer[rear-pointer] = element;
                                 rear-pointer = (rear-pointer + 1)
                                 modulo m;}
                              else raise_exception('buffer_full');
  remove(buffer)::            if not (is_empty_buffer(buffer)) {
                                 element = buffer[front-pointer];
                                 front-pointer = (front-pointer + 1)
                                 modulo m; return(element);}
                              else raise_exception('buffer_empty')
```

Егер артқы көрсеткіштің мәні алдыңғы көрсеткіш мәніне тең болса, Тізім бос (буфер) операциясы ақиқат мәнін қайтарады. Тізім толтырылды (буфер) операциясы 1 с-қа артатын артқы көрсеткіш модуль бойынша арифметикалық операцияларды пайдалана отырып, алдыңғы көрсеткіштің беретінін мәнін тексереді. Бұл артқы көрсеткіштің алдыңғы көрсеткішке жетіп алғанын және ешқандай да бос торлар жоқ екенін білдіреді. Ендіру (буфер, элемент) операциясы алдымен буфердің толық болуын тексереді. Деректер элементі буфер толық болмаған кезде ғана ендіріледі. Деректер элементі артқы көрсеткішпен индекстелген ұяшыққа ендіріледі. Элементті ұяшыққа ендіргеннен кейін артқы көрсеткіш модуль бойынша арифметикалық операцияларды пайдалана отырып 1-сқа артады. Жою (буфер) операциясы егер буфер бос болмаса, алдыңғы көрсеткішпен индекстелген жерден деректер элементін жояды, ал алдыңғы көрсеткіш модуль бойынша арифметикалық операцияларды пайдалана отырып бір бірлікке артады.

2.5 Қысқаша қорытынды

Бағдарламалау тілдерін әзірлеуде және іске асыруда қолданылатын негізгі ұғымдар осы тарауда сипатталған. Бұл ұғымдар бағдарламалау тілдерін әзірлеуде және іске асыруда маңызды рөлді ойнайды және келесі тарауларда өзекті болып табылады.

Бағдарламалау тілдерін зерттеуге қажетті дискреттің құрылым тұжырымдамалары жиынтықты және мультижиын логикалық операторларды, функциялар мен қатынастарды, соңғы автомат пен жиынтық операцияларды қамтиды. Мұндай түрдегі теория жиынтық теориясын кеңінен қолданады, және бағдарламалаудың көптеген тілдері жиынтық хабарларын және жиынтықтар негізіндегі операцияларды қолдайды. Жиынтықтарға жүргізілетін көптеген операциялар, мысалы ішкі жиынның жиынтықтары, декарттық көбейтінді, түпкі бейне және байланыссыз бірігу секілді операциялар талқыланды.

Бульдік логика бағдарламалау негізі болып табылады. *Предикат* деп аталатын класс функциясы (немесе логикалық бағдарламаларда пікір айту) бар, олар тұрақсыз мәндерге байланысты *ақиқат* немесе *жалған* мәндерін қайтарады. Күрделі логикалық жағдайларды тексеру көптеген басқару абстракцияларында қолданылады, мысалы алып тастау (шартты оператор, таңдау оператор және т.б..) және белгісіз итерациялар (WHILEDO циклы, DOWHILE циклы), ерекшеліктерді өңдеу және шартты ауысулар, мысалы, салынған блоктардан шығу немесе егер оқиға болып қойған болса, орындауды біртіндеп тоқтату. Бульдік шарттар логикалық ЖӘНЕ, логикалық НЕМЕСЕ, импликацияны және бо-

лымсыздарды пайдалана отырып, логикалық бағдарламалау өзегін қалыптастырады.

Функция екі жиынтық арасындағы бейнелерді білдіреді: яғни *домен мен өзгеріс саласы,* осылайша, доменнің әрбір элементі өзгерістің бір саласын ғана бейнелейді. Бейнелеу "one-to-one" немесе "many-to-one" ассоциациясы арқылы орындалуға тиіс. Алайда ол әрқашанда осындай бола бермейді. Фнукционалдық бағдарламалау парадигмасы функцияларды толығымен пайдаланады, ал бағдарламалаудың өзге парадигмалары, мысалы императивті бағдарламалау парадигмасы функцияларды бекітулермен қатар пайдаланады. Функционалдық бағдарлама өрнектерді есептеп шығаруды пайдаланады. Императивті бағдарламалау парадигмасында функциялар өрнектерді есептеп шығаруға толықтыру ретінде бекітулерді де пайдаланады.

Қатынас екі (немесе элемент пен атрибут арасындағы) *x* және *y* арасындағы ерекшелікті анықтайды. Қатынас *қайтару, адалдық* (немесе *симметриялылыққа қарсы*) немесе *ауысу* әрекеттері болуы мүмкін. Қатынас ерекшелігі логикалық бағдарламалау, бір мезетте іске қосу үшін бағдарламаны көшіруге арналған деректерге байланысты талдауды және тұрақсыз бағдарламаларды алмастыру секілді бағдарламаларды пайдаланады.

Рекурсия анықтамалардың екі түрімен сипатталады: *базалық оқиға* және *рекурсивті анықтау.* Бағдарламалау тілдерінде рекурсия рәсімдерді анықтауда және деректер құрылымын анықтауда да жүргізіледі. Рекурсивті функциялар бағдарламалауда басты рөл ойнайды. *Соңғы рекурсивті функция* итеративті бағдарламаларды пайдалана отырып модельденуі мүмкін. *Сызықты рекурсивті бағдарламалар* класы итерация және жинақтаушы ұғымдарды пайдалану кезінде іске асырылуы мүмкін. Рекурсияны өңдеуді орындау кезінде жадының өсімі үшін тағайындауды қажетсінеді, себебі деректердің рекурсивті құрылымы мен рекурсивті рәсімдер жадысына қойылатын талаптар компиляциялау кезінде белгіленуі мүмкін. Рекурсивті бағдарламалаудың кейбір үстеме шығындары тиісті итерациялық бағдарламалар үшін соңғы және сызықты рекурсияға айналу жолымен оңтайландырылуы мүмкін.

Соңғы автоматтар – бұл 3-тарауда талқыланатын компиляция кезіндегі лексикалық талдау-сөз орамы кезінде бағдарламалау тілдерінде қолданылатын және жағдайлармен анықталатын әр түрлі оқиғалар арасындағы модельдік ауысу болып табылады.

Стек және кезектер (тізімдер) – бұл әр түрлі ерекшеліктері бар деректердің абстрактілі түрлері: стек LIFO сипатын (тәртіппен, келіп түсудің кері тәртібімен) қамтиды, ал кезек FIFO сипатын (тікелей тәртіппен)

қамтиды. Элементтер стектің бір ұшынан итеріп шығарылады және алынады, ал деректердің элементтері кезектегі бір ұшынан кіргізіледі және басқа ұшынан шығарылады. Стек ағашты тереңдігімен айналып зерттеуде, рекурсивті рәсімдерді орындауда және бағдарламалау тілдерін іске асыруда маңызды рөл ойнайды. Кезек ағашты енімен айналып зерттеуде FIFO сипаты талап етілетін жерде қолданылады. Тереңдігімен айналып зерттеу ағашқа тереңірек енеді, себебі зерттеу оң жақ ішкі ағашты айналып шығудан бұрын сол жақ ішкі ағаштың түпкілікті түйініне жетеді. Енін зерттеуде ағашты деңгей деңгейімен айналып өтеді. Енімен зерттеу 6-тарауда айтылғандай, хиппен басқарудың кейбір тиімді әдістерінде қолданылды.

Хэш-кестелер тиімді енгузі үшін және хэш-фукнциясын пайдалана отырып, индекске бастапқы кілтті салыстыру жолымен динамикалық деректерді зерттеу үшін қолданылады. Хештеу функциясын пайдалану индекстің жақын арадағы тұрақты уақытта теңестірілетініне кепілдік береді. Хэш-кестелер бағдарламалау тілдерін іске асыруда және басқару процестерінде маңызды рөл ойнайды, себебі ақпарат тұрақты уақытта алынуы мүмкін.

Тұрақсыз шама бағдарламада ақпараттың басты қолданушысы болып табылады. Тұрақсыз шама негізгі алты атрибуты қамтиды:*атауы*, *түрі, көлемі, өмір сүру мерзімі, жадыда орналасқан жері*, сондай-ақ *берілген мәні*. *Әрекет ету саласы* бағдарлама бөлігі болып табылады, мұнда тұрақсыз шама оқылуы немесе жазылуы мүмкін, ал *өмір сүру уақыты* – бұл тұрақсыз шама пайдаланылуы мүмкін кезең. *Ғаламдық тұрақсық шамалар* өмір сүру уақыты мен бағдарлама секілді бірдей көлемге ие, ал *жергілікті тұрақсыз шамалар* олар жария етілген рәсімнің көлемімен шектелген. *Статикалық тұрақсыз шамалар* компиляциялау кезінде тіркелген мекенжаймен бөлінеді және жадыға тікелей қол жетімділік арқылы қол жетімді болып табылады. *Өзгергіш тұрақсыз шамалар* деструктивті түрде жаңартылуы мүмкін және мән бергіш операторлар үшін императивтік бағдарламалау парадигмасында қолданылуы мүмкін. Бағдарламаны орындау кезінде, маңызды үш ұғым бар: хабарламалар есебінен өзгеретін *қабықша*; мән бергіш операторлардың есебінен өзгеретін *қойма*; және *дамп* – қабықша стегі және туындайтын рәсімдердің реттілік қоймасы. Бағдарламаны орындау айрықша сипатталған есептеуіш жағдайлар арасындағы тұрақсыз шама реттілігімен немесе үштік түр ретінде (*қабықша, қойма* және *дамп*) не болмаса, конъюнкцияны, дизъюнкцияны және отрицания бульдік предикаттарды терістеуді қамтитын *логикалық мәндер* түрінде модельденуі мүмкін. Екінші тәсіл Нейманның фон машинасын есептеу жағдайын анықтаудан босатады

және бағдарламаның нақтылығын талқылау үшін қолданылады.
Процесс компьютерде іске қосылған бағдарламаның немесе ішкі бағдарламаның белсенді бөлігі болып табылады және ЖСҚ жадысын қамтиды.
Ағын процес ретінде қолданылатын әрекеттердің реттілігін білдіреді, яғни ағын процеске қарағанда жадыны бөлудің азғантай үстеме шығындарын қамтиды және аталық процестің жады кеңістігін пайдаланады. Буферлер екі процес арасындағы, ағындар, кіру/шығу екі құрылығысы арасындағы немесе процессор мен кіру/шығу құрылығысы арасындағы деректерді беру үшін қолданылатын жады кеңістігін білдіреді. Буфер айналма буфер ретінде модельденеді, ол деректер элементтерін жою арқылы босатылған буферден жады кеңістігін қайта пайдалану үшін модуль бойынша арифметикалық операцияларды қолданады.

2.6 Бағалау

2.6.1 «Тұжырымдамалар мен анықтамалар» деп өзгерту

Жинақтаушы; ациклдық граф; симметриялылыққа қарсы; байланыстыру; Бульдік логика; енімен зерттеу; буфер; Декарттық көбейтінді; айналма кезек; команда; есептеу жағдайы; цикл; циклдық граф; тереңдігімен зертеу; ациклдық графқа бағытталған; графқа бағытталған; дамп; динамикалық сала ережесі; қабықша; қолданбалы кванторларды пайдалану; өрнек; соңғы бейне; түпкілікті автомат; бірінші реттің предикаттарды есептеу; функция; граф; хэш-функция; хэш-кесте; өзгермейтін; нәтиже; индексация; сызықты рекурсия; логикалық ЖӘНЕ; логикалық НЕМЕСЕ; бейне; мультижиын; өзгеретін; терістеу; бір мекенжайлық есептеуіш машина; реттелген жиынтық; көрсеткіш; көрсетім жиыны; предикаттарды санау; процесс; өрнектерді есептеу; кванторлар; кезек (тізім), рекурсия; деректердің рекурсивті құрылымы; рекурсивті функция; сілтеме; қайтарымдылық; байланыс; көру ережесі; реттілік; стек; көрудің статикалық саласы; қойма; жол; симметриялылық; соңғы рекурсия; ағын; үш мекенжайлы машина; транзитивтік, ағаш; кортеж; екі мекенжайлы машина; тұрақсыз шама түрі; ортақ кванторларды пайдалану; тұрақсыз шама; көреріну, Нейманның фон машинасы; өлшенген граф; нөлді мекенжайы бар машина.

2.6.2 Тапсырма Шығару.

1.. Екі жиынтықты декарттық көбейтіндімен жазу: {"таң", "кеш", "түн", "күн"} және { "күн", "жарық", "жаңбыр"}.
2.. {"күн", "жарық", "жаңбыр"} деп берілген жиын жиынтығының көрсеткішін жазу. Көрсеткіш жиынының мөлшерін түсіндіру.
3.. "азайту"-5 функциясы берілген, ол зат құрамының сандарынан бес

санды азайтады және 10, 0, 3, 7 және 8 элементтерін бейнелей отырып заттық сандардың кеңейтілген саласында элементтерді бөледі. Кеңейтілген сала да төменгі ⊥ символын қамтитынына назар аудар. Мән саласында тұрақты элементтер үшін бейнеленбейтіндердің барлығы төменгі символ үшін бейненеленеді.

4.. Мынадай тұжырым үшін предикаттарды есептеу шешімін ұсынуды жазу: "әр бір N саны үшін осындай N>1, саны бар, ол бұл санға қарағанда 1-ге кем"

5.. Мынадай тұжырым үшін предикаттарды есептеу шешімін ұсынуды жазу: "Бұл әлемде әрбір адам үшін осы адамды осы әлемдегі кемінде бір адаммен қосатын байланыс бар."

6.. [0..4, 0..7, 0..9] өлшем бірліктерін қамтитын үш өлшемді ауқым және 10,000 базалық мекенжай берілген, яғни әрбір элемент жадының 2 байт көлемін қамтитын жағдайда (3, 4, 2) индексінде орналасқан бастапқы мекенжайды тап. Есептеу нәтижесін түсіндір.

7.. Төрт түр (4-кортеж) берілген (бүтінсан, символ, жылжымалы нүкте, бүтін сан), ескі мекенжай 5000 екенін ескере отырып, үшінші алаңның жылжуын көрсет. Мысалы, бүтін сан төрт байтты қамтиды делік, онда жылжымалы нүкте сегіз байт, ал символ бір байты иеленеді.

8.. Операцияның келесідей реттілігіне байланысты бос стектен бастай отырып, әрбір операциядан кейін стектің жағдайын суреттеп көрсет: жылдамдату (стек, 4), жылдамдату (стек, 5), итеріп шағыру (стек); жылдамдату (стек, 6). Стекті мынадай жүйелі түрде елестет < элемент N, ..., элемент 1>, мұндағы элементN, ..., элемент1–бұл LIFO тәртібінде стекке жылдамдатылған элементтер, және алдыңғы ұшынан қосылған және қойылған элементтер.

9.. Операцияның келесідей реттілігіне байланысты бос кезектен бастай отырып, әрбір операциядан кейін кезектің жағдайын суреттеп көрсет: енгізу (кезек, 4), енгізу (кезек, 5), жою(кезек); енгізу (кезек, 6). Кезекті мынадай жүйелі түрде елестет < элемент 1, …, элемент N >, мұндағы элемент1, …, элементN> бұл тізімдегі элементтер. Элементтер реттіліктің соңғы ұшынан бастап қойылады және реттіліктің алдыңғы ұшынан бастап жойылады.

10. Кезекті пайдалана отырып, енімен зертеу үшін және стекті пайдалана отырып, тереңдігімен зерттеу үшін бағдарлама әзірле және стеке қарсы кезекте сақталған деректер элементінің орташа санын салыстыр. Векторды пайдаланып кезекті іске асыр. Статистикалық талдау үшін кездейсоқ сандардың генераторын пайдалана отырып, орындау кезінде кемінде10 ағашты құрастыр.

11. Кемінде бес тұрақсыз шаманы пайдалана отырып қарапайым бағдар-

лама әзірле және бес қарапайым операторлар мен бірге қарапайым кодты жаз, және әрбір бекітуден кейін және әрбір оператордың орындауынан кейін қабықшамен қойманың қалай өзгеретінін көрсет.

12. Возьмите программу с, по крайней мере, пятью различными процедурами, с, по крайней мере, двумя процедурами
13.Бағанда түйінді белгілей отырып, әрбір рәсімді көрсет. Шақырылатын рәсімнен шақырушы рәсімге бағытталған қырды көрсет. Қырдың салмағы шақыру рәсімдерінің санымен анықталады. Егер рәсім өзін шақыратын болса, онда циклды көрсете отырып, түйіннен өзіне қарай қырды сипаттап бер. Алынған бағанды сипатта.

2.6.3 Толық жауап

13. Предикаттарды есептеу дегенді қалай түсінесіз? Түсіндір. Предикаттарды есептейтін әртүрлі компоненттер қандай? Қарапайым мысалдың көмегімен олардың әрқайсысын түсіндір.
14. Алғашқы тәртіп пен жоғарғы тәртіп предикаттарын есептеу арасындағы айырма қандай? Түсіндір.
15. Стек пен кезектің арасындағы айырманы түсіндір.
16. Ациклдық бағандардың бағытымен дегенді қалай түсінесіз? Олардың ағаштардан қандай айырмасы бар? Ағаштардың көмегімен емес, керісінше бағытталған ациклдық бағандардың көмегімен модельденуі мүмкін, тапсырманың нақты мысалын келтір.
17. Бағдарламалау тілдеріндегі көрсеткіштерді пайдалану артықшылығы мен кемшіліктері қандай? Түсіндір. Бағдарламалау тілдеріндегі көрсеткіштердің кемшілігін азайтуға мүмкіндік беретін үш жобалық шешімді талқыла.
18.Енімен, тереңдігімен зерттеу және олардың айырмасын түсіндір. Үйлестірілген N-тәрізді ағаштың көрнекі мысалын келтір ($n \geq 2$).
19. Кітапта берілмеген қарапайым мысалды пайдалана отырып, көрімділіктің статикалық және динамикалық саласын түсіндір.
20. Қабықша мен қойма арасындағы айырманы түсіндір. Өрнектерді есептеуді, рәсімдерді шақыртуды, мән беретін операторларды, хабарламаларды пайдалану кезінде қабықша мен қойманың қалай өзгеретінін көрсет.
21. Тиімді зерттеу үшін және кітапта берілмеген, бірақ толық мысалды пайдалана отырып, мұрағат деректерінен деректерді алу үшін хэштеу тетігінің қалай жұмыс жасайтынын түсіндір.

Қосымша әдебиет

Давендер С. Малик және Мридул К. Сен. Дискреттік математикалық құрылым: Теория және Практика.
Thomson Course Technology. 2004.
ДавендерС. Малик. С++қолданатын деректер құрылымы, екінші басылым.CourseTechnologyCengageLearning.
2009.
ДэвидА. Паттерсонжәне ДжонЛ. Хеннесси компьютердің құрылымы және компьютерлік жүйелерді жобалау, бесінші басылым. MorganKaufmann. 2012.
Зильбершатц, Абрахам, Галвин, Петер Б., и Гайне, Грег. Операциялық жүйелердің тұжырымдамасы, тоғызыншы басылым.
John Wiley and Sons. 2010.

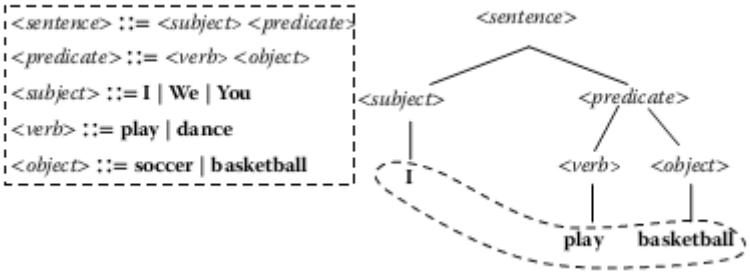
3. Синтаксис және Семантика

Базалық тұжырымдама
Есептеудегі абстракттілік ұғымдар (2.4-бөлім); Бульдік логика (2.2.2-бөлім); Басқару ағынының диаграммасы (1-бөлім); Дискреттік құрылым (2.2-бөлім); Қабықша және қойма (2.4.4-бөлім); Соңғы автоматтар (2.2.5-бөлім); Рекурсия (2.2.4-бөлім); Ағаштар (2.3.5-бөлім); Бағандар (2.3.6-бөлім); Нейманның фон машинасы (2.1-бөлім).
Бағдарламалау тілінің құрылымын ұғынуға арналған екі негізгі компонент бар: *синтаксис* және *семантика. Синтаксис* бағдарламалау тіліндегі оператордың рұқсат беретін құрылымын тексеруден құралады, ал семантика бағдарламалау тіліндегі сөйлемдердің нақты мәнін ұғынуды білдіреді. Бұл тұжырымдаманың екеуі де бағдарламаны ұғыну, компиляторды әзірлеу, және бағдарламалық қамтамасыз етуге қызмет ету үшін өте маңызды. *Синтаксистің* маңызы зор, себебі сөйлемдердің құрылымы тиісті үлгіде расталмаса, ол бағдарламалау тілінің бөлігі болып табылмайды, және дұрыс мәнмен байланысты бола алмайды. *Семантиканың* да маңызы зор, себебі бағдарламалау тілінің құрылымында және бағдарламада пайдаланатын біз әрбір сөздің беркелкі мағынасын бергенше және салыстырып қарағанша бағдарламадағы сөйлемді ұғыну мүмкін болмайды және орындау үшін аралық деңгей коды дұрыс аударылмайды.

3.1. Синтаксис және семантиканы енгізу

Графикалық тілдерді қоспағанда, қытай тілі секілді табиғи тілдердің көбісі синтаксистің негізгі төрт компонентін қамтиды: сөйлемдерді құрау және негіздеу үшін символдардың, сөздердің, сөз орамының және *грамматикалық ережелердің* соңғы жинағы (сондай-ақ өнімдік ережелер ретінде танымал). Символдар *дауысты* және *дауыссыз* болып бөлінеді. Дауысты символдар дауыссыз симолдарды байланыстыру үшін қолданылады және сөздерді айтуға көмектеседі. Сөздер мағына-

ларды байланыстыруға арналған негізгі бірлік болып табылады; символ деңгейінде ешқандай мағыналар байланыспаған. Бұл сөздер сөйлемді қалыптастыру үшін грамматикалық ережелердің (өнімдік ережелер) көмегімен қосымша біріктірілген. Егер жеке сөздердің мағынасы бірегей болып табылса, ол аталмыш сөйлем үшін бірегей мағына алынуы



ің
те
ш,
ік
ді
қа-

<sentence>::=<subject><predicate> - <сөйлем>::=<пән>< предикат>; <predicate>::=<verb><object> - < предикат>::=< етістік>< зат>; <subject>::=I|We|You - <пән>::=Мен|Біз|Сіз; <verb>::=play|dance - < етістік>::=ойнау|билеу; <object>::=soccer|basketball - < зат>::=футбол|баскетбол

3.1 Сурет Қарапайым ағылшын грамматикасы және синтаксистік талқылаудың тиісті ағашы.

Аралық нысан бастапқы сивол болып табылған жағдайда процес аяқталады. Өнімдік ережелерді прогрессивті қолданудың графикалық ұғымы *ағаш* болып табылатын сөйлемнен бастап бастапқы символға дейін жетуге көмектеседі және *синтаксистік талқылау ағашы* деп аталады. Синтаксистік талдау және синтаксистік талқылау ағашы 3.2-тарауда берілген.

3.1 Мысал

Ағылшын сөйлемінің мысалын қарастырайық: "Мен баскетбол ойнаймын." Әрбір сөздің мағынасы бар. Сөйлемнің грамматикалық тұрғыдан дұрыстығын тексеруден бұрын орфографиясын тексеру керек. Дәлірек айтқанда, орфографиясын тексергеннен соң, ағылшын тілінің грамматикалық ережелерінің қарапайым ішкі жиыны қолданылады(3.1-Суретті қара). "Мен баскетбол ойнаймын " сөйлемі аралық нысанға <*субъект*><*етістік*><*объект*> өзгереді, ол өз кезегінде басқа аралық нысанға <*субъект*><*предикат*> өзгереді. Тағы бір өнімдік ережені қолдана оты-

рып, <*субъект*><*предикат*> аралық нысаны бастапқы символға <*сөйлемге*> айналады.

Семантика нақты саланың дұрыс сөйлемінің синтаксистік мағынасын алады. Мысалы, "214" жолы: (1) *21410* (*2 * 102 + 1 * 101 + 4 * 100*) ондық есептеу жүйесінде, (2) *2148* сегіздік жүйеде (*2 (* 82 + 2 * 81 + 4 * 80*) = *14810* (*148-дегі есептеудің 10-дық жүйесінде*) ондық санды және құрылыс саласындағы (3) "екінші нөмір қабаты, нөмірі 1- алаң, және алаңдағы төртінші бөлме" мағынасын білдіреді. Мағына сөз орамы қолданылған семантикалық салаға байланысты болып келеді. Бірегей мағынаны алу үшін, семантикалық сала нақты жазылуға тиіс. Мысалы, есептеудің 10-дық жүйесіндегі «1011» екі санды мағына бұл *1×23 + 0×22 + 1×21 + 1×20 = 1110, 13* сегіздік (8 негізі бойынша семантикалы сала) және "D" он алтылық (16 негізі бойынша семантикалық сала).
Ондық санның мағынасы <*бүтін бөлік*>‘**.**’<*жылжымалыбөлік*>, сандар саласында бұл мағына (<*бүтін бөлік*>) +*мағына* (<*жылжымалыбөлік*>. Егер бейтерминал символдардың мағынасы <*бүтін бөлік*>және<*жылжымалы бөлік*> жеке тәртіптке танымал болса, ал бинарлық оператордың мағынасы '+' тең болса, онда ондық санның мағынасы алынуы мүмкін. *Семантикалық саладағы* сөйлемдердің мағынасын алу үшін қолданылатын операция *семантикалық алгебра деп* аталады. Бағдарламалау тіліндегі сөйлемдердің мағынасын ұғыну үшін, бізге семантикалық сала және тиісті семантикалық алгебра қажет.

3.2 Грамматика

Бағдарламалау тілдері жоғары деңгейдегі нұсқаулықты төмен деңгейдегі балама нұсқаулыққа аудару кезінде әртүрлілігін болдырмау үшін, ағылшын тілі секілді аса күрделі емес. Бағдарламалау тілдерін қарапайым күйде сақтау үшін көп күш жұмсалды:

1. Бағдарламалау тілдерінің негізгі бірліктері ағылшын тіліндегі сөздерге ұқсас; сөздерді құрайтын 26 әріп жоқ. Мұндай негізгі мағыналық бірліктер негізгі *(сақталған) сөздер* деп аталады және олар бірегей мағыналармен байланысқан. Бағдарламау тіліндегі негізгі сөздердің (сақталған) жинағы ағылшын тіліндегі сөздерге қарағанда басымырақ болып келеді.

2. Бағдарламалау тілдері бір негізгі сөз үшін бірнеше мағыналардан қашқақтайды, яғни ол мәтінге сүйене отырып, әртүрлілігін жоюды қажетсінеді. Контекстіге тәуелді көпмағыналы сөздерді пайдалану талдамалы тілдегі сөйлемдер үшін нақты мағынаны қалыптастыру кезінде қателіктерге бейімді және уақытты шығындауға әкеп соғады. Сөйлемнің

мағынасы төмен деңгей кодының іздестірумен байланысты болғандықтан, мағынадағы қателік төмен деңгейдің қате кодына әкеп соғады. Сондай-ақ сөз қорының шектелген саны бірнеше мағыналарды қамтиды және олардың нақты мағынасы мәтін негізінде алынған. Бұл *шамадан тыс* депте аталады және 7-тарауда талқыланған.

Грамматика тілдегі сөйлемнің соңын алу үшін қолданылады. Грамматика төрт компоненттен құралған: *бастапқы символ*, *терминалдық символдар жиыны(сақталған сөздер),бейтерминал символдар жиыны*, сондай-ақ *өнімдік ережелер жиыны. Бейтерминал символдар* өнімдік ережелер жинағын пайдалана отырып, терминалдық және бейтерминалдық символдардың қиысуына дейін кеңейеді. Бейтерминал символдар бағдарламалау тіліндегі сөйлемдердің бөлігі емес, керісінше грамматиканың бөлігі болып табылады. Бағдарламалау тіліндегі барлық *терминалдық символдардың* (сақталған сөздер) жиыны тиісті грамматиканың *әліппесі* деп аталады. Бағдарламалау тіліндегі сөйлем *бастапқы символдан* бастап өнімдік ережелерді пайдалана отырып, әліппеден шығарылады. *Әліппе* әдетте грек символы Σ деп белгіленеді. Ресми түрде грамматика төрт түрге бөлінеді (*бастапқы символS*, *өнімдік ережелер жинағы P, бейтерминал символдар жиыны N*,*әліппе* Σ). Бағдарламалау тілінде сөйлемдерді назарға ала отырып, грамматикадағы өнімдік ережелер сөйлемнің құрылымын тексеру үшін қолданылады. *Бастапқы символға* дейін аталмыш сөйлемді қысқарту мақсатымен бағдарламалау тіліндегі грамматикада өнімдік ережелерді көп мәрте қолданудың мұндай процесі *синтаксистік талдау* деп аталады.Өнімдік ережелердің оң жақ сөйлемшесі сөйлем бөлігімен немесе аралық нысанмен салыстырылады, және өнімдік ережелердің сол жақ сөйлемшесімен алмастырылады. Өнімдік ережелерді пайдалана отырып қалпына келтірудің мұндай процесі *талдау ағашы* деп аталады. Сөйлем синтакситік ағаштың соңғы түйіндерінде орналасқан, ал бейтерминал символдар синтаксистік талқылау ағашының ұшты емес түйіндерінде орналасқан, ал бастапқы символ ағаштың түбір түйінінде орналасқан.

3.2 Мысал

3.1-суретте "Мен баскетбол ойнаймын" сөз орамын ағылшын тілінде беруі мүмкін ағылшын грамматикасының қарапайым ішкі жиын ережелерін пайдаланудың қарапайым мысалы көрсетілген. Грамматика бес өнімдік ережеден тұрады. Бастапқы символ бұл<*сөйлем*>. Әрбір өнімдік ереже '::=' символымен бөлінген сол жақ және оң жақ сөйлемшелерді қамтиды.

Ереженің сол жақ сөйлемшесі бұрышты жақша ішіне салынған '<' және '> ' бір символды қамтиды, ал ереженің оң жақ сөйлемшесі символдардың екі түрінің мынадай тәсілін қамтиды: (1) жақша ішіне салынған символдар және (2) бұрышты жақшалары жоқ символдар. Бұрышты жақша ішіне салынған символдар бей*терминал символдар* деп аталады, ал бұрышты жақшасыз символдар *терминалдық символдар (сақталған сөздер)* деп аталады. Бейтерминал символдардың жиыны {*<сөйлем>*, *<предикат>*, *<субъект>*, *<етістік>*, *<объект>*}, ал әліппе (бейтерминал символдардың жиынын) қамтиды {**Мен,біз,сен, ойнау, билеу, футбол,баскетбол**}.

"Мен баскетбол ойнаймын," сөйлемі берілген, мұнда 3, 4 және 5 ережелері сөйлемдегі қалған бөліктермен өнімдік ереженің оң жақ сөйлемшесін салыстыра отырып және өнімдік ереженің сол жақ сөйлемшесіне сөйлем бөліктерін алмастыра отырып, сөйлемнің әртүрлі бөліктеріне қатысты қолданылады: 3-ереже "I" сөзін бейтерминал символ *<субъектке>* аударады, 4-ереже "ойнау" сөзін в бейтерминал символ *<етістікке>*аударады, ал 5-ереже "баскетбол" сөзін бейтерминал символ *<объектке>* аударады. Жаңа аралық нысан былайша айқындалады *<субъект><етістік><объект>*. Екінші тәсілде *<етістік><объект>* түрінің бейтерминал жұбы 2-ереженің оң жақ сөйлемшесімен салыстырылады және бейтерминал символ *<предикатты>* көрсетеді (2-ереженің сол жақ сөйлемшесі). 1-ереженің оң жақ сөйлемшесімен салыстырылатын жаңадан аударылған аралық нысан түрі *<субъект><предикат>*, және бастапқы <сөйлемді> көрсетеді. Осылайша, өнімдік ережелерді қолдану «Мен баскетбол ойнаймын» сөйлемін құрылымын тексеру кезінде бастапқы символ *<сөйлемге>* дейін қысқартады. Грамматикалық құрылымның сәтті тексеруі сөйлемнің әрқашанда мағыналы болуына кепілдік бермейді. мысалы, "Мен футболда билеймін" сөз орамы өз құрылымында өнімдік ережелердің көмегімен тексеріледі. Дегенмен,сөйлем мағыналық жүктемені қамтымайды.

3.2.1 Грамматика түрлері

Грамматиканың үш түрі бар, олар анықтамаларды қолданылу және бағдарламалау тілдерінде іске асырылуы мүмкін: *регулярлы грамматика*, *контекстіге тәуелді грамматика*, және *контекстіден тәуелсіз грамматика*.

Сондай-ақ *регулярлы грамматика* 3-түрдегі грамматика деп те аталады, қарапайым түрі және *анықталмаған соңғы автоматты* білдіреді. *Анықталмаған соңғы автомат* бұл бір кіріс символына арналған бір жағдайдан екінші жағдайға ауысатын бірнеше тұрақсыз шаманы қамти-

тын соңғы автомат. 2-тараудың 2.2.4-бөлімінде біз тұрақсыз шаманы қабылдайтын соңғы автоматты талқыладық. Соңғы автомат қарапайым грамматиканың көмегімен де айқындалуы мүмкін:

<*S1*> →<*әріп*><*S2*>
<*S2*> →<*әріп*><*S2*>| <*цифр*><*S2*>| ’_’ <*S2*>|**ε**

...Ресми түрде, *регулярлы грамматика* төрттік түрінде берілген (*N, Σ, P, S*), мұндағы *N* бейтерминал символдарының жиыны, Σ терминалдық символдардың жиыны, *P* өнімдік ережелер жинағы, ал *S*– бұл бастапқы символ болып табылады. Жоғарыда берілген мысалда,*N*– бұл{<*S1*>, <*S2*>}, ал Σ-бұл {*кез келген әріп,кез келген цифр,‘_’*}, *P*– бұл өнімдік ереже жинағы, ал*S* - бұл бастапқы символ<*S1*> болып табылады. *Регулярлы грамматика* сөйлемді қабылдау үшін және қабылдау кезінде маркер түріндегі сигналды беру үшін қолданылады. Символдардың реттілігін ескере отырып, регулярлы грамматика тұрақсыз шама меншігі болып табылса, реттілікті қабылдайтын болады.
Регулярлы грамматика тұрақсыз шамалар, литералдар, идентификаторлар және сақталған сөзде секілді бағдарламалау тілінің элементтері үшін лексема–игерілген ұғымдар ағынын тудырушы компиляциялау процесі кезінде талдаудың лексикалық кезеңінде кеңінен қолданылады. Лексемалар синтаксистік талдауды оңтайландыру үшін қажет.
2-түрдегі грамматика деп аталатын контекстісіз (контекстіден тіуелсіз) грамматика, өрттік түрінде берілген (*N, Σ, P, S)*, мұндағы *N* бейтерминал символдар жиыны, Σ терминалдық символдар жиыны, ал *P* өнімдік ережелер жинағы, *S*бастапқы символ болып табылады. Контекстіден тәуелсіз грамматиканың негізгі сипаты мынада, яғни сол бөлігі бірыңғай бейтерминал символды білдіреді. 3.1-суреттегі грамматика контекстіден тәуелсіз грамматиканың мысалы болып табылады. Мұндай шектеу сөйлемді талқылау кезінде кеңістікте және уақыт барысында артықшылыққа ие. Өнімдің ережелердің сол жақ сөйлемше бірыңғай бейтерминал символді білдіретіндіктен, ол кез келген терминалдық символдың қатысуынсыз кеңейтілуі мүмкін. Контекстіден тәуелсіз грамматика қарапайым грамматикаларға қарағанда өте мәнерлі болып табылады. Мысалы, a*n*b*n* жолын қабылдау үшін регулярлы грамматиканың қажеті жоқ. Сондай-ақ төменде көрсетілгендей, біз *an bn* жолын қабылдау үшін контекстіден тәуелсіз грамматиканы жаза аламыз:

<*S*>: = a<*S*>b | **ε**

Жоғарыда берілген грамматика анықтамасы бойынша контекстіден тәуелсіз грамматика болып табылады, себебі ол бастапқы символ <*S*>, набор бейтерминал символдар жинағы{<*S*>}, терминалдық символдар жинағы {a, Ь}, және бір өнімдік ережені қамтиды. Ереже сол жақ мәнді тек бір бейтерминалды символды қамтиды. Грек әріпі ε нөлдік символды білдіреді.

Контекстіге тәуелді грамматика да *1-түрдегі грамматика* түрінде танымал, сондай-ақ бейтерминал символдарға толықтыру ретінде мәннің сол жағында қосымша терминалдық символдарды иеленуі мүмкін. Мұндағы басты шектеу мынада, яғни сол бөліктегі жолдың ұзындығы бірінші бөліктегі символдар санына тең немесе одан кем болуға тиіс. Грамматиканың мынадай ережелерін қарастырайық:

<*S*>::= a<*S*>c | ε
<*S*>c : = b<*S*>cc

Жоғарыда келтірілген мысал контекстіге тәуелді грамматиканы дәлелдейді, себебі екінші өнімдік ереже ереженің сол жақ сөйлемшесінде бейтерминал символ <*S*> толықтыру ретінде терминалдық символ 'c' қамтиды. Грамматика *anbmcm+n* (*m*≥ 0, *n*> 1) түрінің жолын қабылдайды. Сонымен қатар, екінші терминалдық символ 'c' болғанда ғана ашылады. Контекстіге тәуелді грамматика контекстіден тәуелсіз грамматикаға қарағанда аса қуатты болып табылады. Мысалы, контекстіге тәуелді грамматика *anbncn*(*n>0*) жолын қабылдайтын нысанда жазылуы мүмкін. Ал контекстіден тәуелсіз грамматика *anbncn* түрінің жолын құра алады. Сондай-ақ контекстіге тәуелді грамматиканы пайдаланудың салмақты кемшілігі бар: контекст сезімталдығы өнімдік ережелер санын барынша арттырады, бұл сөйлемді талқылау талдау процесін баяулатады. Синтаксистік талдау кезіндегі тиімді түсініктерден бағдарламалау тілін анықтауда контекстіден тәуелсіз грамматиканы пайдаланады.

3.2.2 Бэкус-Наур формасын пайдалана отырып, грамматиканы ұсыну

Компиляторлардың дамыған алғашқы жылдарында компьютер ғылымы саласында жұмыс жасайтын екі ғалым Джон Бэкус және Петер Наур, бағдарламалау тілдері үшін контекстіден тәуелсіз грамматиканың сипатына арналған нысанды ұсынды. Олармен ұсынылған нысан қазіргі таңда *Бэкус Формасы — Наура* немесе *БНФ* ретінде танымал.

БНФ-дағы өнімнің сол және оң жақ тараптарын ':: =,' символымен бөледі, бейтерминал символдар бұрышты жақша ішіне салынған, ал бір

бейтерминал символдар жиыны анықтамалары мынадай -'|тік сызықпен бір-бірінен бөлінген. Сонда-ақ грамматика нөлдік мәнді сипаттау үшін грек символы‘ε’ пайдаланады, және объект түрінің қайталану санын анықтау үшін соңғы рекурсияны пайдаланады (2-тараудың, 2.2.3-кіші тарауын қара). Соңғы рекурсияда рекурсивтік бөлігі анықтаманың соңында келеді.

3.3 Мысал
Бейтерминал символ <*операторлар-жүйелігі*>соңын рекурсивті түрде анықтайды. ‘ε,’ символымен белгіленетін нөлдік оператор базалық жағдай болып табылады, ал соңғы рекурсияның бөлігі бейтерминал символ <*оператор*> ретінде анықталады, одан кейін үтір нүкте қойылады, әрі қарай <*операторлар жүйелігі*>былайша:
<*операторлар жүйелігі*>::= <*оператор*>**‘;’**
<*операторлар -жүйелігі*>| | **ε**

3.4 Мысал
Рисунок 3.2-сурет арифметикалық мәнді оңтайлы түрде анықтау үшін БНФ грамматикасын дәлелдейді. «Контекстден тәуелсіз грамматиканың бастапқы символ <*сөйлемше*>» 13 өнімдік ережені, мынадай {<*сөйлемше*>, <*А-сөйлемше*>, <*Л-сөйлемше*>,бейтерминал символы жинағын, <*салыстыру*>, <*идентификатор*>, <*символдар*>, <*цифр немесе әріп*>, <*сан*>, <*цифра*>,<*А-оператор*>, <*Л-оператор*>, <салыстыру *операторы*>}, және {**0** ... **9**, ‘**a**’ … ‘**z**’, ‘**A**’ … ‘**Z**’, ‘**+**’, ‘**–**’,‘*****’, ‘**/**’, ‘**&&**’, ‘**||**’, ‘**>**’, ‘**<**’, ‘**>=**’, ‘**=<**’, ‘**==**’}.әліппесінқамтиды. Ал '0' | '1' | ... | '8' | '9' жазбасы 0 -ден 9-ға дейінші барлық цифрларды қамтитын ішкі топты білдіреді.

Сөйлемше *(<сөйлемше>*) көпбейінді анықтама болып табылады. Бұл арифметикалық мән (<*А-сөлемше*>) немесе логикалық мән (<*Л-сөйлемше*>) немесе идентификатор (<*идентификатор*>) болуы мүмкін. Арифметикалық өрнек(<*а-өрнек*>) сан (<*сан*>) немесе келесі арифметикалық оператор(<*А-оператор*>)-мен арифметикалық мән (<*А-выражение*>) ретінде, сондай-ақ арифметикалық өрнек (<*А-сөйлемше*>)болып анықталады. Логикалық өрнек (<*Л-сөйлемше*>) *ақиқат* болып табылады*;*

<expression> ::= <A-expr> | <L-expr>
<L-expr> ::= **true** | **false** | <identifier> | **not** <L-expr> | <comparison> | <L-expr> <L-op> <L-expr>
<comparison> ::= <A-expr> <comp-op> <A-expr>
<A-expr> ::= <number> | <identifier> | <A-expr> <A-op> <A-expr>
<identifier> ::= <letter><characters> | <letter>
<characters> ::= ε | <digit-or-letter><characters>
<digit-or-letter> ::= <digit> | <letter>
<number> ::= <digit> | <number><digit>
<digit> ::= '0' | '1' | ... |'8' | '9'
<letter> ::= 'a'| 'b' | ... | 'z'| 'A' | 'B'| ...| 'Z'
<A-op> ::= ' + ' | '–' | '/' | '*'
<L-op> ::= '&&' | '||'
<comp-op> ::= '>' | '<' | '>=' | '=<' | ' ==’

1.2 сурет Сөйлемшені анықтауға арналған БНФ грамматикасы.

жалған, екі арифметикалық өрнекті салыстыру, екі логикалық өрнекті салыстыру (*<Л-сөйлемше>*), логикалық оператормен біріктірілген (*<L-оператор>*), немесе логикалық сөйлемшені терістеу. Салыстыру *(<салыстыру>*) салыстыру операторларын (*<Салыстыру операторы>*) пайдалана отырып, екі ариметикалық өрнекті салыстырады. Идентификатор (*<идентификатор>*) кез келген*<цифр>*немесе*<әріп>* қайталама санына тең бейтерминал символдан (*<символ>*)кейінгі әріп болып табылады. Сан (*<сан>*) келесі цифрмен (*<цифр>*) бірге сан (*<сан>*) ретінде рекурсвиті түрде анықталады. Сол жақ рекурсивтік *<сандарды>* анықтау 3.5-тараудағы детонациялық семантиканы ұғындыруға арналған келесі бөлімде қолданылатын болады. Цифр (*<цифр>*) 0-ден 9-ға дейінгі кез келген элемент болуы мүмкін. Арифметикалық оператор (*<А-оп>*) '+', '-', '*' болып табылады, немесе '/' және логикалық оператор логикалық ЖӘНЕ ('&&') немесе логикалық НЕМЕСЕ ('||') болуы мүмкін. Грамматика операциялар басымдығының болмауынан бір мағыналы емес және 3.12 және 3.13.-суреттерінде берілгендей, бір сөз орамы үшін екі немесе одан да көп талдау ағашын беруі мүмкін.

3.2.3 Бэкус — Наур кеңейтілген пішіні (РБНП)

Көрнекі болғанына қарамастан БНП адамның жеңіл түсінуі үшін кейбір жағдайларды анықтай алмайды:

1. БНП өнім қағидасының бірнеше бөліктері өзгергенде де, бірнеше анықтамаларды пайдаланады. Бұл қағиданы қажетсіз күшейтуге алып келеді. Мысалы, біз *<А-өрнек>* (‘+’|‘–’|‘*’|‘/’) *<А-өрнек>* сияқты арифметикалық теңдеуді есептей аламыз, мұнда (‘+’|‘–’|‘*’|‘/’) конструк-

циялары мүмкіншіліктердің бірін көрсетеді және екі қағиданы жақсы түсіну үшін бір қағидаға біріктіреді.
2. БНП символдың кез-келген қайталануын өңдеу үшін соңғы рекурсияны пайдаланады. Соңғы рекурсияны итерацияны пайдалана отырып тез түсіндіруге болады. 3.1-мысалда < *операторлардың кезектілігі*> анықтамасының орнында ол {<*оператор*> ';'}* ретінде көрсетілуі мүмкін, мұнда '*' белгісі ';' терминалды символынан кейінгі <*оператор*> терминалды емес символының кез-келген қайталануын білдіреді .
3. БНП өнімді қағида бөлімшесінің қосымша қайталануын анықтайды және оларды анықтама немесе ε нөлдік символы ретінде сипаттайды.

РБНП БНП-дегі аталған шектеулерді алу үшін пайдаланылды. 3.1-кесте РБНП-де пайдаланылған ауысуды көрсетеді. Бірнеше анықтамалар тобы дөңгелек жақшалар мен тік күйіндегі түрді (1-нұсқа | 2-нұсқа пайдаланады). Өнімдік қағиданың оң жақтағы өрнегін кеңейтудегі қосымша мүмкіншіліктер [қосымша функция] түріндегі шаршы жақшаны пайдаланып, өрнектеледі. Қайталау '*', '+' сияқты әр түрлі символарды немесе қайталау деңгейін сипаттау үшін әртүрлі цифрлармен толықтырылған қайталанатын ұғымның айналасында фигуралық жақшаларды пайдаланады. Мысалы сан <*цифрдың*> бір немесе бірнеше қайталануларын білдіре отырып, {<*цифр*>} +түрінде анықталады; ал <*идентификатор*> <*әріп*> ретінде анықталады, {(<*letter*>| <*digit*>)}254 254 символдан тұратын (әріптер немесе цифрлар) әріптерді білдіреді. Жақшаны, шаршы жақшаны, бұрыштық жақшаны, тік сызықтарды, арнайы мағыналары бар '+' және '*' пайдаланудың арқасында, осы символдар бағдарламалау тілінде терминалды символдар ретінде пайдалану кезінде тырнақшаға орналасады. Диапазонның төменгі шегі мен жоғарғы шегі арасындағы дефистің көмегімен қысқартылған түрде диапазон сипатталады.

TABLE 3.1 Transforming a BNF Grammar into an EBNF Grammar

BNF Representation	EBNF Representation
<*NT*> ::= Alternative$_1$ | Alternative$_2$	<*NT*> ::= (Alternative$_1$ | Alternative$_2$)
<*NT*> ::= ε | Optional-feature	[Optional-feature]
<*NT*> ::= Symbol <*NT*> | Symbol	<*NT*> ::= {Symbol}$^+$
<*NT*> ::= Symbol <*NT*> | ε	<*NT*> ::= {Symbol}*
<*NT*> ::= '**0**'|'**1**'|'**2**'|'**3**'	<*NT*> ::= '**0**'–'**3**'

Table 3.1 – Кесте 3.1

Transforming a BNF Grammar into an EBNF Grammar – BNF грамматикасын EBNF грамматикасына көшіру
BNF representation – BNF білдіргіші; EBNF representation – EBNF білдіргіші
<NT>::=Alternative$_1$|Alternative$_2$ - <NT>::=Балама$_1$|Балама$_2$;
<NT>::=ε|Optional-feature - <NT>::=ε| Қосымша-ерекшелігі;
<NT>::=Symbol<NT>|ε - <NT>::=Рәміз<NT>|ε;

3.5-мысал
3.3-сурет 3.2-суреттегі сипаттаманың РБНП нұсқасын көрсетеді. *<А-өрнек>, < L-өрнек>* және *<салыстыру>* терминалдық емес символдарын анықтау 3.1-кестеде сипатталғандай РБНП-ның «топтық» ерекшеліктеріне қосу жолымен өзгертілді. *< L -өрнек>* терминалды емес символының анықтамасы РНБП-ның «қосымша» ерекшелігін пайдаланады, ал унарлы оператор *< L-өрнек>* алдында қажеттілігі туғанда қайталана алмайды. Тиісті көптеген анықтамалар *< L-өрнекке>* біріктірілді. «Нөмір» терминалды емес символы «топтау» және «қайталау» үйлесімін пайдаланады және 1-ден 9-ға дейінгі цифрлар тобының бір немесе бірнеше қайталануы ретінде анықталады. Жоғарғы деңгейдің басқа өнімдік қағидаларын өзгерту үшін пайдаланылған өнімдік материалдың кейбірі ары қарай қажет болмайды және олар жоғарғы деңгейдің қағидаларымен біріктірілді.

РБНП грамматикасы (‘+’ | ‘–’ | ‘*’ | ‘/’) топталған операторларының нөлдік немесе көп қайталануымен *(<сан>| <идентификатор>)* тобынан кейінгі *<сан >(*немесе*<идентификатор>) ретінде итеративті <А-өрнекті>* анықтау жолымен кеңейтілуі мүмкін. *< L-өрнек>* терминалды емес символы (**шынайы** | **жалған** | *<идентификатор>*| *<салыстыру>*) салыстыру операторының тобын қайталаудан кейінгі **(шындық | жалған |** *<идентификатор>*|*<салыстыру>*) топтамасы ретінде итеративті анықталады. Оң жақтағы [**не**] *< L-өрнек>* өрнегі “[**не**]” артқа тастау арқылы [**не**] (шынайы| жалған| *<идентификатор>* |*<салыстыру>*) тобына өзгертілді. Жетілдірілген РБНП грамматика 3.4-суретте келтірілген.

3.6-мысал
3.5-сурет РБНП ұғымын пайдалана отырып, синтаксис ұғымын зерттеу үшін *итеративті тұжырымдамаларға* арналған грамматика анықтамасына тағы бір нақты мысал келтіреді. Атауының өзі айтып тұрғандай *<итерация>* көптеген анықтамаларға ие: *<цикл for>*, *<цикл while>*,

```
<expression> ::= <A-expr> | <L-expr>
<L-expr> ::= true | false | <identifier> | <comparison> |
             [not] <L-expr> ('&&' | '||') [not] <L-expr>
<comparison> ::= <A-expr> ('>' | '<' | '>=' | '=<' | '==') <A-expr>
<A-expr> ::= <number> | <A-expr> (' + ' | '–' | '*' | '/' ) <A-expr>
<identifier> ::= <letter> {(<letter> | <digit> )}*
<number> ::= {<digit> }+
```

3.3-сурет. Өрнекті анықтауға арналған кеңейтілген БНП грамматика.

```
<expression> ::= <A-expr> | <L-expr>
<L-expr> ::= [not] (true | false | <identifier> | <comparison>)
             { ('&&' | '||') [not] (true | false | <identifier> | <comparison>)}*
<comparison> ::= <A-expr> ('>' | '<' | '>=' | '=<' | '==' ) <A-expr>
<A-expr> ::= (<number>|<identifier>) {( ' + ' | '–' | '*' | '/' ) (<number> | <identifier>)}*
<number> ::= {<digit>}+
<identifier> ::= <letter> {(<letter> | <digit>)}*
<letter> ::= ( 'a' – 'z' | 'A' – 'Z')
<digit> ::= '0' – '9'
```

3.4-сурет. Өрнекті анықтауға арналған БНКП

```
<iteration> ::= <for-loop> | <while-loop> | <do-while-loop> | <iterators>
<for-loop> ::= for '('<l-value> '=' <expressions> ';' <expressions>; <expressions> ')' <block>
<while-loop> ::= while '(' <L-expr> ' )' <block>
<do-while-loop> ::= do <block> while '(' <L-expr> ' )'
<iterator> ::= foreach '(' <identifier> in (<identifier> '|' <enumeration>) ')' <block>
<block> ::= '{' {<statement> ';'}* '}' | <statement> ';' | '{' '}'
<statement> ::= <assignment> | <if-then-else> | <iteration>
<assignment> ::= <identifier> '=' <expression>
<enumeration> ::= '{' <entity> { ',' <entity> }* '}' | <identifier>
<entity> ::= <integer> | <float> | <string>
<identifier> ::= <alphabet> {(<alphabet> | <digit>)}*
<string> ::= ""{(<alphabet> | <digit>)}*""
<integer> ::= [(' + ' | '–')] {<digit>}*
<alphabet> ::= 'A' – 'Z' | 'a' – 'z'
<digit> ::= '0' – '9'
```

3.5-сурет. РФБН-ге жазылған итерациялық операторларға арналған грамматика

(<*Идентификатор*>) операторлар блогын білдіретін <*блок*> терминал-

ды емес символынан кейінгі дөңгелек жақшаның оң жағындағы индексті ауыспалы қадамның өлшемін көрсету үшін ‘**;** терминалды символынан кейінгі, <*идентификатор*>, терминалды емес символынан кейінгі, <*опер*> терминалды емес символынан кейінгі, <*өрнек*>терминалды емес символынан кейінгі (<*өрнек*>) есептелген өрнектің мағынасы үшін белгіленеді. Қалған өнімдік қағидалар алдыңғы мысалдарда алынған РФБН білімін пайдалана отырып, баламалы түрде саналуы мүмкін.

3.2.4 Атрибутивті грамматикалар

Грамматикадағы өнімдік қағидалар сәулеттің нақты түрінде тиімді орындау үшін қателерді тиімді өңдеу, бағалау, шектеулерді ерекшелеу және төменгі деңгейдегі код генерациясына арналған әр түрлі атрибуттармен байланысты болуы мүмкін. Әр түрлі сәулетшілер қабат өлшемі, қатардың рұқсат етілген өлшемі, идентификатордағы символдардың ең жоғарғы саны және с.с. өнімдік қағидаларға әр түрлі шектеулер қояды. Сонымен қатар, әр өнімдік қағиданың синтаксистік талдау ағашының төменгі деңгейлі кодын генерациялау үшін пайдаланылатын қандай да бір мағынасы бар. Өнімдік қағиданың атрибуты сәулеттің шектелуін немесе тілді әзірлеушілер салған шектеуді және талдау барысында генерирлейтін талдау ағашының ең төменгі кодын генерациялауға қажетті өнімдік қағида мағынасын көрсетеді.

Мысалы, тіл 16 биттік, 32 биттік немесе 64 биттік машинада орындалуы үшін толықтырылуы мүмкін. Прагматикалық себебі мен тиімділік тұрғысынан әзірлеушілер ауыспалыдағы рұқсат етілген символдардың санын шектеуі мүмкін. Осындай түрде сөз өлшемінің негізінде толық санның немесе жүзбелі нүктесі бар санның мағынасы сәулетке байланысты ең жоғарғы мағынамен шектелуі тиіс.Синтаксистің анықтамаларында бұл шектеулер (немесе атрибуттар), сондай-ақ грамматиканың бөлігі болып табылады және өнімдік қағидалармен бірге анықтаалады.

Өнімдік қағидамен байланысты атрибуттарды талдауда шектеулер тиісті түрде орындалғанына көз жеткізу үшін синтаксистік талдау ағашы бойынша жоғары және төмен қозғалуы тиіс. Атрибуттардың таралуы сондай-ақ кодтың аралық генерациясында және егер, атрибуттар бұзылған жағдайда ерекшеліктерді өңдеу үшін кодында пайдалы.

3.7-мысал

3.2-кесте екі өнімдік қағиданы және тиісті енгізілген өнімдік қағидаларды көрсетеді.

<*Тұтас сан*>1 дескрипторы сол жақтағы тұтас санды, ал <*тұтас сан*>2 дескрипторы оң жақтағы тұтас санды білдіреді. *Мағына* функциясы 10

негіз бойынша тұтас сан мағынасын береді, *ұзындығы* функциясы тұтас санды цифрды білдіреді, ал *өлшем* функциясы қатардағы белгілер санын білдіреді.

1- қағидаға арналған атрибуттар *«тұтас сан»* тұтас санның мағынасы − *231* до + *231* − *1* аралығындағы шектелуі мүмкін; *ұзындықпен (*(<*тұтас сан*>1) белгіленген сол жақтағы тұтас сандық цифр саны оң жақтағы тұтас сандық цифр санына қарағанда бір бірлікке көп; сол жақтағы <*тұтас сан*>1 тұтас санның мағынасы 10 болып табылады.
* *тұтас сан мағынасы <тұтас сан>2 оң жақта + цифр мағынасы <цифр>*оң жақта. Балама ретінде, егер <*тұтас сан*> <*цифр*> ретінде анықталса, онда *мағына<тұтас сан>* <*цифр*> параметрінің *мағынасы* сияқты болады.
2-қағида тиімділік тұрғысынан <*идентификатор*> элементінің өлшеміне шектеулер қояды. Шектеу идентификатор өлшемі 255-тен аз немесе тең болуын, ал идентификатордың сол жақ өлшемінің қатардың оң жақ бөлігі өлшеміне қарағанда бір бірлікке артық болғанын қадағалайды.

3.2-кесте. Атрибуттық грамматикадағы өнімдік қағидалар мысалдары

TABLE 3.2 Examples of Production Rules in an Attribute Grammar

Production Rule	Production Rule with Attributes
1 <int> ::= <int><digit> | <digit>	<int> ::= <int><digit> **Attributes:** value(<int>) > -2**31; value(<int>) < 2**31 - 1 value(<int>$_1$) = 10 * value(<int>$_2$) + value(<digit>) length(<int>$_1$) = length(<int>$_2$) + 1
	<int> ::= <digit> **Attribute:** value(<int>) = value(<digit>)
2 <identifier> ::= <letter>{(<digit>|<letter>)}*	<identifier> ::= <letter> {<digit> |<letter>)}* **Attribute:** size-of(<identifier>) =< 255 size-of(<identifier>) >= 1

Кесте 3.2
Examples of Production Rules in an Attribute Grammar - **Төлсипатты грамматикадағы өндірістік ережелердің мысалдары**
Production rule – **Өндірістік ереже**
Production rule with attributes – **Өндірістік ереже атрибуттарымен**

<int>-**<int>**; <digit>-**<сан>;** attributes – **атрибуттар;** value – **құны**; length – **ұзындығы;** <identifier> - **<идентификатор>;** <letter> - **<әріп>;** size-of (<identifier>) – **(<идентификатордың>) мөлшері**.

3.2.5 Гиперқағида мен Мета-анықтамалар

Синтаксис қағидалары өнімдік қағидалардағы жалпы картинаны жинап, мәнерлі қылуға болады. Екі қосымша қағида пайдаланылады: (1) *гиперқағида және*(2) *мета-анықтамалар. Гиперқағида* жалпы схема бойынша бірнеше өнімдік қағидаларды абстракциялайды, ал *мета-анықтамалар* гиперқағидаларда орналасқан бірнеше анықтамаларды көрсетеді. Гиперқағидаға мета-анықтамаларды қоя отырып, балама үлгілерге ие бірнеше өнімдік қағидаларды алуға болады. Мысалы, *<кезектілік>* бағдарламалау тілінің грамматикасында көптеген өнімдік қағидалар бойынша жалпы шаблон болып табылады және мына түрдегі гиперқағида ретінде өрнектелуі мүмкін:

<жүйелілік>: *<анықтама>*';' *<жүйелілік>*| *ε*-**гиперқағида**

<Анықтама> терминалды емес символын анықтауға арналған мета-анықтама төменде келтірілген. Оң жақтағы өрнек атау түсінікті.

<анықтама> :: *<формалды параметр>* | *<өзекті параметр>*| *<хабарландыру>* | *<оператор>* **-мета-анықтама**

Гиперқағидада бір рет бір мета-анықтаманы қолдана отырып, төрт өнімдік қағида құрылады. Тек бір мета-анықтама өнімдік қағидалардың генерациясындағы гиперқағидағы барлық анықтамаларға параллельді бір рет қолданылады; екі немесе одан көп мета-анықтамалар сол гиперқағидада бір уақытта қолданыла алмайды. Өндірістің төрт қағидасы:

*<формалды параметрдің реттілігі >***::=** *<формалды-параметр>';'*
< формалды параметрдің реттілігі >| *ε*
*<формалды параметрдің реттілігі>***::=** *<нақты параметр>* ';'
<формалды параметрдің реттілігі>|*ε*
<хабарландыру *реттілігі* >**::=** <хабарландыру> ';'
<хабарландыру *реттілігі* >|*ε*
<о*ператорлардың реттілігі*>**::=** *<оператор>* ';'
<операторлардың *реттілігі* >| *ε*

Грамматикада өнімдік қағидаларды гиперқағидалардан және метақағидалардан ерекшелеу қажет. Сол жақтағы және оң жақтағы гиперқағидаларда ':' бөлу белгісі пайдаланылады, БНП-дегі өнімдік қағидаларда оң жақ бөлікті сол жақтан бөлетін '::=' белгісі басым.

3.2.6 Абстрактылы синтаксис

Деректер абстракциясы мен бағдарламалау тілдері сыныбына арналған басқарма абстракциясының құрамын түсіну мақсатында синтаксис қағидалары деректер мен басқару абстракциясын пайдалану есебінен қысқартылады. Мысалы, біз бағдарламаларды, блоктарды, итерацияларды, таңдау операторларын, тағайындау операторларын, командаларды, өрнектерді, хабарландыруларды, формалды параметрлерді, нақты параметрлерді, нақты параметрлерді, идентификаторларды, анықтамаларды, ауысу операторларының литералдары мен секвенсорларын, деректер түрін сипаттайтын өрнектерді және т.б. пайдалану арқылы бағдарламалау тілінің конструкциясын абстракциялай аламыз.

Бұл абстракциялар бағдарламалау тілдеріндегі маңызды резервтелген сөздері бар абстрактілік синтаксис қағидаларының көмегімен анықталады. Абстрактілі синтаксис қағидалары бағдарламалау тілінің грамматикасына арналған өнімдік қағидалар жинағынан ерекшеленеді. Бағдарламалаудың барлық тілі үшін әмбебап болып табылатын идентификаторлар, сандар, тұтас сандар, цифрлар, өрнектер, қатарлар және оператор басымдықтары сияқты төменгі деңгейдегі кейбір анықтамалар ескерілмейді, себебі олар бағдарламалау тілін түсінуге ақпарат қоспайды. Сол себеп бойынша төменгі деңгейлі литералдар, бөлінгіштер мен кідірістер ескерілмейді. Абстрактілі синтаксис қысқа және абстракциялар туралы бағдарламашылардың білімінің көмегімен конструкцияны түсіндіреді. Абстракциялық синтакис қағидаларында оларға тән төменгі деңгейдегі ерекшеліктерді өткізу нәтижесінде туындаған белгісіздіктер жасалады. Соған қарамастан, ол қысқа, терминалды емес символдар пішімінде бағдарламалау тіліндегі деректер және басқарудың абстракциясымен байланысты және бағдарламалау тілінің негізінен тұрады.

3.8-мысал

1-мағыналарды, хабарландыруларды, өрнектерді және командаларды басқарудың абстракциясына арналған абстрактілі синтакис қағидалары 3.6-суретте көрсетілгендей бағдарламалаудың императивті тілдерінің негізгі кластары үшін анықталуы мүмкін.

Абстрактілі синтаксис қағидалары 1-мағына абстракциясы құрылымның нақты алаңы, идентификаторы немесе индексті ауыспалы болуы мүмкін (<*l-мағына*> ‘[‘<*өрнек*>’]’жазылады). Абстрактілі өрнек литерал, идентификатор, 1-мағына, өрнектің бинарлық операторлар немесе унарлы операторлмен қосылған жақшадағы екі өрнек болуы мүмкін.

Бағдарламалаудың аталған тіліндегі команда абстракциясы фигуралық жақшалардағы, тағайындау операторларындағы, команда реттілігінде-

гі блок, шартты оператор, WHILE циклі, DO-WHILE циклі, FOR циклі немесе ресімді шақыру болуы мүмкін.Сонымен қатар, ол {**if,then,else, do,while,'{,'and'},'for**} резервтелген сөздерді және тілдің бөлігі болып табылатын басқа конструкцияны хабарлайды. Соған қарамастан, төмендегі деңгейдегі басқа ерекшеліктер алынды. Мысалы, <*өрнек*> арналған абстрактілі синтаксис қағидаларын анықтаудағы оператор анықтамасы операторлардың әр түрлі басымдықтары мен әр түрлі өрнек арасында: арифметикалық салыстырғанда логикалық арасында айырмашылығы жоқ.

<*l-мағына*>**::=** <*идентификатор*> | <*идентификатор*>.<*l-мағына*> | <*l-мағына*>' **[**'<*өрнек*>'**]**'
<*хабарландырулар*>**::= variable**<идентификатор><*ауыспалы түрін сипаттайтын өрнек*> |
<*ауыспалы түрін сипаттайтын өрнек*> [<санвые>] |
structure{<*ауыспалы түрін сипаттайтын өрнек*>} <идентификатор> |
void<*идентификатор*> (<*формалды параметрлер*>) |
<*идентификатор*>*функция*<*идентификатор*> (<*формалды параметрлер*>)
<*өрнек*>**::=** <*литерал*>| <*идентификатор*> | <*l-мағына*> | (<*өрнек*> | <*ор*><*өрнек*> |
<*өрнек*><*ор*><*өрнек*>
<*нақты параметрлер*>**::=** <*идентификатор*> ',' <нақты *параметрлер*>

<*формалды параметрлер*>**::=** <*идентификатор*> ',' <*идентификатор реттілігі* > ';' <*формалды параметрлер*>| ∈

<*командалар*>**::=** { <*командалар*>} | <*l-мағына*>'=' <*өрнек*>| <*команда*>';' <*командалар*>|
if<*өрнек*>**then**<*командалар*>**else**<*командалар*> |
if<*өрнек*>**then**<*командалар*> |
while'**(**' <*өрнек*> '**)**' <*командалар*>|
do<*командалар*>**while**'**(**' <*өрнек*> '**)**' |
for'**(**'<*l-мағына*>'=' <*өрнек*> ';' <*өрнек*>';' <*өрнек*>'**)**' <*командалар*> |
<*идентификатор*>'**(**'<*формалды параметрлер*> '**)**'
<*секвенсор*>**::= goto**<*санвое мағына*>
<*бағдарлама*>**::= main**<*идентификатор*>';' <*хабарландырулар*> ';' <*командалар*>

3.6-сурет. Абстрактілі синтаксис қағидаларына мысал.

3.3 Синтаксистік диаграммалар

Мәтін компьютерде өңдеу үшін пайдаланылады. Соған қарамастан, адамдар көзбен қабылдағанды және түсіну үшін қарапайым графикалық суретті жақсы қабылдайды. Тілді әзірлеушілер бағдарламашылар бағдарламалау тілінің синтаксисін көзбен шолу, түсіну және пайдаланатындай
өнімдік қағидалар үлгісін пайдаланады. Бұл графиктік үлгілер *синтаксистік диаграммалар* деп аталады.

Синтаксистік диаграмма мен синтаксистік грамматика арасындағы өзара байланысты түсіну ерекше маңызды, себебі тілді әзірлеушілер (1) тілдік бағдарламашылар және басқа әзірлеушілер үшін түсінуге оңай синтаксис диаграммаларды құрастыруы тиіс және (2) әзірленетін синтаксистік талдауыштар мен код генераторлары үшін мәтіндік пішінді жаңғыртуы тиіс. Бағдарламашылар, сондай-ақ бағдарламаны әзірлеу кезінде мәтіндік нысандағы синтаксистік диаграммаларды зерттеу барысында алынған синтаксистік мәліметтерді түрлендіруі керек.

Іс жүзінде, синтаксистік диаграмма – бұл тораптар ұсынған терминалды және терминалды емес символдары бар және доға ұсынған өнімдік қағиданың оң жақ бөлігіндегі терминалды және терминалды емес символдар арасындағы өзара байланыстары бар бағытталған циклдік граф. Циклдер артқы рекурсия анықтамасын (немесе БНКП-дегі қайталау) үлгілейді. Синтаксистік диаграмманың сол жақ бөлікте орналасқан өнімдік қағидадағы терминалды емес символдарды сипаттайды,графаның қалған бөлігі өнімдік қағиданың оң жақ бөлігінің үлгісі болып табылады.

Синтаксистік диаграммалар үш негізгі компоненттерді құрайды: *терминалды емес символдар, терминалды символдар және бағытталған доғалар.Ыңғ*айлы болу үшін терминалды емес символдар сопақтарға енгізіледі, осылайша оларды терминалды символдардан нақты айыруға болады. Сызық диаграмманың қозғалыс бағытын көрсетеді. Сол жақтағы шеткі символ анықталуы тиіс өнімдік қағиданың сол жақ бөлігіндегі терминалды емес символды көрсетеді. Синтаксистік қағидалардан синтаксистік диаграммаларға ауысу кезіндегі әр түрлі түрлендірулер 3.7, 3.8-суреттерінде көрсетілген.

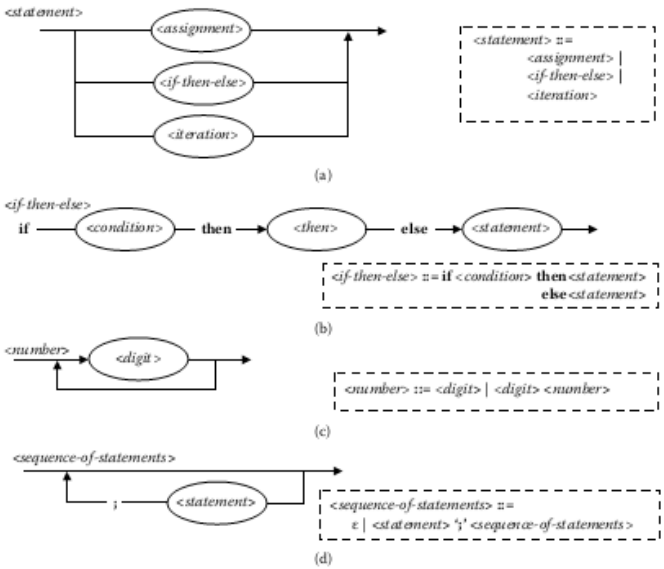
Қайта анықтамалар синтаксистік диаграммадағы бірнеше қозғалыс бағыттары түрінде көрсетілген. Нөлдік символ дереккөз бен символы жоқ белгілену блогы арасында бағытталған доға түрінде көрсетілген. Өнімдік қағиданың оң жақ бөлігіндегі бірнеше символдардың бірігуі бірнеше тораптарды біріктіруші учаске түрінде көрсетілген. Артқы рекурсияны анықтау анықтаманың бірнеше рет кіруін көрсету үшін кері

байланысты контур (цикл) түрінде ұсынылған. Шығатын доғадағы анықтамасы бар кері байланысты контур РНБП-дегі бір немесе бірнеше кірулерді сипаттайды. Белгілену торабының екі таңдауы бар: циклдің қайта өтуі үшін кері қайтатын доғаны пайдалану немесе шығу. Қызғылықты бағытты символдың нөлдік немесе көп санда енуін үлгілеуде көруге болады. Ол кері бағыттағы доғада символдар пайда болатынын, ал символдар алдында бағытталған доғада жоқ екендігін көрсетіп, өзгерістері бар кері байланысты контурды білдіреді.
Ақпаратты анық қабылдау үшін бірнеше өнімдік қағида бір синтаксистік диаграммаға бірігеді. Біріктіру *идентификаторлар; ауыспалы; тұтас сан; ондық сан; цифрлар; арифметикалық өрнек; логикалық өрнек; нақты параметрлер; формалды параметрлер; шартты оператор, WHILE циклі, DO-WHILE циклі, итераторлар мен таңдау операторлары; операторлар блогы; деректер түрін хабарландырулар; бағдарлама және т.б.* сияқты әр түрлі операторларды бағдарламалау тіліндегі абстрактілі элементтерді көрсету үшін пайдаланылады.

3.9-мысал
3.7-суретте қайталама анықтамаларға арналған синтаксистік диаграмма (3.7a-сурет), символдардың өзара байланысының синтаксистік диаграммасы (3.7b-сурет), символдың бір немесе бірнеше кіруін көрсететін артқы рекурсия анықтамаларының синтаксистік диаграммасын (3.7c-сурет) және символдардың нөлдік немесе одан да көп кіруіне арналған синтаксистік диаграмманы (3.7d-сурет) көрсетеді.

3.7a-суретте көрсетілгендей бір тараптан басталатын және басқа тараптан кері біріктіретін екі торап арасындағы бірнеше бағдарлардың тармақтануы қайталанған анықтамаларды білдіреді.
<*Оператор*> терминалды емес символына арналған синтаксистік диаграмманың бірнеше бұтақтары бар: (1) бірінші бұтақ <*тағайындау*> деген терминалды емес символдан құралған; (2) екінші бұтақ <*шартты оператор*> деген терминалды емес бұтақтан құралған; және (3) үшінші бұтақ <*итерация*> деген терминалды емес символдан құралған. Ол <*оператор*>**::=** <*тағайындау*> | <*шартты оператор*> |<*итерация*> түріндегі синтаксистік қағидаға тең.

```
<statement> ::= <assignment> | <if-then-else-statement> |
                <iteration>
```



<statement> - **мәлімдеме**; <assignment> - **тапсырма;** <if-then-else-statement> - **егер-содан кейін-тағы-мәлімдеме**; <iteration> - **итерация;** <if-then-else**> - егер-содан кейін-тағы;** if – **егер;** condition – **жағдай;** then - **содан кейін;** else – **тағы;** number – **нөмір;** digit – **сан;** sequence of statement – **мәлімдемелердің реті.**

3.7-Сурет
Синтаксистік диаграммалар.

3.7b-суретте <*шартты оператор*>**::=if**<*шарт*>**then**<*оператор*>**else**<*оператор*> синтакистік қағидасы көрсетілген. Бірінші бөлігі белгілі тәртіптегі {**if, then,else**}терминалды символдары мен и {<*шарт*>, <*оператор*>} терминалды емес символдардың бірігуін білдіреді. Синтаксистік диаграмма синтаксистік қағидадағы бірігу тәртібін сақтайды.
3.7c-суретте <*сан*>**::=**<*цифр*>| <*цифр*><*сан*> өнімдік қағидасына арналған синтаксистік диаграмма көрсетілген.
Анықтама <*сан*>**::=** <*цифр*> негізгі жағдайға және анықтама <*сан*>**::=**

<цифр><сан> артқы рекурсиядағы анықтамаға ие. Бұл *<цифрдың>* бір немесе бірнеше рет кіруі түрінде бейнеленеді, себебі артқы рекурсияның анықтамасын тағы бір *<цифр>* қосу үшін пайдалануға болады, ал анықтама *<сан>* терминалды емес символының анықтамасын кеңейтуді аяқтау алдында тағы бір *<цифр>* қоса отырып, негізгі жағдайды қолданып аяқтауға болады. Анықтама *<цифрдың>* бір немесе бірнеше рет кіруін білдіреді. Синтаксистік қағиданың БНКП нұсқасы *<сан>***::=** **{***<цифр>***}+** деп жазылады. Тиісті синтаксистік диаграмма кері байланысты контурды білдіреді, онда тік бағыттағы доға *<цифр>* терминалды емес символынан құралған, ал белгілену доғасынан негізгі көз торабына қарай кері бағытта доға өтеді. Тік бағыттағы қозғалыс бірнеше рет өтуі мүмкін. Тік бағытта қозғалыс қайталанғанда әр кезде *<цифр>* кезектілігін қалыптастыру үшін қосымша *<цифр>* қосылады.
3.7d-суретте операторлардың нөлдік немесе үлкен саны кіруі үшін үлгілейтін синтаксистік қағиданың синтаксистік диаграммасы көрсетілген. БНП-дегі тиісті өнімдік қағида рекурсияның негізгі бөлігінен және артқы бөліктен тұрады. Негізгі бөлік ε бос символынан тұрады, ал артқы рекурсия төменде көрсетілгендей

*<операторлардың реттілігі>***::=** *<оператор>***';'**
<операторлардың реттілігі > | **ε**

'**;**' символымен сүйемелденетін <оператор> терминалды емес символын қамтиды.

БНКП-де символдардың нөлдік немесе көп санының енуі дөңгелек жақшаларға орналастырылған символдар ретінде жазылады және төменде көрсетілгендей

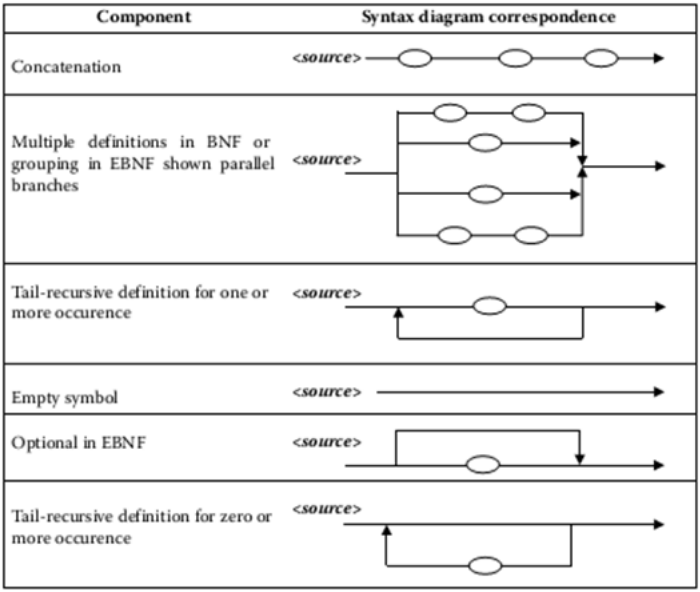
*<жүйелілік-операторов>***::=** {*<оператор>***';'**}*
оң жақ дөңгелек жақшадан кейін дереу "*" белгісі қойылады.

Символдардың нөлдік немесе үлкен кіру саны кері байланыстағы контур түріндегі синтаксистік диаграмма түрінде бейнеленеді, яғни тік бағыттағы доға символсыз тік сызық түрінде көрсетіледі, ал кері бағыттағы доғада символдар бар. 3.7d-суретте кері бағыттағы доғада *<сөйлем>* терминалды емес символы бар, одан кейін '**;.**' терминалды символы орналасқан

3.3.1 Синтаксистік диаграммадағы синтаксистік қағидаларды аудару

Синтаксистік қағидалар мына қағида бойынша тиісті синтаксистік диаграммаға аударылады:

1.Синтаксистік диаграммалардағы терминалды және терминалды емес символдардың айырмашылығы айқын.
2.БНП-мен немесе БНКП-дегі қайта анықтамалары бар өнімдік қағидаларды екі торап арасындағы параллельді бұтақтар ретінде елестетуге болады, ол көп жолақты жол сияқты.
3.Оң жақтағы бірнеше символдардың бірігуі бір бұтақтағы бірнеше символдар түрінде көрсетіледі.
4.Артқы рекурсия анықтамасы кері байланысы бар контур түрінде көрсетіледі, оң қыры бастапқы анықтамамен қосылған.
5.Бос символ ε белгіленеді, екі торап арасында сызығы бар тік сызық түрінде көрсетіледі.
6.Міндетті емес анықтама екі бұтақ түрінде көрсетіледі, бір бұтақ бір тораптан екіншісіне өтетін сызықты тік желіні білдіреді, ал екінші бұтақтың анықтамасы бар.
3.8-суретте БНП-дегі және РНБП-дегі синтаксистік қағидалардың тиісті компоненттеріне арналған синтакистік диаграмма көрсетілген. Эллипс мәтін нысанында ұсынылған символды (терминалды немесе терминалды емес) білдіреді.
Бірінші қатарда өнімдік қағидаға тиісті синтаксистік диаграмма ұсынылған, онда үш символ оң жақ бөлікте орналасқан. Эллипстерді біріктіретін үш доға ыңғайлы болу үшін біріктірілген. Екінші қатарда өнімдік қағида көрсетілген, онда бір уақытта төрт әр түрі анықтама бар:
1.Жоғары доғада орналасқан бірінші анықтама оң жақ бөліктегі екі символдан құралған.
2.Екінші және үшінші анықтамалар оң жағында бір символдан құралған.
3.Төменгі доғада орналасқан төртінші анықтаманың оң жағында екі символы бар.
Үшінші қатарда өнімдік қағиданың оң жақ бөлігіндегі символдың бір немесе бірнеше енуінің артқы рекурсиясының анықтамасы көрсетілген. Бір немесе бірнеше кірулер тең мағыналы.



Component – **Құрамдас;** Syntax diagram correspondence - **Синтаксис диаграмма хаты;** concentration – **концентрация;** multiple definitions in BNF or grouping in EBNF shown parallel branches – **BNF-тағы бірнеше анықтамалар немесе EBNF-тағы топтастық параллельдік салаларда көрсетілген;** tall-recursive definition for one or more occurrence - **бір немесе бірнеше оқиғалар үшін биік-рекурсивті анықтама**; empty symbol – **бос рәміз;** optional in EBNF – **EBNF-та міндетті емес;** tall-recursive definition for zero or more occurrence - **нөл немесе бірнеше оқиғалар үшін биік-рекурсивті анықтама.**

3.8-Сурет. Синтаксистік диаграммалар мен синтаксистік қағидалар арасындағы сәйкестік.

БНКП-дегі {<*символ*>}+ және БНП<*анықтама*>**::=** <*символ*><*анықтама*> | <*символ*> артқы рекурсиясының анықтамасы, мұнда <*символ*> терминалды және терминалды емес символдың үйлесуімен көрсетілуі мүмкін. Төртінші қатарда бос ‘ε.’ Символы үшін синтаксистік диаграмма көрсетілген. Осы жағдайда символ жоқ болғандықтан, тиісті синтаксистік диаграмма тік сызықты білдіреді.

Бесінші қатарда міндетті емес символдың синтаксистік диаграммасы көрсетілген.

БНКП–де [<*символ*>] ретінде жазылған міндетті емес символ тік бағыттағы екі маршруттан тұратын (<*символ*> | **ε**) тобына сәйкес келеді: бір маршрута кіріктірілген торап түрінде <*символ*> тұр, ал екінші маршрутта кіріктірілген торап жоқ. Алтыншы қатарда нөлдік немесе ауқымды кіру символдары көрсетілген. Нөлдік немесе ауқымды кіру символының синтаксистік диаграммасында сондай-ақ бір немесе бірнеше кіруге тең кері доға бар. Соған қарамастан, нөлдік немесе ауқымды кіруге арналған синтаксистік диаграммада символ тік емес, керісінше маршрутта көрінеді.

<*Символ*> нөлдік немесе ауқымды кіру анықтамасы БНКП-де{<*символ*>}* ретінде және БНП-де <*анықтама*>**::=** <*символ*><*анықтама*> | **ε** артқы рекурсияның анықтамасы ретінде жазылады, мұнда <*символ*> терминалды немесе терминалды емес символ болуы мүмкін.

Мәтіннен синтаксистік диаграммаға ауыстыру үшін грамматиканы деректерді абстракциялау немесе басқару деңгейіндегі бірқатар мағынаға ие маңызды қызметтік бірліктерге топтайды. Грамматиканы өнімдік қағидалардың ішкі топтарынан тұратын топтарға бөледі, онда әр ішкі топ *ауыспалы, формалды параметрлер, блок* және т.б. сияқты қызметтік бірлік бағдарламасын құруда сәйкес келеді. Содан кейін итеративті жоғары деңгейдегі қағида синтаксистік диаграммаға түрленеді, ал түрленетін синтаксистік диаграммалар ішіндегі терминалды емес символдар басқа төменгі деңгейдегі синтаксистік диаграммаларды қосу нақты қызметтік блоктың ары қарайы түсінуін жеңілдетпегенше, басқа синтаксистік диаграммаларға дейін түрленеді.

3.10-мысал

<*Идентификатор*> анықтамасында үш терминалды емес символдан: {<*идентификатор*>, <*әріп*>, <*цифр*>} тұратын топты пайдаланатын үш өнімдік қағида бар. Мұнда <*әріп*> пен <*цифр*> <*идентификатор*> терминалды емес символын анықтау үшін пайдаланылады. Осылайша, үш өнімдік қағида бір синтаксистік диаграммаға бірігеді.

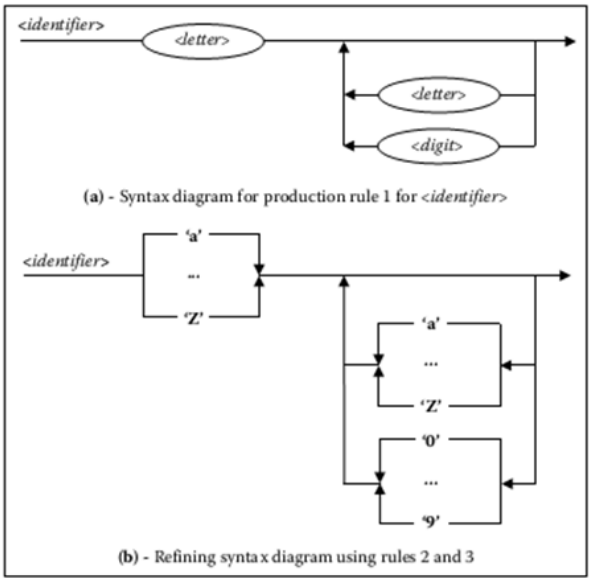
<*идентификатор*>, <*әріп*>және<*цифр*> төменде көрсетілгендей:

<*идентификатор*>**::=** <*әріп*> {(<*әріп*> | <*цифр*>)}*

<*әріп*>**::= ‘A’ – ‘Z’ | ‘a’ – ‘z’**

<*цифр*>**::= ‘0’ – ‘9’**

Алдымен <*идентификатор*>терминалды емес символын анықтайтын өнімдік қағидаға арналған синтаксистік диаграмма құрылады.
<*Идентификатор*> анықтайтын өнімдік қағиданы бірінші таңдайды, себебі <*идентификатор*> анықтамасы өзіне оң жақ бөліктегі терминалды емес символдардың анықтамасын қамтиды, бұл терминалды емес символдарды кейінірек ашуға болады.
<*Әріп*> және <*цифр*> терминалды емес символдарын синтаксистік диаграмманы нақтылау үшін ашады. Өнімдік қағиданың сыныптары арасындағы сәйкестікке байланысты өнімдік қағида синтаксистік диаграммаға ауысады. Соңғы синтаксистік диаграмма 3.9-суретте көрсетілген.



Identifier – **идентификатор;** letter – **әріп;** digit – **сан;** syntax diagram for production rule 1 for <identifier> - **<идентификатор> үшін 1 өндірістік ереженің синтаксис диаграммасы;** refining syntax diagram using rules 2 and 3 – **2 және 3 ережелерді пайдаланып синтаксис диаграмманы өңдеу**

3.9 - Сурет. Бірнеше өнімдік қағидаларды пайдалану арқылы синтаксистік диаграмма құру

3.3.2 Синтаксистік диаграммаларды синтаксистік қағидаларға аудару

Синтаксистік диаграммаларды синтаксистік талдауышта қағидалар генераторында оларды пайдалану үшін мәтін нысанына түрлендіру қажет. Осылайша, тіл әзірлеушісі немесе бағдарламашы синтаксистік диаграмманы синтаксистік қағидалардың тиісті жинағына түрлендіруі мүмкін.

Синтаксистік диаграмманы өнімдік қағидаға аудару өнімдік қағидалардың синтаксистік диаграммасына кері процесс. Синтаксистік диаграммаға кіріктірілген синтаксистік диаграмманың сол бөліктері және тек терминалды символдар арқылы анықталатындар және анықталған терминалды емес символдар өнімдік қағидаға өнімдік қағидалар мен 3.8-суретте көрсетілген синтаксистік диаграммалар арасындағы терминалды емес символдардың сәйкестіктерінің көмегімен аударылады.

Жаңа маңызды терминалды емес символдар кіріктірілген синтаксистік диаграммалар үшін құрылады.

Өнімдік қағиданы құрған соң синтакистік диаграмманың тиісті бөліктерін жаңа анықталған терминалды емес символмен ауыстырады және үдеріс қайталанады. Процесс синтаксистік диаграмма толығымен бір терминалды емес символға ауысқанда тоқтайды.

3.11-мысал

3.7-суретте көрсетілген синтаксистік диаграммаға арналған қағида жиыны құру үшін кіріктірілген бөліктердің көпшілігі - (‘**0**’ | ‘**1**’ | ... | ‘**9**’) тобын анықтайтын бірнеше параллельді маршруттар және *(‘***a**’|...| ‘**z***’)* тобын анықтайтын бірнеше параллельді маршруттар *<әріп>* **::=** *‘***a**’|...| ‘**z**’және *<цифр>* **::=** *‘***0***’* | ... | *‘***9***’түрленеді. <әріп>* мен *<цифр>* терминалды емес символдарын енді бірнеше параллельді маршруттардың орнына қояды, синтаксистік диаграмма 3.7а суретте көрсетілген диаграммаға дейін қысқартылады.

3.7а суретте *<әріп>* терминалды емес символы мен (*<әріп>*|*<цифр>*) баламалы символдарының нөлдік немесе үлкен санының кіруін көрсететін артқы рекурсияның анықтамасы көрсетілген.

Осылайша, жаңа өнімдік қағиданың түрі мынадай: *<идентификатор>***::=***<әріп>* {(*<әріп>* | *<цифр>*)}*.

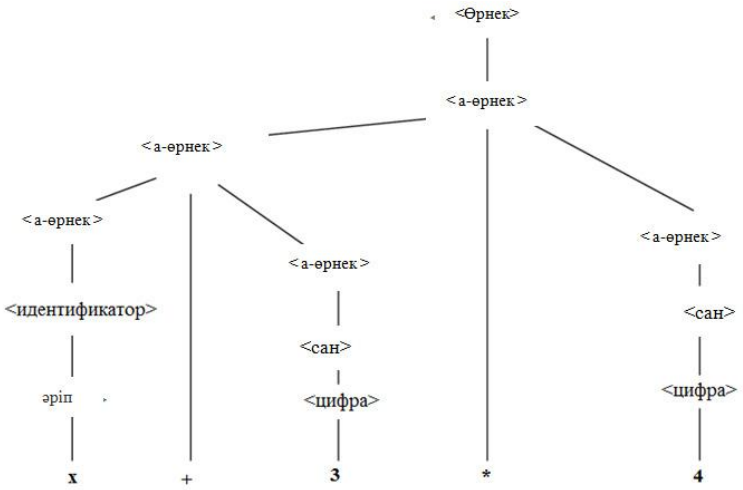
3.4 Сөйлем құрылымын тексеру

2.Бағдарламаның түрлену үдерісі сөйлем құрылымын тексеруден басталады. Сөйлем құрылымын тексеру екі сатыдан тұрады: (1) "токендердің" көмегімен сөйлем символын ішкі форматқа түрлендіру және (2) осы сөйлемнің ішкі форматын тілдің грамматикалық қағидалардың көме-

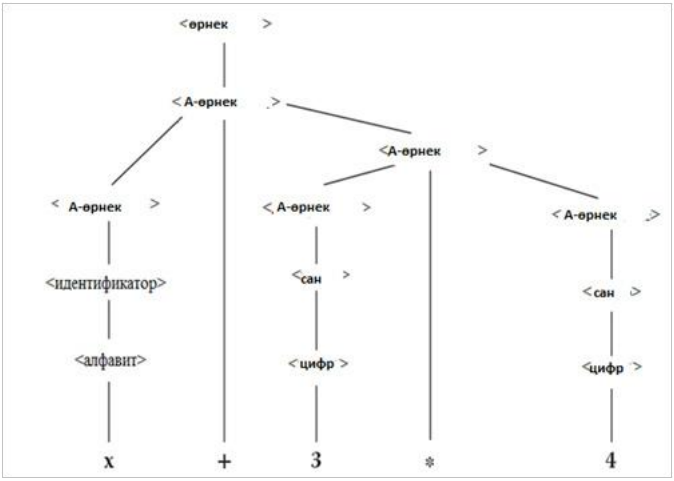
гімен тексеру. Бірінші саты *лексикалық талдау* деп аталады, ал екінші саты *синтаксистік талдау* деп аталады. Екінші сатының нәтижесі болып *талдау ағашы* табылады. *Талдау ағашы* -S = s0, …, sm жүйелігі түріндегі сөйлем
3.*Түбір* бастапқы символы;
Шығу: Т синтаксистік талдау ағашы
{.қысқартылған нысан = S;
...талдау қатесі = жалған;
...Т = нөлдік ағаш;
While ((қысқартылған нысан ≠ түбір) **&&not**(талдау қатесі))
{ Егер қысқартылған түрдегі s*i*.. s*j* жүйелілігі болса, s*i*… s*j*== оң жақ бөлік (pk∈**R**),**мұнда** 1 =<k =<n{ терминалды емес = сол жақ бөлік (pk);
қысқартылған нысан = ауыстырғыш (қысқартылған нысан, s*i*… s*j*, терминалды емес символ);
Т = Т + доға (s*i*..s*j*→ сол жақ бөлік (pk));
}
әйтпесе талдау қатесі = шындық;
}
Егер қайту болмаса (талдау қатесі) (Т); **әйтпесе баспа** (‘талдау қатесі’);
}
3.11-Сурет. Төменнен жоғары типті сөйлемді синтаксистік талдаудың оңайлатылған схемасы.

Өнімдік қағидаға сәйкес келетін тиісті жүйелілікті табу процесі - LL(K)- және LR-талдауыштар сияқты синтаксистік талдаудың әр түрлі автоматтандырылған тәсілдерін қолдану арқылы шешілетін күрделі тапсырма. Автоматтандырылған талдауыштар 3.4.5-тарауда қысқаша сипатталған.

3.13-мысал
Қанекей 3.2-суретінде көрсетілген грамматиканы пайдалана отырып, “x + 3 * 4,”талдайық. Сөйлем бес символдан құралған: ‘x’, ‘+’, ‘3’, ‘*’,және ‘4’. Бұл символдар лексикалық талдауышта тиісті токендерде түрленетін болады. Соған қарамастан, ыңғайлы болу үшін токендердің орнына символдарды олардың бастапқы нысанында пайдаланатын боламыз. Кез-келген символ немесе 5 символдан тұратын жүйелілік өнімдік қағиданың оң жақ бөлігіне сәйкес келуі мүмкін. 3.3-суретте ұсынылған грамматиканы пайдалана отырып, 3.12 және 3.13-суреттерде көрсетілген талданған екі баламалы ағашты аламыз. *Бір сөйлем үшін бірден көп талдау ағашын беретін грамматика бір мәнді емес деп аталады, оны қолданбаған жөн.*



3.12-Сурет. Грамматикадағы әр мәнділікпен байланысты алынған дұрыс емес талданған ағаш. әріп



3.13-СУРЕТ. x + 3*4 өрнегіне арналған дұрыс талданған ағаш

3.4.3 Грамматикалық әр түрлі мәнді жою

Әр түрлі мәнді грамматика сөйлем жеке мағынаға ие болу керек деген бағдарламалау тілінің негізгі принциптерінің бірін бұзады. Бір сөйлемге

арналған екі әр түрлі талдау ағашы төменгі деңгейдегі екі әр түрлі нұсқаулар жиынына түрленуі мүмкін, ол екі әртүрлі есептеулерге алып келеді. Грамматикалық әртүрлі мән туындайтын екі негізгі есептеу жолы бар: (1) өнімдік қағидада операторлардың басымдығын ескермейді және (2) енгізу деңгейі резервтелген сөздермен бөлінген енгізілген құрылымдарда конструкцияларды салыстыру бір мағыналы болмайды.

3.14-мысал

3.2-суретте <*өрнекті*> 3.5 арқылы анықтауға арналған грамматика бір өнімдік қағидада операторларды топтау кезінде арифметикалық өрнектер мен логикалық өрнектердің бірнеше анықтамаларын пайдаланғанына байланысты бір мәнді емес. Арифметикалық қағидаларда '*' және '/' бинарлық операторлары '+' и '−' бинарлық операторларына қарағанда үлкен басымдыққа ие. Сол сияқты, логикалық өрнектерде 'not' унарлық операторлар '&&' (логикалық И) қарағанда жоғары басымдыққа ие, ал '&&' (логикалық И) '||' (логикалық НЕМЕСЕ) қарағанда жоғары басымдыққа ие.

Өрнектерде әр түрлі мәнмен күресудің бір тәсілі – дөңгелек жақшаны пайдалана отырып, өрнекті нақты бөлу. Соған қарамастан,басымдықтың кезектілігі грамматикалық қағидаларда бірнеше өнімдік қағидаларға арналған өнімдік қағиданы бөлу арқылы кодталуы мүмкін. Анықтамалары көп өнімдік қағида жаңа терминалды емес символдарды енгізу кезінде өнімдік қағидаларға бөлінеді, олар кейіннен операторлардың арту басымдығы бар басқа терминалды емес символдарының арасында анықталады және соңғы өнімдік қағида аса жоғары басымдықты операторларды пайдаланады. Алдымен басымдығы жоғары операторлар талданады, себебі жоғары басымдықты операторлар талданады, аса жоғары басымдықты операторлар бірінші кезекте жоғарыдан төмен типті синтаксистік талдауда қолданылады. Өрнектің тиісті әртүрлі мәнді грамматикасы 3.14-суретте көрсетілген.

<*А-өрнек*> арифметикалық өрнекке арналған өнімдік қағида <*көп анықтамалы өрнек*> және <*А-шарт*> екі қосымша терминалды емес символын пайдалана отырып,үш өнімдік қағидаға бөлінген. Осыған ұқсас, <*L-өрнек*> логикалық өрнегіне арналған өнімдік қағида <*өрнек-И*> және <*L-шарт*> екі қосымша терминалды емес символдарын пайдаланып, үш өнімдік қағидаға бөлінген. Терминалды емес <*А-өрнек*> символы «көп анықтамасы бар өрнек» арқылы анықталды. «көп анықтамасы бар өрнек» терминалды емес символы «А-шарт»қатысты анықталды. «А-шарт» терминалды емес символы (<*А-өрнек*>), немесе <*идентификатор*>, немесе <*сан*> қатысты анықталды. <*L-өрнек*> терминалды

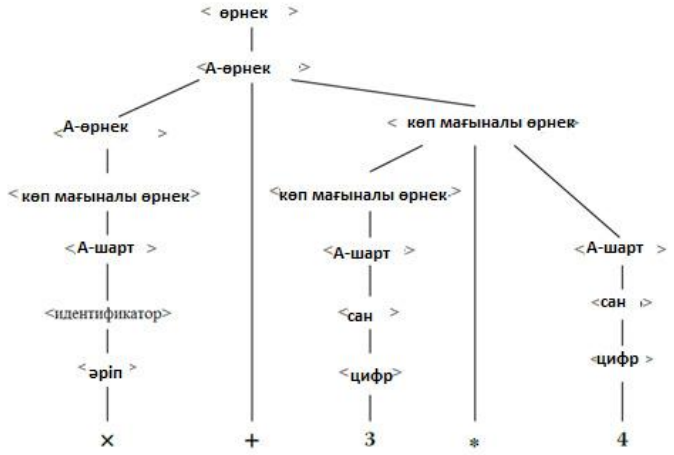
емес символы <*өрнек-И*>.терминалды емес символымен анықталады. <*Өрнек*-«and» > терминалды емес символы <*L-шарт*> терминалды емес символымен анықталады. <*L-шарт*> терминалды емес символы артынан жақшада логикалық өрнектер тобы немесе фигуралық жақшаларды салыстыру немесе терминалды емес <*идентификатор*> символы немесе , «**true»** терминалды символы немесе «**false**» терминалды символы еретін «**not**» міндетті емес терминалды символына қатысты анықталады.

<*өрнек*>**::=**<*А-өрнек*>|<*L-өрнек*> (1)
<*А-өрнек*>**::=** <*А-өрнек*> (‘ + ’ |‘-‘)<*көп анықтамалы өрнек*>|<*көп анықтамалы өрнек*> (2)
<*көп анықтамалы өрнек*>**::=**<*көп анықтамалы өрнек*>(‘*’|‘/’)<*А-шарт*>|<*А-шарт*> (3)
<*А-шарт*>**::=** ‘**(**‘ <*А-өрнек*> ‘**)**’ |<*идентификатор*>|<*сан*>
(4)
<*L-өрнек*> *::=* <*L-өрнек*> *‘||’*<*өрнек-И*> (5)
<*өрнек-И*>**::=** <*өрнек-И*>‘**&&**’<*L-шарт*> (6)
<*L-шарт*> *::=* [**not**](‘**(**‘<*салыстыру*>‘**)**’|‘(‘<*L-өрнек*> ’)’ | <*идентификатор*> | **true** | **false** (7)
<*салыстыру*>*::=*<*А-өрнек*> (‘>’ | ‘<’ | ‘>=’ | ‘=<’ | ‘==’) <*А-өрнек*>
...(8)
<*идентификатор*> *::=* <*әріп*>{(<*әріп*>|<*цифр*>)}* (9)
<*сан*>**::=** [(‘ + ’|’−‘)] {<*цифр*>} + (10)
<*әріп*>**::=** ‘**a**’| ‘**b**’ | … |’**z**’| ‘**A**’ | ‘**B**’ | … | ‘**Z**’ (11)
<*цифр*>**::=** ‘**0**’ | ‘**1**’| … |’**9**’ (12)

3.14-СУРЕТ. Өрнектің әр түрлі мәнді грамматикасы

3.15-мысал
Қанекей 3.14-суретте көрсетілген грамматиканы пайдалана отырып, “x + 3 * 4” сөйлемін талдайық. ‘3’ және ‘4’ литералдары мына қағидалардың кезектілігі бойынша: 12-қағида, 10-қағида, 9-қағида және 4-қағиданың көмегімен <*А-шарт*> терминалды емес символына дейін қысқарады. Қысқартылған“<*А-шарт*>− <*А-шарт*>” нысаны 3-қағидасының көмегімен<*көп анықтамалы өрнек*> терминалды емес символына дейін қысқарады және жаңа қысқартылған нысан “x + <*көп анықтамалы өрнек*>.” *түріне ие болады.* x символы мына қағидалардың кезектілігі бойынша: 11-қағида, 9-қағиданың көмегімен терминалды емес <*идентификатор*> символына дейін қысқарады және аралық нысан “<*иден-*

тификатор> + <*көп анықтамалы өрнек*>.” болады» Терминалды емес <*идентификатор*> символы ары қарай 4-қағиданың, 3-қағиданың, 2-қағиданың кезектілігі арқылы терминалды емес “<*А-өрнек*> символына дейін қысқарады» Жаңа қысқартылған нысан “<*А-өрнек*> + <*көп анықтамалы өрнек*>” түріне ие болады, ол 2-қағиданың көмегімен <*А-өрнек*> терминалды емес символына дейін қысқарады. <*А-өрнек*> терминалды емес символы 1-қағиданың көмегімен <*өрнек*> бастапқы символына дейін қысқарады. Алынған бірегей және дұрыс ағаш 3.15-суретте ұсынылған..



3.15-Сурет. Бір мағыналы грамматиканың көмегімен орындалған “x + 3 * 4” өрнекке синтаксистік талдау.

3.3-КЕСТЕ. Енгізілген шартты оператордың сәйкес келмеуін түсіндіру

Дұрыс емес түсіндіру	**Дұрыс түсіндіру**
if(x>4) **then**	
if(y> 0) **then**	
return(1);**else return**(0)	
(a) **Else** бөлік сыртқы **if** салыстырылады	
if(x> 4) **then**	
if(y> 0) **then**	
return(1); else	
return(0);	
(b) **else** бөлік жақын **if** салыстырылады	

3.4.3.1 Енгізілген құрылымдардың әр түрлі мәнділігі

Шартты операторлың екі нұсқасы бар: (1) **if**<*оператор*>**then**<*оператор*> немесе (2) **if**<*оператор*>**then**<*оператор*>**else**<*оператор*>. Бірінші нұсқа салыстырылмайды, себебі else бөлігі жоқ, ал екінші нұсқа салыстырылады, себебі else бөлігі бар. Енгізілген шартты операторды құру үшін салыстырылатын және салыстырылмайтын нұсқаның үйлесуі әр түрлі мәнділікке алып келеді, себебі 3.3-кестеде көрсетілгендей, else бөлігі сыртқы if-пен немесе ішкі if-пен салыстырылатыны түсініксіз болады.

Екі операторлың да екі "if" кіруі және бір "else" кіруі болады. Бағдарламалау қағидасы else жақын if кіруімен байланысатынына құрылған. Дұрыс түсіндіру үшін екі әрекет нұсқасы бар: (1) бөлгіштердің көмегімен ішкі салыстырылатын бөлікті жазу, ол оны сыртқы блоктардан нақты ажырату үшін жасалады немесе (2) енгізілген шартты операторды өңдеу үшін бір мәнді грамматиканы жазу. Бір мәнді грамматикаға мысал төменде келтірілген

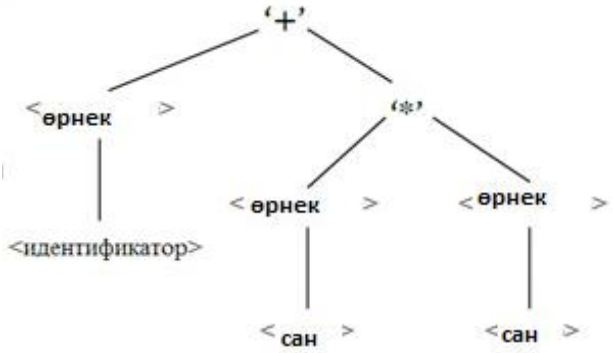
<*шартты оператор*>**::=** <*салыстырылатын-шартты оператор* > |
<*салыстырылмайтын-шартты оператор*>
<*салыстырылатын-шартты оператор*>**::= if**<*шарт*>**then**
<*салыстырылатын-шартты оператор*>
else<*салыстырылатын-шартты оператор*>|
<*басқа операторлар*>
<*салыстырылмайтын-шартты оператор*>**::=if**<*шарт*>**then**
<*шартты оператор*> |
if<*шарт*>**then**
<*салыстырылатын-шартты оператор*>
else<*салыстырылмайтын-шартты оператор*>

3.4.4 Абстрактілі синтаксистік ағаш

Абстрактілі синтаксис қағидасынан жасалған талдау ағашы *абстрактілі синтаксистік ағаш* деп аталады. Абстрактілі синтаксистік ағашқа абстрактілі синтаксис қағидаларында жоқ нақты синтаксистік қағидалардағы барлық терминалды емес символдар кірмейді. Өрнек операторлармен байланысады. Мысалы, "x + 3 * 4" өрнекке арналған абстрактілі синтаксистік ағаш 3.16-суретте көрсетілген.

Нақты синтаксистік ағаштар (нақты синтаксистік қағидалардан құрылған ағаштар) артық терминалды емес символдарды және сөйлемге тікелей мағына қоспайтын аса төменгі деңгейдегі терминалды емес символдарын жою көмегімен абстрактілі синтаксистік ағаштарға

дейін қысқарады.Мысалы, 3.15-суретте көрсетілген “x + 3 * 4” өрнекке арналған нақты синтаксистік ағаш тікелей семантикаға жатпайтын және 3.16-суретте көрсетілген абстрактілі ағаштан алынатын {<*А-шарт*>, <*көп анықтамалы өрнек*>, <*цифр*>} сияқты көптеген төменгі деңгейдегі терминалды емес символдардан тұрады.



3.16 –Сурет. “x + 3 * 4.”өрнекке арналған абстрактілі синтаксистік ағаш

Абстрактілі синтаксистік ағаш бағдарламалау тілдерінің тілдік конструкцияларын түсіну үшін пайдалы, себебі нақты синтаксистік ағаштардың маңызды емес бөлшектерін жасырады. Абстрактілі синтаксистік ағаштарды сондай-ақ бағдарлама кодының синтаксистік басқарылатын генерациясында пайдаланады: синтаксистік талдаудан кейін нақты синтаксистік ағаш тиісті абстрактілі синтаксистік ағашқа дейін қысқарады, ал содан кейін аралық деңгейдегі кодты құру үшін абстрактілі синтаксистік ағашқа семантикалық талдау жүргізіледі. Абстрактілі синтаксистік ағаштарды сондай-ақ бағдарламаны талдауда және бағдарламаның абстрактілі құрамын түсіндіргенде пайдаланады.

3.4.5 Автоматтандырылған синтаксистік талдау

Алдыңғы тарауда біз талдау ағашын құру кезінде өнімдік қағиданың оң жақ бөлігін салыстыратын тиісті жүйелілік автоматы түрде анықталатын болады деп болжадық және соңғысы тиісті өнімдік қағиданың сол жақ бөлігіндегі терминалды емес символына дейін қысқаратын болады. Осындай жүйелілікті анықтау үшін адамдар өзінің интеллектуалдық мүмкіншіліктерін пайдаланады, компьютерлерге өнімдік қағиданың оң жақ бөлігіне сәйкес келетін осындай бір мәнді жүйелілікті анықтау үшін синтаксистік талдау бағдарламасын автоматтандыру қажет.

Синтаксистік талдаудың автоматтандырылған тәсілдерін екі негізгі

санатқа бөлуге болады: *жоғарыдан төмен* типті синтаксистік талдау және *төменнен жоғары* типті синтаксистік талдау. *Жоғарыдан төмен типті синтаксистік талдауды* орындау кезінде, басқаша рекурсивті түсу тәсілі деп аталатын, ең шеткі сол жақты терминалды емес символ өнімдік қағиданың оң жақ бөлігіне дейін кеңейеді және талданатын сөйлемнің тиісті бөлігімен салыстырылады. Синтаксистік талдау бағдарламасы бастапқы қалыпқа қайтып оралады – балама шешімді табу үшін кері бағытта жүреді – егер салыстыру сәтсіз болса, балама қағидалар ұсынады.

Бастапқы қалыпқа қайта оралу – есептеу тұрғысынан тиімсіз тәсіл. Тиімділікті арттыру және бастапқы қалыпқа қайту қажеттілігін жою үшін *терминалды символды алдын ала көру* пайдаланылады. *Болжамалы синтаксистік қарау* $M \times N$ өлшемді синтаксистік талдау кестесін пайдаланады, мұнда *M*– грамматикадағы терминалды емес символдар саны және *N*– терминалды символдар саны (алдын ала қарау үшін пайдаланатын).Синтаксистік талдау кестесінің әр ұяшығы анықтаманы көрсететін бір немесе бірнеше өнімдік қағидаларға сілтемеден тұрады. Егер ұяшық бірегей өнімдік қағидаға сілтеме жасаса, онда өнімдік қағида таңдалған болып саналады және алдын ала қарау процесі аяқталады. Олай болмаған жағдайда алдын ала көп символ қаралады және бір өнімдік қағида анықталмағанша, кестенің тиісті ұяшықтары зерттелетін болады.

Төменнен жоғары типті синтаксистік талдау, сондай-ақ *ұлғаймалы синтаксистік талдау* деген атауы бар талдау талданатын сөйлемнен басталады және жоғарыға жылжиды. Синтаксистік талдау аралық қысқартылған нысандарда тиісті жүйелілікті анықтайды және оларды өнімдік қағиданың сол жақ бөлігіндегі терминалды емес символға дейін қысқартады. Ұлғаймалы талдаудың негізгі тобына *LR(k) синтаксистік талдау жатады*: *k* синтаксистік талдауды орындау кезінде бір мәнді шешім қабылдау үшін алдын ала қарау қажет символдар максималды санын білдіреді.

LR(k) синтаксистік талдау сөйлемнің оң жақ шеткі бөлігін бастап, кері бағытта талдау ағашын құрады.

LR синтаксистік талдау бірқатар басымдыққа ие: (1) ол рекурсивті емес, (2) ол бастапқы қалыпқа қайтып оралмайды, (3) оның көмегімен барлық бағдарламалық конструкцияларда синтаксистік талдау жүргізуге болады және (4) ол грамматикадағы әр түрлі мәнді таба алады. LALR (алдын ала қаралатын LR синтаксистік талдауыш) атауына ие синтаксистік талдауыштың ерекше LR(K) ішкі класы бағдарламалау тілдеріндегі синтаксистік талдау бағдарламалары үшін кеңінен пайдаланылады және

синтаксистік талдауыштың автоматтандырылған генераторларының көмегімен құрылады. Автоматтандырылған синтаксистік талдауыштарды зерттеу компиляторларды зерттеу курсына кіреді.

3.5 Семантика

Бағдарламалау тілдерінде семантиканың бес негізгі түрі пайдаланылады: *операциялық семантика, аксиомалық семантика, денотациялық семантика, әрекет семантикасы* және *мінез-құлық семантикасы*. Келесі бөлімдерде семантиканың әр түрінің ерекшеліктері түсіндіріледі.

3.5.1 Операциялық семантика

Операциялық семантика бірінші рет ALGOL 68 бөлігі ретінде сипатталды және кейіннен Плоткин тұжырымдады. Оның негізгі міндеті сөйлемнің абстрактілі командалардың тиісті жиыны көмегімен абстрактілі машинаның абстрактілі жай-күйінің анықтамасына әсер етуін сипаттау арқылы мағына беру болып табылады. Жоғары деңгейдегі басқарудың абстракция мағынасы *кішкене қадамдық абстрактілі командалардың* жүйелілігін білдіреді, орындау нәтижесінде есептеу жай-күйі жаңа есептеу жай-күйіне ауысады. Есептеу жай-күйі абстрактілі машинаға байланысты, өз кезегінде бағдарламалау парадигмасына және бағдарламалау тілдеріне негізделеді. Әр түрлі абстрактілі командаларды орындау кезінде есептеу жай-күйіндегі өзгерістер операциялық семантиканы сипаттайды. *Кішкене қадамдық абстрактілі командалардан* құралған операциялық семантика *ішкі қадамды операциялық семантика* деп аталады. *FOR циклі, шартты оператор* немесе *WHILE циклі* сияқты деректерді және жоғары деңгейлі басқару абстракциясын пайдалану арқылы есептеу жай-күйіне ауысу мәселесімен айналысатын операциялық семантика *үлкен қадамды операциялық семантика* деп аталады.
Біздің жағдайымызда төменгі деңгейлі абстрактілі командалармен жоғары деңгейлі бағдарламаларды машинамен аударуды түсіну үшін біз негізіне 2.1-тарауда сипатталған фон Нейман машинасы енгізілген машинаны алуды шештік. Абстрактілі машина императивті бағдарламалау парадигмасын үлгілей алады. Есептеу жай-күйі үштік нысанды (*орта, жад блогы, дамп*), ал бағдарлама – *ауыспалы хабарландыру, өрнектер және командалардың* жүйелілігін білдіреді. *Ауыспалы хабарландыру* ортаны өзгерту жолымен есептеу жай-күйін түрлендіреді, *команда* есептеу жай-күйін жад блогын өзгерту жолымен өзгертеді, өрнек есептеу жай-күйін өзгертпей жад блогын оқиды. Ішкі бағдарламаны шақыру ортаны, сондай-ақ жад блогын өзгертеді. Оператор *құрама* болуы мүмкін және ортаны, сондай-ақ жад блогын өзгертуі мүмкін. *Мысалы, integerx=*

10 операторлар ауыспалы хабарландыруды, сондай-ақ тағайындау операторларн да пайдаланады:
Integer ауыспалы хабарландыру ортаны өзгертеді, ал *x*= 10 тағайындау операторлар жад блогын өзгертеді.
2.4-тарауда сипатталғандай, есептеу жай-күйі σ грек әрпімен, ал есептеу жай-күйінің арасындағы ауысу үлгісі үшін оператор *S* символымен белгіленеді. Операторлың операциялық семантикасы (*S*, σ0) → σ1 ретінде анықталады. Құрама оператор <*S*; *S*s> түрге ие, мұнда *S* символы бірінші операторды білдіреді, ал *S*s символы - қалған операторлар.

(бос команда, σ) → σ
(литерал, σ) → литерал
(идентификатор, σ) → r-мағына(идентификатор) (σ) егер ((идентификатор →l-мағына) εσE и (l-мағына →r-мағына) εσS(***жаңа*** идентификатор, <σE, σS, σD>) → <σE⊕ (идентификатор →l-мағына), σS⊕ (l-мағына → белгісіз), σD> (exp1 орехр2, σ) → мағына1 ор мағына2 иσ өзгермейді мұнда (exp1, σ) → мағына1 және (exp2, σ) → мағына2, және
орε {қосу, азайту, көбейту, бөлу} (идентификатор = exp, <σE, σS, σD>) → <σE, σS⊕ (l-мағына(идентификатор) → мағына, σD>
мұнда (exp, <σE, σS, σD>) → мағына

$(\text{no-op}, \sigma) \rightarrow \sigma$
$(\text{literal}, \sigma) \rightarrow \text{literal}$
$(\text{identifier}, \sigma) \rightarrow \text{r-value(identifier)}\ (\sigma)\ \text{if}\ ((\text{identifier} \mapsto \text{l-value}) \in \sigma^E \text{ and } (\text{l-value} \mapsto \text{r-value}) \in \sigma^S$
$(\textbf{\textit{new}}\ \text{identifier}, <\sigma^E, \sigma^S, \sigma^D>) \rightarrow <\sigma^E \oplus (\text{identifier} \mapsto \text{l-value}), \sigma^S \oplus (\text{l-value} \mapsto \text{undefined}), \sigma^D>$
$(\text{exp}_1\ \text{op}\ \text{exp}_2, \sigma) \rightarrow \text{value}_1\ \text{op}\ \text{value}_2 \text{ and } \sigma \text{ does not alter}$
$\text{where } (\text{exp}_1, \sigma) \rightarrow \text{value}_1 \text{ and } (\text{exp}_2, \sigma) \rightarrow \text{value}_2, \text{ and}$
$\text{op} \in \{\text{add, subtract, multiply, divide}\}$
$(\text{identifier} = \text{exp}, <\sigma^E, \sigma^S, \sigma^D>) \rightarrow <\sigma^E, \sigma^S \oplus (\text{l-value(identifier)} \mapsto \text{value}, \sigma^D>$
$\text{where } (\text{exp}, <\sigma^E, \sigma^S, \sigma^D>) \rightarrow \text{value}$

3.17- Сурет. Қарапайым абстрактілі командалардың операциялық семантикасы.
Құрама операторлардың мағынасын түсіну үшін бізге операторлардың жүйелілік семантикасын түсіну қажет. Операторлар жүйелілігінің операциялық семантикасы мынадай түрге ие: (<*S*; *Ss*>, σ0) → (*Ss*, σ1) →* σfinal, мұнда (*S*, σ0) → σ1. "→*" символы саны операторлардың қалған жүйелілігіндегі операторлар санына тең ауысуды білдіреді,
соңғы есептеу жай-күйін білдіреді.

3.17-суретте *бос команда, литералды бағалау*, *идентификаторды көру, идентификатор мәнінің өзгеруі*, *қарапайым өрнекті бағалау, құрама өрнекті бағалау, жаңа идентификаторды жариялау және тағайын-*

дау операторлар сияқты ішкі қадамдық аса кең таралған абстрактілі командаларға арналған операциялық семантика көрсетілген.Операторлардың әрекетін түсіндіру үшін σесептеу жай-күйін <σE, σS, σD> үш операторлардың нысаны түрінде белгілейміз, мұнда σE– орта – идентификаторлар жиынының l-мағыналы жиынға түрленуі, σS жад блогын білдіреді —l-мағыналы жиынның r-мағыналы жиынға түрленуі, ал σD дампты білдіреді — шақырылатын процедураны орындау уақытында мұрағатталған немесе жабық (көрінбейтін) процедура тізіліміндегі жеке орта мен блоктар жүйелілігі.

Абстрактілі машинаның есептеуіш жай-күйі бос командаларды орындаған соң, литералды бағалаған соң, идентификаторды қараған соң және өрнекті бағалаған соң сақталады. σE ортасында идентификаторды анықтау кезінде оның l-мағынасы қаралады, ал 1-мәннің тиісті мағынасы тиісті r-мәнін табу үшін σS жад блогында қаралады.

Құрама өрнекті бағалау есептеу жай-күйін өзгертпейді. Ағымдағы σ есептеу жай-күйіне қатысты екі өрнек өлшенеді, ал екі мағынаның нәтижесін алу үшін бинарлық оператор қолданылады. <*ident*> жаңа идентификаторын жариялау ортаны өзгертеді:

σE ортасында жаңа (<*ident*>→l-мағына) байланысу қосылады.Оның үстіне тиісті σS жад блогына жаңа (l-мағына → белгісіз) байланысу қосылады.

Тағайындау операторлар бұл – құрама оператор. Тағайындау операторларының мағынасы былай сипатталады: өрнектің оң жақ бөлігін ағымдағы σ0 есептеу жай-күйінің көмегімен бағалау (1) және ағымдағы σS жад блогының жаңаруы өрнектің оң жақ бөлігінің есептелген мағынасы бар идентификатордың 1-мәнінің байланысының өзгерісі есебінен.

Ішкі қадамды абстрактілі командалардың мағынасы абстрактілі машинада анық танылса, солай жоғары деңгейдегі абстракцияның құрама мағынасы біздің санамыздың жоғары деңгейлі абстрактілі командаларға тең ішкі қадамды абстрактілі командалардың кезектілігін түсіну үшін айқын болады.

Жоғары деңгейлі басқару абстракциясы абстрактылы командалар жүйелігіне басқару ағындары диаграммасы түрінде жоғары деңгейлі басқару абстракциясының жүйелілігіне және ары қарай басқару ағындарын диаграммаға жоғары деңгейлі басқару абстракциясы сияқты түрдегі есептеу жай-күйі әсер ететін, кіші қадамды абстрактылы командалар жүйелігіне түрлендіріледі. Шартты оператор, WHILE циклі, FOR циклі, процедураны шақыру сияқты жоғары деңгейлі басқару абстракциясын түрлендіру және оларды төменгі деңгейлі абстрактылы командаларға үйлестіру 5-тарауда сипатталады.

3.5.2 Аксиомалық семантика

Бағдарламалар негізгі сәулет пен абстрактілі машинаға тәуелсіз, предикаттарды есептеу сияқты математикалық логика негізінде әзірленуі мүмкін. Бағдарламаның мағынасы фон Нейман үлгісіне негізделген абстрактілі машинаның есептеуіш жай-күйінің түрлену жүйелілігі ретінде емес, логикалық өрнек түрінде қабылданады.

Предикаттарды есептеуге негізделген *аксиомалық семантика* бірінші рет Ч. Э. Р. Хоар есептеу кезінде есептеудің жай-күйін анықтауға арналған логикалық өрнектің көмегімен сипаттады Операторды орындау логикалық өрнектің өзгеруіне байланысты есептеудің жай-күйін өзгертеді. Оператор мағынасы *шарттан кейін* және *шарт алдында* арасындағы айырмашылықтан алынады. *Шарт алдында* бұл – оператордың орындауы алдындағы логикалық өрнек, *ал шарттан кейін* бұл – оператордың орындағаннан кейін дерек шындыққа айналатын логикалық өрнек. Тағайындау операторларндағы сияқты жаңа шартқа ауысу салдарынан операторлардан туындаған аксиомалардың қарама-қайшылығы болса, алғышарттағы аксиома жаңа алынған аксиома пайдасына түседі. Болжап көрелік, командаларды орындағанға дейінгі бастапқы шарттар – {**P**}, ал операторды орындағаннан кейінгі шарт – {**Q**}. Онда аксиомалық семантиканы сипаттайтын символдық жазу мына сипатқа ие: {**P**} *S*{**Q**}, мұнда *S*– оператор.

3.16-мысал

Мысалы, тағайындау операторларн орындағаннан кейін $x = 4$ болса, логикалық шарт $x == 4$ *шынайы* болады. Егер алдыңғы шарт $x == 10 \vee Y == 9$ операторды орындағанға дейін *шынайы* болса, онда $x = 4$ тағайындау операторларн орындағаннан кейін $x == 10$ логикалық өрнегін $x == 4$ логикалық өрнегіне ауыстырғаннан кейін, $x == 4 \wedge y == 9$ логикалық өрнегі *шынайы* болады. $y = 5$ операторларн орындау $x == 4 \wedge y == 5$ *шынайы* шарттан кейінгіні құруға алып келеді. $y = 5$ операторларның мағынасы шарттан кейінгі $x == 4 \wedge y == 5$ *шынайы* мен шарт алдындағы $x == 4 \wedge y == 9$ *шынайы* арасындағы айырмаға тең.

Бағдарламалаудың қарапайым тілінде команда, командалардың жүйелілігі, *if* операторлар, итерациялық конструкциялар, мысалы, *WHILE циклі мен процедураларды шақыру* сияқты басқару абстракциялары бар. Командалардың мағынасын түсіну және аксиомалық семантиканың көмегімен соңғы шарттарды алу үшін біз осы басқару абстракцияларын орындағаннан кейінгі шарттан кейінгіні алуымыз тиіс. Тағайындау операторларның әсері 3.15-мысалда көрсетілген. Операторлардың жүй-

елілігін орындау кезінде алдыңғы операторды орындағаннан кейін кейінгі шарт алдыңғы і операторы (хабарландырулар, командалар немесе өрнек) үшін алғышарт болып табылады. Келесі қағидада {**P**} символы алғышартты білдіреді, ал *Ss* символы ағымдағы операторды орындағаннан кейін операторлардың жүйелілігін білдіреді.
S операторларн орындағаннан кейін қалған *Ss* операторлары үшін жаңа алғышарт бұл- {**Q**}, мұнда {**Q**} –*S* операторларн орындағанға дейін алғышарт {**P**} болса, {**P**} (S; Ss) → {**Q**} (Ss) мұнда {**P**} S {**Q**}
If B then S1 else S2 басқару абстракциясының аксиомалық семантикасы {**Q1**} V {**Q2**} құрама кейінгі шарт түрінде көрсетіледі.
{**Q1**}кейінгі шарт *S1* операторларн орындағаннан кейін алынады, егер егер логикалық өрнек *B* логикалық өрнегі {**P**} алғышартында болса, {**Q2**} кейінгі шарты алынады, егер *B логикалық өрнегі* {**P**}алғышартта *жалған* болса. Біз шартты операторлар үшін аксиомалық семантика қағидасын былай жазамыз:

P} if B then S*1* else S*2* {**Q1** ∨**Q2**} мұнда {**P**} S*1* {**Q1**}, егер шынайы болса B,
немесе {**P**} S*2* {Q**2**}, егер жалған B
Дизъюнкция (логикалақы немесе) then-часть және else-часть біріктіреді. Кейінгі шарт іске асырылады немесе шынайы then-бөлікпен немесе шынайы else-бөлікпен, себебі бұл бөліктер бірін-бірі болдырмайды.

3.17-мысал
Мына бағдарламалық кодты талдайық:

x = 10; z = 4; if (x > 4) then value = x; else value = z;

Бастапқы шартқа {**P**} қабылдайық. *x* = 10 операторын орындаған соң {**P**^*x*== 10} кейінгі шарты *шынайы* болады. Бұл кейінгі шарт келесі *z*= 4 тағайындау операторлар үшін алғышарт болады. *z* = 4 операторларн орындаған соң {**P**^*x* == 10 ^*z* == 4} кейінгі шарты *шынайы* болады және шартты оператор үшін алғышарт болып табылады.
Шартты операторлың кейінгі шарты {**P**^*x* == 10 ^*z*== 4 ^ (((*x*> 4) ^ (*value*== *x*)) ∨ (*x* ≤ 4) ^ (*value*== *z*)))}.
"**while***BS*" басқару абстракциясының аксиомалық семантикасы күрделі болып табылады. Қағидада WHILE циклін орындау алдындағы {**I**^**B**} алғышарты екі бөлікке бөлінеді: **I** инвариантты бөлік және қалған **B** логикалық өрнек, ал {**I**^¬ **B**} кейінгі шарты сол **I инвариантты бөліктен және** ¬ **B** терістеуден тұрады. *Инвариантты шарт* итерацияны орын-

дау уақытында өзгеріссіз қалады.
I инвариантты шартын тікелей есептеуге болмайды. WHILE цикліне арналған аксиомалық семантика мынадай түрде бейнеленген:

{**I**^**B**} while BS; {**I**^ ¬ **B**} мұнда {**I**} S {**I**}% **I**– инвариантты шарт

Аксиомалық семантиканың негізгі басымдығы мынада:

1.Ол шын мәнінде орындамалмаған бағдарламаны орындағаннан кейінгі соңғы шартты алу үшін қолданылады. Осы шектеулі деңгейдегі сипат бағдарламаны орындамай бағдарламаның дұрыстығын тексеруге көмектесуі мүмкін. Идеясы - аксиомалық семантиканың көмегімен *FC* соңғы шартын және оны қолданыстағы *FI* шартымен салыстыру. Егер соңғы алынған шарт *FC⊆FI* болса, онда бағдарламаның *қатесі болмайды* (VII қосымшаны қараңыз), ал егер соңғы орындалған шарт *FC⊆FI* және *FI⊆FC* болса,онда бағдарлама орындалған және қатесі жоқ. Егер схема кішкентай бағдарламалар үшін жұмыс істесе, онда ол үлкен бағдарламалар үшін есептеуге тиімсіз болады, (1) баламалы логикалық өрнектерді салыстыруға арналған есептеу *шығындарына байланысты*, (2) бір логикалық өрнекте сипатталған шарт басқа логикалық өрнек шығындарымен байланысты және (3) инварианты шарттарды анықтауға арналған шығындарға байланысты.
2.Болжамалы *FI* соңғы шартын біле отырып, бағдарлама кері логикалық қорытындының көмегімен бірнеше сатылар түрінде құрылуы мүмкін: бағдарлама конструкциясының кейінгі шарты мен сатылы әрекеті шынайы алғышарттар алу үшін пайдаланылады. Алғышарттар алдыңғы оператор үшін кейінгі шартқа айналады.бағдарламаны осындай кері эвристикалық талқылау 4-тарауда сипатталғандай сатылы құрылуы мүмкін.

3.5.3 Денотациялық семантика

Денотациялық семантика басқару абстракциясы немесе деректер мәнін синтаксистік қағиданы *математикалық функция* мен семантика саласындағы *семантикалық алгебраның* көмегімен түрлендіру көмегімен алады.
Сөйлем мағынасын алу үшін синтаксистік талдаудағыдай талдау ағашы құрылады, айырмашылығы талдау ағашының доғасында синтаксистік қағидаларға сәйкес семантикалық қағидалар қолданылады, ал сөйлем бөліктерінің мағыналары ішкі тораптар түрінде алады. Сөйлемнің толық мағынасын талдау ағашының бастапқы торабынан алады. Күрделі

сөйлемнің мағынасын жеке сегменттерді қызметтік құрастыру арқылы алады.
Операциялық семантика мен денотациялық семантиканың айырмашылығы мынадай: операциялық семантика есептеуіш жай-күйіндегі өзгерістерді абстрактілі машинадағы абстрактілі командаларды орындау нәтижесі ретінде сипаттайды, ал денотациялық семантика абстрактілі синтаксистік ағашты және сөйлем мағынасын алу үшін математикалық формулалармен берілген семантикалық қағидалардың үйлесуін пайдаланады. Денотациялық семантикада есептеу жай-күйі мен абстрактілі машина ұғымы жоқ.
Денотациялық семантика мен операциялық семантиканың бірнеше ұқсастықтары бар:

1.Денотациялық семантика операциялық семантика сияқты орта мен жад блогын пайдаланады.
2.Операциялық семантика да, денотациялыққ семантика да өзінің анықтамаларында абстракцияны пайдаланады.

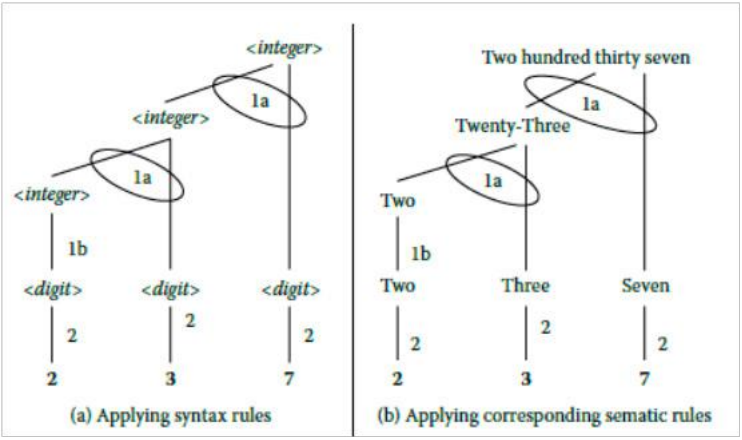
3.18-мысал
10-дық негізді жүйедегі белгісі жоқ тұтас саннан ғана тұратын қарапайым грамматиканы пайдаланып, денотациялық семантика ұғымын талдайық. 10-дық негізі бар тұтас санға сәйкес келетін грамматика төменде келтірілген

Синтаксистік қағида #1: <*тұтас сан*> ::= <*тұтас сан*><*цифр*> | <*цифр*>
Синтаксистік қағида #2: <*цифр*>::= ‘**0**’ | ‘**1**’ | ‘**2**’ | **…** | ‘**9**’

Тұтас санды алуға арналған семантика саласы бұл – 10-дық есептеу жүйесі бар тұтас сандар, Z10 ретінде белгіленеді, мұнда индекс 10-дық есептеу жүйесі негізін көрсетеді. Мәнді анықтау үшін көбейту және қосу операциясы қажет. Осылайша, семантика саласындағы семантикалық алгебра *плюс*, *көбейту* белгілерімен көрсетілген: Z10 × Z10 → Z10 бұл кіру деректері болып тұтас сандар жұбы табылады, көбейту мен қосуды орындағанда сондай-ақ 10-дық есе*птеу жүйесі негізді тұтас сан алынады.* “×” символы декартово туындыны білдіреді. Семантикалық қағида синтаксистік қағиданы Z10 семантика саласындағы тиісті мағынаға түрлендіретін ‘ĭ’ функциясын пайдаланады.

Семантика саласы: Z10

Семантикалық алгебра: қосу, көбейту: Z10 × Z10 → Z10
Семантикалық қағида # 1: ĭ (<*тұтас сан*>лч) : =ĭ (<*тұтас сан*>пч) *көбейту*
он *плюс* ĭ(<*цифр*>) | ĭ (<*цифр*>)
Семантикалық қағида # 2: ĭ (<*цифр*>) : = **нөл** |бір | **екі**| **…** | **тоғыз**
бұл біздің он, жүз және т.б. мағыналарының астында не бар екендігі туралы жасырын көрініс. Осыны болжай отырып, біз 10-дық есептеу жүйесінде цифрдың кез-келген жүйелілігін ала аламыз. Қанекей, 237 үш цифрларнан тұратын жүйелілік грамматикасының көмегімен талдап көрейік, содан кейін тиісті семантикалық қағидаларды қолдану арқылы талдау ағашын пайдалана отырып, Z10 семантика саласында екі жүз отыз жеті анықтаманы белгілейік. Талдау ағашын құрастыру мен тиісті семантикалық қағидалар арқылы мағына алу 3.18-суретте көрсетілген.
Талдау ағашында көрсетілгендей (3.18a сурет), әр символ 2-синтаксистік қағида бойынша <*цифрға*> бөлінеді. 1b синтаксистік қағидасы бойынша біз сол жақ шеттегі <*цифр*> символын <*тұтас сан*> ретінде таныстыра аламыз, ал содан кейін 1a синтаксистік қағидасын пайдалана отырып, <*тұтас сан*> анықтамасы бар басқа екі цифрды екі рет байланыстыра аламыз.
Талдау ағашының негізі <*тұтас сан*> бастапқы символы болады.
3.18b-суретте ішкі тораптағы ағаш астындағы мағыналарды және түбір торабындағы сөйлем мағынасын анықтауға арналған талдау ағашы көрсетілген.
2-семантикалық қағида ‘*2,*’ ‘*3,*’ және ‘*7*’ символдарына екі, үш және жеті деген мағына береді. 1b семантикалық қағидасы цифрды *екі* тұтас санына түрлендіреді. 1a семантикалық қағидасын қолдана отырып, *екі көбейту он қосу үш =жиырма үш* мағынасын аламыз.
Қайтадан 1a семантикалық қағидасын қолдана отырып, *жиырма үш көбейту он қосу жеті = екі жүз отыз жеті* мағынасын аламыз.



Integer – **бүтін;** digit – **сан;** two hundred thirty seven – **екі жүз отыз жеті;** twenty three – **жиырма үш;** two – **екі;** three – **үш;** seven – **жеті;** applying syntax rules - **синтаксис ережелерін қолдану;** applying corresponding sematic rules - **тиісті семантикалық ережелерін қолдану.**

3.18-сурет. Талдау ағашында семантикалық қағидаларды қолдану көмегімен мағыналарды алу.

3.19-мысал

Денотациялық семантиканы түсіну үшін бағдарламалау тілдеріндегі санды анықтау мысалын талдайық. Натурал сан домені бұл – тұтас сан домені мен нақты сан доменінің байланысты емес бірлігін қамтитын құрама домен. Соған қарамастан тұтас сан домені егер арифметикалық әрекет тұтас санды, сондай-ақ нақты санды қамтыған жағдайда нақты сан доменіне алып келуі мүмкін.Сандарға арналған синтаксистік қағида 3.19-суретте көрсетілген. Синтаксистік қағидалар жеңіл оқылады, мына тұжырымдамадан басқасы: <*тұтас теріс емес сандар* > мен <*құбылмалы нүктесі бар саны*> терминалды емес символдарын анықтау <*цифрлар*> жүйелілігін түзеді.

Соған қарамастан, семантикалық түсіндіруді жақсарту үшін оларды басқа түрде жазуға болады: <*тұтас теріс емес сан*> сол жақ рекурсивті анықтама арқылы цифрлар жүйелілігін тудырса, <*құбылмалы нүктесі бар саны*> анықтамасы артқы рекурсия анықтамасын пайдаланады. Соңғы нәтиже бірдей болғанына қарамастан, екі синтаксистік қағидалармен байланысты мағынаны беру тәсілі әр түрлі. Сол жаққа

жылжығанда *<тұтас теріс емес сан>* терминалды емес символының мағынасы *онға көбейтіледі, <құбылмалы нүктесі бар саны>* терминалды емес символының мағынасы оң жаққа жылжығанда *құбылмалы нүктесімен 10-ға бөлінеді.*

<Тұтас теріс емес сан> мағынасының тұтас сан доменінен және нақты сан доменінен айырмашылығы бар: тұтас сан доменінде ол тұтас сан ретінде қаралады; нақты сандар доменінде сан мағынасы нақты сандар доменіндегі тиісті санға *келтіріледі.*

1-қағида: *<сан>* **::=** *<тұтас сан>* | *<нақты сан>*
2-қағида: *<тұтас сан>* **::=** *<белгі><тұтас теріс емес сан>*| *<тұтас теріс емес сан>*
3-қағида: *<нақты сан>* **::=** *<белгі><белгісі жоқ нақты сан>* |*<белгісі жоқ нақты сан>*
4-қағида: *<белгісі жоқ нақты сан>* **::=** *<тұтас теріс емес сан>* '.' *<құбылмалы нүктесі бар саны>*|*<тұтас теріс емес сан>* '.' *<құбылмалы нүктесі бар саны>* '**E**'*<тұтас сан>*
5-қағида:*<тұтас теріс емес сан>* **::=***<тұтас теріс емес сан><цифр>*|*<цифр>*
6-қағида: *<құбылмалы нүктесі бар саны>* **::=** *<цифр><құбылмалы нүктесі бар саны>*| *<цифр>*
7-қағида: *<цифр>* **::=** '**0**' | '**1**' | '**2**' | … | '**9**'
8-қағида: *<белгі>* **::=** ' + ' | '−'

Rule 1: *<number>* ::= *<integer>* | *<real>*
Rule 2: *<integer>* ::= *<sign>* *<whole-number>* | *<whole-number>*
Rule 3: *<real>* ::= *<sign>* *<unsigned-real>* | *<unsigned-real>*
Rule 4: *<unsigned-real>* ::= *<whole-number>* '.' *<float>* |
<whole-number> '.' *<float>* 'E' *<integer>*
Rule 5: *<whole-number>* ::= *<whole-number><digit>*|*<digit>*
Rule 6: *<float>* ::= *<digit><float>* | *<digit>*
Rule 7: *<digit>* ::= '0' | '1' | '2' | ... | '9'
Rule 8: *<sign>* ::= ' + ' | '−'

3.19-сурет. Сандарды тексеруге арналған синтаксистік қағидалар.

Түсіндіру қағидаларына арналған семантика саласы бұл – *негізі 10-ды құрайтын натурал сандар домені,* N-мен елгіленеді. N натурал сандар

домені -бұл мен Z тұтас сан домені R нақты сандар доменін байланыссыз біріктіру

Соған қарамастан, zεZ әрбір элементі сондай мағынаға ие rεR бірегей түріне ие. Тұтас сан доменінен нақты сандар доменіне түрлену бұл – 7-тарауда талқыланған ақпараттың жоғалуынсыз түрлену *типін келтіруге* мысал.

Семантикалық алгебра *тұтас сандарды қосуды* (‘+I’), *тұтас сандарды көбейтуді* (‘×I’), нақты сандарды *қосуды* (‘+R’), нақты сандарды *көбейтуді* (‘×R’), нақты сандарды бөлуді (‘/R’) және деңгейге шығаруды (‘^’) қамтиды. *Қосу* және *көбейту* операциясының қосымша функциясы бар: қосу тұтас сандар доменінде тұтас сандарды қосу және нақты сандар доменінде құбылмалы нүктелі сан қосу болады. Соған ұқсас, көбейту операциясы тұтас сандар доменінде көбейту болады және нақты сандар доменінде құбылмалы нүктелі сандарды көбейту болады.

3.20-суретте семантикалық қағида синтаксистік қағида тәртібінде көрсетілген.

ň функциясы N натурал сандар доменінде синтаксистік қағиданы түсіндіруді білдіреді, ř функциясы R нақты сандар доменінде синтаксистік қағидаларды түсіндіруді білдіреді, ал ĭ функциясы ′ тұтас сандар доменінде түсіндіруді білдіреді.

Қанекей, әр семантикалық қағиданы зерттейік. 1-қағида ň(<*сан*>) деп белгіленген сан мағынасы ř(<*нақты сан*>) деп белгіленген нақты сан мағынасы тең немесе ĭ(<*тұтас сан*>) деп белгіленген тұтас сан мағынасына тең. 2-қағида <*тұтас сан*> терминалды емес символының мағынасы <*белгісіз тұтас сан*> терминалды емес символының мағынасына немесе <*белгісіз тұтас сан*> терминалды емес символының мағынасына көбейтілген тұтас сан доменіндегі <*белгі*> терминалды емес символының мағынасы сияқты.

3-қағидада <*нақты сан*> терминалды емес символының мағынасы нақты сандар доменіндегі <*белгі*> терминалды емес символының мағынасы <*белгісі жоқ нақты сан*> терминалды емес символының мағынасына немесе <*белгісі жоқ нақты сан*> терминалды емес мағыналарына көбейту арқылы алынады.

Semantic Domains: integer domain $\mathbb{Z}_{10}$
real number domain $\mathbb{R}_{10}$
number domain: $\mathbb{N}_{10} = \mathbb{Z}_{10} \uplus \mathbb{R}_{10}$ % $\uplus$ denotes disjoint-union
Semantic algebra in $\mathbb{R}_{10}$: real-add ‘$+^{\mathbb{R}}$’; real-multiply ‘$\times^{\mathbb{R}}$’ : $\mathbb{R}_{10} \times \mathbb{R}_{10} \to \mathbb{R}_{10}$
Semantic algebra in $\mathbb{Z}_{10}$: int-add ‘$+^{\mathbb{I}}$’; int-multiply ‘$\times^{\mathbb{I}}$’ : $\mathbb{Z}_{10} \times \mathbb{Z}_{10} \to \mathbb{Z}_{10}$
Semantic algebra in mixed domain: exponent ‘$\wedge$’: $(\mathbb{R}_{10} \times \mathbb{Z}_{10}) \to \mathbb{R}_{10}$
Semantic functions: ň; ř; ĭ;

Rule 1: ñ(<*number*>) ::= Ĩ(<*integer*>) | ř(<*real*>)
Rule 2: Ĩ(<*integer*>) ::= Ĩ(<*sign*>) ×^{I} Ĩ (<*whole-number*>) | Ĩ(<*whole-number*>)
Rule 3: ř (<*real*>) ::= ř(<*sign*>) ×^{R} ř(<*unsigned-real*>) | ř(<*unsigned-real*>)
Rule 4: ř (<*unsigned-real*>) ::= ř(<*whole-part*>) +^{R} ř(<*decimal-part*>) |
(ř(<*whole-part*>) +^{R} ř(<*decimal-part*>)) ×(10.0 ∧ Ĩ(<*integer*>))
Rule 5a: ř(<*whole-part*>1) ::= ř(<*whole-part*>2) ×^{I} 10.0 +^{R} ř(<*digit*>) | ř(<*digit*>)
Rule 5b: Ĩ(<*whole-number*>1) ::= Ĩ (<*whole-number*>2) ×^{I} ten +^{I} Ĩ(<*digit*>) | Ĩ (<*digit*>)
Rule 6: ř(<*decimal-part*>1) :: ř(<*digit*>) +^{R} ř(<*decimal-part*>2)) / 10.0 | ř(<*digit*>) / 10.0
Rule 7a: ř(<*digit*>) ::= **float zero | float one** | ... | **float nine** % float 1 is 1.0
Rule 7b: Ĩ(<*digit*>) ::= **zero | one | two** | ... | **nine** % interpretation of digits
Rule 8a: ř(<*sign*>) ::= **plus float-one | minus float-one** % + 1.0 or –1.0
Rule 8b: Ĩ(<*sign*>) ::= **plus one | minus one**

Семантика саласы: тұтас сандар домені Z10
нақты сандар домені R10
натурал сандар домені: N10 = Z10 ⊎R10 % ⊎ байланыспайтын бірігуді білдіреді
R10 –дегі **семантикалық алгебра:** нақты сандарды қосу '+R';
нақты сандарды көбейту '×R' : R10 × R10 → R10
Z10 –дегі семантикалық алгебра: тұтас сандарды қосу '+I'; тұтас сандарды көбейту '×I' : Z10 × Z10 → Z10 **Аралас домендегі семантикалық алгебра: дәрежеге шығару** ^': (R10 × Z10) → R10
Семантикалық функциялар: ň; ř; ĭ;

1-қағида: ň(<*натурал сан*>) ::= ĭ(<*тұтас сан*>) | ř(<*нақты сан*>)
2-қағида: ĭ(<*тұтас сан*>) ::= ĭ(<*белгі*>) ×Iĭ (<*тұтас теріс емес сан*>) | ĭ(<*тұтас теріс емес сан*>)
3-қағида: ř (<*нақты сан*>) ::= ř(<*белгі*>) ×Rř(<*белгісі жоқ нақты сан*>) | ř(<*белгісі жоқ нақты сан*>)
4-қағида: ř (<белгісі жоқ нақты сан>*) ::=* ř(<*тұтас теріс емес бөлік*>) +Rř(<*ондық бөлшектің бөлшек бөлігі*>) |ř(<*тұтас теріс емес бөлік*>) +Rř(<*ондық бөлшектің бөлшек бөлігі*>)) ×(10.0 ^ ĭ(<*тұтас сан*>))
5a-қағида: ř(<*тұтас теріс емес бөлік*>1) ::= ř(<*тұтас теріс емес бөлік*>2) ×I10.0 +Rř(<*цифр*>) | ř(<*цифр*>)
5b-қағида: ĭ(<*тұтас теріс емес сан*>1) ::= ĭ (<*тұтас теріс емес сан*>2) ×I десять +I ĭ(<*цифр*>) | ĭ (<*цифр*>)
6-қағида: ř(<*ондық бөлшектің бөлшек бөлігі*>1) :: ř(<*цифр*>) +R ř(<*ондық бөлшектің бөлшек бөлігі*>2)) / 10.0 | ř(<*цифр*>) / 10.0
7a-қағида: ř(<*цифр*>) ::= **құбылмалы нүктелі нөл | құбылмалы нүктелі бір** | … | құбылмалы нүктелі он % 1 құбылмалы нүктемен 1,0 тең
7b-қағида: ĭ(<*цифр*>)::= **нөл | бір| екі** |…| **он** % цифрды түсіндіру

8a-қағида: ŕ(<*белгі*>) ::= **плюс бір құбылмалы нүктесі бар сан** | **минус бір құбылмалы нүктесі бар сан** % + 1.0 or −1.0
8b-қағида: ĭ(<*белгі*>)::= **плюс бір**| **минус бір**

3.20-сурет. Синтаксистік қағидаларға сәйкес келетін семантикалық қағидалар.

1.4-қағида <*белгісі жоқ нақты сан*>-‘int’, ‘float’, ‘char’, ‘Bool’, and ‘;’, резервтелген сөздерді, сандар мен идентификаторларды енгізетін соңғы автоматты жасаңыз, резервтелген сөздер, сандар мен идентификаторлар бір реттік бос орындар бөлінуі мүмкін, ал қалған барлық басқа бос орындар жойылуы тиіс.
2.Сегіздік санды қабылдайтын грамматиканың қарапайым БНП көрінісін жазыңыз. Грамматиканы кеңейтілген БНП-ге түрлендіріңіз.
3.Келесі синтаксистік қағидаларға арналған синтаксистік диаграмманы құрыңыз:

<*блок*>**::=** ‘**{**‘ <*хабарландыру*>**;** <*операторлар*> ‘**}**’
<*операторлар*>**::=** <*оператор*>**‘;’**<*операторлар*> |
<*оператор*> ‘**;** ‘| ε
<*оператор*>**::=** <*шарт операторы*> | <*WHILE циклі*>|
<*тағайындау*> |
<*FOR циклі*> | <*процедураны шақыру*> | **return**
(<*өрнек*>)
<*WHILE циклі*>**::= while**<*логикалық өрнек*>**do**<*операторлар*>
<*шартты оператор*>**::= if**<*логикалық өрнек*>**then**<*оператор*>**else**
<*оператор*>
<*тағайындау*>**::=**<*идентификатор*> = <*өрнек*>
<*логикалық өрнек*>**::=** <*логикалық өрнек*><*l-операция*><*логикалық өрнек*> | <*предикат*> |
true| **false**
<*l-операция*>**::= ‘&&’** | **‘||’** | **‘not’**
<*предикат*>**::=** <*идентификатор*><*с-операция*><*идентификатор*>
<*с-операция*>**::= ‘==’** | **‘>’** | **‘>’** | **‘>=’** | **‘=<’**
<*өрнек*>**::=** <*өрнек*><*а-операция*><*өрнек*>
<*а-операция*>**::=** ‘ + ’ | **‘–‘** | **‘*’** | **‘/’**
<*хабарландыру*>**::=** <*тип*><*идентификаторлар жүйелілігі*>; <*хабарландыру*> | ε
<*тип*>**::= int**| **float**| **Bool**| **string**
<*идентификаторлар жүйелілігі*>**::=** <*идентификатор*>**’,’**<*идентифи-*

каторлар жүйелілігі> |
<идентификатор>

4. Логикалық өрнек пен операторды қамтитын WHILE циклі үшін қарапайым БНП-ні жазыңыз және тиісті синтаксистік диаграмманы салыңыз.
5.C++ -дегі параметрлердің берілуін зерттеңіз және тиісті синтаксистік қағиданы жазыңыз. Синтаксистік қағиданы синтаксистік диаграммаға түрлендіріңіз және синтаксистік диаграмманы оңтайландырыңыз.
6.if шартты операторының конструкциясына арналған БНКП-ге қарапайым бір мәнді грамматиканы, оған толық логикалық өрнекті қоса алғанда, C++ -ге немесе Java-ға жазыңыз және тиісті синтаксистік диаграмманы сызыңыз.
7.Синтаксистік грамматиканы (3.14-сурет) және сан мәнін табу үшін семантикалық қағидаларды (3.15-сурет) пайдалана отырып, құбылмалы нүктесі 23,416 санға арналған талдау ағашын құрыңыз.
8. "Анықталмаған" жай-күйден бастап кодтың келесі фрагментіне арналған алғышарттар мен кейінгі шарттарды жазыңыз.

x = 4; y = 6; z = 7;
if (x > y) then max = x else max = y;

9.Орта σE = [x→ 1, y→ 2, z→ 3], ал жад блогы σS = [1 → 43, 2 → 5, 3 → 10] және дамп σD= [] болған жағдайда, есептеу жай-күйі (σE, σS, σD) үштік ретінде ұсынылғанын болжай отырып, "intw = 5;" хабарландырудан кейін есептеу жай-күйін жазыңыз, одан кейін "x = x + 4; y = z;" командасы орындалады.
10. Он алтылық санды қабылдайтын грамматиканы жазыңыз және тұтас сандар саласында тиісті денотациялық семантиканы жазыңыз. Саланы, семантикалық алгебраны және бағалау функциясын анықтаңыз.
3.7.3 Толық жауап
11. Грамматиканың үш түрін: тұрақты грамматиканы, мәтіндік еркін грамматиканы және мәтіндік тәуелді грамматиканы салыстырыңыз.
12. Лексикалық талдау үшін грамматиканың басқа түрлерімен салыстырғанда тұрақты грамматика дұрысырақ? Түсіндіріңіз.
13. Символдар кестесінің лексикалық талдаудағы рөлі қандай? Қарапайым мысалда көрсетіңіз.
14. Абстрактылы синтаксис пен нақты синтаксис арасындағы айырмашылық қандай? Қарапайым мысалда түсіндіріңіз.
15. Шартты оператордағы бір мәнді еместіліктің ықтимал көзін және бір

мәнді еместілікті жою тәсілдерін қараңыз.
16. Операциялық семантика мен денотациялық семантика арасындағы айырмашылық қандай? Түсіндіріңіз.
17. Неге логикалық семантика аксиомалық семантикаға кіреді? Қарапайым мысалда түсіндіріңіз.
18. Әрекеттер семантикасын семантиканың басқа түрлерімен салыстырыңыз және әрекеттер семантикасының басымдығын көрсетіңіз.
19. Мінез-құлықтық семантика бағдарламалау тілдерінде не үшін қажет? Түсіндіріңіз.

4. Бағдарламалардағы абстракция және ақпаратпен ауысу

Негізгі тұжырымдамалар

Абстрактылы синтаксис (3.2.6-тарау); Есептеулердегі абстрактылы ұғымдар (2.4-тарау); Грамматика (3.1-тарау); Басқарушы логика (1.4.2-тарау); Деректер құрылымының тұжырымдамалары (2.3-тарау); Дискреттік құрылымдар (2.2-тарау); Графалар (2.3.6-тарау); Бағдарламалау жөніндегі білім; Ағаштар (2.3.5-тарау); Синтаксистік диаграммалар (3.3-тарау).

Бағдарлама мақсаты - соңғы шарттың талаптарын қанағаттандыратын болжамалы есептеу мәніне қол жеткізу үшін құрылымдалған деректерді басқару. Бағдарламалау бірнеше деңгейлерде орындалуы мүмкін: машиналық кодтағы бағдарламалау, ассемблер тіліндегі бағдарламалау, төменгі деңгейдегі процедуралық бағдарламалау, жоғары деңгейдегі процедуралық бағдарламалау, декларативті бағдарламалау және т.б. Абстракция саны мен анық басқару деңгейі жоғары деңгейдегі бағдарламалау тілдері мен төменгі деңгейдегі бағдарламалау тілдерін бөледі. Абстракция мақсаттарының бірі – ең аз өзгерістері бар бағдарламалық қамтамасыз етуді қайта пайдалануды қамтамасыз ету, яғни талаптарды көбейту немесе технологиялық өзгерістер кезінде бағдарламалық қамтамасыз етудің дамуына арналған шығыстарды азайту.

Алдында сипатталған абстракцияның екі түрі бар: *деректер абстракциясы және басқару абстракциясы.Деректер абстракциясы н*ақты объектілерді олардың атрибуттарының ішкі топтары үшін пайдаланылады, *ал басқару абстракциясы* командалардың жүйелілігін құру үшін пайдаланылады, олардың қабылдануы мен пайдалануын жақсартады. Деректер элементі атрибуттар жиынымен сипатталады және қордаланған проблемаларды шешуге қажетті атрибуттардың тиісті тобының көмегімен абстракциялануы мүмкін.

Деректер элементтері әдетте бағдарламадағы хабарландыру тарауында сипатталады, ал құрама басқару бағдарлама ішінде болады. Алдында

айтып өткендей, бағдарламаның хабарландыру тарауы ортаны өзгертеді, ал тағайындау операторларынан құралған бағдарламаны басқару тарауы бағдарлама жадының блогын өзгертеді.
Бағдарламалау тіліне байланысты, деректер мен басқару арасындағы шектеу нақты немесе бұлдыр болуы мүмкін:
1.Бағдарлама блогы деректер ретінде қаралуы мүмкін, ал бағдарлама деректерді орындау уақытында үдемелі режимде құрылуы мүмкін, ары қарай бағдарлама блогына түрленуі тиіс.
2.Бағдарлама мета бағдарламалардағы деректер ретінде басқа бағдарламаны пайдалануы мүмкін.
3.Класс деректер абстракциясы мен басқару абстракциясының бірыңғай пакетін құруы және осы пакеттегі элементтерді жариялай отырып, басқа есептеу объектілерінің әрекетін не ашық, не қорғалған, не жабық реттеуі мүмкін.
Алдында айтып өткендей, бағдарлама бір-бірімен ақпаратпен алмасатын бірнеше ішкі бағдарламалардан немесе модульдерден тұрады. Әзірлеуші реттеген немесе инкапсуляция көмегімен жасырылған ақпарат, сондай-ақ абстракцияның бөлігі болып табылады. Инкапсуляция деректерді жасыруға арналған табиғи шекара болып табылады. Инкапсуляцияны ішкі бағдарламалар, объектілер және кластар, модульдер немесе олардың үйлесімі арқылы жүзеге асырылады.
Кластар деректер абстракциясы, сондай-ақ басқару абстракциясы ретінде инкапсуляция арқылы ақпараттың жасырынуын қамтамасыз етеді. Деректер мен басқарудың осындай көрінісі *импорт-экспорт тетігі* немесе *мұралануды пайдалану* көмегімен реттелуі мүмкін. *Экспорт-импорт тетігі* екі элементтен тұрады: басқа модульдер мен бағдарламалар үшін инкапсулданған деректер мен бағдарламалық модульдерді *экспорт тетігінің* көмегімен көрінетін етеді және модульден деректер немесе бағдарламалық модульдерді *импорт тетігінің* көмегімен алады. Экспорт тетігі жабық деректерді ашық қылады, ал импорт тетігі экспортталған элементті жергілікті ортала көруге мүмкіншілік береді. Импорт пен экспорты тікелей бағдарламашылар жариялайды.
Императивті бағдарламалауда деректер абстракциясы мен басқару абстракциясынан құралған негізгі бірлік бұл – ішкі бағдарлама немесе объектілік бағытталған бағдарламалау парадигмасындағы модуль, негізгі бірлік бұл- ішкі класс немесе объект – класс немесе ішкі кластың қолданыстағы өкілі.
Модульдер негізіндегі тілдерде ішкі бағдарлама немесе деректер элементі модуль ішіне салынуы мүмкін және деректер элементін немесе ішкі бағдарламаның элементін қолдану *экспорт-импорт тетігін қолда-*

ну есебінен реттелуі мүмкін.
Осы тарауда біз деректер және басқару абстракциясын, бағдарлама бөліктерінің арасындағы ақпараттың алмасуын, әр түрлі абстракциялық бағдарламалық конструкцияларды зерттейміз. Соған қарамастан, императивті бағдарламалау парадигмасынан ерекшеленетін бағдарламалау парадигмаларымен байланысты нақты мысалдар парадигмалар қалай толығырақ қаралуына байланысты тиісті тарауларда сипатталатын болады.

4.1 Деректер абстракциясы

Деректер абстракциясындағы ең маңызды ұғымдардың бірі өңделуі қажет ақпаратты сақтау үшін пайдаланылатын абстрактылы деректердің элементін құру болып табылады. Мұнда ақпарат бағдарламалық модульдің әр түрлі орындарында бірнеше рет пайдаланылуы мүмкін және оны жеңіл өзгертуге болады.
Анықтама *нақты* немесе *жалпы* болады, бағдарламаны орын ауыстыру тетігін пайдалана отырып, параметрлер немесе белгілерін беру тетігі арқылы деректердің нақты абстракциясына орындау уақытында инстанциялауға болады.
Деректер элементі «литерал» немесе «есептелмеген константа» сияқты *бірлік элемент б*олуы мүмкін, оны ары қарай бөлуге болмайды немесе элемент нақты белгіленген операцияларды орындалатын жалпы қасиеттерінің бірқатарына ие элементтердің *бірігуін* білдіреді.
Жалпы қасиеттердің жиыны деректердің жалпы типтегі объектілермен шатастыруға болмайды. Біріктірілген элементтің әр түрлі элементтеріне ықтимал қайталанатын жалпы операция жасайтын бірқатар жалпы сипат жалғыз шектеу болып табылады. Егер олар деректердің бірдей типіне ие болса, мысалы, тұтас сандар сияқты, онда жалпы операция тұтас сандар операциясына жатады. Бірақ көптеген жағдайларда операциялар құрылыммен байланысты, мұнда деректердің ерекше типін алу қажеттіліг туындайды. Бұл айырмашылық 7-тараудағы типтер теориясын зерттеген соң түсінікті болады.

Деректер элементінде көрінетін аймақ бар, есептеулер осы аймақта жүргізіледі. Элементтің көрінуін көз жеткізу үшін бағдарламалау тілі нақты шекаралары бар бағдарламаның бөліктерін анықтау қажет, онда деректер элементі көрінетін болып табылады. Егер біз бағдарламада бірдей атауы бар бірнеше деректер элементінде болғымыз келген жағдайда, деректер элементінің көрінетін шекараларын белгілеу қажет. Әйтпесе, атауының сәйкес келуіне байланысты қате туындауы мүмкін, себебі бірдей атау әр түрлі деректер элементтеріне жатады. Бағдарлама

бөлігінде деректер элементіне қол жеткізуді алу үшін екі тәсіл бар:
(1) Деректер элементіне атау беру және (2) орналасқан шекараларды анықтау, мұнда ол жария етілген ұстанымға байланысты көрінуі мүмкін. Атауды пайдалану бағдарламалық модульдің әр түрлі орындарындағы деректер элементіне қол жеткізуді оңайлатады және орналасқан жердің шектері деректер элементінің көрінуі шекараларын анықтайды.
Деректер элементіне бағдарламаның әр түрлі бөліктеріне арналған бірнеше атауды пайдалана отырып, жүгінуге болады. Деректердің бір элементіне жататын бірнеше атау *бүркеншік атау* деп аталады, ары қарай толығырақ сипатталатын болады. Біз деректер элементін жариялағанда, жалпы құрылымды шаблон қажеттілігі пайда болады, себебі деректердің бірнеше элементтері деректердің осындай шаблонымен байланысты. Мұндай құрылымдарды *деректер абстракциясы* деп атайды.
Жақсы анықталған сипатқа ие деректер элементтерінің белгісіз санымен ішкі бағдарлама жұмыс істеу үшін *тұрақты* және *ауыспалы* ұғымын түсінуіміз қажет. Тұрақты – компиляция кезінде деректер элементіне берілуі мүмкін және ары қарай өзгермейтін мәнді сақтағыш. Ауыспалы жад ұяшығында көрсетіледі, ол бағдарламаны орындау процесінде бірнеше реті қайта жазылуы мүмкін. Ауыспалы жасырын компиляторомен жасалуы мүмкін немесе бағдарламаны орындау уақытында тілді орындау жүйесі немесе бағдарламашы жасауы мүмкін. Элементтегі ауыспалының көрінуі бағдарламаны орындау уақытында бірнеше рет өзгеруі мүмкін, мысалы, императивті тағайындау парадигмасында деструктивті тағайындау кезінде немесе бір рет мағына тағайындалған соң тіркелуі мүмкін. Императивті бағдарламалау парадигмасы бағдарламашыға қалауы бойынша жад ұяшығы немесе жад блогы ұғымын енгізу көмегімен өзгертуге мүмкіншілік береді. Идентификатор→ жад ұяшығы жад ұяшығының пайда болуына байланысты, ал императивті бағдарламалау парадигмасында ауыспалы деректер жад ұяшығы → деректер элементі өзгеретін болады және *өзгеретін объектілер* деп аталады. Керісінше, декларативті тілдер бағдарламашыға идентификаторды деректер элементіне тек бір рет қана түрлендіруге мүмкіндік береді: идентификатор→деректер элементі. Мағына берілген соң бағдарламашы жаңа мағына беру үшін жаңа ауыспалыны құру қажет. тек бір рет қана мағына берілетін объектілер *өзгертілмейтін объектілер* деп аталады.
Бағдарламалаудың заманауи тілдерінде деректер абстракциясының осындай кластары кеңінен таралған. Барлық заманауи бағдарламалау тілдері жекеленген элементтер мен бірігулерді қолдайды. Бірігулер *құрама типтерді, жиындарды* және *деректердің кеңейтілген құрылымдарын* қамтиды. Көптеген парадигмалық бағдарламалау тілдері деклара-

тивті, сондай-ақ императивті бағдарламалауды тануды қолдайды және *өзгеретін* және *өзгермейтін* объектілерді қолдайды.

4.1.1 Деректердің бірлік элементтері

Деректер элементін анықтауға арналған абстракция нақты объектілерді түрлендірумен байланысты және олардың басқа объектілермен өзара байланысымен байланысты. Нақты объекттің көптеген атрибуттары бар, ал бұл атрибуттар бөлінуі мүмкін. Мұндай атрибуттар негізгі математикалық салалардың, мысалы, тұтас сандардың, құбылмалы нүктелі сандардың, символдар мен қатарлардың немесе есептеу салаларының, мысалы, битер мен байттардың, семафорлардың немесе пайдалану белгілеген есептелетін салалардың көмегімен ұсынылуы мүмкін. Әр түрлі заттық салалардың әр типті элементтері бар. Көптеген тілдер қатарларды индекстелген символдар жүйелігі ретінде өңдейді, ал бірнеше тілдерде, мысалы, Java-ад немесе C#-де "қатар" деректер типінің негізгі хабарландыруы аясында қаралады, ол сондай-ақ егер қажет болса, символдар жүйелігі ретінде өңделуі мүмкін.

4.1.2 Деректердің құрама элементтері

Құрама объектігің бірден артық атрибуты бар. Құрама элементе әр алаң өзі деректің құрама элементі болуы мүмкін. Құрама объектінің үлгісі үшін кортеждерді пайдаланады. Кортеждер атаулы немесе атауы жоқ болуы мүмкін. Көптеген бағдарламалау тілдері атаулы кортеждерді пайдаланады, деректердің құрама элементін үлгілеу үшін жазу немесе структты пайдаланады.

Кортеждер мен жазулар немесе структтар арасындағы жалғыз айырмашылық – атауы бар жүйелілік шаблондары. Деректер типі хабарландыруындағы кортеждердің атаулы шаблондарының басымдығы бағдарламаны құруда ыңғайлы әр түрлі ауыспалымен байланысты бірнеше кортеждерді құру үшін пайдаланылады. Атауды енгізу бірнеше күрделі пайдаланушылық типін анықтау үшін қайта пайдаланылуы мүмкін. Бағдарламалаудың әр түрлі тілдерінде пайдаланатын синтаксистік қабылдауға тәуелсіз құрама объектілердің барлық анықтамалары кортеж түрінде көрсетіледі. Кортеждің рекурсивті анықтамасы деректердің кеңейтілген элементі үлгісінде қолданылады. Мысалы, байланысты тізім жұп түрінде көрсетіледі (ақпараттық алаң, байланыстыратын тізім); ал ағашты үштік ретінде көрсетуге болады (ақпараттық алаң, сол жақ ағаш, оң жақ ағаш).

4.1-мысал

« «Сыныпты» анықтау құрама элементтің мысалы болып табылады (4.1-сурет). Сынып бұл – түрдің 6 кортежі (*курс нөмірі, курс атауы, оқытушы, студенттер, орын, уақыт*). К*урс нөмірі*, *курс атауы* және оқытушы *атрибуттары бұл* – бірлік элементтер; *орын мен уақыт* – жұптар; ал *студенттер атрибуты* – бұл деректер элементтерінің жиыны.

«Сынып» анықтамасында көрсетілгендей, деректердің құрама элементі деректер элементінің атауынан, атрибуттар санынан және әр түрлі атрибуттардың сипаттамасынан тұрады. Әр атрибут деректер абстракциясын немесе деректер абстракциясы жиынын немесе бірлік элементті қамтитын кортеж болуы мүмкін. Мысалы, *орын* – жұп түрінде үлгіленген деректер абстракциясы. Соған балама, *студенттер* бұл – *студенттер* жиыны.

Студент бұл – пішіннің 4-кортежі (квадруполь) (студенттің_*id, студент аты, кафедра атауы, ЖОО-дағы жыл*). Әр алаң бұл – бірлік элемент. Орын түр жұбы (корпус, кабинет) түрінде абстракцияланауы мүмкін. Уақыт түр жұбы ретінде абстракцияланауы мүмкін (*басталу уақыты, ұзақтық*).

```
data abstraction: class
    abstraction-type: tuple
    attribute-size: 6
    begin attribute-description
        attribute1: single-entity course-number
        attribute2: single-entity course-name
        attribute3: single-entity instructor
        attribute4: bag of student
        attribute5: tuple location
        attribute6: tuple time
    end attribute-description
end data abstraction

data-abstraction: student
    abstraction-type: tuple
    attribute-size: 4
    begin attribute-description
                attribute1: single-entity student-id
                attribute2: single-entity student-name
                attribute3: single-entity department-name
                attribute4: single-entity years-in-college
    end attribute-description
end data-abstraction
```

```
data-abstraction: location
    abstraction-type: tuple
    begin Attribute-description
                attribute1: single-entity building
                attribute2: single-entity room
    end attribute-description
end data-abstraction

data-abstraction: time
    abstraction-type: tuple
    begin Attribute-description
                attribute1: single-entity start-time
                attribute2: single-entity duration
    end attribute-description
end data-abstraction
```

Деректер абстракциясы: сынып
абстракция типі: кортеж
атрибут өлшемі: 6
атрибуттарды сипаттауды бастау
1-атрибут: бірлік элемент курс нөмірі
2-атрибут: бірлік элемент курс нөмірі
3-атрибут: бірлік элемент оқытушы
4-атрибут: студент жиын
5-атрибут: кортеж орын
6-атрибут: кортеж уақыт
атрибуттарды сипаттауды аяқтау деректер абстракциясын аяқтау

деректер абстракциясы: студент
абстракция типі: кортеж
атрибут өлшемі: 4
атрибуттарды сипаттауды бастау
1-атрибут: id-студенттің бірлік элементі
2-атрибут: студент атының бірлік элементі
3-атрибут: кафедра атауының бірлік элементі
4-атрибут: ЖОО-дағы бірлік элемент
атрибуттарды сипаттауды аяқтау деректер абстракциясын аяқтау

деректер абстракциясы: орын
абстракция типі: кортеж
атрибуттарды сипаттауды бастау
1-атрибут: бірлік элемент корпус
2-атрибут: бірлік элемент кабинет

атрибуттарды сипаттауды аяқтау
деректер абстракциясын аяқтау

деректер абстракциясы: орын
абстракция типі: кортеж
атрибуттарды сипаттауды бастау
1-атрибут: бірлік элемент басталу уақыты
2-атрибут: бірлік элемент ұзақтық
атрибуттарды сипаттауды аяқтау
деректер абстракциясын аяқтау

4.1-сурет. "Сынып» деректері элементін абстракциялау.

Кешенді сан, рационалды сан және уақытша интервал түріндегі математикалық объект абстракцияларында пайдаланылатын екі атрибуттан тұратын кортеждің арнайы типтерінің жұбы . мысалы, кешенді сан түр жұбын білдіреді (нақты бөлім, жорамал бөлім), ал рационалды сан түр жұбын білдіреді (алым, бөлім). Оқиға сияқты абстрактылы ұғымдар жұп ретінде үлгіленуі мүмкін (оқиға атауы, оқиғаның басталуы), мұнда оқиғаның басталуы бұл – түр жұбы (уақыт, орын). Көптеген бағдарламалау тілдері, мысалы,
ADA, кіріктірілген типтер ретінде кешенді және рационалды типтерді қабылдайды. Полиномдардың үштік жиыны бар, мұнда әр үштіктің түрі бар (коэффициент, ауыспалының атауы, деңгей).

4.1.3 Деректер элементтерінің жиыны

Деректер элементтерінің жиыны бұл – құрама объектіден айырмашылығы бар көптеген элемент. Құрама объектіде әр атрибут бұл – бір объекті абстракциясының бөлігі және ол индекстелмейді. Жиында немесе жиынтықта әр элементе бұл – индекстелетін жекелеген элемент. Деректер элементтері жиынын үлгілейтін бірнеше тәсілдер бар: (1) жүйелілік— реттелген жиынтық немесе (2) түрдің жиынтық жұбы (кілт, элемент), мұнда кілт– бұл деректер элементіне қатысты жеке идентификатор.
Реттелген жиынтықта әр деректер элементі белгілі бір қалыпқа жатқызылады және индекстелетін жүйелелік түрінде, мысалы, массив түрінде немесе вектор түрінде ұсынылуы мүмкін.
Жүйелілік массивтерді, байланысты тізімдер мен векторларды пайдалану есебінен іске асырылады.
Массивтер мен векторлар индекстеледі. Байланысты тізімдер индек-

стелмейді; *i* элементті алу үшін, алдымен (*i*− 1) элементтерді өту керек. Массивтер *статикалық, жартылай динамикалық және динамикалық* болуы мүмкін. Жартылай динамикалық массив өлшемі шақыратын ішкі бағдарламаның параметрлерінің түрінде беріледі және шақырулар арасындағы аралықта өзгеруі мүмкін. Динамикалық массив жадта орналасады, мысалы, ағаш ретінде деректердің рекурсивті құрылымында іске асырылады және кеңейтілген құрылым болып табылады. Мысалы, динамикалық массив өрнегі тәсілдерінің біріне шаршы ағашы жатады. Шаршы ағашында төрт бұтақ бар.

Егер жиынтық өлшемі 1-ден 4 аралығында болса, онда бір деңгейдегі ағаш қажет болады. Егер жиынтық өлшемі 4+ 1 және 42 болса, онда екі деңгейлі ағаш қажет болады, ал егер өлшем 42 + 1 және 43 аралығында болса, онда үш деңгейлі ағаш қажет. Деректердің барлық элементтері парақтық торапта сақталады. Осындай ұсынымның басымдығы мынада: жиынтықты кеңейту үшін ағаштағы деректер элементінің дәл қалыбын индекс мәніне қарай отырып, кез-келген сәтте есептеуге болады және бұл мән ағаш теңестірілген жай-күйге жеткенде, логарифмдік уақытта орын ауыстыруы мүмкін.

Жұп жиынын (кілт, деректер элементі) декларативті тілдерді *ассоциативтіі тізімдер* деп атайды және функционалдық бағдарламалау мен объектілік-бағытталған бағдарламалауды біріктіретін заманауи мультибағдарламадағы тілде Scala –дағы *тар* де аталады. Кілт деректер элементін кілтті сақтауға кеткен қосымша жад есебінен анық емес жеке реттеуден тәуелсіз қылады; деректер элементтері әр түрлі тәртіпте кез-келген жиынтық ұяшықта орналасуы мүмкін. Деректер элементін алу үшін хэштегтеу немесе бинарлық іздеу сияқты іздеу тетігі қажет болуы мүмкін, олар қажетті кілтті табады және тиісті деректер элементін алады.

Деректер элементтерінің кез-келген динамикалық жиыны екі негізгі сипатты қамтуы мүмкін: (1) индекс немесе кілт мәні бойынша деректер элементтеріне тиімді іздеу және (2) кіру-шығару тетігі, сондай-ақ бағдарламалық генерация арқылы келіп түсетін деректердің жаңа элементтерін орналастыруға арналған бағдарламаны орындау уақытында деректер абстракциясының жиыны өлшемінің қосымша кеңеюі. Деректер элементтерін тиімді іздеуге арналған екі негізгі схема бар: (1) хэш-функция индекс мәнін табу үшін тұрақты уақыттағы кілтті орналастырады, ал содан кейін индекс тиісті массивтен деректерді алу үшін пайдаланылады және (2) іздеу ағашын логарифмдік уақытта пайдалану. Сатылы көрініске, ішінде екілік немесе үштік ағаш түріне байланысты

кілт іздеудің тиісті алгоритмінің көмегімен алынады.

4.1.4 Кеңейтілетін деректер элементтері

Кеңейтуді құру үшін рекурсивті анықтамалар мен кортеждер пайдаланылады. Деректердің белгіленген жиынтығы екі алаңнан: деректер элементі және бағдарламаны орындау уақытындағы деректер жиынтығының шектеусіз кеңейтілуіне арналған сілтемелік алаңнан тұратын кортеж ұғымын түсіну есебінен абстракцияланады.

Жиынтық хэш кестеден немесе ағаштан тұрады. Хэш кестеде кілттің немесе мәндердің реттелу қажеттілігінің жоқтығына байланысты жиынтықты сатылы іске асыру іздеудің тиімді логарифмдік уақыты үшін реттелуі мүмкін.

Жүйелілік рекурсивті *<жүйелілік> ::= (<деректер элементі><жүйелілік>)|нөл* ретінде анықталады. Осындай ұсынылғанда деректер элементтері сызықтық кеңейтіледі және кез-келген кеңею санында жинақ *нөл* негізгі сценариі арқылы тоқтатылады. Деректер жиынының өлшемі шектеулі болып табылады және бағдарламаны орындау уақытында кеңейтілуі мүмкін. Байланысты тізімдер, массивтер мен векторлар кезектілікті құру үшін пайдаланылады.

1.Кортеждерді пайдаланатын рекурсивті абстракцияның басқа мысалының түрі - *Реттік* хабарландыруларда D1 ортаны σE –ден σ’E-ге дейін жаңартады, ал D2 өзінің анықтамасында D1 хабарландырумен байланысты ауыспалы жиынтығын алу үшін жаңа σ’E ортасын пайдаланады. Мысалы, *Lisp* функционалдық бағдарламалау тілінде (*let* ((X* 4*)(Y(+ X* 4*))))* хабарландыру X ауыспалысының 4 –мәнмен байланыстыруға және *Y* ауыспалысының 8-мәнге байланыстыруға сәйкес келеді, себебі бірінші хабарландыру *Y* жарияланғанда орта бөлігі болып табылады, ол ортадан *X* байланысуды алып, *X* мәніне 4-ті қосады, нәтижесінде мән 8 болады. X 4-ке тіркелгеннен кейін орта σ’E= σE⊎ {*X*→ 4}, ал екінші хабарландырудан кейін орта σE⊎ {*X*→ 4} ⊎ {*Y*→ 8} болады.

2. *Параллельді* хабарландыруларда D1 де, D2 де байланысуды алдыңғы σEортасынан алады, ал D2 4.2.-мысалында көрсетілгендей, D1 әрекеті-не байланысты өзгермейді.

4.2-мысал

Lisp бағдарламалау тілінен мысалды талдап көрейік.

(**defun**my_square(b) % функцияны анықтау

(**let**((b (+ b 5)) (c (+b 6)))% параллельді хабарландыру
(+ (* bb) (* cc)) % b2 + c2 есептеу) % let хабарландырудың көріну аймағын жабу)
% хабарландыру функциясының көріну аймағын жабу

Let конструкциясы *b* және *c* жергілікті ауыспалылары үшін жергілікті ортаны құрады. my_square*(4)* функциясын шақыру *b* мағынасын (*b*) + 5 = 9 мағынасы ретінде есептейді, ал с мағынасы *b* + 6 = 4 + 6 = 10 ретінде есептейді.

b мағынасы жергілікті ортада 9-ға өзгереді, екінші инициализацияда параллельді тағайындау семантикасының *let* конструкциясымен шақырылған алдыңғы b = 4 мәні алынады. Соңғы жауап: 92 + 102 = 181.

Процедураны немесе функцияны шақырған кезде ортада өзгеріс болады, себебі шақыру процедурасына арналған жергілікті байланыстардың көпшілігі жасырын, яғни шақырылатын ішкі бағдарламалардың көріну аймағында көрінбейді. Ішкі бағдарламала шақырған жаңа ортада ғаламдық ауыспалы, жергілікті емес ауыспалы, импортталған ауыспалы байланыстары ғана және формалды параметр байланыстары ғана сақталады.

4.2 Басқару абстракциясы

Деректерді басқаратын командалар бағдарламалау парадигмасына байланысты абстракцияланады. Бағдарламалау парадигмасына байланыссыз

командаларды бөлуге болады: (1) *конструкторлар* — деректердің жаңа элементін құрады; (2) *мутаторлар* — идентификаторға тіркеулі, мәнді өзгертетін командалар; (3) *селекторлар* — деректердің құрама объектісінен немесе деректер объектілерінің жиынтығынан деректер элементтерін іріктеуді жүзеге асыратын командалар; (4) шарт *командалары* —шартқа байланысты нұсқалардың бірін таңдайтын командалар; (5) *итераторлар*—ұйымдастырушылық тәртіпте деректер объектілерінің жиынтығына бір операцияны циклді орындайтын командалар; (6) *есептеуіштер* —өрнекті есептейтін командалар; (7) *секвенсорлар*— басқаруды қандай да бір белгі арқылы орындайтын командалар; (8) *шақыру командалары*— қандай да бір мәнді есептеу үшін функцияны немесе процедураны белсенді ететін командалар.

Тағайындау операторы ретінде *мутатор* жад ұяшығында сақталған мәндерді түрлендіру есебінен жад блогын өзгертеді. Кейбір тілдерде жад блогы сондай-ақ деректердің жаңа элементін құруда немесе хабарлауда мәнді инициализациядау кезінде өзгереді. Мутаторлардың жүй-

елігі жағдайында әр мутатор сатылы түрде жад блогын жаңасына түрлендіреді. Жад блогы ағымдағы параметрдің мәні формалды параметрге жататын жад ұяшығымен байланысқанда немесе ауыспалы хабарландыруда нақты мағынаға инициализацияланғанда өзгереді.
Командалар жүйелілігі шарт командаларымен, ауысу операторларымен немесе итераторлармен өзгеруі мүмкін. Бағдарламаны орындау уақытында орындалған командалардан кейінгі командалар жүйесі *жалғасу* деп аталады.
Жалғасу бағдарламаның мінез-құлқының себебін түсіндіру үшін қажет және бағдарламалаудың нақты тілдеріндегі денотациялық семантиканы анықтаудың ажырамас бөлігі болып табылады. Таңдау функциясының жалғасуында, мысалы, шартта операторда бірнеше нұсқалар бар, олардың бірі шартты бағалағаннан кейін бағдарламаны орындау уақытында таңдалады. Осындай, итерациялық оператордың жалғасы шарттың шынайылығын анықтағанда циклді көп ретті айналуын қамтиды. Итеративті циклдің немесе шартты операторлардың жалғасын болжау қиын, себебі ол шартты бағалау нәтижесіне байланысты.

4.3-мысал
Ұсынылған бағдарламаның жалғасын қарастырайық.

x = 4; z = 6;
goto L; z = 8;
L: y = 5;
while (z > 4)
{x = y + 5; z = z − 1;}

x = 4 операторының жалғасы {*z* = 6; *y* = 5; **if** *z* > 4 **then exit**; *x* = *y* + 5; *z* = *z* − 1; **if** *z*> 4 **then exit;***x* = *y* + 5;*z* = *z* − 1; **if** *z*> 4 **then exit**} болып табылады. Мәтіндік тәртіпте "**goto** L" секвенсоры бағдарламаны орындау тәртібін мәтіндік тәртіпте өте жақсыға өзгертеді.осындай түрде, операторлар циклінде *Z* 4-ке тең болғанша, WHILE цикліндегі операторлар екі рет орындалады және одан кейін басқару *Z* 4-ке тең болғанша, WHILE цикліне кіреді. Осылайша, жалғасы есептеуге байланысты болады және оны бағдарламаға қарап болжау қиын.

4.2.1 Тағайындау және командалар жүйелілігі
Тағайындау операторы жад ұяшығын тағайындау операторының оң жақ бөлігіне жазылған өрнекті анықтауға арналған идентификатормен байланыстырады. Байланыстыру анықтаған соң жүзеге асырылады. Жалпы

алғанда тағайындау бір байланыстан тұрады, ол операторлың <*идентификатор*> = <*өрнек*> түрге ие екендігін білдіреді. Соған қарамастан, бірқатар тілдерде, мысалы ALGOL-68, C++, Python және Ruby бір өрнектің анықтамасымен байланысты бірнеше ауыспалысы бар тізбекті тағайындау (немесе көптеген тағайындау) қолданады. Функционалдық және объектілік-бағытталған бағдарламауды қолдайтын Ruby заманауи тіліндегі синтаксисте көптеген тағайындаулар былай жазылады:
x = y = 4 + 5 + 6 (Python және Ruby қолданады)
x, y = 4 + 5 + 6 (Scala қолданады)

Оператор “4 + 5 + 6” өрнегін есептейді және 15- мәнді *x* және *y* ауыспалы жад ұяшығымен байланыстырады.
Кейбір тілдерде, мысалы, Ruby, C++, Python, Perl және Lua, *бір реттік тағайындау* немесе *параллельді тағайындау* қолданылады. Бір реттік тағайындауда тиісті өрнектердің мәні бір уақытта бірнеше жекелеген байланысты емес ауыспалыға тағайындалады,ал өрнекті есептеу бастапқы ұяшықтың мәнін қабылдайды. Бір уақыттағы тағайындауға қойылатын негізгі талап – ауыспалы жадтың жекелеген ұяшықтарында жазылуы тиіс.
Бір уақыттағы тағайындау мына түрге ие:

var1, var2, var3, …, varN= exp1, exp2, exp3, …, expN

Жоғарыдағы оператор семантикасы expi($1 \le i \le n$) әр өрнегі σS жадтың бастапқы блогына қатысты есептеледі, ал есептелген мәндер vari($1 \le i \le n$) ауыспалыға сәйкес жад ұяшығымен байланыстырылады. Мысалы, Interactive Ruby тілінің синтаксисінде келесі көптеген x1 с 4, x2 с 5 және x3-ты 6-мен байланыстырады. Соған қарамастан, егер екі ауыспалы бүркеншік болса, онда жад блогы соңғы байланысудан тұрады.

x1, x2, x3 = 4, 5, 6 % x1 → 4; x2 → 5; x3 → 6
y, y = 8, 9 % y→ 9

Бағдарламалау тілінің парадигмасына байланысты C1; C2 түрін тағайындау операторларының жүйелілігі жад блогын сатылы немесе параллельді өзгертуге қабілетті. Жад блогын *сатылы өзгерту кезінде* алдыңғы тағайындау операторы бірінші жад блогына өзгерістер енгізеді, ал кейінгі тағайындау операторлары жадтың өзгертілген блогымен жұмыс істеуді жалғастырады. Жад блогын *параллельді өзгерту кезінде* екі немесе одан да көп тағайындау операторы бір жадтың бастапқы бло-

гымен жұмыс істейді. Жадтың әр түрлі ұяшықтарында өзгерістер орын алуы тиіс; жад ұяшығы жиынтығын оқу немесе жазу *жарыс жай-күйін* тудырады; ол – жад блогы келісілмеген және егер, бір операторлар жиыны бірнеше рет есептеу жүргізсе, әр түрлі мән беруі мүмкін. Жарыс жай-күйі толығырақ 7-тарауда сипатталған.

4.4-мысал

Мысалы, "x= 4; y=x;" түріндегі сатылы тағайындау алдымен σ'S= σS⊕(x→4) жад блогын жасау үшін алдымен σS жад блогын бұзады.мұнда σ'S= σS⊕(x→4) жад блогын жасау үшін "→" символы императивті бағдарламалау парадигмасында деструктивті жаңартуды білдіреді. Енді "y= x" тағайындау операторы σ'S жад блогынан x ауыспалы мәнін оқиды және жаңа өзгертілген σ''S= σ'S⊕(y→4) = σS⊕(x→4) ⊕(y→ 4) жад блогын құрады.

Императивті бағдарламалау парадигмасын қолдайтын бағдарламалау тілдерінде тағайындау операторы болып *мутатор* табылады. Бұл идентификатордың деструктивті жаңартылуы мүмкін жад ұяшығымен байланысты екендігін білдіреді. Алдында айтылғандай, бұзып жаңартудың басымдығы - алдыңғы есептеу нәтижелерін жою мен бағдарламалау үшін маңызды мәнге ие негізгі математикалық принциптерді бұзуы мүмкін бағдарламаның қолайсыз режимдерінің жанама әсерлері есебінен жадты қайта пайдалануда.

Декларативті бағдарламалау парадигмасы бір реттік тағайындау қасиетін пайдаланады және ауыспалы өрнектің мәнін тек бір рет қана тағайындайды және байланыстыруды анық жою мүмкін емес. Соған қарамастан, бағдарламалаудың кейбір тілдері, мысалы, Prolog, артық қайтарымдарды қолдайды, қайтарымдар – шешім нұсқасын іздеудегі алдыңғы есептеуге қайтып оралу. Артық қайтарымдарды орындау уақытында Prolog негізгі іске асыру машинасы ауыспалыны алдыңғы мәнінен ажыратуды жүзеге асырады.

Логикалық бағдарламалауда тағайындау операторының нұсқасы болып *бірегейлендіру* ұғымы табылады. Бірегейлендіруде екі логикалық терм теңестіріледі. Логикалық терм бұл – ауыспалы, тұрақты литералдарды және басқа енгізілген логикалық термдерден құралған құрама құрылым. Теңестіру кезінде есептеу жүргізілмейді, себебі термдер өрнек болып табылмайды. Оның орнына, сол жақтағы логикалық термдер мен оң жақтағы логикалық термдер әр позиция сайын салыстырылады. Егер термдердің бірі ауыспалы болса, онда екі логикалық термдердегі ауыспалының пайда болуы декларативті бағдарламалау парадигмасындағы бір реттік тағайындау әсерінен тиісті логикалық термге тіркеледі.

Егер екі тиісті терм-литерал болса, онда олар солай салыстырылады. Мысалы, $X + 4 + 3$ термін $5 + Y + 3$ термімен бірегейлегендіру кезінде $5 + 4 + 3$ аламыз, X ауыспалы 5 литералмен байланыстырылады, Y ауыспалы 4 литералмен байланыстырылады, ал 3 және 3 литералдар сәтті салыстырылады. Керісінше, $X + 4 + 3$ және $5 + Y + 2$ сәйкестендіруін жүзеге асыру мүмкін емес, себебі тиісті 3 және 2 литералдар бірбірімен салыстырылмайды.
Бірегейлендіру 10-тараудағы логикалық бағдарламалауда толық зерттеледі. Соған қарамастан, ол мұнда шартты оператор ұғымы туралы толық түсіну үшін айтылған.

4.2.2 Шартты операторлар

Шартты конструкциялар бағдарламалау тілдерінде бірдей құрылымға ие. Шарттың операторлық негізгі конструкциясы абстрактылы синтаксистік мына қағидасында ұсынылған:

<*шартты оператор*>**::= if**('<*шарт*>')' **then**<*оператор*>
[**else**<*оператор*>];
<*шарт*>**::=** <*шарт*>**&&**<*шарт*> | <*шарт*> ||
<*шарт*> |**not**<*шарт*> |
<*арифметикалық-өрнек*><*с-операция*><*арифметикалық-өрнек*> |
true| **false**
<*с-операция*>**::= '>' | '<' | '>=' | '=<' | '=='**

Әр түрлі тілдерде әр түрлі резервтелген сөздер қолданылады. Мысалы, '.**and**' орнына '**&&**' немесе '**.or**' - '||' тұруы мүмкін.
(if-then-else) шартты операторында then операторы мен else операторының ықтималдығы тең және шартты теріске шығаруды тексергеннен кейін орындау тәртібін өзгерту функционалды түрде (if-then-else) шартты операторының бастапқы нысанына функционалды тең болады, бұл "if (<*шарт*>) then<*оператор-then*>else<*оператор-else*>" функционалды if (not <*шарт*>) then<*оператор-else*>else<*оператор-then*> тең екендігін көрсетеді, себебі <*шарт*> және not<*шарты*> өзара тыйым салуды талап етеді.
Then операторы да, else операторы да (if- then-else) басқа шартты операторын қамтитын кез-келген оператор болуы мүмкін. 3-тарауда енгізілген шартты оператор жағдайында else бөлігі барлық уақытта жақын then операторына жатқызылады.
Тармақталудың басқа конструкциясына *таңдау операторы* жатады. Таңдау операторының құрылымының түрі:

case(<*өрнек*>) of :
< *1 мәндер жиыны*>: < *1 командалар жүйелілігі*>;
…
< *N мәндер жиыны*>: < *N командалар жүйелілігі*>;
otherwise: <*N+1 командалар жүйелігі*>
end case

Таңдау операторында өрнек есептеледі. Көп мәнді нәтиже мүмкін және әр кіру мәні немесе шығу мәндерінің салалары үшін әрекеттердің әр түрлі жүйелігі мүмкін. Әрекеттердің әр түрлі жүйелілігіне ие барлық нұсқалар орындалады, ал соңында егер басқа барлық нұсқалар орындалатын болса, қолданылатын catch-all операторы бар. Өрнекті бағалау негізінде бір таңдау жасалса, таңдау операторы анықталған болатынына назар аударыңыз
Lisp мына түрдегі шартты операторлар жиынын пайдаланады:

(**cond**((<*предикат1*><*өрнек1*>)
…
(<*предикатN*><*өрнекN*>) (**t**<*catch-all-өрнек*>)))

Мұндай функция жоғарыдан төмен тексеріледі; егер <*предикаті*> кез-келгені шынайы болса, онда тиісті функция орындалады; әйтпесе <*catch-all-өрнек*> орындалады.
1.Бірнеше функционалдық тілдерде қолданылған басқа шартты конструкция, мысалы Lisp-те, -Жалпы блок ол пайдаланылатын әр ішкі бағдарламада жариялануы тиіс.
2.Жалпы блокта жарияланған бірнеше ауыспалының сәйкестігі сенімсіз болып табылады.

Блоктық-құрылымданған тілдер әр бағдарламалық блокқа ақпаратты көрінетін қылуы үшін ғаламдық ауыспалыны және енгізілген бағдарламалық блоктар үшін жергілікті емес ауыспалы пайдаланады. Жергілікті емес ауыспалылар тек кіріктірілген бағдарламалық бағдарламалар үшін ғана қолжетімді. Сыртқы бағдарламалық блоктарға арналған ақпарат сыртқы бағдарламалық блокқа енгізілген бағдарламалық блокқа арналған жергілікті емес ауыспалы болып табылатын сыртқы блокқа арналған ауыспалы, жергіліктінің көмегімен берілуі мүмкін. Сонымен қатар, блокты-құрылымданған тілдер ауыспалы атауын немесе жад ұяшығының мекенжайын немесе шақырушы бағдарламалық блок пен шақы-

рылатын бағдарламалық блок арасындағы ақпаратты беруге арналған ауыспалы мәнін пайдаланады. Параметрлерді беру кезінде шақырушы бағдарламадағы аргументтер *нақты параметрлер,* ал шақырылатын бағдарламалық блоктағы тиісті аргументтер *формалды параметрлер* деп аталады. Параметрлерді беру кезінде параметрлер сәйкестігі төрт жолмен алынуы мүмкін:

1.Нақты және формалды параметрлер сол жақтан оң жаққа беттеседі, бұл позициядан позицияға дейінгі салыстыру түрі. Бұл параметрлерді салыстырудың жалпы режимі көптеген бағдарламалау тілдерінде.
2.Нақты және формалды параметрлердің атауы атау ассоциациясының көмегімен, мысалы, ADA бағдарламалау тілінде салыстырылады. Егер атау ассоциациясы пайдаланылса, онда позициялар бойынша салыстыру қажеттілігі жоқ.
3.Нақты және формалды параметрлердің сәйкестігі ішкі бағдарлама шақыруында анықталады. Атауды араластырған жағдайда бір параметр нақты параметр екі басқа формалды параметрлермен байланысады – нақты параметр атрибуттарын формалды параметрлермен байланыстыру үшін позицияларды реттеу пайдаланылады.
4.Егер ресім шақыруында аргументтер саны мен формалды параметрлер саны салыстырылмаса, онда формалды және нақты параметрлерді салыстырғаннан кейін қалған формалды және нақты параметрлер үнсіздік бойынша инициализацияланады.
5.Формалды параметрі кеңейтілетін типті болуы мүмкін, мысалы, тізім түрінде, мұнда аргументтердің белгісіз саны шақырылатын бағдарламаға берілуі мүмкін, мысалы, “param” хабарландыруының көмегімен C#.

Бағдарламалаудың объектілі-бағытталған тілдері кластардың енгізілген құрылымдарға ие болуы мүмкін. Осылайша, ауыспалы класта сипатталуы мүмкін және енгізілген ішкі кластарда көрінуі мүмкін. Класта жарияланған ауыспалы *кластың ауыспалысы* деп аталады, ол осы кластағы деректер элементтері мен барлық тәсілдер арасында көрінеді. Ауыспалы *статискалық ғаламдық ауыспалы* болуы мүмкін, ол барлық кластарға және тиісінше барлық объектілер үшін қолжетімді болады. Ауыспалы тәсіл үшін сипатты болуы мүмкін. Ақпарат тәсілдер мен объектілер арасындағы хабарламаны беру арасындағы параметрлерді беру, ғаламдық ауыспалының, класс ауыспалының көмегімен объектілер арасында берілуі мүмкін. Егер тіл модульдерді, кластар мен объектілер, мысалы, Modula-3-ті қолдаса, онда тәсілдер мен ауыспалылар импорт-экспорт

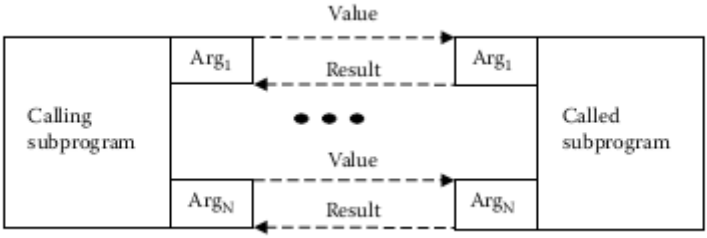
тетігі арқылы модульдер арасында пайдаланылуы мүмкін.
Бағдарламалаудың функционалды тілдерінде ақпаратпен алмасудың қуатты тетігі бар. Олар шақырылатын функцияда шақырылуы мүмкін параметр сияқты толық функцияны беруі мүмкін. Деректер, сондай-ақ функциялар сияқты функцияның екі түрлі табиғаты бұл – функционалдық тілдердің ерекшелігі.

Функционалдық бағдарламалау сондай-ақ өрнекті есептеу алдында нақты параметр атына формалды параметрді мәтіндік ауыстыруды пайдаланады. Егер өрнекті есептеу қажет болғанша кейінге қалдырылса, онда ол шақырылатын функцияға ақпаратты бергенше есептелуі мүмкін. Ол мәтіндік ауыстырылуы тиіс. Формалды параметрден нақты параметр атауына мәтіндік ауыстыруды пайдалану есебінен ақпаратты беру қасиеті *атау бойынша шақыру* деп аталады және келесі тарауда қаралады.

4.4 Параметрлерді беру

Императивті тілде ауыспалы *аты→жад ұяшығы →r-мән* деп аталады. Ауыспалығы үш атрибут жатады: *аты, жад ұяшығы* және *мән*. Осы үшеуін бергенде ғана түпкілікті r-мәнді алуға болады. Бұл параметрлерді береді. Егер атау нақты параметрде аргурмент ретінде берілсе, онда параметрді беру тетігі *атауы бойынша параметрлерді беру* деп аталады. Егер жад ұяшығы аргумент ретінде берілсе, онда параметрлерді беру тетігі *сілтеме бойынша параметрлерді беру* деп аталады, ал егер r-мәні берілсе, онда параметрлерді беру тетігі *параметрлерді мәні бойынша беру* немесе *параметрлерді көшірмесі бойынша беру* деп аталады.

Көптеген жағдайларда шақырылатын бағдарлама шақыратын бағдарламаға кері, шақырылатын ішкі бағдарламала жергілікті есептеуден алынған нәтижелерді береді. Нәтижелер ғаламдық ауыспалының, жергілікті емес ауыспалының, жадтың жалпы аймағы мен параметрлерді беруді пайдалануды бөлудің көмегімен берілуі мүмкін.



Value – **құны;** calling subprogram – **кіші бағдарламаны шақырушы;** Arg – **аргумент;** result – **нәтиже;** called subprogram – **шақырылғын кіші бағдарлама** 4.4-СУРЕТ.

4.4 –сурет. Аргументтердің r-мәнін көшіру арқылы параметрлерді беру.

Параметрлерді атауы бойынша және сілетеме бойынша беру шақырушы ішкі бағдарламадағы ауыспалымен байланысты жад ұяшығын анық емес өңдейді.

Мән бойынша параметрлерді берудің 4.4-суретте көрсетілгендей үш нұсқасы бар.

Шақыратын бағдарлама нақты параметрден формалды параметрге ақпарат бере алады. Соған қарамастан, нәтиже шақырылатын ішкі бағдарламадан шақыратын ішкі бағдарламаға кері берілмейді. Параметрді берудің осы типі *параметрлерді мәні бойынша беру* деп аталады.

Параметрлерді нәтиже мәні бойынша беруде ақпаратты беру екі нұсқамен жүзеге асырылады. Соған қарамастан, нәтиже шақырылатын бағдарлама аяқталғаннан кейін ғана кері беріледі. Ақпаратпен алмасу белгілі тәртіпте жүреді. Әдетте, позицияға сәйкес сол жақтан оң жаққа. r-мән анық берілетін, параметрді берудің соңғы тетігі *параметрлерді нәтижесі бойынша беру* деп аталады. Параметрлерді мән бойынша беруде ақпарат ішкі бағдарламаны шақыру жүріп жатқанда, нақты параметрден формалды параметрге өтпейді; формалды параметрлер үнсіздік бойынша инициализацияланады. Соған қарамастан, шақырылатын ішкі бағдарламаны аяқтағаннан кейін нәтиже формалды параметр мен нақты параметрдің сәйкестігі арқылы шақырылатын бағдарламаға қайтып оралады.

Әдетте, параметр ретінде берілетін аргументтер саны нақты анықталып, белгіленген. Бірақ, кейбір тілдер шақырушы ішкі бағдарлама беретін нақты параметрлердің ауыспалы санына рұқсат береді. Нақты параметрлердің ауыспалы саны формалды параметрдегі деректердің кеңейтіл-

ген абстракциясын білдіреді, ол деректердің кеңейтілетін абстракциясының өлшемін алатын және кеңейтілген деректер абстракциясының әр элементін өңдейтін итерациялық циклде өңделуі мүмкін. Осындай ерекшелікті C# және Java тілдері қолдайды. Мысалы, C# "paramstring[] names" формалды параметріндегі хабарландырудың көмегімен шақырушы бағдарламадағы "Mike," "Karen," и "Ambika" сияқты үш аргументті белгілейді.

"param" резервтелген сөзі шақыратын ішкі бағдарламаға ішкі бағдарламаның қалған аргументтерімен деректердің кеңейтілген абстракциясы сияқты қолдану пәрменін береді. Мұнда "names" – "Mike," "Karen" және "Ambika" деректерінің үш элементінен тұратын вектор.

Кеңейтілетін параметрлерді вектор ретінде беру басымдығы мынада: оны вектор ретінде көрсетіп, өңдеуге болады.

Логикалық бағдарламалау парадигмасында параметрлерді берудің тетіктері болып *бірегейлендіру* жатады. Бірегейлендіру мен параметрлерді берудің басқа тетіктеріндегі айырмашылық мынада: бірегейлендіру ақпарат пен тағайындауларды екіжақты беруге рұқсат береді, бірақ өрнекті есептемейді. Бірегейлендіру 10-тарауда зерттелген.

4.4.1 Параметрлерді мәні бойынша беру және оның түрлері

Өрнекте параметрлерді мәні бойынша беру (*параметрлерді көшірмесі бойынша беру*) немесе параметрлерді режимі бойынша беруде алдымен нақты параметр есептеледі, содан кейін нәтижелі мән шақыратын бағдарламаға беріледі және тиісті формалды параметрмен байланысады. Формалды параметр жергілікті ауыспалы сияқты мәртебеге ие, ал нақты параметрдегі өрнек мәні бар формалды параметрлерді байланыстыру тағайындау операторы тәрізді. Шақыратын ішкі бағдарлама ортасында шақыратын ішкі бағдарламаға жүгіну кезінде жадтың жаңа ұяшықтары жасалады және жад ұяшықтарына тиісті нақты параметрлерден есептелген өрнектердің мәндері беріледі. Бірақ көшіру тек бір бағытта ғана жүргізіледі: нақты параметрдің тиісті жад ұяшығынан формалды параметрдің жад ұяшығына. Формалды және нақты параметрлер арасында мәндерді көшірген соң ешқандай өзара әрекеттер болмайды. Шақыратын ішкі бағдарлама қандай да бір ақпарат бермей, есептеуді жүргізеді. Параметрлерді мәні бойынша беру шақыратын функциялар үшін пайдаланылады, олар нәтижені параметрлер сәйкестігінің көмегімен қайтармайды. Бірақ функциялар 4.8-мысалда көрсетілгендей "return (өрнек)" көмегімен мәнді анық қайтарады.

4.8-мысал

Келесі кодты қарастырайық. Бағдарлама “square_ sum” функциясының параметрлері мәні бойынша беру көмегімен x және y ауыспалы мәндерін береді, ал функция есептелген мәнді басты шақырушы бағдарламаға қайтарады. Ұғымды түсіндіруге арналған жалпы синтаксис.

```
Program main
{ integerx, y, z;
read(x, y);
z = square_sum(x, y)
print(“square sum of the numbers: ~d and ~d is ~d”, x, y, z);
}
function integer square_sum(a, b)
{ return(a*a + b*b);}
```

Формалды параметр “a” формалды параметрі “x” нақты параметріне сәйкес келеді, ал “b” формалды параметрі “y” нақты параметріне сәйкес келеді. Формалды параметрлер нақты параметрлер сияқты типке ие екендігіне назар аударыңыз. “a”, сондай-ақ “b” бұл – *square_sum* функциясының ортасында қосымша жад бөлетін “x” және “y” ауыспалысының көшірмелері. Параметрлерді мәні бойынша берудің негізгі басымдығы мынада: нәтиже тиісті нақты параметрге кері берілмейтіндіктен, шақырылатын ішкі бағдарлама шақырылатын процедура ортасында күтпеген деструктивті жаңартуларға байланысты бағдарламаның кездейсоқ теріс әрекетін, жанама әсерлерді болдырмау үшін жадтың тиісті ұяшығын деструктивті жаңартпайды. Параметрлерді мәні бойынша беру сондай-ақ 4.9-мысалда көрсетілгендей, параметрлердің бастапқы мәндеріне ғана қажет есептеулер үшін шақырылатын ішкі бағдарламадағы жағдайларда ғана пайдаланылады.

4.9-мысал

Келесі бағдарлама: *main* және *my_print* екі бағдарламалық блогынан тұрады. *main* бағдарламасы әрқайсысының өлшемі 100-ді құрайтын “a” and “b” екі массивін өлшейді және оларды элемент бойынша қосу үшін және нәтижесін басып шығару үшін *my_ print* ішкі бағдарламасын шақырады.

```
program main
{     integer x[100]; y[100];
for (i = 0, i = < 99; i++) read(x[i], y[i]);
call my_print(x, y);
```

```
}

subprogram my_print(integer a[100], b[100])
{ integer c[100];
for (i = 0; i = < 99; i++) {
c[i] = a[i] + b[i];
for (i = 0; i = < 99, i++)
print("c[~d]= " ~d~n", i, c[i]);
}
}
```

Параметрлерді мәні бойынша берудің екі кемшілігі бар: (1) көшіру жад ұяшықтарының қосымша санын қажет етеді, нақты параметрлерді сақтау үшін осындай қажет. Егер нақты параметрі үлкен болса, мысалы, ірі масштабты ғылыми есептеулерде матрица өлшемі 10000 × 10000 болса, онда қосымша 100 миллион × шақыратын ішкі бағдарламаны орындау үшін жад ұяшықтарының (деректердің жекелеген элементтерінің) өлшемі, және (2) шақыратын ішкі бағдарлама жадынан шақырылатын ішкі бағдарламаның жадына деректердің ірі құрылымдарын көшіруге арналған шығындар үлкен болады.

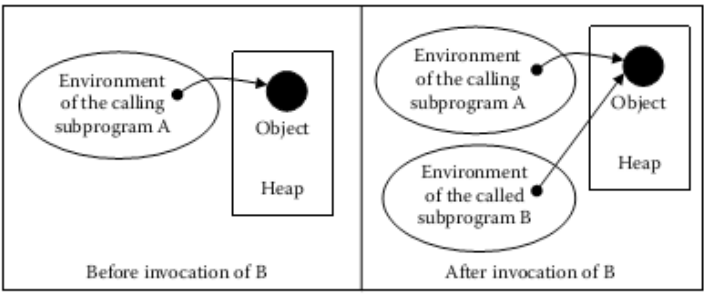
4.4.1.1 Күрделі объектілерді бөлу үшін параметрлерді мәні бойынша беру

Деректердің кеңейтілетін құрылымдарының күрделі объектілері мен бағдарламалаудың объектілік тілдеріндегі динамикалық объектілері жалпы ғаламдық кеңістіктің үдемелік саласында сақталады. Бұл объектілер сілтеменің көмегімен бағдарламалық блок ортасынан алынады. Сілтеме динамикалық аймағында сақталған деректер объектісінің негізгі мекенжайына көрсетеді. Шақыратын ішкі бағдарламаға шақырылатын ішкі бағдарламаның объектісімен бөлісу қажет болса, 4.5-суретте көрсетілгендей, параметрлерді мәні бойынша беру көмегімен шақыратын бағдарламаның ортасы көшіріледі. Сол жағы шақыратын ішкі бағдарламаға жүгінгенге дейін әрекеттер жүйелілігін көрсетеді. Көшіріп алынған ақпарат мекенжайды білдіреді, формалды параметр объект мекенжайын ауыстырады. Бұл тетік динамикалық салада сақтаулы динамикалық объектілерді қолдайтын әр тілде пайдаланылады.

4.4.2 Параметрлерді сілтеме бойынша беру және оның түрлері

Параметрлерді сілтеме бойынша беру, сондай-ақ *параметрлерді рұқсат бойынша беру* деп те аталады, 1-мәнді немесе нақты параметрдің жад ұяшығын формалды параметрге береді.

Формалды параметр бұл – нақты параметрге арналған сілтеме. Шақыратын бағдарламада нақты параметрдің жад ұяшығы формалды параметрді түсіндіруде қолжетімді, ал нақты параметрге сәйкес жад ұяшығы оқылады немесе жаңартылады.



Environment of the calling subprogram A – **А шақырушы кіші бағдарламаның қоршаған ортағы;** object – **зат;** heap – **үйінді;** Before invocation of B – **В-ны шақырудың алдында;** Environment of the calling subprogram B - **В шақырушы кіші бағдарламаның қоршаған ортағы;** after invocation of B – **В-ны шақырғаннан кейін.**
4.5-сурет. Объектілерге арналған сілтемелерді көшіру үшін параметрлерді мәні бойынша беруді пайдалану.

Деректер элементтерін, мысалы, массивті немесе векторды жинауда деректердің бірінші элементтерінің жад ұяшықтары беріледі, ал деректердің басқа элементтері жылжу тәсілінің арқасында қолжетімді болады, деректер элементінің *i* мекенжайын есептеп, *i** (деректердің бір элементін) жылжу өлшемін қоса отырып және оны формалды параметрінде сақтаулы жад ұяшығына қоса отырып,есептеледі.

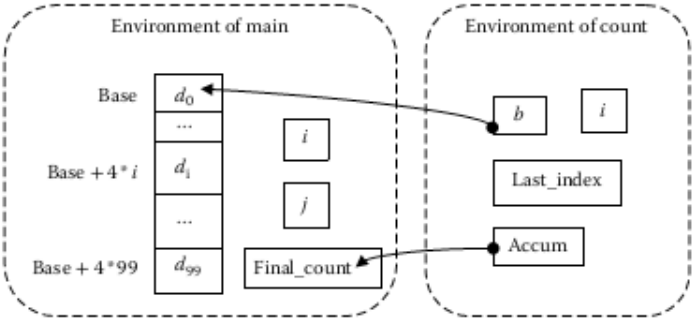
Формалды параметрдің әр түрлі атаулы мәнінің өзгерісі нақты параметрдің жад ұяшығындағы рұқсатпен және оның жаңаруымен сәйкес келетіндігі хабарландырудан айқын болады. Шақырылатын ішкі бағдарламадағы шақыруы ішкі бағдарламадағы нәтижелерді беру қажеттілігі жоқ.

Параметрлерді мәні бойынша берудің бірнеше басымдықтары бар:

1.Формалды параметр бұл – деректердің күрделі құрылымының жадының бірінші ұяшығындағы тек көрсеткіш қана және ол деректер құрылымының өлшеміне байланысты емес.

2.Жад ұяшықтарының ресурстары мен орындау уақытын артық шығындай отырып, нақты параметрдің мәнін дәл көшірудің қажеттілігі жоқ.
3.Шақырушы бағдарламаа есептеу нәтижелерін анық берудің қажеттілігі жоқ, себебі нақты параметрлердің жад ұяшықтары тиісті формалды параметрлер өзгергенде, тұрақты түрде жаңарып отырады.

4.10-мысал
Келесі бағдарлама *main* және *count* екі бағдарламалық блогынан тұрады. *Main* бағдарламалық блогы 1-ден 200-ге дейін тұтас сандарды жасау үшін бағдарламалар кітапханасында кездейсоқ сандар генераторын пайдаланады, *count* шақырылатын ішкі бағдарламасы сандық мәні 100-ден аса *d* массиві бөлігінің элементтерін санайды. Тұтас сан 4 байтты алады және көрсеткіш 4 байтты алады.
main бағдарламасында *d* тұтас сандар массиві мен *i,j,* және *final_count* үш қосымша тұтас ауыспалы сандар бар. *main* бағдарламасы үш нақты параметрді: деректерді сілтеме бойынша беру арқылы *d* массиві; және параметрді мән бойынша беру арқылы *j* ауыспалы; және деректерді сілтеме бойынша беру арқылы *final_count* ауыспалыны пайдалана отырып, *count* ішкі бағдарламасын шақырады.
j ауыспалы аргументі *d* массивінің соңғы индексі мәнін сақтайды. "&" символы мекенжайдың шақырылған ішкі бағдарламаға берілгендігін білдіреді, ал "*" символы келесі символ нақты параметрге сілтеме екендігін және r-мәннің рұқсаты атауын өзгертудің қажеттілігі жоқ екендігін білдіреді.



Environment of main - **Басты қоршаған орта;** base – **негіз;** final count - **соңғы санау;** environment of count - **санаудың қоршаған ортасы**; last index - **соңғы индекс;** accum – **жинау.**
4.6-сурет. Параметрлерді сілтеме бойынша берудің принципиалдық схемасы

count бағдарламалық блогында үш формалды параметр бар: ауыспалы *b* – d[0] деректер элементіне сілтеме, ауыспалы *last_index –j* нақты параметрінің көшірмесі, *accum* сілтемелік ауыспалысы – *final_count* нақты параметріне сілтеме. *b* және *accum* сілтемелік ауыспалысы өздері көрсететін деректер элементтерінің өлшеміне тәуелсіз. Бағдарламадағы *b[i]* элементінің мекенжайы *address(d[0])* + 4 * *i* ретінде есептеледі.
j нақты параметрі мәнді беру тетігі бойынша берілетіндігіне, ал *last_index* формалды параметрі *j* мәнінің көшірмесін алатындығына назар аударыңыз. 4.6-суретте сілтеме бойынша параметрлерді берудегі сілтемелер көрсетілген.

```
program main
{ integer d[100], i, j, final_count;
for (i = 0; i = < 99; i++) d[i] = random_number(1, 200);
j = 50;
call count (&d,j, &final_count);
}
subprogram count (integer *b, last_index, *accum)
{ integer index;
*accum = 0;
while (index =< last_index)
{ if (*b[index] > 100) *accum = *accum + 1; % end_if
}% end_while
}% end_while
}
```

Параметрлерді сілтеме бойынша беруде бірнеше кемшіліктер бар:

1.Шақырылатын ішкі бағдарлама шақыратын бағдарламаның жад блогын жаңартады. Шақырылатын бағдарламаны аяқтаған соң, шақырушы бағдарламаның жад блогы шақырылатын бағдарлама орындағанға дейін өзінің жай-күйіне қатысты өзгереді. Егер шақырушы бағдарламаға жаңару қажет болмаса, онда ол қатемен орындалады, себебі ол жадтың бұрмаланған блогындағы есептеу мәнін оқитын болады.
2.Нақты параметр қолданылған сайын жадқа екі мәрте жүгіну болады: бірінші жүгіну нақты параметр жадының ұяшығына және ары қарай нақты параметрдің жад ұяшығын r-мәнге қол жеткізу үшін пайдалану.
3.Бөлінген есептеулерде, шақырушы және шақырылатын ішкі бағдарламаларда әр түрлі процессорларда (немесе компьютерлерде) болса, олардың әр түрлі мекенжайлық кеңістіктері болады. Әр түрлі мекенжайлық кеңістіктерге қол жеткізу деректерді беру хаттамаларын, сондай-ақ шығындарды біршама арттыратын деректерді орау және ашуды

қажет етеді және сонымен қатар, деректерді беру арналары төменгі сенімділікпен сипатталады.

Сілтемелік параметрлерді дұрыс емес пайдалану салдарынан шақырушы бағдарламаның жадындағы блоктың өздігінен жаңаруымен байланысты проблеманы жою үшін бірнеше тілдер, мысалы, C++ және Modula-3 тілдерде *параметрлерді сілтеме бойынша өзгертпей беру опциясын* (тек оқу үшін қолжетімді) қолдайды. *Параметрлерді сілтеме бойынша өзгертпей беруде* нақты параметрдің мәні формалды параметрдің жад ұяшығында сақтаулы көрсеткіштің көмегімен ғана есептелуі мүмкін; тиісті нақты параметрге сәйкес жад ұяшығын деструктивті жаңартуға болмайды.

Басқа нұсқада параметрді сілтеме бойынша берудің бірінші параметрін беру параметрді мәні бойынша кейіннен берумен үйлеседі. Шақырылатын ішкі бағдарламалардың барлығының нәтижесінде бастапқы нақты параметрге қолжетімді болады: параметрлерді мәні бойыша беру көмегімен сатылы көшірілетін бірінші сілтеме әр келесі шақырылатын бағдарламада сілтеме жасайды. Мысалы, “A”бағдарламалық блогы «B» шақырылатын ішкі бағдарламасында «x» нақты параметрін беру үшін параметрлерді сілтеме бойынша беруді пайдаланады. «B» бағдарламалық блогындағы «y» формалды параметрі «B» шақырылатын ішкі бағдарламасындағы «x»-қа сілтеме болады.

Егер “B” бағдарламалық блогы басқа “C”бағдарламалық блогын шақырады және “y”-ті параметрлерді мәні бойынша беру арқылы “C” бағдарламалық блогындағы «z» формалды параметріне береді.

“B” және “C”бағдарламалық блоктары өзінің сілтемелері арқылы «x» нақты параметріне қолжетімділікке ие. Мұндай схема көптеген тілдерде, әсіресе C#, Java және C++ сияқты объектілік тілдерде кездеседі және *алмасу бойынша шақыру* деп аталады.

4.4.3 Нәтиже бойынша шақыру

Нәтиже бойынша шақыру, сондай-ақ *режимді көшіру* көмегімен параметрлерді беру деп аталады, ол мән бойынша шақыруға қарама-қарсы болып табы*лады*. *Нәтиже бойынша шақыру* уақытында нақты параметр формалды параметрдің жад ұяшығына көшірілмейді. Оның орнына, формалды параметр объект типі бойынша үнсіздікпен мәнге инициализацияланады. Шақырылатын процедураның соңында формалды параметр мәні кері нақты параметрге көшіріледі. Мән бойынша шақыру сияқты нәтиже бойынша шақыру да формалды параметрлерді жергілікті ауыспалы ретінде қабылдайды. Шақырылатын бағдарламаны орын-

дау уақытында формалды параметр мен нақты параметр арасында бір-бірімен ешқандай өзара әрекет жоқ.

4.4.4 Нәтиже мәні бойынша шақыру

Нәтиже мәні бойынша шақыру, сондай-ақ параметрлерді кіру-шығу режимі бойынша беру деп аталады, r-мәнді екі тәсілмен береді: нақты параметрде формалды параметрдегі өрнекті анықтаған соң, ол шақырылатын ішкі бағдарлама аяқталған соң нәтижені кері береді. Нәтижені кері беру белгіленген тәртіпте жад ұяшығында жүзеге асырылады. Шақырылатын ішкі бағдарламаны орындау уақытында шақырушы және шақырылатын ішкі бағдарлама арасында ешқандай өзара әрекет орын алмайды.

Мәні бойынша шақыру сияқты, нәтиже мәні бойынша шақыру да формалды параметрлерді жергілікті ауыспалы сияқты қабылдайды және шақырылатын ішкі бағдарламаның жергілікті ортасында жад ұяшығын құрады. Нәтиже мәні бойынша шақыру жадын бөлу шығындары мән бойынша шақырудағы сияқты. Бірақ нәтижені мәні бойынша шақыруда көшіру шығындары мән бойынша шақыруға қарағанда екі есе көп, себебі нәтиже мәні бойынша шақыру сондай-ақ нақты параметрлердің нәтижелерін кері береді.

4.11-мысал

Келесі бағдарлама x[100], y[100] мен z[100, 100] жергілікті ауыспалысын жасайды және y[100]-те x[100], b[100] , ал z[100, 100]-те c[100, 100]» деп өзгерту керек тиісті жад ұяшықтарына a[100] мәнін көшіреді. Түпкілікті нәтиже “multiply” ("көбейту") ішкі бағдарламасын аяқтаған соң кері беріледі, яғни x[100] –тан нәтиже a[100] –ге көшіріледі; y[100]-тен b[100]-ге көшіріледі; ал z[100, 100]-ден c[100, 100]-ге көшіріледі.

```
Program main
{   integer a[100], b[100], c[100, 100], i;
for (i = 0; i =< 99; i++) read(a[i]);
for (i = 0; i =< 99; i++) read(b[i]);
call multiply (value-result a[100], b[100], c[100, 100]);

}
subprogram multiply(integer x[100], y[100], z[100, 100])
{   integer i, j;
for (i = 0; j = < 99; i++)
```

```
for (j = 0; j = < 99; j++)
z[i, j] = x[i] * y[j];
}
```

Нәтиже мәні бойынша шақыру есептеуді жүзеге асыру нәтижесін шақырушы бағдарламаға кері бергеніне қарамастан, сілтема бойынша шақыру мен нәтиже мәні бойынша шақыруда бірқатар айырмашылықтар бар:
1.Сілтеме бойынша шақыру нақты параметрлердің өзгеруін сақтайды, себебі есептеу шақырылатын ішкі бағдарламада өтеді және созылмалы деструктивті жаңаруға байланысты соңғы мәнін сақтайды. Нәтиже мәні бойынша шақыруда түпкілікті мән процедура шақыруындағы аргумент тәртібіне байланысты болса да, нақты параметрлер шақырылатын ішкі бағдарлама сәтті аяқталғаннан кейін де жаңара береді. Нәтиже мәні бойынша шақырудағы нақты параметрдің түпкілікті мәні 4.12-мысалда көрсетілгендей, сілтеме бойынша шақырудағыдай болуы міндетті емес.

main ( );	**main** ( );
integer i;	**integer** i;
{i = 1; sub(**&**i, **&**i);}	{i = 1; sub(**value-result** i, **value-result** i);}
void sub(**integer** *j, *k);	**void** sub(**integer** j, k);
{*k = 4; *j = 2}	{k = 4; j = 2}

4.7-сурет. *Сілтеме бойынша шақыру* мен *нәтиже мәні бойынша шақыруды* салыстыру.

2.Сілтеме бойынша шақыру үшін деректердің үлкен құрылымына қолжетімді сілтемені сақтауға арналған жадтың бір ғана ұяшығы қажет, ал нәтиже мәні бойынша шақыру нақты параметрдің көшірмесін жасайды. Осылайша, деректердің үлкен құрылымына арналған жадты бөлу шығындары мән нәтижесі бойынша шақырумен салыстырғанда сілтеме бойынша шақыруда өте аз.
3.Сілтеме бойынша шақыруда қосымша жад алуына байланысты нақты мәнге қол жеткізуде қосымша шығындар бар. Егер шақырушы бағдарлама үлкен көлемді есептеулермен, үлкен массивтермен, векторлармен немесе деректер элементтерінің үлкен жиынымен байланысты операцияларды орындаса, онда жадқа қолжетімділік шығындары нәтиже мәні бойынша шақырудағы көшірудегі шығындарға қарағанда көбірек болуы мүмкін.
4.Бөлінген есептеулерде, шақырушы және шақырылатын бағдарламалар

екі түрлі процессорларға немесе компьютерге тиесілі, сілтеме бойынша шақырудағы жадқа қолжетімділік біршама болады және нақты параметрдің жергілікті көшірмесімен жұмыс істеген тиімді.

4.12-мысал.
4.7-суретте көрсетілген бағдарламаның көмегімен нәтиже мәні бойынша шақыру мен сілтеме бойынша шақыру арасындағы айырмашылықты белгілейік. Сол жақтағы бағанда сілтеме бойынша шақыру бағдарламасының нұсқасы, ал оң жақ бағанда сол бағдарламаның нәтиже мәні бойынша шақыру бағдарламасының нұсқасы көрсетілген.
«&» белгісі ауыспалының мекенжайын білдіреді, ал "*" белгісі нақты параметр мәніне қолжетімділік көрсетішкінің атауын өзгерту үшін қолданылады. Сілтеме бойынша шақыру бағдарламасының нұсқасында нақты параметрдің жад ұяшығы операторларды орындау тәртібінде жаңарады. Тиісінше, *i* ауыспалысының түпкілікті мәні 2-ге тең болады. Нәтиже мәні бойынша шақыру бағдарламасының нұсқасында нәтиже сол жақтан оңға қарай тәртіпте шақырылатын ресімнің соңында кері қайтарылады. Тиісінше, *i* ауыспалысының түпкілікті мәні "*k*" формалды параметрінің түпкілікті мәні болады, яғни 4-ке тең.

4.4.5 Атауы бойынша шақыру

Атауы бойынша шақыру бұл – формалды параметр нақты параметрдің тұтас өрнегіне қандай-да бір есептеусіз орын ауыстырғанға дейін орнын басатын параметрлерді берудің негізгі үшінші санаты және осы орнын басқан шақырылған ішкі бағдарламаның денесі *ауыстыру* деп аталатын тәсіл көмегімен сауал бойынша шақырушы процедура ортасында орындалады. Ауыстыру бұл – шақырушы процедура ортасы нақты параметрге қол жеткізсе, әрбір кезде есептелетін,есептелмеген өрнекті параметрсіз процедура. Ауыстыру нақты параметрінің мекенжайын өрнек шақырушы процедура ортасында есептелген әрбір ретте қайтарады. Егер орнын басқан соң шақырылатын ішкі бағдарлама денесі шақырушы бағдарламада жарияланған ауыспалыда атау қақтығысы туындайтын жергілікті ауыспалыдан тұрса, онда шақырылатын ішкі бағдарлама денесіндегі жергілікті ауыспалының атауы атау қақтығысын болдырмау үшін өзгереді. Атауы бойынша шақыру мен мәні бойынша шақырудағы айырмашылық мынада: атау бойынша шақыру кезінде нақты параметрдегі өрнек орнын басу алдында тікелей есептелмейді. Керісінше, ол орнын басу орындалғанша кейінге қалдырылады және нақты параметрге қолжетімді болғанда әрбір ретте қайта есептеледі.
4.13-мысал

4.8-суретте *атауы бойынша шақыру* параметрін беретін бағдарлама көрсетілген. Сол жағында нақты бағдарлама, ал оң жағында "sub"-шақырылатын ішкі бағдарламасына жүгінгеннен кейінгі орындау уақытындағы бағдарлама көрсетілген.

a, b, және *w* формалды параметрлері қандай да бір есептеулерсіз *x+ y*, *x+ z* өрнегіне және *w* –ге орны ауыстырылады.

Тағайындау операторы өрнегінің оң жақ бөлігіндегі *z* ауыспалысында *main* бағдарлама ортасында жарияланған *z* ауыспалысына сәйкес келеді *және sub* ішкі бағдарламасында z жергілікті ауыспалысымен атау қақтығысына түседі.

Осылайша, *z* ауыспалысы *z1* –ге атауын өзгертеді және шақырылатын ішкі бағдарламасындағы орнын басқан дене логикалық түрде шақыру орнына қойылады. Нәтиже 4.8-суреттің оң жақ бағанында көрсетілген. Жалпы жағдайда, өрнекті қойғаннан кейін *sub* ішкі бағдарламасының денесі шақырылатын бағдарламасы блогының қызметін атқарады. Бағдарламаны орындау кезінде *z1* (3 + 4) * (3 + 4) + (3 + 5) *(3 + 5) есептеуімен байланыстырылады, түпкілікті мәнді *w* ауыспалысы алады.

Атауы бойынша шақыру бұл – қуатты тетік. Соған қарамастан, оның кемшілігі бар:

1.Ауыстыруды пайдалану себепті есептеуді тоқтату, ол бағдарламаны орындау мен есептеуде қосымша шығындарға алып келеді.

2.Атау қақтығысын шешу сондай-ақ қосымша шығындарды қажет етеді.

Ауыстыру кезінде өрнекті есептеуді тоқтату жад ұяшығындағы идентификатордың көрінісі орындау уақытында өзгеруі мүмкін екендігін білдіреді, ал бұл құрылымы бойынша бағдарламаны түсінуден айырмашылығы бар бағдарламаның орындалу уақытындағы бағдарлама сипатында маңызды проблемаларды тудырады.

Бағдарлама	sub ішкі бағдарламасын шақырғаннан кейінгі бағдарламаның сипаты

program main
{ **integer** x, y, z;
real r;
x = 3; y = 4; z = 5;
call sub(x + y, x + z, w);
}
subprogram sub (**name** a, b, w)
{ **integer** z;
z = a * a + b * b; w = square_root(z);
}
	program main
	{ **integer** x, y, z;
	real w;
	x = 3; y = 4; z = 5;
	{ **integer** z1; % rename the variable
	z1 = (x + y) * (x + y) + (x + z)
	* (x + z);
	w = square _ root(z1);
	}
	}

4.8-сурет. Атауы бойынша шақыру тетігіне мысал.

```
program main
{   integer  i, j, k; integer a[5];
  k = 3; j = 2;

for(I=0; I=<4; i++) a[i]=0;
swap(name j, k);
swap(name k, a[k];
}
subprogram swap(name m, n);
{   integer temp;
temp = m; m = n; n = temp;
}
```

```
program main
{   integer  i, j, k; integer a[5];
   k = 3; j = 2;
for(I=0; I=<4; i++) a[i]=0;
{        integer temp;
temp = j; j = k; k = temp;
}
{ integer temp;
temp = k; k = a[k]; a[k] = temp;
}
}
```

4.9-сурет. *Атауы бойынша шақырудың* жад ұяшықтарындағы түсініспеушілік проблемасы.

Атауы бойынша шақыру бағдарламаларында, әсіресе a[i] индекстелетін элементтері бар бағдарламаларда тұжырымдамаларды бекіту өте қиын, себебі "i" мәні 4.9-суретте және 4.14-мысалда көрсетілгендей, белгіленген жад ұяшықтарымен байланысты, a[i] үшін жазылуы мүмкін бағдарламадағы "a[i]" идентификаторымен байланысты жад ұяшығын өзгерте отырып, өрнек ретінде есептелуі мүмкін.

4.14-мысал

4.9-суретте *swap* ішкі бағдарламасына бірнеше жүгінулердің көмегімен ауыспалының мәнін ауыстыратын бағдарлама көрсетілген.

Бірінші кезекте ол *j* және *k* мәндерін ауыстырады, содан кейін *k* және *a*[*k*] мәндері ауыстырылады. Ауыстыру ресімі стандартты болып табылады. Суреттегі оң жақ бағанда атауды шақыру кезіндегі бағдарламаның сипаты көрсетілген.

swap(*j, k*)-ге бірінші жүгінуде, барлығы дұрыс жұмыс істейді, *j* = 3 және *k*= 2 мәні беріледі. Бірақ екінші шақыруда *k* = *a*[*2*], 0-ге тең мән беріледі. *k* 0-ге тең болса, онда *a*[*2*] жаңаруының орнына, бағдарлама *a*[*0*]-ны 2-ге

тең ескі k мәніне дейін жаңарады.

Атауы бойынша шақыру жад ұяшықтары шатасатындықтан, императивті тілдерде қолданылады. *Атауы бойынша шақыру* алдында ALGOL-60 тілінде көрсетілген. Соңынан оны басқа бағдарламалау тілдерінің императивті тілдерінде пайдалануды тоқтатты. Соған қарамастан, *қажеттілігі бойынша шақыру* деп аталатын *атауы бойынша шақыру* нұсқасы функционалдық бағдарламалау тілдерінде пайдаланылады, ол өрнекті есептеуді қажет болғанға дейін ұстап тұрады. Haskell тілі бұл – бағдарламалау тіліне мысал, мұнда қажеттілігі бойынша шақыру пайдаланылады.

4.4.6 Қажеттілігі бойынша шақыру

Қажеттілігі бойынша шақыру бұл – *атауы бойынша шақыру* нұсқасы. Атауы бойынша шақыруға қарағанда мұнда нақты параметрдің мекенжайы өрнектің әрбір есептелуінде есептеледі, қажеттілігі бойынша шақыру бірінші есептеу уақытындағы мәнді кэштейді және өрнекті әрбір есептеу кезінде кэштелген мәнді алады. Бірінші есептеу баяуланады. Бірақ келесі есептеулер тоқтамайды.

Көптеген жағдайларда индекс 4.9-суреттегі мысалда көрсетілгендей, қайта есептелмесе, мекенжай өзгермейді және теория бойынша қажеттілік бойынша шақыру өзін әр ретте мәнді алу қажет етпейтін тиімді *атау бойынша шақыру* ретінде ұстайды. Кэштеу нәтижелері бір рет есептеледі және келесі жағдайларда көшірме жасалынады. *Қажеттілігі бойынша шақыру атауы бойынша шақыру* ретінде қаралуы мүмкін, одан кейін *мәні бойынша шақыру* жүреді, мұнда мән кэштен көшіріледі.

4.15-мысал

Атауы бойынша шақыру мен қажеттілігі бойынша шақыруды тиімді пайдалануды қарапайым мысалда салыстырайық. 4.8-суретте “z1 = (*x* + *y*) * (*x* + *y*) + (*x* + *z*) * (*x* + *z*);” тағайындау операторының оң жақ бөлігінде екі рет есептелген *x* + *y* және *x* + *z* екі өрнегі көрсетілген. Атауы бойынша шақыру өрнекті екі рет есептейді, ал *қажеттілігі бойынша шақыру* *x*+ *y* өрнегін бір рет есептеп, есептеу нәтижесін кэшке сақтайды. Келесі жолы ол *x* + *y* өрнегімен ұшырасқанда, ол кэштен мәнді алады. Осындай түрде, атауы бойынша шақыру *x*+ *z* өрнегін екі рет есептейді, ал қажеттілігі бойынша шақыру *x* + *z* өрнегінің бірінші пайда болуын есептейді және есептеу нәтижесін кэшке сақтайды және өрнек келесі пайда болғанда кэштен мәнді алады.

Қажеттілігі бойынша шақыру бағдарламалаудың функционалдық тіл-

дерінде, мысалы, Haskell-де пайдаланылады, мұнда өрнекті есептеу ол керек болмағанша, кейінге қалдырылады. Қажеттілігі бойынша шақыру өрнек мәнін сақтау және өрнектің басқа орнында пайда болған дәл осындай өрнекті есептеуді болдырмау үшін нәтиже мәнін пайдалану есебінен есептеу жылдамдығын арттырады.

4.4.7 Параметрлер түріндегі ішкі бағдарламаларды беру

Lisp, Scheme, Haskell, Ruby және Scala сияқты функционалдық бағдарламасы парадигмасын қолдайтын тілдер және ALGOL және Pascal сияқты бағдарламалаудың императивті тілдері параметрлер түріндегі функциялардың/ішкі бағдарламалардың берілуін қолдайды. Параметрлер түріндегі функцияларды беру функцияның бірінші нұсқаулығына арналған сілтемелерді және санды тексеруге арналған тиісті ортаға сілтемені және орындау уақытындағы аргументтер типтерін қамтиды. Бұлар деректер типі қатаң статикалық бақылаудағы тілдер үшін күрделі саналады, мұнда деректе элементтерінің типі компиляция уақытында жарияланады, себебі әр түрлі ішкі бағдарламалар орындау уақытында әр түрлі сауалдар арқылы шақырылуы мүмкін, ал компиляция уақытындағы кодты талдау шақырылатын функциямен алынған аргументтер типін қамтитын орта параметр түрінде берілген функциялардың аргументтер типі мен ортасына сәйкес келуін тексеру қажет.

Деректер типтері динамикалық бақыланатын бағдарламалаудың функционалдық тілдерінде осындай проблемалар туындамайды, себебі деректер типтерін тексеру орындау уақытында жүзеге асырылады.

4.4.8 Бөлінген есептеулерге арналған параметрлерді беру

Бөлінген есептеу әр түрлі процессорлардан қашықтықтан процедураларды шақыруды қажет етеді. Бұл әр түрлі ішкі бағдарламалар әр түрлі мекенжайлық кеңістіктерде орындалатындығын білдіреді. Бөлінген есептеу параметрлерді берудің үш типін пайдаланады: (1) *жылжу бойынша шақыру,* сондай-ақ *көшірме бойынша шақыру* деп аталады; (2) *сілтеме бойынша шақыру*; және (3) *келуі бойынша шақыру,* сондай-ақ *көшіру/қалпына келтіру бойынша шақыру* деп аталады. Параметрлерді берудің осы тетіктері бағдарламалау тілдерін бір процессорлы іске асыру нәтижесі бойынша мәні бойынша шақыру, сілтеме бойынша шақыру, нәтиже бойынша шақырудың мекенжайлық-кеңістіктік ұқсастықта бөлінеді. Қашықтық процедурасын шақыру өрнекті есептеуді, объект көшірмесін қашықтық торабына ауыстыруды және оны формалды параметрмен байланыстыру, қашықтық процедурасын есептеу және шақыратын процедураға кері нәтижені немесе объектіні қайтаруды қамтиды.

Келуі бойынша шақыру қашықтық процессордағы объектінің көшірмесін жасайды және шақырылатын процедураны сәтті аяқтаған соң уақытша кері көшіреді.
Орнын ауыстыру бойынша қашықтық процессорында шақырылатын процедураны есептеуге арналған объектінің көшірмесін жасайды.Бірақ объект кері көшірмейді. Келуі бойынша шақыру нәтиже мәні бойынша шақыруға ұқсас, ал орнын ауыстыруы бойынша шақыру мәні бойынша шақыруға ұқсас. Сілтеме бойынша шақыру күрделі объектіге арналған сілтемені көшіреді.
Шақыратын процедура процедурасынан шақырылатын процедурасының қашықтық процедурасына объект мекенжайын беру өткізгіштік қабілетке арналған шығындардағы бөлінген есептеудегі біршама шығындармен және жүйелік ішкі бағдарламалардың шоғырланған мекенжайлық кеңістік арқылы өтуімен байланысты. Сілтеме бойынша шақырудағы осындай шығындарға қарамастан, объектіге сілтемені деректерді беруге арналған қосымша шығындарсыз бөлінген процессорлар арасындағы жеңіл берілуі мүмкін басымдығы бар.
Emerald бөлінген бағдарламалау тілі параметрлерді берудің үш схемасын пайдаланады, соның ішінде сілтеме бойынша шақыруды, келуі бойынша шақыруды және орын ауыстыру бойынша шақыруды пайдаланады Параллельді бағдарламалауға арналған бөлінген есептеуге арналған параметрлерді беру 8-тарауда қайта қаралады

4.5 Жанама әсерлер

Процедураларда ғаламдық ауыспалының, жергілікті емес ауыспалының, шақырушы процедулар мен ұзақ уақытты деректер объектілер ортасына арналған сілтемелердің болуы себепті жергілікті емес ауыспалы құрған жад блоктарына рұқсаты бар. *Жанама әсер* шақырылатын ішкі бағдарламадан артық өмір сүретін әсер ретінде анықталады. Жанама әсер (1) жад ұяшығында өтетін, жергілікті ауыспалы құрған ортаға қатысты емес және процедураның осы сәтінде есептейтін жаңарулардан; (2) сыртқы ортамен қадағаланатын өзара әрекеттен, мысалы, файлға немесе ағынға жазу; немесе
(3) тыйым салудың пайда болуынан туындайды. Тиісті жергілікті ауыспалы блокта туындаған өзгерістер процедураны орындаған соң тоқтаса, ішкі бағдарламадан артық өмір сүрген жадтың басқа блоктарындағы өзгерістер, тұрақты объектілердегі, динамикалық объектілердегі, деректердің рекурсивті құрылымдарындағы өзгерістер осы сәтте есептеліп жатқан процедура аяқталса да, сақталады.
Жанама әсерлерге негізделген бағдарламалау нәтижені императивті тіл-

дердегі шақырушы ішкі бағдарламаға кері нәтиже беру үшін пайдаланылады. Бірақ, егер жад блогы шақырылатын ішкі бағдарламада *жадтан тыс жедел есептеу* арқылы жүйесіз өзгерсе, онда басқа бағдарламалық блоктар жад ұяшығының зақымдалғанын білмей, қасақана зақымдалған мәнді пайдалануы мүмкін. Бұл дұрыс емес есептеу нәтижесіне алып келетін, болжап болмайтын нәтижеге алып келеді.
Жанама әсердің маңызды проблемаларыны бірі – *коммутативтілікті жоғалту*. Коммутативтілік бұл – қосу және көбейту сияқты өрнекті есептеу кезіндегі көптеген операторлардың негізгі қасиеті. e1 + e2 өрнегі берілді, e1 және e2 екі өрнегі де жадтың бастапқы блогының мәнін есептеу арқылы анықталады және олар жад блогын өзгертпеуі тиіс. Соған қарамастан, егер e1 немесе e2 өрнегін есептеу кезінде жанама әсер орын алса, онда бастапқы жад блогы e1 немесе e2 өрнегін есептегеннен кейін өзгереді, ал келесі есептелетін өрнек 4.16-мысалында көрсетілгендей, өрнекте айырмашылығы бар мәнді бере отырып, жадтың жаңа блогындағы мәнді есептейтін болады.

4.16-мысал
square_sum шақырылатын функциясы параметрлерін беруге арналған сілтеме бойынша шақыруды пайдаланатын келесі бағдарламаны қарастырайық.
square_sum функциясы *x* және *y* екі ауыспалысының шаршы дәрежелейді, алынған мәнді қосады және алынған нәтижені басты бағдарламаны кері жібереді. Бірақ ол *x* және *y* нақты параметрлерін 9-дан 16-ға дейінгі мәндерге деструктивті жаңартады. Қайтқан соң, *x* мәні *square_sum* функциясының орындау нәтижесіне қосылады, B мәні 34-ке тең, ол *x*+*square_sum*(*x*, *y*) өрнегін ауыстыру кезінде алынатын 28 мәнінен айырмашылығы бар.

```
program main
{ integer A, B, x, y;
  x = 3; y = 4;
A = square_sum(&x, &y) + x; % A 28-дің орнына 34 болады
B = x + y;% B 7-нің орнына 25 болады
print(A, B, x, y);

}

function integer square_sum(integer *x, *y);
{ *x = *x * *x; *y = *y * *y; % тағайындау x және y ауыспалысын өзгер-
```

теді
return(*x + *y);
}

4.5.1 Аттар біріктіру және жанама әсерлер

Аттар біріктіру бір жад ұяшығындағы екі идентификатордың немесе бір жад ұяшығына бағыттайтын екі көрсеткіштің болуы ретінде анықталады. Егер бір ауыспалыға мән берілсе, басқа ауыспалы автоматты түрде жаңарады. Бұл егер, бағдарламашы бүркеншік аттау туралы білмесе немесе шақырылатын ішкі бағдарлама шақыратын ішкі бағдарламадан бөлек есептелсе, қатты әсер ету мүмкін. Оны 4.17-мысалда талдап көрейік.

4.17-мысал.

Келесі бағдарламада атаудың қосарлануымен байланысты туындауы мүмкін бағдарламаның болжап болмайтын дағдысын көрсету аттар біріктіру мен сілтеме бойынша шақырудың үйлесуі сипатталады. Бір-біріне тәуелсіз *main* және *swap* екі бағдарламалық блогы да дұрыс орындалады. Бұл блоктар бөлек есептеліп, кейіннен қосылуы мүмкін немесе swap (ауыстыру) бастапқы кодты білмей басқа модульден жүктелуі мүмкін.

Басты бағдарлама *swap* ішкі бағдарламасын екі рет шақырады: (1) бірінші шақыруда ол екі әртүлі жад ұяшықтарында сақтаулы мәнді ауыстырады және (2) екінші рет ол бір жад ұяшығының мәнін өзгертеді. Ауысудың ішкі бағдарламасы ауыстырылуы қажет кез-келген жергілікті ауыспалыны пайдалануды болдырмау үшін стандартты емес жазылған. Оның орнына ол арифметикалық өрнекті пайдаланады. Жаңа *x* мәні ескі *x*ескі + *y*ескі мәніне тең. Жаңа *y* мәні *x*жаңа – *y*ескі → *x*ескі + *y*ескі – *y*ескі → *x*ескі мәніне тең, ал жаңа *x* мәні *x*ескі + *y*ескі– *y*жаңа → *x*ескі+*y*ескі – *x*ескі→ *y*ескі мәніне тең.

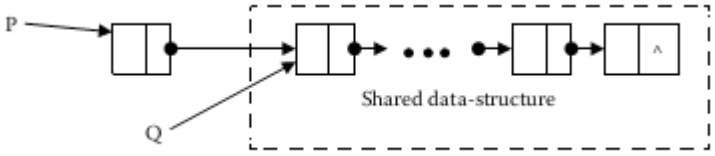
Ішкі бағдарлама қатесіз *y* және *z* мәндерін ауыстыратыны белгілі. Бірақ е *z* мәндерін ауыстырады. *swap(&x, &x)* шақырылса,екі формалды параметр де жадтың бір ұяшығына жатады және **x*= **x*+ **y* бірінші тағайындау операторы x нақты параметрінің мәнін екі еселейді. Екінші тағайындау операторы x нақты параметрінің жад ұяшығында 0-ді сақтай отырып, өзінен мәнді азайтады.

```
program main
{ integer x, y,z;
x = 3; y = 4; z = 5;
swap(&y, &z); % y және z мәндерін ауыстыру
```

```
swap(&x, &x); % жад ұяшығында 0-ді сақтау
x – болжап болмайтын сипат
print(x, y, z) % x 0-ге тең болады; Y = 5 және z = 4
}
subprogram swap(integer *x, *y);
{ *x = *x + *y;
*y = *x - *y;
*x = *x - *y;
}
```

Бірнеше көрсеткіштер бір құрылымды жадтың динамикалық аймағына бағыттаса, онда бір көрсеткішті алу, сол көрсеткішті қайта пайдалану үшін жад ұяшығын немесе 4.10-суретте көрсетілгендей жадты тазалауды көрсетеді.

4.10-суретте P және Q екі көрсеткіші көрсеткіш тізімінің екі әр түрлі бөлігін бағыттайды: P деректер басын, ал Q деректердің бірнеше ұяшығын көрсетеді. P көрсеткішін алған соң, көрсеткіштердің тұтас тізімі қайта пайдалану үшін жарамды болады, жадты тазалау бағдарламасы оны қайта пайдаланатын болады, ал жад ұяшықтары басқа деректердің динамикалық құрылымына берілетін болады. Осылайша, жад ұяшығы зақымдалады, ал бағдарлама қатемен орындалатын болады.
Жад блогында қажетсіз өзгерістер туындататын басқа проблема - орындау уақытында деректер абстракциясының шекараларының бұзылуы, оны мекенжайлық арифметика, сондай-ақ деректер құрылымының жоғарғы шекарасынан шығатын массив немесе векторға рұқсат алатын индекс мәні тудыруы мүмкін. Мысалы, егер біз *a*[*100*] массивін жарияласақ және орындау уақытында *i* индексті ауыспалысының мәні 130 болса, онда *a*[*i*] деректер құрылымына бөлінген *a*[*100*] жад шекарасынан өтіп, қайтадан қате мәнді алып келеді.



Shared data-structure - **Ортақ деректер құрылымы**

4.10-сурет. Деректер құрылымын біріге пайдаланатын көрсеткіштер.

4.5.2 Жанама әсерлерді реттеу

Бағдарламаның дұрыс орындалмауына алып келетін, жанама әсерлердің пайда болуының әр түрлі себептері бар. Бұл себептерге: (1)көріну аймағы жергілікті болып табылмайтын ауыспалыдағы жедел жадтан тыс есептеу, (2) мекенжай арифметикасы, (3) жадта көрсеткіштерді тәуелсіз орналастыру және олардың деректер құрылымын тәуелсіз көрсету мүмкіншілігі және (4) көріну аймақтары жергілікті емес ауыспалының немесе деректер объектілерінің деструктивті жаңару жатады. Жанама әсерлермен байланысты проблемаларды шешудің бірнеше тәсілдері бар. Тәсілдер тіл деңгейінде шектеуді, ал бағдарламалаудың реттелген тәсілді қамтамасыз етуі тиіс. Мына тәсілдер бар:

1.*Бағдарламаның тәртібі:* Жедел жадтан тыс есептеуге арналған жергілікті ауыспалыны пайдалану. Көріну аймақтары жергілікті ортадан тыс шығатын ауыспалылар ақпаратты беру үшін ғана өзгеруі мүмкін. Бүгінгі күні бұл - көріну аймағы жергілікті ортадан тыс шығатын ауыспалының өзгерістерінен байланысты туындаған қажетсіз жанама әсерлерді басқарудың ең танымал тәсілі.

2.*Мекенжайлық арифметикадан бас тарту:* Көрсеткіштер қандай да бір арифметикалық әрекеттерге ұшырамайды. Бұл орындау уақытында деректердің элементінен тыс көрсеткіштің шығу типі бұзылған жағдайға жатады

3.*Тәуелсіз көрсеткіштерден бас тарту:* Көрсеткіштер деректердің рекурсивті құрылымын және бағдарлама ішіндегі объектіні көрсету кезінде ғана жариялануы мүмкін. Бұл деректердің әр түрлі типтерінің үйлеспейтін әрекеттерінің проблемасын шешеді.

4.*Деструктивті жаңарудан бас тарту:* ауыспалыларды бағдарламалық бойынша мәнмен тек бір рет қана байланыстыруға болады. Деструктивті жаңару бұл шақырылатын ішкі бағдарламадағы жад блогындағы өзгерістердің басты себебі болса, онда осы жанама әрекеттің көзін жою қажет. Аталған проблеманы шешу декларативтік тілдердің алдыңғы нұсқаларында сыналды. Мәнді тағайындау қасиеті бір рет: (1) жадты қайта пайдалануды шектейді; (2) ауыспалыны артық құру себебі болып табылады; және (3) әр циклде индекстік ауыспалыны деструктивтік жаңартуды қажет ететін итерацияны қолдаудың жоқтығынан бағдарламалаудың рекурсивті стилін пайдаланады.

5.Өзгеретін объектілерді шектеп пайдалану рекурсивті бағдарламалаудың бірнеше қызғылықты стильдерінде және декларативті бағдарламалау тілдерінде итераторларды пайдалануда көрініс табады. Итераторлар соңында көптеген заманауи тілдерден алынған, себебі олар бөлшектер

деңгейі туралы ақпаратты жасыру деңгейін қамтамасыз етеді.

4.5.3 Маңызды мысал

Осы тарауда мәні бойынша шақыру, сілтеме бойынша шақыру, нәтиже мәні бойынша шақыру пайдаланылатын бағдарламалауға болжалды мысал және атаудың қосарлануымен туындаған ұяшық қатесін пайдалану қалай жүретіні туралы сипатталады. Сілтеме бойынша шақыруда нақты параметр алдында
мекенжайдың берілетіндігін білдіретін "&" белгісі қойылады және формалды параметрдің алдында нақты параметрдегі мәнге рұқсатты қамтамасыз ету үшін атауды білдіретін "*" белгісі қойылады. Нақты параметр алдында "#" символдың болуы параметрлерді беру нәтиже мәні бойынша шақыруды пайдаланатынын білдіреді.
"$" символының орналасуы параметрдің нәтиже бойынша шақыру көмегімен берілетіндігін білдіреді. 4.11-суретте *main* бағдарламасы және *messy* ішкі бағдарламасы мысалында параметрлерді берудің әр түрлі тетіктерінің әсер етуі көрсетілген.
Messy ішкі бағдарламасын шақырған соң ауыспалы *a*[*1*] ауыспалысы нәтиже бойынша шақыру арқылы беріледі, *a*[*2*] ауыспалысы сілтеме бойынша шақыру көмегімен беріледі, *j* ауыспалысы үшінші және төртінші аргументте сілтеме бойынша шақыру көмегімен беріледі, *a*[*3*] бесінші аргументі нәтиже мәні бойынша шақыру көмегімен беріледі, ал *k* ауыспалысы нәтиже бойынша шақыру көмегімен беріледі.
A формалды параметрі *a*[*1*] ауыспалысының 10-дық берілген мәнін алады, *B* формалды параметрі *a*[*2*] нақты параметріне көрсеткіш болып табылады, ал *C* және *D* формалды параметрлері *j* нақты параметрінің жад ұяшығын көрсетеді.
C және *D* формалды параметрлері бір жад ұяшығын білдіретіндіктен, *C* формалды параметрінің жад ұяшығында сақтауды көрсеткіштің жад ұяшығындағы мәндердің өзгерісі **D* мәнінің өзгеруіне және керісінше алып келеді.
E формалды параметрі *a*[*1*]= 10 мәнінің көшірмесін алады. *F* формалды параметріне 0 мәні беріледі, себебі нәтиже бойынша шақыру мәнді көшірмейді.
Бірінші оператор *a*[2] = 10 нақты параметрінің мәнін алады және *j* = 0 нақты параметрінің *A* мәні 10-ға тең болатындай етіп оларды жинайды. Екінші оператор а *j* = 0 нақты параметрінің мәнін алады және 10 деген мән берілген (*a*[3] мәнінің көшірмесі) *E* мәнін қосады және 0 деген мән берілген *F* мәні 10 мәнін алу үшін *a*[2] нақты параметінің жад ұяшығына кері жазылады. Үшінші оператор *A* және *E* ауыспалыларының мәндерін

санайды, оларды 20 мәнін алу үшін қосады және *j* нақты ауыспалысының жад блогында сақтайды.
Төртінші оператор *j* нақты параметрінің мәнін есептейді және *j* нақты параметрінің жад ұяшығында сақтайтын 0 мәнін алу үшін *j* нақты параметрінің мәнін азайтады.
Бесінші оператор *a*[2] нақты параметрінің мәні мен *j* нақты параметрінің мәнін есептейді және *E* ауыспалысына берілетін 10 мәнін алу үшін оларды қосады. Соңғы оператор 10 мәні тағайындалған E ауыспалысының мәнін есептейді.

```
program main( )
{ integer i, j, k, a[6];
  i = 0; j = 0; k = 2;
for (i = 1; i =< 5; i++) a[i] = 10;
messy(a[1], &a[2], &j, &j, #a[3], $k);
}

subprogram messy(integer  A, *B, *C, *D, E, F)
{ A = *B + *C;  % A = мағына(a[2]) +мағына(j) = 10 + 0 = 10
*B = *D + E + F; % a[2] = мағына(j) + мағына(E) + мағына(F) = 0 + 10 +
0 =10
*C = A + E; % j = мағына(A) + мағына(E) = 10 + 10 = 20
*D = *C – *D; % j = мағына(j) – мағына(j) =  20 – 20 = 0
E = *B + *C;  % E = мағына(a[2]) + мағына(j) = 10 + 0 = 10
F = E + A; % F = мағына(E) + мағына(A) = 10 + 10 = 20
}
}

subprogram messy(integer  A, *B, *C, *D, E, F)
{ A = *B + *C;  % A = (мағына) значение(a[2]) +( мағына) значение(j) =
10 + 0 = 10
*B = *D + E + F; % a[2] = (мағына) значение(j) + (мағына) значение(E) +
(мағына) значение(F) = 0 + 10 + 0 =10
*C = A + E; % j = (мағына) значение(A) + (мағына) значение(E) = 10 +
10 = 20
*D = *C – *D; % j = (мағына) значение(j) – (мағына) значение(j) =  20 –
20 = 0
E = *B + *C;  % E = (мағына) значение(a[2]) + (мағына) значение(j) = 10
+ 0 = 10
```

```
F = E + A; % F = (мағына) значение(E) + (мағына) значение(A) = 10 + 10
= 20
}
```

СУРЕТ 4.11 Параметрлер ықпалы мен аттар біріктіруінің үйлесімділігінің мысалы

Ауыспалы A-ға 10-ға тең мағынаны қосады, және 20 алынған мағынаны ауыспалы *F қосады*.
messy процедурасы аяқталғаннан кейін кері тек ауыспалы *E* және *F* мағыналары ғана көшіріледі. Ауыспалы *E* мағынасы *a*[3] нақты параметрінің жадысының ұяшығына, ал ауыспалы *F* мағынасы *K* нақты параметрінің жадысының ұяшығына көшіріледі. Нақты мағыналары болып табылады: $i = 0, j = 0, k = 20, a[0] = 10, a[1] = 10, a[2] = 10, a[3] = 10, a[4] = 10, a[5] = 10$.

4.6 Ерекше жағдайларды өңдеу

Бағдарламаны орындау кезіндегі басты мәселелердің бірі болып табысты аяқтау және "filenotfound" ("файл табылмады"), "invalidmemoryaccess" ("жадыға жол берілмейтін айналым"), "divide by zero" ("нөлге бөлу"), "unable to open a file" ("файлды ашу мүмкін емес") немесе "array out of bounds" ("массивтың шектен шығуы") тәрізді жарамсыз қателердің салдарынан күрт үзілудің алдын алу табылады. Көп жағдайларда қателер мәліметтерге байланысты болады, және қисынды қателерге жатқызылмайды. Кей жағдайларда, қателер кіріс мәліметтерінен немесе жүйенің жағдайына байланысты пайда болуы мүмкін. Мұндай жағдайларда бағдарламаның, ресурстарды блоктай отырып, мерзімінен бұрын аяқталудың орнына, файлдер, буферлер және құрылғыны қосу/шығару тәрізді пайдаланушымен үлестірілген барлық ресурстардан босап, аяқталу мүмкіндігі бар.
Қате жүйелі болуы мүмкін, мысалы, "filenotfound" ("файл табылмады"), немесе пайдаланушылық. Қателер, команда соңында қайталанатын, бағдарламалардың үзілуі- *тұзақ көмегімен* операциялық жүйемен түзетілуі мүмкін. Алайда, тұзақтар операциялық жүйенің деңгейінде болады, және қате жағдайды анықтай алмайды. Бағдарлама уақытысынан бұрын аяқталмайтындай, қате жағдайды анықтау үшін, бағдарламалау тілінде, бағдарламалық құрылғымен анықталатын, ерекше жағдайларды өңдеушіні құру қажет.
Ерекше жағдайларды өңдеу (қателерді) – пайда болу себебінен есептеу нәтижесі қатені құрайтын болатын, патологиялық жағдайды араңдата-

тын немесе бағдарламаны мерзімінен бұрын үзе алатын, қате немесе жағымсыз жағдайға жауап ретінде әрекеттер қабылдайтын, бағдарламалау тілінің деңгейіндегі абстракция. Ерекше жағдайлар жүйелі немесе пайдаланушы болуы мүмкін. Ерекше жағдайлар ұсталынатын, тілдер пайдаланушымен анықталған ерекше жағдайларды ұстана алады. Жанжақты ерекше жағдайлар, ерекше жағдайларды өңдеушіде тексеріледі. Пайдаланушымен анықталатын, ерекше жағдайлардың өңдеушісін *ерекшее жағдайлардың түрі* ретінде хабарлау қажет. *Ерекше жағдайлар түрі*– бұл, туындалатын қосалқы бағдарламалардың тізбегінде қосалқы бағдарламамен айналымда тексерілетін, статистикалық қисынды ауыспалы . Әр түрлі тілдер, ерекше жағдайларды өңдеушілермен, және оны туындататын шаралар арасындағы ерекше жағдайлардың көрерлігі саласындағы өңдеудің әр түрлі тетіктерін қолданады. Бір сызба ерекше жағдайларды өңдеушінің қосалқы бағдарламасына ерекше жағдайды беру үшін мағынасы бойынша шақыруды пайдалануды қарастырады.

Егер де ерекше жағдайлар өңдеушісі ерекше жағдайлардың күйінің белгісін растай алса, онда тиісті бағдарлама немесе орнатылған командалардың реттілігі орындалатын болады. Ерекше жағдайлардың тиісті өңдеушісі болмаған жағдайда, басқару жоғары деңгейдегі шақырылатын келесі қосалқы бағдарламаға беріледі. Басқаруды келесі деңгейге беру алдында қосымша блок орындалады, оның артынан сақталған "*finally"* сөзі ереді. Ақырғы блоктың хабарламасының міндетті емес екендігіне назар аударыңыз.

Егер де басқарма осы деңгейде тиісті өңдеушісін тапса, онда ерекше жағдайларды өңдеуші орындалады. Әйтпесе, үдеріс қайталанады, және басқарма басты бағдарламаға ауыспайынша, туындатушы қосалқы бағдарламалар тізбегіндегі келесі деңгейге басқарма ауысады. Егере де ерекше жағдайлардың тиісті өңдеушісі табылмаса, онда бағдарламаның басты пішініне қол жетілгеннен кейін, бағдарлама аяқталады.

Әр түрлі тілдер ерекше жағдайлардың өңдеушісі үшін әр түрлі синтаксиспен сипатталады (exception-handlers). Ерекше жағдайлардың өңдеушісінің абстрактілі ұғымы келесідей көрінеді:

<*кеңейтілген оператор*>**::= try**<*оператор*>
if<*мағына 1*>**raise**<*ерекше жағдай 1*>;
if< *мағына 2*>**raise**<*ерекше жағдай 2*>;
…
if<*выражениеМ*>raise<*ерекше жағдай М*>;
ерекше жағдайларды өңдеуші
{ when<*ерекше жағдай 1*>**:**<*блок1*>
when<*ерекше жағдай 2*>**:** <*block2*>

…
when<*ерекше жағдай N*>**:** <*блокN*>;**}**
[**finally**<*блокF*>]
Ерекше жағдайлардың өңдеушісі бірнеше әрекеттерді орындай алады, нақтырақ: (1) басқа қосалқы бағдарламада басқаруды беру; (2) ресурстарды босату; (3) қате жағдайдың көзін түзету және әрекетті қайталау үшін басқаруды беру; (4) командаға ерекше жағдайларды өңдеушіден кейін басқарудың қайтып келуі, бағдарламалаушы құрылғыға қате жағдай туралы хабарлағаннан кейін; (5) ерекше жағдайларды өңдеушіге жүгіну және жоғары деңгейдегі туындатушы қосалқы бағдарламаға қайтып оралу; (6) қатені туындататын, мәліметтерді жіберу; (7) туындатушы қосалқы бағдарламаға қайтып оралу.
Бірдей операторды орындаушы бағдарламалардың әртүрлі бөліктері, операторменбайланысты әртүрлі ерекше жағдайларды өңдеуші болуы мүмкін Әр түрлі операторларда ерекше жағдайларды жалпы өңдеуші болуы мүмкін.
Жағдайларды өңдеу алғаш рет PL I қолданылды. ADA, Java, C++ және Ruby тәрізді көптеген қазіргі замаңғы тілдерге ерекше жағдайларды қатты өңдеу кіреді.

Мысал 4.18
4.12 суреттегі бағдарламада қатесіз болжамды синтаксисті пайдаланумен ерекше жағдайларды өңдеу көрсетілген. Бағдарлама “myfile” файлын ашады, “mystream” сәйкес мәліметтер ағымы, несие картасымен жұмыс істейтін компания үшін қарыз сомасын анықтау үшін есептеледі. Бағдарлама, егер де файл табылмаса, қондырылған “file-not-found” ерекше жағдайын береді. Егер де шоттағы қалдық нөлден көп болса, пайдаланушылық ерекше жағдай “incorrect_debt” болады. Ерекше жағдайлардың бірінші өңдеушісі “Account-filemissing” операторын шығарады, және шақырушы қосалқы бағдарламаға кері оралады. Ерекше жағдайлардың екінші өңдеушісі операторды шағарады, ағымды жабады және қайта оралады.
subprogram illustrate _ exceptions
integer i;
real deposit, account;
file myfile;
streammystream;
exceptionincorrect _ debt; % пайдаланушымен анықталған, ерекше жағдай
open _file (myfile, mystream, read);

exception-handler {when file-not-found: **write** (‘Account file missing’); **return** }
read (mystream, account);
if(account> 0) **raise**incorrect _ debt; % пайдаланушымен анықталған ерекше жағдайды беру
exception-handler {% пайдаланушымен анықталған ерекше жағдайды өңдеу
when incorrect _ debt: **write** ('Incorrect debt'); **close** (mystream); **return**;}
close (mystream); **return**
СУРЕТ 4.12 Ерекше жағдайларды өңдеу мысалы.

4.7 Анықталмаған есептеулер

Анықталмаған есептеулер бағдарламаны орындау кезінде бағдарламаны басқарудың ағымын қамтамассыз етеді. Анықталмаған есептеулер үшін жалғыз шарт ол соңғы күйінде тапсырманың дұрыс шешімін беруде болып табылады Егер бағдарламаның орындалуы сызбаның әрбір түйіні есептуіш жай-күі болатын бағандарды айналып өту мәселесі ретінде қарастырылатын болса, онда программа анықталмаған болады, егер де біз бастапқы жай-күйінен соңғысына дейін кем дегенде бір бағыт таба алатын болған жағдайда. Егер соңғы жай-күйі есептеу операторларының орындау тәртібін тәуелді болмаған жағдайда есептеу бөлігі анықталмаған болып табылады. Анықталмаған есептеуді ұстанатын бағдарламалаудың бірнеше негізгі қасиеттерін ерекшелеуге болады:

1.Арифметикалық өрнектер және логикалық өрнектері операторларының коммутативтілігі анықталмаған өрнек есептеулерін төменгі деңгейде есептеуге ықпалын тигізеді. Анықтамаға сәйкес "Коммутативтілік" есептеу ретін тәуелді емес. Кейбір операторлар қолдайтын "коммутативтілік" - бұл қосу, көбейту, логикалық ЖӘНЕ, логикалық НЕМЕСЕ болып табылады. Мысалы егер біз аралық деңгейдегі командалар жиынының көмегімен 4 + 5 + 3 + 9 өрнегін есептейтін болсақ, сонда оны бір мезгілде екі санның қосуының бірнеше нұсқасы түрінде көрсетуге болады. Түпкі нәтиже қосындысы бірдей болады, өйткені қосынды өзінің сипаты бойынша ассоциативті және коммутативті болып келеді. Осыған ұқсас, егер біз логикалық (exp1 exp2 exp3) өрнегін есептейтін болсақ, онда біз нәтижесі өрнек қандай тәртіппен есептегенімізге қарамастан шешімі бірдей болады, себебі "логикалық және" және "логикалық немесе" - бұл ассоциативті және коммутативті операторлары.

2.Егер yS блогы жадын y S ⊎ (y S ⊎ σ2S) түріндегі үндеспеген бірлестіктер жиындарына бөлуге болса, онда C1 ; C2 жиындарының бірізділігі орындалады, бұл жерде C1 σ1 – де орындалады және C2 σ2S – кезінде

орындалады, ал σI C1 және C2 кезінде ардайым тұрақты болады. Сонымен қатар σ1 σ2S – ге тәуелді емес, оларға деген әсер олардың арасына әсерін тигізбейді. C1 командасынан кейін σ S σ’ S-тің жаңа болк жадында көрініс табады, ал C2 командасынан кейін σ2S σ’2S –тің жаңа болк жадында көрініс табады. Осылайша, жалпы C1; C2 командалар тәртібінің ықпалы σ’S = σ S ⊎ (y’ S ⊎ y’2S) жаңа блок жадын құрайды.
3.Егер біз командалар тәртібін өзгертетін болсақ, онда олар y’S = yI ⊎ (y’2 ⊎ y’ S) секілді түпкі шешімін береді. Мысалға, егер біз X және Y псевдоним болмайтын иемденуі x = 4; Y = 5 болатын екі команда алсақ, онда олар әртүрлі екі жады уяшықтарын өзгертіп, кез келген тәртіппен орындалуы мүмкін.
4.Егер (then Else If) шартты оператор секілді командалық таңдауды алсақ, және оны барлық шартты операторлары байланыстан босатылатын формада және командалар ағымынан құралған күйде жазсақ, және есептеу үшін тең дәрежеде мүмкіндікте деп қарастырсақ, онда командалардың орындалу ретінің өзгерісі соңғы нәтижеге әсер етпейді.
5.Дейкстрдің атақты қоғалған бағдарламалар туралы мақаласында келтірілген мысалды талқылайық:
if (x >= y) smaller = y;
elseif (y >= x) smaller = x;
Жоғарғы коды екі мәннен аз болып табылады және 4.1 кестесінде көрсетілгендей семантикалық тең екі кодқа алмастырылуы мүмкін.
Оператор анақталмаған болып табылады, өйткені операторды есептеп, бірдей түпкілікті нәтижже алудың екі әдісі бар. Егер x > y болған жағдайда бағдарлама келісідей болып есептілінеді: егер бірінші болып тексерілсе логикалық y >= x орындалмайды, және оператор бірінші болып есептелінеді. Жалпы жағдайда анықталмаған шартты команданы келесі абстракттті синтаксис арқылы жазуға болады:

if { *<шартты-өрнек1>⟶ <команда1>*|
< шартты - өрнек 2>⟶ <команда2>| |
*< шартты - өрнек n>⟶ <командаn>***}**
Жоғарыда жазылған абстрактілі құрылым басқару командаларының жиілігіне тәуелді емес, демек (<шартты-өрнекі> → <командаі>) ($1 \leq i \leq N$) түріндегі шартты команда бірдей мүмкін дегенді білдіреді. Әрбір шартты оператор басқасымен ‘|’символымен белгіленетін логикалық НЕМЕСЕ арқылы байланысады. ‘→’ симболы команданың бір бөлігін логикалық өрнекті сәтті есептеп болғаннан кейін есептеу керек дегенді білдіреді. Нәтиже алу үшін кем дегенде логмкалық өрнектің *<логикалық-өрнекі>* ($1 \leq i \leq N$) біреуі орындалуы керек. Егер барлық логикалық

өрнектер есептелмесе, онда қате шығады. Мүмкін болатын нәтижені алу үшін жасалатын алғашқы логикалық өрнек сәтті өткеннен кейін басқа шарттары тексерілмейді.

Кесте 4.1 Семантикалық баламалы бағдарламалар

if(X> = Y) **then** smaller = Y|	**if**(Y> = X) **then** smaller = X
if (Y > = X) **then** smaller = X |	**if** (X > = Y) **then** smaller = Y

4.7.1 Қорғалған командалар

Дейкстра анықталмаған бағдарламалаудың моделін ұсынды, онда ең логикалық өрнек әлсіз қажетті шарт болып саналды, және ол шартты орператорға есептелуін жіберу үшін шындық болуы керек. Есептелу бағдарламасы шартты операторлардың біреуіне өткен кезде басқа шартты операторлар тексерілмеді, дегенмен басқа шартты өрнектекер де сәтті орындалуы мүмкін еді. Дейкстра бұл шартты өрнектерді қорғаныш деп атады, ал шартты операторларды- қорғалған командалар деп атады. Қорғалған команда екі құрамдастан тұрады: қорғау және командалар. Қорғау - бұл жады блогын өзгертпейтін логикалық өрнектер. Командалар тағайындау операторларын қамтиды, және шартымен үшін алғышарттарды өзгертеді..Команданың кейінгі шартқа арналған алғышарт иемденуді операторлары бар және өзгертеді.

Қорғалған командалар мынадай қасиеттерге ие:

1. Бұл қорғалған командаға көшу үшін ең әлсіз шарт.
2. Қорғау қажетті, бірақ шарт жеткіліксіз жеткіліксіз. Қорғалған командалар кез келген ретпен тексерле алады.
3. Егер қорғаныс сәтті жүзеге асырылса, онда тиісті команда орындалып, басқа қорғаулар тоқтатылады.
4. Тек қана бір қорғау командасы орындалғандықтан, және қорғау қажет болғандықтан, бірақ жеткіліксіз шартты болғандықтан, онда қорғау команданы орындау бөлігінде шешімін алмай-ақ қорағау сәтті орындала береді. Алайда, басқа да қорғалған қорғау табысты орындау жариялан аннан кейін команданың тексерілмейді. Осылайша, қоғанысы бар бағдарлама орынсыз болып табылады: шешім таба алмауы мүмкін, бірақ бұл шешімі бар болуы мүмкін.

Q кейінгі шарты мен S оператор берілсін делік, *wp(S, Q) = P* белгіленуі P- анықталмаған оператор үшін ең әлсіз алғышарт болып, ал Q –кейінгі шарт болып табылады. Екінші жағынан *P* алғышарты мен S операторы берілген, Q-ең қатаң кейінгі шарт болып табылады. Ең әлсіз алғышарттардың бірнеше қасиеттерін бөледі, мысалға, егер *wp(S, Q) = P*1 және *wp(S, R) = P*2, бұл жағдайда *wp(S, Q ^ R) = P1 ^ P2*. Осыған ұқсаса wp(S,

Q ∨ R) *P*1 ∨ *P*2-ке тең, барлық кейінгі шарттар үшін *wp(skip,Q) = Q; ал wp(abort, Q) = fail,* бұл *abort (үзіліс)*деген мәнге тең түпкі нәтижені алу үшін алғашқы шарт *(fail)* болып орындалмауы керек деген мағынаны білдіреді.

*S*1*; S*2 операторларының жүйелігі берілген, және де соңғы шарт Q, оператордың кейінгі шарты *S*1 бұл *S*2 кейінгі операторының ең әлсіз алғышырты болып табылады. Осылайша *wp(S*1*; S*2*, Q) = wp(S*1*, wp(S*2*, Q))* теңдеуін жазға болады.

Қорғалған командаларда құрылымдардың екі түрі бар: қорғалған командалар таңдауы және қорғалған команданың итерациясы. Қорғалған команданың таңдауы келесідей түрге ие болады:

if

{ *<қорғау1>⟶ <команда1>* |

< қорғау 2>⟶ <команда2>|

…

*< қорғау N>⟶ <командаN>***}**

'→' символы қорғанышты сәйкес командадан әр қорғалған командада бөледі, ал '|' символы қорғалған командаларды бөледі.

Жоғарыда ұсынылған құрылымы қорғауды кез келген тәртіпте тексереді. Егер қорғау ойдағыдай бекітілген болса, онда тиісті команда орындалады, ал қалған қорғалған бағдарламалары тексерілмейді.

Қорғау сәтті орындалмаған болса, онда басқа қорғау барлық қорғау тексеріліп бітпейінше немесе қорғаулардың біреуі сәтті тексерілмейінше тексеріледі.

Қорғалған команданың итерациялық жобасы келесі түрде болады:

айналым

{ <қорғау*1*>⟶ *<команда1>* |

< қорғау 2>⟶ *<команда2>* |

< қорғау *N*>⟶ *<командаN>***}**

Интерактивті құрылым бағытталуды тексергеннен кейін барлық қорғау қате болғанға дейін сақьайды. Егер қорғаудың біреуң тесеруден сәтті өтсе тиісті команда орындалып, келесі итерациалдық айналым басталады. Егер басқару итерациалық жобадан шығып кетсе онда барлық қорғаулар тексерістен өткен жоқ , бұл кейінгі шарттың *Q* тең *P* ^ ¬ *қорғау*1 ^ ¬ *қорғау*2 ^ ... ^ ¬ *қорғауN* дегенді білдіреді. Де Морган теоремасын пайдалана отырып, түпкілікті шартты *P* ^ ¬ *(защита*1 ∨ *защита*2 ∨ ...∨ *защитаN)* ретінде жазу болады. Итерациалық жоба *(защита*1 ∨ *guard*2 ∨ ... ∨ *защитаN)* логикалық шартын алғышартқа қосады.

Мысал 4.19
Анықталмаған программаға арналған Дейкстра іргелі жұмыстарынан алынған келесі мы біз анақталмаған итерацияны көреміз. Біз *a0, a1, a2*, және *a3* төрт сандардың жүйелігін құрғымыз келеді делік. Бұл мәселені шешу үшін, біз анықталмаған итерацияны пайдала аламыз

айналым
{ a0 >a1 → ауыстыру(a0, a1) |
a1 >a2 → ауыстыру(a1, a2) |
a2 >a3 → ауыстыру(a2, a3)}

```
loop
{       a0 > a1 → swap(a0, a1) |
        a1 > a2 → swap(a1, a2) |
        a2 > a3 → swap(a2, a3)
}
```

Бастапқы жағдайлары үшін *P*, итерация циклін орындағаннан кейін, соңғы шар мынған тең болады *P ^ not (a0 > a1) ^ not (a1 > a2) ^ not (a2 > a3)*, оны былайша кері жазуға болады *P ^ (a0 ≤ a1) ^ (a1 ≤ a2) ^ (a2 ≤ a3)*.Кейінгі шарттар мен алғышартты салыстыра отырып итерациялық айналым келесі әрекетті орындайды *(a0 ≤ a1) ^ (a1 ≤ a2) ^ (a2 ≤ a3)*, ол жоғарылау бойынша сұрыптауға балама болып табылады a0 ≤ ≤ a1 a2 ≤ a3 .

4.7.2 Бағдарламаны біртіндеп құру
Болжамды соңы жағдай мен аксиоматикалық семантика алдыңғы операторды есептеуге қажет ең әлсіз алғышартты табуға қолданылуы мүмкін және біртіндеп анықьалмаған бағдарламаны құру үшін процесс кері ретпен қайталануы мүмкін.
Ақырғы жағдай мен бастапқы жай-күйі ескере отырып айырмашылық анықталады. Егер айырмашылық конъюнктивті нысан B1 ^B2 ^ ... ^Bn ретінде елестетуге болса онда сіз төмендегідей итерациялық жасақтаманы пайдалана аласыз: қорғау алдыңғы бөлімде сипатталған әлсіз алғышарттарды пайдалану ескере отырып де Morgan заңнамасына сәйкес айқындалады. Де Морган заңын сәйкес *not (B*1 ^ *B*2 ^ ... ^ *Bn)* қорғау1-ді not (B1) күйінде; қорғау2 – ні not (B2) күйінде т.с.с көрсетеді. Сол сияқты, егер дизъюнктивты нысанда *B*1 ∨ *B*2 ∨ ... ∨ *Bn*, айырмашылық бар болса қорғалған команданың таңдауды қорғау1 *wp(command*1*, B*1*)* болатындай ал қорғау2 *wp(command*2*, B*2*)* болатындай етіп қолдануға

болады. Сол себептен әрбір әлсіз алғышарт *wp(commandi, Bi)* бастапқы Р шартына енгізіледі.

Мысал 4.20
үлгісі 4.2 істі қарастырайық, және анақталмаған бағдарламасын құру мақсатында қарама-қарсы бағытта деп ойлайнайық. Жиілікті сұрыптау үшін соңғы жағдайы мына түрде болады a0 ≤ ≤ a1 a2 ≤ a3 , және конъюнктивті түрінде (A0 ≤ a1) ^ (a1 ≤ a2) ^ (a2 ≤ A3) болып жазылады. Конъюнктивті түріне қарап, біз, анықталмаған итерациялық дизайнын пайдалану керектігін білеміз. Қорғаныс1 *not (a0 ≤ a1) болады, бұл қорғаныс1-ді* a0> a1 ретінде ұсынылуы мүмкін екенін білдіреді. Сол сияқты, сіз басқа қорғаныстарды *a1 > a2* және *a2 > a3* алуға болады.

4.8 Мәлімет тәрізді бағдарламалар
Бағдарламалау тілдерінде, жалпы, деректер командадан бөлінген және деректер - бұл түрлендірілетін нәрсе ал команда өзгергериейді,. Алайда, көптеген жағдайларда, мысалы, жасанды интеллект жағдайда, деректер бойынша операцияларды орындауға болатын, бағдарламаны айырбастайтын деректер объект ретінде бағдарламасын құру қажеттілігі туындайды.
Бағдарлама деректер түрінде дамиды, содан кейін бағдарламаға айналады, онда бағдарлама *бірінші деңгейдегі объектісі* ретінде қаралатын болады. *Бірінші деңгейдегі нысан* идентификаторымен параметр ретінде өтпеген немесе есептеу нәтижесінде қайтарылған байланысты өңдеу кезінде құрылуы мүмкін.
Бағдарлама деректер түрінде дамыған, содан кейін бағдарламаның түрлендіріледі болса алғашқы нысан urovnya.Obekt Бірінші деңгей идентификаторы параметр ретінде өтпеген немесе есептеу нәтижесінде қайтарылған байланысты өңдеу кезінде құрылуы мүмкін, өйткені, онда бағдарлама қарастырылады.
Сол сияқты, көптеген жағдайларда, бағдарламаға басқа бағдарламаны дерек ретінде талдап немесе өңдеу керек болады . Мысалы, редактор басқа да бағдарламаны деректер ретінде қарастыруы тиіс.
Тілді өзінің абстарктілі саласындағы атқарылуы туралы өздерінің қорытындыларын жасайтын етіп жазуға болады. Барлық осы мысалдар бағдарлама деректер ретінде қарастырылғандағы жағдайлар. Кейбір деректер құрылымдардың көмегімен деректер ретінде бағдарламаны ұсыну процесі реификация деп аталады.

4.8.1 Бірінші деңгейдегі нысан ретіндегі функциялар

ML, Haskell, Lisp, Prolog сияқты декларативтік тілдер бағдарламаны Бірінші деңгейдегі нысандар ретінде қарастырады. Мысалы, көптеген функцианалдық бағдарламалау тілдерінде функция атауын және тиісті істі деректер түрінде алатын және деректерді белсенді функциясының түрлендіреді функционалдық режимі бар. Lisp бағдарламалау тілінде (apply 'first '(Arvind Programs)) функциясы екі деректер дәлел қабылдайды- имя функции *first* функция аты және *'(Arvind Programs)* тізімі , және деректерді *'(Arvind Programs))* функциясына алмастырады. Lisp тілдерді отбасында деректер элементтері функциясы шатастырмас үшін қосымша тырнақша бар екенін ескеріңіз. Осы сипаттамадағы синтаксис Lisp және Scheme тілдерінде пайдаланылады, және басқа да функционалдық программалау тілдердегі синтаксистен ерекшеленеді.

4.8.2 Метабағдарламалау және рефлексивтілік

Метабағдарламалар – басқа бағдарламаларды да өздерінің деректер секілді іске қосып және талдайтын бағдарламалар. Егер тіл өз бағдарламалық қамтамасыз ету үшін мета-бағдарламаларды жаза алса онда ол *рефлексивті тілі* деп аталады. Бірінші деңгейдегі объектілер тарапынан қолдау декларативтік тілдері, сондай-ақ, мета-бағдарламаларды қолдайды. Жарнама табылған болса, онда мета-бағдарламалар ортаны жаңартылады және бұл команда жолына сай келсе, онда ол бағдарламаға айналады және орындалады. Мета-бағдарламалар мысалына оны орындау барысында бағдарламаның әрекетін талдау, абстарктілі саласындағы басқа да бағдарламалар туралы қорытындылар жасайтын абстрактілі аудармашылар болып табылады. Абстрактілі саласының мысалы - белгілі құндылықтарды олардың тиісті түрлеріне түрлендіретін аймақ түрі. Мета-бағдарламалардың тағы бір мысалы тілі бағдарламасын орындау механизмін жазу болып табылады. Метабағдарламалар, сондай-ақ басқа да бағдарламаларды жасау үшін пайдаланылады. Метабағдарламалар кодын құру уақыты мен құнын төмендететін, бағдарламасы бағдарламалаушыға оңай толтыру үлгілерін жасау үшін пайдаланылады. Метабағдарламалар Ruby, Scala, Lisp және Prolog сияқты тілдерінде қолданылады.

4.9 Бағдарламалық қамтамассыз етуді қайталама қолдану

Бағдарламалық қамтамасыз ету мөлшері күрделі автоматтандыру міндеттеріне байланысты өседі, ол императивті болады, ол дегеніміз қайталануын болдырмау бағдарламалық қамтамасыз етуді әзірлеу уақытын қысқартуға және бағдарламалық қамтамасыз қателерді азайту

мақсатында, жаңа бағдарламалық қамтамасыз етуді әзірлеу, алдыңғы бағдарламалық қамтамасыз қайта пайдалану дегенді білдіреді. Әртүрлі парадигмалардың көмегімен тиімді нақты тапсырмаларымен кодтауға болады. Кешенді бағдарламалық қамтамасыз етуді әзірлеу бірнеше бағдарламалау парадигмасын пайдаланып дамыған интеграциялық бағдарламаларды қажет етеді.

Әртүрлі тілдер түрлі бағдарламалау парадигмасын (I қосымша) пайдаланады. Әр түрлі бағдарламалау парадигмалары деректер және бақылау қолдауымен жүзеге асыру абстракцияларымен ерекшеленеді. Мысалы, императивтік тілдер деректер жиынның жүзеге асыру үшін дәстүрлі бірнеше индекстелген жиымдарды немесе векторларын пайдаланылады. Декларативтік тілдер шартты өзгеріссіз деректер жиыны элементтерін жүзеге асыру үшін пайдаланылады. Парадигмалар тіпті қолдайтын олар деректер мен бақылау абстракциянда да әртүрлі болып келеді. Кейбір деректер жинау және абстракция басқару кешенді бағдарламалық қамтамасыз етуді дамытуға қажетті міндеттерді аудандарында әр түрлі сыныбы үшін неғұрлым қолайлы болып табылады. Бірнеше парадигмалардың интеграциялары өзара алмасуды талап етеді - басқа тілдерде бар өзара іс-қимылдар және басқа да тілдерде құрылған кітапхананы өңдейтін қабілеті.

Қазіргі заманғы бағдарламалық қамтамасыз ету екі тәсілдер пайдаланады: пайдалануға дайын сыртқы кітапханаларды пайдалану және басқа тілде әзірленген бағдарламалық қамтамасыз етуі бар өзара іс-қимыл. Кітапхана - жұртшылыққа және дамыған бағдарламалық қамтамасыз етуге жіберілетін функционалдық жиынтық. Кітапхананы «импорт»<кітапхана-аты> функциясын пайдаланып импорттауға болады.

4.9.1 Тағы да өзара әрекетте мүмкінділік туралы

Тілдің өзіндік қабілеті басқа тілде жазылған бағдарламаны шақыруға, басқа тілдердегі функцияны бағдарламалауға сондай-ақ өндірістік қызметінің тиімділігін жақсартуға мүмкіндік береді.

Тілдердің өзара іс-қимылының негізгі проблемасы шақырылатын және шақыратын бағдарламлардың бірігуі болып табылады. Бұл міселені шешудің екі тәсілі бар: (1) ортақ тіліне аударылған барлық тілге ортақ аралық тілді дамыту, немесе(2) екі функционалды үйлесімді тілдерді арқылы интерфейс деректер пішімі айырбастау дамыту.Осы әдістердің екеуі де қолданылды.

Аралық бағдарламалық қамтамасыз етуді ортақ тіл үлгі жүйесі мен метадеректерді көрсете отырып, функционалдық үйлесімділіктің қамтамасыз етеді. Әдеттегі жүйесі әрбір өзара әрекеттесу интерфейсін пай-

далану үшін, стандартты түріне ауыстырылды. Метадеректер дәстүрлі стандартты тілінде интерфейстің түрі, сондай-ақ басқа да тілдерде қоймасына ақпарат түріне аудару жұмысатқару үшін бірыңғай тетігін қамтамасыз етеді және жазбалардың жалпы түрі туралы ақпаратты шығарып алуды қамтамасыз етеді.. Метадеректер құрастырылған бағдарлама туралы ақпаратты қамтитын кестелер мен деректер құрылымдарын қамтиды.Кесте кластар, өрістер мен олардыі түрлері туралы, экспорталатын түрлер жайында, сонымен қатар бейтарап тілдік интерфейстегі басқа метадеректер кестелерге сілтемелер жайнда мағұлматты қамтиды. Соның салдарынан белгілі бір интерфейс тілінің және теңшелетін код өзара іс-қимылды анықтау қажеттідігі жоғалады.

Жалпы үлгідегі, және басқа жүйенің бір түріне аудару үшін метадеректер арқылы көрсеткен қолдауы қарамастан, түрлі абстрактілі деректер түрлі тілдерге қолдау көрсетеді, және жүзеге асыру процесін функцияны толық пайдалану арқылы осындай деректерді жүзеге асыру ретінде басқа абстрактілі деректерінен ажырату қиын.

Көптеген бағдарламалық тілдер үшін ортақтілдік тәсіл .NET Framework платформасында жүзеге асырылды, жіне ол Майкрософт секілді компаниялардың C#, C++ және Visual Basic тілдерінде қолданылады. NET Framework платформасы ортақтілдің инфраструктураның ерекшеліктерін (ОИЕ) сипаттайды, бұл жерде ортақтілдің ерекшелітері мен ережелері сипатталады. XML интернет-течнологиялар мен деректер базасында құрылған тілдер үшін стандартты ортақтілдік интерфейске арналды.

4.10. КЕЙС-ӘДІС

Бұл бөлімде біз қазіргі заманғы программалау тілдерін бірнешеуін қарастырып және олар қолдайтын абстракцияларды талқылауда. Бұл тарауда талқыланатын абстракцияны барлық емес, барлық бірдей тілдер қолдамайтынын атап өткен жөн. Қарапайым мысал ретінде ADA 2012, C++, C#, Fortran 2009, Java, Modula-3, Ruby, и Scala. ADA тілдерін алайықғ олар- объектілер мен модульдер бірге, аманатты бағдарламалау стилін қолдайтын бай тіл. Fortran 2009 объектілерін тұжырымдамасын қолдайтын құрылымдық зерттеу құрылғымен күттірмес бағдарламалау тілі болып табылады. C ++ - императивтік және объектілі программалау комбинациясы. Java, сондай-ақ императивтік және объектілі-бағдарланған бағдарламалау парадигмасын қолдайды. Lisp абстракцияның бай деректерді басқару және императивті бағдарламалау стилі шектеулі мөлшерде бірге, функционалдық бағдарламалау парадигмасын қолдайды. Modula-3 анық бағдарламалау модульдерді, сондай-ақ, объектілі программалауды қолдайды. Ruby кешенді функционалдық және объ-

ектілі программалау парадигмасын қолдап, сынып модульдер ұғымдар біріктіреді.
Scala – ол функционалды және бағдарламалау парадигмасы интегралдайтын қазіргі мультипарадигмалдық тілі. Логикалық бағдарламалау тілін білдіретін Prolog тілі бұл тізімге қосылмайды, себебі ол ЖӘНЕ-НЕМЕСЕ деген бірігу және агаш сияқты көптеген тұжырымдамаларды талап етеді. Бұл басқа тілдердегі сабстракциялардан ерекшеленеді және 10 тарауда егжей-тегжейлі қаралады.

4.10.1 Бағдарламалау тіліндегі мәліметтер абстаркциясы

Қазіргі бағдарламалау тілдерінде көптеген деректер абстракциясы жиі кездеседі. Барлық дерлік қазіргі заманғы тілдер күрделі түрлерін, жиындарын және кеңейтілетін деректер құрылымын қолдайды. Декларативтік және императивтік бағдарламаларын қолдайтын мультипарадигмалы тілдер нысандардың екі түрін сақтай отырып, өтпелі және мызғымас нысандарды саралайды. Айнымалы нысандар жою болатындай жаңартыла алады және мызғымас нысандар меншік бір рет тағайындауды қолдайды.

4.10.1.1 Бірыңғай Абстракция

Математикалық әлемнен шығатын бүтін сан, логикалық өрнек ретінде, нақты саны мен симболдық сияқты бір объектілер өте кең таралған. ADA, Pascal, Modula, C және C++ сияқты көптеген тілдерде пайдаланушы есептік айқындай алады. Дегенмен, көптеген Modula 2 мұрагері Oberon – нан шоттық жойылды. Поддиапазон хабарламасы мен оның массивткарда падалануы ADA, Fortran, ALGOL, Pascal, Modula-2 де стандартты болып саналады. Алайда, түрі тобы байланысты объектінің шектеулі пайдалану C, C ++ және Оберона жойылады, ал массивтің төменгі шекара үшін «0.» орнатылған. Енгізу түрі Pascal, Modula-2 де және Scala-ның соңғы шыққан нұсқасында қолданылады. Алайда, бұл бағдарламашылары шектеулі пайдалану салдарынан бағдарламалау жиынтығы негізінде коммерциялық бағдарламалық қамтамасыз етуді әзірлеу кезінде көптеген тілдерде бекітілген жоқ. Төменде Modula-2 синтаксис орнату түрі үлгісі келтірілген:

```
Type week = (Mon, Tue, Wed, Thu, Fri, Sat, Sun)
Var workdays: set of week;
```

Аптаны көрсетудің бірінші түрі жарияланды. Айнымалы түрі тыныссымен апта ретінде жарияланды. Айнымалы уақыт аптаның тиісті ішкі

жиыны болып табылатын кез келген мәнді, қабылдай алады.
Scala жинақты жасау үшін сақтаулы сөз «орнатылған» пайдаланады. Мысалы, негізгі айнымалы = Set тұжырымдамасы («нұсқаулық сегментін тіркеу», «Математика», «Қаржы») үш элементтен жиынтығы ретінде жасайды.
ADA мысалында көрсетілгендей аудару түрі, сондай-ақ, ауқымы болуы мүмкін:

```
Type    week is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
Type    year is 1..12;
Subtype  weekend is week range Sat..Sun;
Subtype  workingdays is week range Mon..Fri;
Subtype  fall_semester is year 8..12
Subtype  summer is year 6..8
```

Ada, сондай-ақ бастапқы түріне барлық қасиеттерін иеленеді қосалқы түрін қолдайды, және 7-тарау егжей-тегжейлі талқыланды. Oberon қысқа бүтін арифметикалық бүтін ⊆ ⊆ ұзын бүтін ⊆ ⊆ барлық нақты бірнеше түрлерін енгізу арқылы Modula-2 жүйесі түрлерін кеңейтеді. Енгізілген түрдің мәнін түрін қоса алғанда айнымалы түрін қоса беруге болады. Мысалы, егер N - бүтін, және M ұзын бүтін болса, ддұрысы M = H болады. Бұл шын мәнінде, тарауда сипатталғандай 7, мысал болып табылады.
Scala немесе оның кеңейтілген нұсқасы Escala, функционалдық бағдарламалау интеграцияны қолдайтын қазіргі заманғы мултипарадигмалқ тіл, ол объектілі программалау және (Escala қолдау) оқиға-бағдарланған бағдарламалау негізінде, бүтін және логикалық білдіру сияқты негізгі түрлерін қолдайды. Сондай-ақ, ол жарнама жолды өңдейді.
Абстракцияның кезекті бірлігіне қосқанда, көптеген тілдер 8-тарау талқыланатын мониторлар, сопрограммы және семафоров сияқты параллель орындау үшін қажетті түрлерін қолдайды.

4.10.1.2 Құрамалы абстракция

Кортеждер құрамдас объектілер болып табылады.
Аталған кортеждер Algol, C және C ++ тілдерінде «құрылым» деп аталады, және осындай ADA, Pascal және Module отбасы тілдерінде «жазба» деп аталады.
"Структ" C++ тілдерінде сақталып қойған «struct» деп жазылады және содан кейін атауы ал одан кейін түрлі құрылымдар жүреді. Өріс тобы сол және оң жақ мүсінді жақша жатыр. Атауы, факультеті, дәрежесі: үш өрістерді бар *студент* жазуын мысалын алыңыз. Барлық үш жолдың өріс

десек, С ++ тілінде компонент абстракциясы төмендегідей «құрылымы» болып табылады :

```
struct student {
        char* name;
        char* department;
        char* degree;}
```

онда char * таңбалардың жиымын белгілеу үшін пайдаланылады.
Бұл дегеніңіз жоғарыдағы атауында таңбалардың алабына көрсететін сілтеме болып табылады. Сол сияқты, факультет таңбалардың алабына меңзегіш сілтеме болып табылады. Бағдар жад алабының бірінші орынның көрсетеді.

Кортежді төмендегідей студенті түрі айнымалыны көмегімен арнайы кортеж мәндерінің байланысу арқылы инициалиялауға болады:

student plstudent = {“John”, “Computer Science”, “BS”};
ADA аталған кортежді " жазу" пайдалана отырып ұсынды:

```
Type PERSON is
record
   Name :        STRING(1..30);
   Department : STRING(1..20);
   Degree :  STRING (1..20)
end record;
```

Түрі басқа түрдегі ұйғарымда пайдаланушының қалауы бойынша пайдалануға болады. Мысалы, студент жазбалар келесі анықтамасын далалық жыл қосу арқылы түлегі енген жаңа анықтамасына ұзартылуы мүмкін.
Ұлғаймалы өріс қосу жолымен айқындау мынадай айқындама жоқ жазба жазу мүмкін например студент түлек шығарылған жылы. Oberon үшін ол төмендегідей жазуға тең болады:

```
student = RECORD
        name: ARRAY 30 of CHAR;
        department:  ARRAY 20 of CHAR;
        degree:      ARRAY 20 of CHAR;
  END
alumni = RECORD
        person: student;
```

```
        graduationYear: INTEGER;
  END
```

ADA, Pascal сияқты көптеген тілдер мен Modula 2 нұсқалы жазуды қолдайды. Нұсқалы жазба құрамдас құрылым болып табылады, ол екі бөліктен тұрады: тұрақты бөлік және айнымалы бөлік. Топтың айнымалы бөлігі еске сол алқапта орналастыру мүмкін өзара ерекше өрістер болып табылады. Нұсқалы жазба нөмірленген айнымалы мәніне негізделген, жазба нұсқасы бөлігінде ортақ жад аймағын түсіндіреді. Егер нөмірленген айнымалы мәні орындалу уақытында өзгерген болса, сол есте сақтау ауданын деректердің түрлі түрі ретінде түсіндіруге болады. 7-тарау сипатталғандай пайымдау, осындай қате ретінде бұзылуына әкелуі мүмкін.

Мысал 4.21

ADA-ның синтаксис бойынша композициялық объектісі болып табылатын келесі мысалда хабарландырулар нұсқасы рекордын көрсетеді. DATA түрі диапазондарда түрі бүтін үш жолдардан тұрады тіркелгісі болып табылады. Тағайындау түрі жолдың бірі ретінде жазбаның атын қамтитын нұсқа рекордтық болып табылады. «problem, solved» және «date_assigned» тіркелген өрістер болып саналады, ал нөмерленген «submission» түрінен басталатын өрістер нұсқаулық жолы болып табылады. Үш мүмкіндік кездеседі: submitted, tobe, және missed. Егер мәні «submission submitted» болса, онда бір бүтін сан түрі іске қосу нұсқасы ауқыммен 1. 100 болады. Егер мән «to be» болса онда өріс «DATA» дерегі ретінде қарастырылады. Егер мән «missed» болса онда өрістің мәні бүтін 0 санына ие болады. Алайда нұсқаулы жолдың түсіндіруі өрістің «submission» мәніне тәуелді болады.

```
type Date is
  record
        month: INTEGER range 1..12;
        day:  INTEGER range 1..31;
        year: INTEGER range 2012..2015
  end record
type status is (submitted, tobe, missed);
type assignment (submission : status) is
  record
        Problems: Integer range 1..10;
        solved: Integer range 1..10;
```

```
        date_assigned: DATE;
        case submission is
                when submitted => score:
                        Integer range 1..100;
                when tobe => expected: Date;
                when missed => score: Integer := 0
        end case
  end record
```

ADA, Pascal және Module нұсқаулы жазбаны қолдайды. Алайда, нұсқа рекордтық Oberon-нан алынып тасталды және анық 7-тарау сипатталғандай, байланысты айнымалы бөлігіндегі түріне бұзу қателігі әсерінен көптеген тілдерде ұсынылмайды

Prolog және Scala (1, 2, 3) формадағы динамикалық кортежді қолдайды. Scala кортеждерді new tuple<no-of-entries> ‘(‘<tuple-values>’)’ командасын пайдалана отырып құра алады. Мысалы, бекіту val instructors = new Tuple2(“Arvind,” “Paul”) бола алады, ал жеке элементтеріне instructors._1 және instructors._2 арқылы шығуға болады.

4.10.1.3 Ақпараттық Нысандардың Бірыңғайлығы

Бірыңғайлық индекстелген жиымдарды, ассоциативті, немесе векторларының тізімдерін пайдалана отырып іске реттілігі ретінде модельдеу арқылы жңзеге асырылады. Алқаптар (1) жиым көмегімен (2) әртүрлі өлшемдердегі мәндер беру немесе (3) әрбір өлшем бойынша анықталмайтын кеңейтілетін деректер элементтерінің болуы мүмкін біркелкі алапты құру арқылы жариялана алады.

Мысал 4.22

Мысалы, C # -тегі массивті декларациялаудың әртүрлі схемаларына назар аударайық.

```
int[] x = new int[5];// 5 өлшеміндегі бір өлшемді массив
int[] y = new int[] {1, 30, 44, 33, 8};
//элементтердің өлшемі - саны
int[] y = {1, 30, 44, 33, 8}; //элементтердің өлшемі - саны
int[,] z = new int[2, 3]; // Екі өлшемді массив
int[,] w = {{1, 2, 3}, {4, 5, 6}}; //матрица 2 × 3
int[] [] jagged1 = new int[6][]; // 6 қатар тегіс емес массив
jagged1[0] = new int[3] {10, 20, 30};
```

jagged1[1] = {1, 30, 44, 33, 8};

Сол сияқты, Ruby массивтер төмендегідей қарқынды кеңейтілуі мүмкін белгілі бір мөлшері немесе анықталмаған өлшемі үшін мәлімделген болады:

x = **Array.new** # анықталмаған мөлшерін жаңа динамикалық массивін құру үшін
x = **Array.new**(3) # мөлшері 3 жиымын жасау
x[0] = **Array.new** # бірінші элементі - кері массив
x[1] = [4, 5, 6, 7] # екінші массивтің 4 мөлшері бар

Ruby, сондай-ақ pop, push , shift, және unshift сияқты абстрактілі операцияларды пайдаланады, , сондай-ақ ол индекстелген рет болып саналады. Pop операциясы етпен соңғы элементін жояды. Push операциясы жүйелі соңында орналастырылған элементін қосады. Shift операция ретпен бірінші элементін жояды және бос орынды толтыру үшін, бірінші орынға бір элементпен массивтің барлық элементтерін жылжытады. Unshift операциясы соңғы күйіне барлық элементтерді жылжытады және алаптың басында берілген элементті кірістіреді.

Мысал 4.23
Интерактивті Ruby үшін келесі формулировкіні енгізейік:

p = [10, 20, 30, 40] # Төрт бүтін сандардың жиымын жасау
p. **pop**# Соңғы элементті алып тастау, сондай-p мәні ие болады [10, 20, 30]
p. push(80) # 8-бағаn - соңында және мәні p ие болады [10, 20, 30, 80]
p. shift# Бірінші элементті алып тастау, сондай-p мәні ие болады [20, 30, 80]
p. unshift **(**100) # Бірінші элементі ретінде 100 салып, p болып табылады[100, 20, 30, 80]

```
p = [10, 20, 30, 40] # Create an array of four integers
p.pop # Remove the last element and p will have value [10, 20,
  30]
p.push(80) # Insert 8- at the end and p will have value [10,
  20, 30, 80]
p.shift # Remove the first element and p will have value [20,
  30, 80]
p.unshift(100) # Insert 100 as the first element and p will be
  [100, 20, 30, 80]
```

ADA, Fortran, C, C++, C#, Java, Паскаль, Modula-2 және Modula-3 сияқты аманатты бағдарламалау парадигмасын қолдайтын тілдер, алаптарды қолдайды. ADA, сондай-ақ жартылай динамикалық массивтер қолдайды. C #, Ruby және Scala қолдау динамикалық массивтер сияқты объектілі-бағытталған тілдер, массивтер қарқынды құрылуы мүмкін кез келген басқа нысаны ретінде ұсынылуы мүмкін. Lisp және Prolog көптеген іске асыру ретінде динамикалық тілдері, сондай-ақ, динамикалық массивтер қолдайды. Динамикалық массивтер бағдарламалар реттелмеген алапта орналастырылады. Әр түрлі тілдердегі бағдарламалау синтаксис массивтер кейбір жарнамалар мысалдары 4.2-кесте көрсетілген.

Modula-3 секілді тілдер де массив типті декларацияда сериялық нөмірін қолдайды. Мысалы, Modula-3-те жиым, сондай-ақ ARRAY ['A'..'D'] OF INTEGER, ретінде ұсынылуы мүмкін, мұнда «A» индексі 1, және «D» индексі 4 пен теңесіріледі.
Модуль-3 секілді Тілдер, сондай-ақ, массив декларацияда сериялық түрі санын қолдайды. Модуль-3 Mysaly массив, сондай-ақ, [«D» .. «A»] kakMASSIV ұсынылуы мүмкін IZTSELOGO «A» индексі 1, және «D» индексі 4. салыстырғанда қайда саны осы алаптың баламалы ARRAY табылады [1. .4] бүтін. Бұл жиым Array [1 .. 4] бүтініне тең.

4.2 кесте таңдалған тілдердегі декларация құрылысы

TABLE 4.2 Array Declaration in Selected Languages

Language	Array Description
ADA	**array**(1..5) **of INTEGER**
C, C++:	**int** x[5]
C#	**int**[] x = **new int**[5];
Java	**int**[] x = **new int**[5];
Modula-3	**VAR** x : = **ARRAY** [1..5] **OF INTEGER**
Ruby	x = **Array.new**(5)
Scala	**var** x = **new Array** [**integer**] (30)

Language – **тіл;** array description – **құрылысының сипаттамасы;** ADA, C, C++, Java, Modula-3, Ruby, Scala – **ADA; C, C++, Java, Modula-3, Ruby, Scala;** array of integer – **бүтіндердің құрылысы;** int – **бүтін;** new int – **жаңа бүтін;** var – **әр түрлі;** array – **құрылысы;** new – **жаңа;** integer – **бүтін.**

4.10.1.4 Кеңейтілген ақпараттық нысандар

Жүзеге асыру деңгейіне меңзерді қолдау барлығы дерлік қазіргі заманғы тілдері, не тікелей немесе жанама күйінде, кез келген көрсеткіштер осындай байланысты тізімдерді ағаштар мен графикалар сияқты рекурсивті деректер құрылымдарын пайдалана отырып құрылуы мүмкін. Көрсеткіштер қатты Pascal және Modula-3 сияқты, рекурсивті құрылымдық деректермен анық байланысты болуы мүмкін. Мұндай ADA, C, C ++ сияқты көптеген тілдерде, жылы, көрсеткіштер кез келген деректер құрылымына байланысты болуы мүмкін тәуелсіз объект ретінде сақталады. Индексі көрсетілетін жады жасушаларын оқуға арналған механизм, - сілтегіш өзгерту арқылы индексі деректерінің құрылымының негізінде элементке қатынасу мүмкіндігі. Lisp және Prolog сияқты декларативтік тілдері, тілді енгізу үшін меңзерді салып, бағдарламашыларға көрсеткіштерді тікелей басқаруға мүмкіндік бермейді; құрылымдық деректер, одан теріс сияқты жоғары деңгейдегі ядро операторларының көмегімен кеңейтіледі, мысалы cons, insert және append. Бұл операциялар 9 және 10 тарауларында талқыланады.

ADA тіліндегі байланыс тізімінің хабарлануы

```
type List_Pointer is access My_List;
 type My_List is
  record
    Info : INTEGER;
    Next : List_Pointer;
  end record;
 Start : List_Pointer;  – Always points to start of the list
 Last : List_Pointer;  – Points to the end of the list
```

C және C++ тіліндегі байланс тізімінің хабарлануы

```
struct My_List
{  int info;
   My_List* next;
}
```

Modula-3 тіліндегі байланыс тізімінің хабарлануы

Modula-3 бастапқы түрді және кеңейтілген ақпараттық нысандарды қолдап, меңзерлерді қолдану арқылы құрылды

```
TYPE
My_List = RECORD
```

```
    info: INTEGER;
    next: Ref My_List
END
```

Type list_Pointer is my access My_List – **Жаз, тізім көрсеткіш менің тізіме қол жеткізуші болады;** type My_List is – **жаз, менің тізімім ол;** record – **жазып алу;** info: integer – **мәлімет: бүтін;** next: List_Pointer – **келесі: тізім көрсеткіш;** end record – **жазуды аяқтау;** start: List_Pointer – **бастау: тізім көрсеткіш;** always points to start of the list – **әрқашанда тізімнің басталуына көрсетеді;** last: List_Pointer – **соңғы: тізім көрсеткіш;** Points to the end of the list – **тізімнің соңына көрсетеді;** Linked list declaration in C and C++ - **C және C++-тағы қосылған декларация тізімі;** struct my list – **менің тізімімді құрастыру;** int info – **int мәлімет;** my_list* next – **менің тізімім* келесі;** linked list declaration in Modula-3 – **Modula-3-ке қосылған декларация тізімі;** Modula-3 supports reference type, and extensible data entities are built using pointers - **Modula-3 анықтама түрін қолдайды, және кеңейтілетін деректер тұлғалары көрсеткіштерді пайдаланып құрастырылды**; Type – **жаз;** My_List = record – **Менің тізімім = жазып алу;** info:INTEGER – **мәлімет: БҮТІН;** next: Ref My_List – **келесі: анықтама менің тізімім;** end – **соңы.**

Байланысты тізімді хабарлау Scala:
Scala - бұл бағдарламашы деңгейіне сілтемелерді пайдалану жасырады жоғары деңгейдегі мултипарадигмалы тілі. Оның тізімі Lisp және Prolog-қа ұқсайды. Төменде көрсетілгендей Scala тізім, әр түрлі тәсілдермен ұсынылуы мүмкін. Тізіміге қолтаңбаны жариялауға болады, немесе автоматты түрде алдын ала белгілі бір мәнмен қорытынды жасауға болады.

val num : **List[Int]** = List(1, 2, 3) // num - бүтін тізімі
val num = **List**(1, 2, 3) // Тізімнен шығару түрі
val num = **List**()//Num — Кейінірек кеңейтілуі мүмкін бос тізімі,
val nested = **List** (**List** (1, 2, 3), **List**(4, 5, 6)) // кірістірілген тізімі

Оператор белгілеген құрастырмалы ақпараттық объект тізімі '::';, осылайша тізімі (1, 2, 3), сондай-ақ Valnum = 1 :: тізіміне (2,3) ретінде ұсынылуы мүмкін. Lisp, Ruby және Scala секілді көптеген тілдері негізгі құндылықтарды жұбын қолдайды. Бұл жұп негізгі құндылықтары Lisp-те ассоциативті салыстырулар, Scala-да жылы карталар және Rub-де қа-

жетсіз деректер деп аталады.

4.10.2 Бағдарламалау тілдерінде абстракцияларды басқару

4.10.2.1 Мутация

Императивтік бағдарламалау парадигмасын қолдайтын тілдер отыратын заттарды және деструктивті жаңартуларды қолдайды. Мысалы, Ada, Java, C ++, C #, Fortran және Pascal сияқты тілдер деструктивті жаңартуды қолдайды. Функционалдық бағдарламалау тілдері мызғымас объектілерді қолдайды. Сонымен қатар Lisp, Ruby және Scala сияқты көптеген функционалдық программалау тілдері де қозғалмалы объектілер мен қозғалатын объектілерге қосымша деструктивті жаңартуды қолдайды. Lisp деструктивті жаңарту мен жаһандық айнымалылар қолдайды, сондай-ақ итерация аясында түрлі мәндеріне қоса тіркелуге оператор ішіндегі цикл айнымалы индексті береді. Ruby және және Scala сияқты мультипарадигмалы тілдері деструктивті жаңарту, сондай-ақ мызғымас және өтпелі нысандарды қолдайды. Әдетте, байланысты тізімдер және жолдар осындай шарттармен индекстелген тізбектері ретінде өзгермейтін объектілерін, ретінде қаралады, массивтер отыратын объектілері ретінде қаралады.

4.10.2.2 Шартты бекітулер

Бағдарламалау тілінде шартты есептілігі, әдетте стандартты болып табылады. Барлық дерлік тілдер енгізілген шартты қолдайды. Көптеген тілдер сондай-ақ оператор параметрін қолдайды. Haskell сияқты кейбір тілдер және Parlog және GHC сияқты кейбір параллель логикалық қорғауды қолдайды. Функционалдық бағдарламалау тілдері шартты сөйлемнің функционалдық нұсқасы бар.
Lisp-те жалпы топтау мақсаттағы үшін сол шартты бірнеше терминдердегі шартты бекіту бар.

4.10.2.3 Итерациялар және Итераторлар

Аманатты бағдарламалау парадигмасын қолдайтын барлық дерлік тілдер «ҮШІН» циклі мен шексіз итерация негізінде айнымалы индексін қолдайды, мысалы жалғасы бар шартты цикл немесе «ӘЗІРГЕ»циклі. С секілді кейбір тілдер «дейін» циклін «ҮШІН» циклына ұқсас етіп жасалған. Lisp, Ruby және Scala сияқты көптеген функционалдық бағдарламалау тілдері қайталап шақыру функциялары үшін «ӘЗІРГЕ» және «ҮШІН» циклдері сияқты анықталмаған итерацияны қолдайды. Деструктивті жаңарту айнымалыларды қолдайтын функционалдық тілдері сондай-ақ, итерациалды циклдарды да қолдайды. Ruby және Scala

осындай мультипарадигмалы тілдердің мысалы болып табылады. Scala жалғасы бар шартымен циклды, «ӘЗІРГЕ» циклін, «үшін» циклі және Итераторды қолдайды; ал Ruby жалғасы бар шартымен циклды, «үшін» циклін, анықталмаған циклді және итераторларды қолдайды. Анықталмаған цикл Итерацияны шығу жағдайында арқылы шыққанша сақтайды. Ruby-де әрбір деректер элементі объект болғандықтан, итераторы және циклдар Ruby тілінде әдістері ретінде қарастырылады. Төменде Ruby-дегі итерация құрылымдарының мысалы келтірілген.

[4, 5, 6].**for each** {|i| **puts** i} will write 4, 5, and 6 in that other.
10.**times** {|i| **puts** i} will write 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in separate lines
For each-әр,put-алынып, will write-жазады, and-және, in that order-осы ретпен
Times-рет, in separate lines-бөлек жолақта

4.10.3 Бағдарламалау тіліндегі мәліметтермен алмасу

Әртүрлі тілдер трансмиссиялық тетіктерін түрлі параметрлерді қолдайды. Мысалы, мәні бойынша шақыртулар мен «С» тіліндегі сілтемесі бойынша шақырту объектің деректерінің мекен-жайын көшіру арқылы салыстырады. ң құны бойынша қоңырау пайдалану және қоңырау деректер объектінің мекен-жайы көшіру модельдеу. «С ++» тілді, сілтеме арқылы шақыру, және қоңырау «сілтеме тұрақты шақыру» пайдаланады. «Сілтеме арқылы шақыру» беру параметрлерін механизмі, тек оқу қатынасына рұқсат береді және нақты параметрге жазу мүмкіндік бермейді.

Үйінді сақталған динамикалық нысандарына қатынасу әрқашан шақыру мәні арқылы беріледі.Бұл сіз нысандарды ортақ пайдалануға мүмкіндік береді. Java мәні бойынша шақыруды пайдаланады. Көрсеткіштерді пайдалану арқылы үйінді нысандарына қол жеткізу болғандықтан, мәні-бойынша қоңырау көрсеткіштер көшіру арқылы заттарды ортақ пайдалану мүмкіндігін береді.

ADA сілтеме (қол жеткізу) бойынша нәтижесі мәні (IO режимі) және шақыру , мәні (кіріс режимі) арқылы нәтижесінде шақыруды (шығыс режимі) пайдаланады. Нысандар жағдайына байланысты салыстырылған, шақырушының бағдарламасының нақты параметрлерге қол нақты параметрдің алдына сақтаулы «қорғалған» сөзін пайдалана отырып өзгертуден қорғауға болады. ADA сондай-ақ нақты және формальды параметрлердің икемді түптеу ресми параметр нақты параметр → қамтамасыз түрінде аталған параметрлерін пайдалануға мүмкіндік береді; аталған параметрлері тиісті ұстаным болуы тиіс емес.

С тілі мәні бойынша шақыруды пайдаланады. Алайда, С тіл сілтемені шақыруғасәйкес келетін деректер құрылымын мекенжайын жібере алады. С ++, С #, F # Modula, Modula--2, Modula-3, Pascal-дың соңғы нұсқасы, Fortran 90, PHP, және Python екі параметрлер трансмиссиялық механизмін қолданады: сілтеме арқылы мәні мен сілтеме бойынша шақыру; және сілтемені шақыру бойынша шақыру оны бөлу үшін, сақталған сөз белгіленген.

Барлық объектілердің сілтемелер мәніне шақыруды пайдаланып беріледі, және бақылау дестесін сақталады нысандарына кез келген сілтемелер шақыру арқылы беріледі. Сондай-ақ, бақылау дестесін сақталған аз бірлігі, және деректер құрылымы ақпараттық объектілер, пайымдап шақыру арқылы көшіріледі. Ruby мәні бойынша шақыру арқылы параметрлерді өтеді. Ruby барлық ақпарат нысандар үйіндісіне сақталған объект болып табылады; және әрбір айнымалы тиісті объектіге сілтеме болып келеді. Айнымалы мәнді тағайындау жаңа нысанға жаңа нысанды және ұпай құру баламасы болып табылады.

Басқа айнымалы бір айнымалы мәнді тағайындау объектінің мекен-жайы көшіруге тең. Ruby шақыру мәні бойынша нысан мекенжайын көшірмелерін жасайды, сондықтан формалды параметр объектіге қол жеткізуден басталады. Интерактивті Ruby келесі бағдарламасы осы принципті түсіндіреді:

```
def entityId(Object);
  puts(“passed parameter: #{Object.object_id}\n”);
end
  y = 4.5;
puts(“actual address: #{y.object_id}\n”);
entityId(y)
```

entity-элемент, put-қою, passed parameter-өткен параметр, actual address-негізгі мекенжай, object-объект

Жоғарыдағы код екі жағдайда бірдей шешуге мүмкіндік береді: *y*.**объект_идентификаторы** ID *y* параметрін беру және *Объект*.**объект_идентификатор** идентификатор мекенжай нысанның мекен-жайы формальды параметр объектісі бойынша атап береді.

Emerald тіл бағдарламалауы (Изумруд) үйінді нысандарына айнымалыларды Emerald көрсеткіштерін сақтайды, өйткені объект сілтеме бойынша шақырылып, сондай-ақ объектінің мәніне алмасу үшін шақыру параметрін қолдайды. Emerald таратушы есептеуін қолдайды, және қашықтағы процессор нысан сілтемесін жібере отырып баяулатады. Осылайша, Emerald сондай-ақ қашықтан процессорлар шақырған

тәртіппен өңделген болуы мүмкін объектілердің жергілікті көшірмелерін жасауға жылжыту арқылы келуге және шағымдануға шақыруды пайдаланады.
Scala анықтамалық объектілерін беру үшін мәніне функционалдық бағдарламалау және қоңырау үшін кешенді білдіру беру үшін аты бойынша шақыруды қолдайды. Функциялар бірінші класты объектілері болып табылады және параметрлерін ретінде берілуі мүмкін. Бағдарламаларды функционалдық стиль, және объектілі-бағдарланған стилі пайдаланып дамытуға болады.
Haskell трансмиссиялық параметрлерін кезінде кешенді өрнектердің өңдеу үшін есімімен шақыруды орнына қажетінше қоңырау пайдаланады. Қажетті қоңырау артықшылығы күрделі өрнектер тек бір рет бағаланады және кэштелген және келесі жолы кэштелген мәнінің көрінісін пайдаланады.
ADA, C++, Clojure, Стандартты Lisp, Fortran 90 және Python, Ruby және F # параметр әдеткі мәндеріне рұқсат етілген. Бұл тиісті нақты параметрі нақты кіші қоңырау емес болса, формальды параметр кіші органға әдепкі мәнді өтеді дегенді білдіреді.

4.11 Қысқаша қорытындылар

Бұл тарауда біз абстракцияның түрлі бағдарламаларын талқыладық. Абстракцияның бағдарламалары деректер, мета бағдарламалар, ақпарат алмасу тетіктерін және тілдер арасында өзара іс-қимыл, сондай деректер абстракция, абстракция басқару, деректер жинау және инкапсуляция бақылау бағдарламасын қамтиды.
Деректер абстракция жеке элементі, бірнеше атрибуттары бар құрама нысандар, деректер объектілердің жиынтығы, ақпарат кеңейтілетін орнатылған уақытты, әлемдік ауқымдағы және тұрақты объектілері ақпараттық объектілер мен өтпелі объектілерін желісін іске асыратын нысандар болып табылады. Жекеленген ақпараттық объектілер бүтін сан, өзгермелі (немесе нақты), түртіндінің жолының немесе есеп жиынтығы элементі ретінде осы базалық түрі жарнама арқылы моделденеді.
Композициялық атрибут аталған кортеж сияқты әртүрлі типтер мен абстрагцияланатын әртүрлі өрістер тұрады. Бағдарламалау тілдері реттелуіне және жолаққа жақсы қол жеткізу үшін атындағы өрісі бар аталған кортеждерді пайдаланады. Ақпараттық объектілердің жинағы жиынтық, реттелген жиынтық, мултикөптік, мултикөптік орнату немесе мултикөптік (негізгі, мән) жұп ретінде абстрактілеге болады. Негізгі (басты, мән) жұптарының жиынтығы әр деректер нысаны үшін бірегей болып табылады. Осы барлық жинақтар бағдарламасын бірге орындау кезін-

де қол жеткізу және жаңарту жеңілдету үшін атауларымен байланысты болады. Түрлі деректер құрылымын ақпараттық объектілердің жиынтығын іске асыру үшін пайдаланылуы мүмкін. Ол байланысты тізімдер, индекстелген жиымдарды, векторлары және хэш кестелерді пайдаланып жүзеге асырылуы мүмкін. Кеңейтілген байланысты индекстелген, және кеңейтілетін - - ағаштар мен хэш үстелдер тізімдерін, векторларды пайдалана отырып жүзеге асыруға болады.

Бақылау абстракциялары Итератор, Рекурсия және рәсімдерді қоса алғанда, бірнеше тағайындау, командалар тізбегінен, командалық блогында, шартты есептілігін, итерациялық жобалау, оның ішінде тағайындау операторлар сияқты функциялар мен процедуралар индекстелген айнымалылар болып табылады.

Блоктар кем анықталатын құрылымдар хабарландырулар және командалар сериясы болып табылады, және жарнама облысы блок шектелген болады. Блоктар деректер мен код көрінуін бақылауға табиғи шекарасы қамтамасыз етеді. Ат қойғанда көріну реттеу блоктар мен модульдер ішінде аттас идентификатор икемділігін қамтамасыз ету үшін, сондай-ақ жанжалдарды болдырмау үшін маңызды болып табылады.

Модульдер ендірілген кіші және деректер абстракцияның табиғи шекарасын қамтамасыз ететін абстракцияның басқа деңгейін білдіреді. Блок және модуль арасындағы айырмашылық ол блоктар кіші рәсімдерді қамтиды және модульдер бірнеше рәсімдерді қамтиды. Модульдері болашақта жеке-жеке пайдалану үшін құрастырылған және мұрағаттауға болады. Басқа модульдерді кіші және деректер абстракция экспорттық-импорттық механизмі көмегімен пайдалануға болады. Модулдегі объект басқа модульдерге экспортталғаннан кейін ғана көрінеді, және модулде тек анық импорт декларациялау объектісінен кейін пайдалануға болады. Сынып деректер жинау және деректер абстракцияның үшін жұмыс тәртібін қамтитын пассивті үлгісі болып табылады; нысандар - белсенді ингредиенттер бар сынып, жағдайлары, және жұмыс уақыты мәртебесіне ие.

Рәсімдер арасындағы ақпарат алмасу (1) жаһандық айнымалы, (2) бірнеше айнымалылар, (3) объектілі-бағытталған тілдерінде айнымалылар, (4) төмен деңгейлі іске асыру және FORTRAN бұрын іске асыруға жадындағы жалпы ауданы, және(5) параметр беруді пайдалана отырып туындауы мүмкін құрылымдық бөлімшенің бар тілдерінде параметрі. Тасымалдаушы айнымалы мән болғандықтан, айнымалылар R-мәні шығарып алуға болады, барлық атрибуттары беріліс параметрлерін пайдалануға болады. Жалпы үш атрибут бар: атауы, жады ұяшығы және R-мәні. Нәтижесінде біз параметр өту тетіктерін бес негізгі түрлерін

талқылаймыз: аты бойынша шақыру, сілтеме бойынша шақыру, мәні бойынша шақыру, мәннің нәтижесі бойынша шақыру және нәтиже бойынша шақыру.

Мәні бойынша шақыру нақты параметр және көшірмелерін жергілікті айнымалы ретінде ресми параметр мен процестер формальды параметрді бағалайды. Байланыс біржақты тәртіппен жүреді, және шақырушының ескі есептелген нәтижелер қайтып берілмейді. Сілтеме бойынша шақыруда формальды параметр нақты параметрдің жад мекенжайын тасымалдайды. Композициялық ақпарат объектілері немесе ақпараттық деректер жиынтықтары жағдайында деректер құрылымын базалық адресі ресми параметрге беріледі.

Деректер құрылымы абоненттік ортада болып табылады және оған қатынасу базалық мекен-жайы + жеке элемент (немесе күрделі объектілерді жағдайда астыртын) арқылы жүзеге асырылады. Нысан бағдарланған тілдер объектілерін сақтау үшін үйіндісін пайдаланады. Үйіндіде сақталған күрделі объектілер және динамикалық объектілерге сілтеме, сілтеме деңгейінде беріледі. Мәні бойынша шақыруды пайдалана отырып, нысан сілтемені көшіруге болады. тек оқу және алмасу үшін қоңырау сілтеме қоңырау: Біз сондай-ақ қысқаша сілтеме бойынша шақыру екі параметрлерін талқыладық:оқу үшін сілтеме бойынша шақыру және айырбастау бойынша шақыру.

Сілтеме бойынша шақырудың ресми параметрін сақталған сілтемелерді пайдаланып оқға болады, бірақ олар жаңартылмаған болады. Ол кез келген кездейсоқ деструктивті жаңғырту бағдарламасына дұрыс мінез-құлық әкелуі мүмкін нақты параметрлерін болдырмау үшін маңызды болып табылады. Айырбастау бойынша шақыру бірінші сілтеме бойынша шақыруды көрсетеді, ал содан кейін басқа шақырулар кіші сілтемелері алмасып мәнге шақыруды пайдаланып, қоңырау тізбегінде рәсім қоңырау орындалады. Біз басқа идентификатор дисплей жады болуы мүмкін жұмыс циклі индекстелген айнымалы, есептеу атынан үндеу жүгіру уақытындағы күтпеген мінез-құлық мәселелері талқыланды. Қажеттілік бойынша шақыру аты бойынша шақырудың бір түрі болвһып саналады және онда өрнек бір рет есептеліп, ал сол есепке крек келесі кіріспенің мәні кэштеледі. Кейінгі қосалқы өрнектерді бағалаудың орнына, кэштелген мән тиімділігін арттыру үшін пайдаланылады.

Біз бірақ қызмет белгі бағдарламалар салыстырғанда неғұрлым ұзақ кезеңімен, шақырушының бағдарламасы бойынша қолжетімді және өзгертілетін, сақталуын өзгертетін жанама әсері талқыландық. Жанама әсер нәтижесі модификация әсері тіпті кіші аяқталғаннан кейін бөлінбеген болып табылады. Егер әсер арнайыланбаған болса, онда

коммутативті өрнектерді счияқты негізгі бағдарламалау принциптері мінез-құлық бағдарламасын орындау кезінде өзгеруі мүмкін. Тиімділігі қолдану ұштастыра отырып, тізбекті актілерінде сондай-ақ бағдарламалардың ретсіз мінез-тудыруы мүмкін.
Бірінші деңгейдегі нысандар іске қосу уақытында деректер түрінде салынған, содан кейін бағдарламаның айналдыруға болады. Метабағдарлама басқа бағдарламаны бағдарламаның мінез-түсіндіру қасиеттерін алу абтрактілі доменінді деректер ретінде немесе бағдарлама ретінде қабылдайды.
Ерекшелік өңдейтін бағдарламалар неғұрлым сенімді етеді және тастаудың біртіндеп босату мүмкіндік береді. Реакция интерфейс деректер түрі мен абстракцияның деректер шақыру өңделген мен бағдарламаларды деп атауға болады. Бұл анықтау тілдері нақты интерфейсінің көмегімен немесе екі түрлі программалау тілдері арасындағы айырбастау мәліметтерін анықтайтын ортақ тіл сипаттамасы және метадеректерді пайдалана отырып жасауға болады.
Соңында, ADA, C, C ++, C #, Java, Modula-3, Ruby және Scala сияқты тілдердің әр түрлі деректер абстракцияның және менеджментіндегі кейбір мысалдарды қарадық
Нақты паргадигмаларға арнадған нақты парадигмалары бар тілдердің басқа мысалдар мына тарауларда беріледі: функционалдық бағдарламалау парадигмасын 9-тарау Lisp, Scheme, и Haskell мысалдары; логикалық бағдарламалау парадигмасын 10-тарау Prolog-та бағдарламалау және ақпарат алмасу тетігін мысалдар; C # объектілі-бағдарланған мүмкіндіктерін көрсететін Толық мысалдар, C # ,, Java, Ruby, Scala және Modula-3, объектілі-бағытталған программалаудың парадигмасын 11-тарау; 14-тарауында Python, Perl және Lua сияқты сценарийлер программалау тілдерінің мысалдары.

4.12 Баға

4.12.1 Концепциялар және Анықтамалар

Нақты параметр; бірлігі; тегістеу; бекіту; хабарландыру тақтасы; блок; көшірмесін жасауға шақыру; қозғалыстағы міндет; аты бойынша шақыру; қажеттілігі бойынша шақыру; сілтеме бойынша шақыру; нәтиже бойынша шақыру; алмасу бойынша шақыру; мәні бойынша шақыру; мән нәтижесі бойынша шақыру; кіріс бойынша шақыру; тарату тізбегі; класс; ақпараттық объектілердің жиынтығы; ортақ тіл ; таралған түрі; композициялық ақпарат нысан; шартты өтініш; абстракция басқару; деректер абстракция; жарнама; деструктивті жаңарту; ерекшелікті өңдеу; кеңейтілетін ақпарат нысан; бірінші класс объектісі; формальды пара-

метр; жаһандық айнымалы; қорғау; сақталған команда; өзгермейтін нысан; импорттық-экспорттық; ақпарат алмасу; ақпараттық жасыру; мұралық жер; Итерация; итератор; карта; метадеректер; метапрограммалау; модуль; бірнеше мақсаты; айнымалы объект; мутаторы; өзара рекурсия; атындағы теру; анықталмаған бағдарламалау; жергілікті емес айнымалы; нысан; нысан сыныбы.

4.12.2 Тапсырманы шешу

1. жоғары жылдамдықты бағдарламасы Modula-3, C ++ және Java бағдарламалау тілдерінде жазу, және программалау тілдері пайдаланылатын түрлі абстракция салыстырыңыз.
2. Сұрыптауды орындайтын «moya_sort» деп аталатын біріктіру модулін Modula-3, және ADA 2012 бағдарламасында жазып содан кейін ретпен деректер элементтерін оқи және жаза үшін бағдарламаның негізінде оны пайдалану
3. Қонақ үй екі өлшемді массив hotel[5, 120] ретінде абстрагталады, онда бірінші өлшеу қабаттарды көрсетеді, ал екінші өлшеу нөмірлері шегінде болды. Әрбір объект b байт деректер алады. Теңдеуі есептеу үшін hotel[i, j], ығысу беріңіз, және оны алу үшін жылжу үшін орналасқан бөлме нөмірі 18, 4-ші қабатында орналасқан деп алыңыз. Мысалы, нөмірлеу 1, 0 басталады.
4. Класс оқушылары деп аталатын 30 композициялық деректер объектілерін массив ретінде модельдеу. Әрбір құрылтай деректер нысан кортеж түрінде (аты, жасы, факультеті, жыл) ретінде модельдеу. атауы 4 байт жасы бүтін ретінде модельдеу 32 кейіпкерлердің белгіленген өлшемді жолы ретінде модельдеу деп есептейік, факультеті 4 байт бүтін ретінде модельдеу, және 4 байт жылына бүтін ретінде модельдеу. Хабарландырдағы түрлі салалар ауыстыруын [I] .name деп есептеңіз.
5. Тілі параметр өту төрт түрін қолдайды деп есептейік: мәні бойынша шақыру, сілтеме бойынша шақыру, нәтиже мәні бойынша шақыру, нітиже бойынша шақыру. Келесі бағдарлама үшін жолдар мәлімдеме көрсететін сондай-ақ әр түрлі бағандар айнымалылар және ресми параметрлерін көрсететін екі өлшемді матрица түрінде әрбір мәлімдемесінен кейін жолды көрсету қажет. & сілтеме өткізетін нақты параметр деп, ал # нәтиженің мәні боәйынша шақыруды білдіретін нақты параметр деп, ал $ нәтиже бойынша шақыру арқылы параметрлерді жіберуді пайдаланатын нақты параметр деп есептейік.

Program main ( )
integer i, j, k, a[6];

```
{ i = 0; j = 0; k = 2; a[0] = 1
for (i = 1; i =< 5; i++) a[i] = a[i-1] * 2;
messy(a[3], &a[4], &j, &j, #a[3], $a[4]);
}
void messy( integer a, *b, *c, *d, e, f)
{ f = 2;
  a = *b + *c + e;
  *b = *d + f;
  *c = a + *b;
  *d = *c - *d;
  e = *b + *c + e;
  f = e + a;
}
```

6. Көпіршікті сұрыптауды ADA, Modula-3, C #, Java-да жазыңыз және төрт бағдарламада пайдаланылылған жинау және трансмиссия параметрлерін салыстырыңыз.

4.12.3 Толық жауап

7. Кеңейтілетін деректер абстракция қалай жүзеге асырылуда? Талқылау.
8. Көп кітапхананы дамуындағы экспорты мен импорты артықшылығы неде? Түсіндіріңіз.
9. Класс тұжырымдамасы және модульдер тұжырымдамасы арасындағы айырмашылық неде? Айқын түсініктеме беріңіз.
10. Айты бойынша шақырудың қандай проблемаларыбар? Қарапайым мысалдарды пайдаланып түсіндіріңіз.
11. Қажеттілік бойынша шақыру денегіміз не? Оның аты бойынша шақырумен салыстырғанда тиімділігін қандай ? Қарапайым түсінікті мысалдарды пайдаланып түсіндіріңіз.
12. Алмасу бойынша шақыру дегенім не? Оның мәні бойынша шақырумен ұқсастығы қандай? қарапайым схемасын пайдалана отырып, түсіндіріңіз.
13. Сілтеме бойынша шақыру мен мәні бойынша шақыру арасында қандай айырмашылық бар? механизмі және тиімділігі тұрғысынан түсіндіріңіз.
14. Модулдер мен экспорттық-импорттық механизмдердің жергілікті емес айнымалылар қолдайтын розетка процедураларынан артықшылықтары қандай? Түсіндіріңіз.
15. Қандай жағдайда нәтиже мәні бойынша шақыру сілтеме бойынша шақыруға қарағанда қолайлы болып табылады?

16. Таратылады есептеу үшін параметрлерді жіберіге арналған механизмдерді түсіндіріңіз және салыстыру.
17. Әр түрлі программалау тілдері арасындағы өзара әрекеттесті қамтамасыз ететін механизмдер қандай? Түсіндіріңіз.
18. Ерекшеліктерді өңдеу механизмдер бар? Түсіндіріңіз.
19. Экспорттық-импорттық механизмін жергілікті емес айнымалылар және мұрагерлікпен салыстырыңыз.
20. Модульдер, сыныптар мен объектілер арасында қандай айырмашылық бар? Түсіндіріңіз.

Қосымша әдебиет

Абельсон, Гарольд, Сассман, Джеральд Дж. және Сассман, Джули. *Компьютерлік бағдарламалардың құрылымы және интерпретация***сы, 2-е басылым. MIT Пресс. 1996.**
Американдық ұлттық стандарттар институты. Ada бағдарламалау тілі. CSAISO/IEC 8652:201z. 2012. Онлайн қол жетімді http://www.adaic.org/ada-resources/standards/ada05/
Биррелл, Эндрю Д. и Нельсон, Брюс Дж. «Қашықтан шақыру процедураларын жүзеге асыру.» *Компьютерлік жүйелер ACM операциялар*, 2(1). 1984. 39-59.
Блэк, Эндрю, Хатчинсон, Норман С., Джул, Эрик и Леви, Генри М. «Emerald бағдарламалау тілін құрастыру». Бағдарламалау тілдері тарихы бойынша материалдар HOPLIII ACMSIGPLAN Үшінші Конференциясы бойынша жұмыс. 2007. 11-1-11-51.
Колинбьерн, Хью. *Ruby тілі туралы кітабы*. NoStarch Пресс. 2011.
Дейкстра, Эдсгер У. *Бағдарламалауды дисциплиналау*. Прентис Холл. 1976.
Хоар, Чарльз А. Р. «Компьютерлік бағдарламалаудың аксиомалық негіздері». *ACM-ға мәдімет жіберу*, 12(10). 1969. 576-583.
Гудак, Пол, Хьюз, Джон и Джонс, Саймон Р. «Haskell тарихы: Класстарға күш жімсама». Тілдер бағдарламалау ACMSIGPLAN әңгімелері үшінші конференциясының жұмысы. 2007. 12-1-21-55.
Кеннеди, Кен, Колбэл, Чарльз и Зима, Ханс. «Өнімділігі жоғары Fortran тілі жетістіктері мен жеңілістері: Тарихи нысан сабағы.» Тарих ACM SIGPLAN бағдарламалау тілдері бойынша үшінші конференциясының жұмысы. 2007. 7-1-7-22.
Кляйн, Питер. «Module-3 Бағдарламалық қамтамасыз етуді әзірлеу». T*ехникалық есеп 94-16*. Информатика III, Аахен технология университетінің департаменті. 1994.
Лисков, Барбара и Гуттаг, Джон. *Бағдарламаны әзірлеу абстракциясы*

*және спецификациясы.*МІТПресс. 1986.
Лисков, Барбара и Гуттаг, Джон. *Java Бағдарлама дамыту: абстракцияның, нақтылау, нысан-бағытталған жобалау.*Эддисон-Уэсли. 2000.
Одерски, Мартин, Спун, Лекс и Веннерс, Билл. *Scala бағдарламалау: толық нұсқаулық*, 2-е басылым. Artima Инкорпорейшн. 2011.
Страуструп, Бьерн. «Нақты әлемдегі және әлем үшін тіл эволюциясы: C++ "1991-2006". ACM SIGPLAN бағдарламалау тілдері тарихы үшінші конференциясының жұмысы. 2007. 4-1-4-59.
Ватт, Дэвид А. *Бағдарламалау тілдері ұғымдар мен парадигмалары*. Прентис Холл, 1990.
Вирт,Николас.«Module-2 және Oberon». ACM SIGPLAN бағдарламалау тілдері тарихы үшінші конференциясының жұмысы.2007.3-1-3-10.

5. Императивті тілдердегі үлгілерді үлестіру

Негізгі концепциялар

Абстракция және ақпарат алмасу (4-тарау); есептеу аннотациялық ұғымдар (2.4 бөлім); деректер құрылымы тұжырымдамасы(2.3 бөлім); Бағдарламалар мен бөліктері (1.4 бөлім); Рекурция (бөлім 2.2.4); Фон Нейман машинасы (2.1 бөлім).

Осы тарауда және келесі тарауларда біз төмен деңгейлі абстрактілі машинаның көмегімен бағдарламаларды іске асыруды оқимыз. дерексіз машина аралық деңгейде жоғары деңгейлі бағдарламалау конструкциялық арқылы жандандыру туралы абстрактілі түсініктеме береді. Абстрактілі моделді іске асыру көмегімен біз төмен деңгейлі абстракцияның динамикалық мінез түсінеміз және талдаймыз, Ол үшін (1) басқара алады жақсы және барынша тиімді бағдарламалар жазу және (2) компилятор үшін генератор кодын дамыту керек.

Төмен деңгейлі түрлендірілетін коды жад бөлу схемалары төрт түрін пайдалана отырып жүзеге асырылуы мүмкін: (1) статикалық тарату ;(2) буманың бөлу негізінде; (3) үйілген бөлу негізінде; және оңтайлы сақтау үшін алғашқы үш табыстарды біріктіретін (4) гибридті тарату сұлбасы. Бұл тарауда біз стек негізделген бөлу және гибридті бөлу статикалық бөлу қарастырамыз. Біз (1) стеке жергілікті динамикалық айнымалылар таратады деп гибридті бөлу (2) үйіндінің рекурсивті және динамикалық деректер құрылымдар таратады, сондай-ақ (3) тиімді Жад тікелей қол статикалық айнымалы статикалық таратуды пайдалануға бағытталған.

Статикалық бөлу схемалары компиляция кезінде бірінші кезекте қажетті жад бөледі сондай-ақ іске қосу уақытында жадтың өсуін қолдамайды. Осы схема артықшылығы әрбір деректер элементі бірегей жадындағы салыстырылған және жад қол жеткізу арқылы тікелей кіруге болады. Алайда оның кемшіліктер де бар. Статикалық бөлу қолдау көрсетпейді (1) рекурсивті процедуралар, өйткені жад орындау кезінде рекурсивті рәсімін тудыруы мүмкін-өздерін белгісіз саны есеге өсуін қажет етеді (2)

рекурсивті деректер құрылымы сияқты байланысты тізімдер, ағаштар мен векторлар ретінде кеңейтілетін рекурсивті деректер құрылымын уақыт белгіленбеген мерзімге ұзартылады және орындау кезінде жадын көбейту керек болуы мүмкін; және (3) орындау кезінде объектілердің динамикалық құру, өйткені ол уақыт орындау бойынша динамикалық объектілер мен естеліктер салыстыру қажет етуі мүмкін.

Орындау кезінде жады артуы статикалық бөлу арқылы қолдамайды, өйткені барлық үш шектеулер пайда болады. Әрбір объект бірегей жады ұяшығында салыстырылған, себебі статикалық бөлу, сондай-ақ, ысырапқорлық жады болып табылады, және тарату қызметі қазіргі уақытта таратылады нысанның өткеннен кейін жад ұяшықтарының қайта бөлуге мүмкіндік бермейді.. Сондықтан, статикалық бөлу схемасы (1) бағдарламалардың құрылымдық блогы қолдайтын бірнеше қоңыраулар бойынша функциялары мен рәсімдерін кіші бағдарламасы шеңберінде, өйткені ол қайта пайдалану жад қолдайдтындықтан (2) білім берудегі объектілі-бағдарлы программалау жасайтын динамикалық объектілер; (3) орындау кезінде ұзартылуы мүмкін рекурсивті деректер құрылымын; және (4) орындау кезінде жад өсуін талап ететін рекурсивті сияқты ретінде тілдік функциялард үшін қолайлы емес.

Стек негізделген тарату жергілікті деста және бақылау стек деп аталатын стек пайдаланады. Стек пайдаланудың көптеген артықшылықтары бар. Бөлу негізінде стек рекурсивті рәсім қолдайды, өйткені стек рәсімдерді шақырта отырып өсуді, және кезінде компиляция тіркелмейді. *Стек мөлшері операциялық жүйесі арқылы ғана шектеледі. Стек өсуі мүмкін болғандықтан стек басқармасының шақыру кезінде кіші бағдарламаны орындау кезінде, өйткені жергілікті қоймаға шақырылатын процедураға беріледі. Шақырыған программа аяқталғаннан кейін бағдарлама деп аталатын жергілікті сақтау орны шығады, және де ол орынды босатады, және келесі шақырылатын бағдарламаға үшін қайтадан орындалады. Стек негізделген бөлдің негізгі кемшілігі оларды құрылған ескі өмір арқылы жүріп, тізімдер, ағаштар, векторлардың немесе қарқынды құрылған объектілер сияқты қарқынды кеңейтілетін деректер құрылымды қолдаудың жоқтығы болып табылады.*

Үйіндінің бөлу негізінде үйінді деп аталатын ортақ жад аймағын пайдаланады, оны онымен қызмет мерзімі бірдей барлық бағдарламалар көріп, бұл кеңейтілетін деректер құрылымдардың, оның ішінде және қарқынды объектілерін құрады, деректер құрылымдардың барлық түрлерін орналастыруға болады.

Үйме бойынша бөлу үймеге сақталған процессордың тізілімдердің не-

месе деректер құрылымында бірінші деректер ұяшыққа арналған стек бақылау тапсырмаларын меңзермен негізделген, содан кейін сол логикалық деректер құрылымын басқа деректер ұяшығын айналып деректер жасушаларының арасындағы ішкі көрсеткіштерді пайдаланады. Әрбір көрсеткіші жад мекенжайы болып табылғандықтан рекурсивтық құрылымдар айналымы жадқа арналған көптеген өтініштерді талап етеді. Орындау кезінде операциялық жүйе үйме кеңістігі автоматты түрде ұзартылуы мүмкін немесе бағдарламалау нұсқаулары бағдарламасымен енгізілуі мүмкін. Таратылатын деректер құрылымын жойғаннан кейін, жад босатылады және қайта циклы үшін деп белгіленеді. Бөлінген жады кеңістік толығымен толтырылады кейін, содан кейін екі нұсқасы бар: (1) операциялық жүйесі арқылы жады кеңістігі арттыру, немесе (2) қайта бөлу үшін босаған орынды пайдалану. Бірінші тәсіл операциялық жүйенің қол жетімді жадқа байланысты. Екінші тәсіл еске қайта кәдеге жарату өндіреді және қоқыс жинау деп аталады.

Барлық үш модельдердің өзіндік артықшылықтары мен кемшіліктері бар. Статикалық іске асырудың негізгі артықшылығы айнымалы орналасқан жадқа қатынау бірыңғай жады қосылысты қолдана отырып алуға болады. Стек негізделген бөлу негізгі артықшылықтары (1) рекурсивті өңдеу процедураларын мүмкіндігін және (2) кіші аяқталғаннан кейін еске қалпына келтіру есебінен жадты қайта пайдалану болып табылады. Үйме негізінде бөлудің негізгі артықшылықтары 1) кеңейтілетін деректер объектілерін тарату және серпінді объектілер құру және 2) қарқынды мерзімі аяқталғанда және кеңейтілетін құрылымдық деректерді немесе динамикалық нысандарды бөлінген кезде жадты қайта қолдану болып табылады. Қазіргі заманғы программалау тілдері барлық үш тәсілдерді пайдаланатын гибридті дистрибуциялау моделін пайдаланады. Гибридті тарату моделі, статикалық айнымалы және жаһандық айнымалылар үшін статикалық тарату, бағдарлама блогындағы динамикалық жергілікті айнымалылар үшін стек негізіндегі бөлуді, және рекурсивті деректер құрылымы және серпінді жасалған нысандар үшін үйінді негізіндегі бөлуді пайдаланады. Бұл тарауда біз басқару абстракцияның төмен деңгейлі код өтуін, дестесін негізделген статикалық бөлу схемасы және бөлу схемасын талқылаймыз. Біз сондай-ақ әр түрлі параметрге өту тетіктерін өңдеу қатарын басқару тәртібін талқылаймыз. Келесі тарауда біз үймені пайдаланып динамикалық жады басқару талқылаймыз.

5.1 Абстрактілі есептеу машиналары

Абстрактті бағдарламасын іске асыру машинада фон Нейман (машинасында) негізгі бес компоненттерден тұрады: деректердің облысы, код-

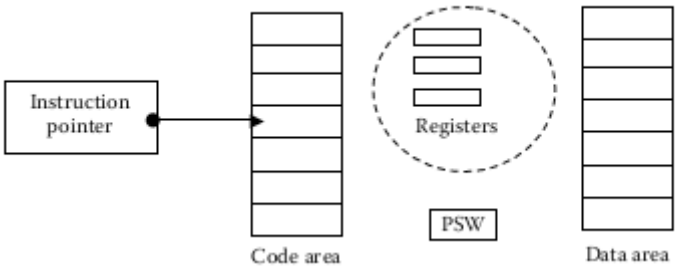
тардың облысы, қадамдық нұсқаулық үшін көрсеткіш нұсқаулықтардың облыстар коды, тіркелімдер үшін қара есептеулер мен сөздер жай-күйі бағдарламасы (PSW) —арнайы тіркелімде сақталатын жалаушалар жинағы. Әрбір нұсқаулықтан кейін тиісінше PSW жалаушалары қойылады және орындалатын бағдарламаның жай-күйін есептеуде маңызды бөлігі болып табылады. Код сақталатын облыс жады *облыс коды* деп аталады , ал деректер сақталатын облыс жады *деректер облысы* деп аталады. Бірінші сыныпты объектілерді қолдамайтын тілдер үшін облыс коды тіркелген болып табылады және қайталап пайдаланылған дегенді білдіреді, әр кез сайын шақырылатын кіші бағдарлама бастапқы жай-күйден яғыни басынан бастап басталады.

Деректер аумағы үш сызықтық құрылым (үйінді, стек басқару, статикалық облыс) болып табылады. Кучи және стек басқару қарама-қарсы бағытта әр түрлі соңдарынан бастап. Олар қолда бар жад кеңістігін барынша максималды пайдалану үшін бір-біріне қарама-қарсы өседі. Стек басқармасы кадрлар реттілігін білдіреді, көрсететін шақырушы кіші программа үлгісі соңғы келді - бірінші қызмет көрсетілді тәртібімен жүргізіледі. Әрбір кадр, сондай-ақ жазылып белсендіру деп аталады, жад облыстарында индексирлі реттілікті білдіретін іске қосуды сақтайды, ақпаратты сақтайды (1), қажетті ақпаратты есептеу үшін ағымдағы кіші бағдарлама (2), қайтушы басқару элементі кері шақырушы бағдарламасы және (3) шақырылатын кіші бағдарлама жандандырылған кездегі шақыратын бағдарламаның мұздатылған күйі. Кадрдағы ақпарат мыналардан тұрады: (1) жергілікті динамикалық айнымалылар үшін сақтау бағыттары - статикалық айнымалылар болып табылмайтын айнымалылар; (2) шақырушы кіші бағдарламада суреттерді тіркеу жаңартылады;(3) шақырушы бағдарламасының ақпараттық объектілерді, аймақ кодын және бақылау бумасының жоғарғы түрлі индекстері;(4) бағдарламалық қамтамасыз ету, бағдарламалық ортаны және қазіргі уақытта орындау тәртібін сақтау мұрағатталған бөлігі;(5) қарапайым динамикалық объектілерде кіші бағдарлама бір рет құрылады қызмет мерзімі кіші бағдарлама мерзімінен аспайды.(6) ағымдағы шақырылатын кіші бағдарламаның шақыру кезіндегі мұздатылған күйі; (7) шақырылатын кіші бағдарламалардың берілетін параметрлері;

Кіші бағдарламаның әрбір жандандыруы стек шыңына тиісті кадрды орналастырады. Аяқталғаннан кейін шақырылатын кіші бағдарлама, жадыны босату үшін кадрды жойылады. Рамка индекстеледі қолдау үшін қол жеткізу әр түрлі ақпараттық объектілері сақталатын кадрда және сілтемелер үшін басқа кадрлар немесе байламы . Аяқталғаннан кейін шақырылатын кіші отыскивается жағдайы шақырылатын кіші

және орындау кодын облысы қиқар бағдарламасы жаңартылады келесі нұсқаулықты шақырғаннан кейін орындалған кіші бағдарама.
Аудан коды - төмен деңгейлі командалардың блоктар тізбегі. Әрбір блок командасы күнделікті төмен деңгейлі командалар тізбегі болып табылады. Бақылау логикалық келесі командасының қадам пайдаланады немесе өтпелі командасын пайдалану арқылы бір команда басқасына секіру. Бақылау элементі бірінен екіншісіне кіші секіру нұсқаулығымен жылжытылады. Код облысында бақылау логикасы *көрсеткіштік командаңы* пайдалана отырып, ол ұқсас есептегіш командалар, командалар деңгейінде орналасу. Сипатталған нұсқаулық команда мен есептегіш команда арасында бір басты айырмашылық болып табылады және процессор бағдарламасы есептегіш: нұсқаулық сілтегіш ағымдағы команданың сәтті аяқталғаннан кейін артып, процессор бағдарламасы қарсы дереу ағымдағы командасына қоңырау кейін өседі. Бұл айырмашылық біздің ыңғайлы тұжырымдамасын түсіндіру және ауыстыру өңдеу шашылуды азайту үшін түсіндіріледі. Секвенсоры - көшу командасы - орнына көршілес командалардың тізбегінен, берілген мәнімен пайдалаушы тағайынды командамен бақылауға алуға, көшу командасы мұндай дәстүрлі конструкцияларын және итерациялық дизайн сияқты төмен деңгейлі хабар тарату басқару абстракцияның, үшін маңыздысы болып табылады. Келесі бөлімде біз түрлі абстракция басқаруды төмен деңгейлік абстрактілі командалар көмегімен диаграммалар логика басқарушы трансляциялауды талқылаймыз
Төмен деңгейдегі командалар командалар ішкі жиынына ретінде жіктеледі ; (1) жадында тұрақтылар маңызы сақтауға арналған (2) жад тіркелімі мәндерін сақтау үшін тізіліміне (3) тұрақты жүктеу үшін жады кеңістікте сақталған мәндерді тиеу үшін (4) және (5)команда подмножеств санатына жатады өрнектердің бағалау үшін) арифметикалық және логикалық операциялар (6) арифметикалық және логикалық өрнектер салыстыру операциялар командасы (7) шартты және шартсыз секіру (8) бақылау стеке деректерді жіберу және (9) бақылау дестесін деректерді басу .
Біз жадқы қол жеткізудің үш түрін болжаймыз: (1) жадқа тура рұқсат, (2) жадқа жанама қол жеткізу және (3) жадқа ығысу негізінде қол жеткізу. Жатқа тура қол жеткізу бастап берілген жад ұяшық (дейін) деректерді оқу немесе жазу үшін жады қатынасады, және ең жылдам болып табылады. Статикалық айнымалы жадқа тікелей қатынас. Жанама жады басқа жадқа немесе мәні жадында сақтайды ұяшықты, кіру үшін сілтегішті пайдаланады. Жанама қол жад құнына қол жеткізу көрсеткіштер сериясы арқылы өту және бірнеше жад қол жеткізуді қажет етеді. Жа-

нама жадқа қатынау тіркелген жад орындардан тәуелсіздігін қамтамасыз етеді. Алайда, бұл байланысты бірнеше жад қатынаған баяу жүреді. Әдетте жанама қол жеткізу пайдаланылатын көп рет көрсеткіштер тиімділігін арттыру үшін процессор тізілімдері сақталады. Офсеттік негізінде жад қол ақпаратқа қол жеткізу үшін пайдаланылатын құрама деректер объектінің шегінде ақпараттық заттарды немесе астыртын бірге нысандарды. Базалық мекенжайын, және объект немесе астыртын жылжуы туралы ақпаратты көрсетеді өрнекті: еңісі негізінде сөйлеген сөзінде екі ақпарат бөліктерін талап етеді. Базалық мекен-жайы —бірінші объект орналасқан ұяшықты мекен-жайы ОЗУ.
PSW жалаушалар күйін өрнекті бағалау нәтижелерін өңдеу үшін қажетті болып табылады. Салыстыру оң болса салыстыру теріс мәні «0» берсе терістеу биті N «1» белгіленеді, нәтижесі емес нөлдік логикалық өрнектің мәні болып табылады, егер салыстыру нөлге тең мәні береді және «0» егер салыстыру оң мәні береді. Нөлдік бит мәні "1" болған кезде салыстыру мәні нөл "0" болса, нәтижесі болып табылады маңызы жоқ, нөлге тең. Операторлар шартты филиалы далалық командаларға шартты секіру шалу үшін осы құсбелгілерді қажет. Кейбір операторлар breq (бәрібір-не нәрсеге тармақталған), brne (емес, сондай-ақ қандай тармақталған), brle (menee- үшін тармақталған, brgt (неғұрлым-кем тармақталған) (аз-кем тармақталған) шартты секіру brlt болып табылады астам немесе тең-қабылдау) және brge (а тең қарағанда--көп немесе--не дейін тармақталған).



Instruction pointer - **нұсқаулық көрсеткіш;** code area - **код орыны;** registers – **регистрлер;** PSW – **PSW;** Data area - **деректер орыны**
5.1 –сурет. Үлестіруге арналған абстрактілі машинаның сызбасы

Сурет 5.1 программалау тілдерін іске асыру үшін төмен деңгейлі реферат машинаны көрсетеді. Бірде, құрастырылған деректер мен бақылауды сақтау үшін аударылған абстракция код беріледі. Аудан коды және

деректер аумағы бір өлшемді массивтер түрінде қалыптастырылған. коды саласындағы жад ұяшық [коды-индексі] ретінде айқындалған кезде, есептеу білдіру «коды индексі» аймақ кодын бірінші командаға офсет берген. Деректер саласындағы жад ұяшық индексі деректер саласындағы базалық мекен-жайы, қазіргі уақытта көрінетін жылжуын есептеу кезінде бұл деректер өрнек болып табылады D [Index деректер], сондай-ақ белгіленеді.

Бағдарламалау парадигмасын, деректер саласындағы, сондай-ақ аймақ коды байланысты одан әрі атап айтқанда, бағзы машинаны анықтау үшін тазартылған болады. Реферат машиналар көпшілігі бақылау бумасының, үйіндісіне, сондай-ақ жаһандық және статикалық айнымалы ауданы тікелей қолжетімді жады кем дегенде талап етеді. Схемаларын жылы жасалғаннан кейін статикалық тарату базасы мекенжай 0 болып табылады.

Алайда стек бөлу негізінде схемада, жақтаулар дестесін түрлі жады жасушаларының орналастырылады, сондай-ақ (1) сілтегіш пайдаланып базалық жақтау мекенжайларын жақтау индексі деп аталады және (2) жақтау базалық мекен-жайы қатысты өзара есепке алу арқылы деректер элементтеріне қол жеткізіеді.

Төрт мұржалары бар SECD машина деп аталатын функционалдық бағдарламалау парадигмасын қараапайым дерексіз машина: мұндағы S, бағалауды білдіру үшін стек; E, бағдарламалық ортаның сақтауға арналған стегі; C, командалық жолдарды сақтауа арналан стек; D, дампының сақтауы үшін стек - соңғы келіп-бірінші қызмет негізін Тапсырысқа сериялық бағдарламалау рәсімі қоңырау. Кері қайтып есептеулер кейбір болдырмау және шешімдерді табуға балама жолдарын көріңіз жолын - логикалық бағдарламалау парадигмасы құрылады курс қолдайды. Қайтару іске асыру стек деп аталатын қосымша дестесін, сондай-ақ стандартты бақылау бумасын талап етеді. Стек қайтып барып, баламалы іс-әрекеттерді табу үшін болдырмау үшін басқару ұпай санын қадағалап отыратын.

Нысан бағдарланған тілдер виртуалды машина негізіндегі дестесін, Java виртуалды машинасын пайдалану, сондай-ақ сақтау объектілерін байламы пайдаланамыз. Императивтік тілдері бір дестесін және көптеген негізінде іске асыруды қолдайды.

Бұл тарауда біз аманатты бағдарламалау парадигмасын фон Нейман реферат машинаны талқылаймыз. Тиісті тараулар да басқа дерексіз машиналар қаралады. Мысалы SECD машинасы 9 тарауда талқыланды, сондай-ақ логикалық бағдарламалауға арналған Уоррен (WAM) абстракты машинасы 10-тарауда талқыланды. Объектілі-бағытталған тілдерді схе-

малық асыру 11-тарауда талқыланды.

...5.2 Басқару абстракциясын аудару
...Төмендегідей аударма басқару абстракция топтастыруға болады: өрнек бағалау (1) трансфер, (2) тағайындау оператор беруге, дәстүрлі итерациялық конструкцияларын құрылыстар, 4) беру, сондай-ақ (5) қоңырау аудару күтімінің (3) трансфер-. Бұл бөлімде біз бөлімдерінде 5.3 және 5.4 бөлек болады кіші қоңырау, сонымен қатар, басқа да бақылау абстракцияның барлық талқылайтын боламыз.

5.2.1 Айқындамалардың аудармасы

Процессор тізілімдер және жад жерлерде тіркесімін пайдалана отырып есептеу өрнектер. қол жад шығындарды азайту үшін, есептеудің кейін аралық мәні процессордың тізілімдері сақталады. Бірыңғай бағыныңқы өрнек бір рет есептеледі және болашақта пайдаланылуы тиіс тізілімдері сақталған; ортақ бірінші есептеу кейін, оңтайландыру әдістерін пайдалана отырып, түрлі компиляции талдау кезінде жүзеге бөлу тіркеліңіз. төмендегідей мысалы, білдіру (X + Y + 5) + 2 * (X + Y) бағалау түсіндірілген болады:

```
load X, R1 % load value stored in memory-loc(X) into register R1
add Y, R1, R2 % add value stored in memory-loc(Y) into register R2
add #5, R2, R3 % add constant 5 to R3 and store in the register R1
multiply #2, R2, R4 % value-in(register R4) = 2 * value-in(register R2)
add R3, R4, R4 % add the values stored in the registers R3 and R4
```

жүктеме X, R1 тіркелімінде жад ұяшық (X) сақталған мән, Y қосу R1% жүктеме, R1, R2,%, Тіркелу R2 жад ұяшық (Y) сақталған мәнді қосу, № 5 қосып, R2, R3%, R3 тұрақты 5 қосу және Тіркелу R1 сақталған, # 2 көбейту, R2, R4% мән-жылы (кейс R4) = 2 * маңыздылығы кіріс (кейс R2), R3 қосу, R4, R4%, сақталған мәндерді қосыңыз R3 және R4 тіркейді.
Жоғарыда мысал тиісті жады ұяшықтардан айнымалы мәндерін жүктеу үшін бағзы командалар төменгі деңгейде өрнектеледі, үзілу ретпен есептеу суреттейді және уақытша тізілімдері ішінара нәтижелерін сақтайды. Тек бір рет есептеледі және болашақ қайта пайдалану үшін тіркелімі R2 сақталады ортақ бағыныңқы өрнек (X + Y) екенін ескеріңіз. Болашақта, басқа да басқару абстракцияның беруді талқылау үшін, біз білдіру беруді болдырмау болады және есептеу (өрнек) ретінде тағайындаймыз.

5.2.2 Иелену операторының аудармасы

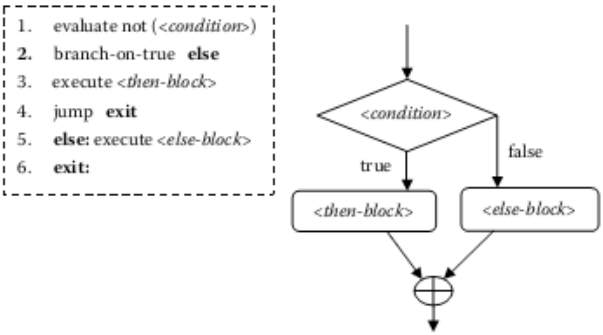
Тағайындау оператор <айнымалы> = <өрнек> бірінші өрнекті бағалай-

ды, содан кейін сол жақ айнымалы жад ұяшықтағы мәнді сақтайтын реферат тапсырмаларын төмен деңгейлі ретпен, балама болып табылады. Мұнадай балама X = Y + 5 мысалы осындай тапсырма операторы:

```
load Y, R0 % load the value in Y-location into register R0
load # 5, R1 % load constant 5 into register R1
add R0, R1, R0 % add value of R0 and R1, and store the result in R0
store R0, X % store value-in(R0) into location of variable X
```

Тіркелу R1 үшін Y, Y тіркеліміне R0 жүктеу # 5 ұяшықта R0% тиеу мәні, R1% тұрақты жүктеме 5 жүктеме
R0, R1, R0% қосылған құнды R0 және R1 қосып, және К0 сақтау R0 нәтиже сақтап, X% мәні-жылы (R0) сақтап X ұяшықта

Тағайындау операция (<өрнек>, <айнымалы>) қарапайым команданы тағайындау үшін тағайындау операцияны жәйттерді, тиеу, есептеу және сақтау операцияларының тіркесі.



1.бағаламау (<шарт>); 2. дұрыс тармағында, тағы 3. Орындау <содан кейін – блок> 4. Шығу 5. Егер тағы басқа болса, онда <тағы-блок> орындау 6. Шығу.
Condition – шарт; true – дұрыс; false – қате; then-block – содан кейін-блок; else-block – тағы басқа-блок.
Сурет 5.2 Абстрактілі машинадағы логикалық басқарудың өту диаггрaма схемасы

5.2.3 Шартты операторлар құрылымының аудармасы

Шартты логиканы Басқарушы үш кезеңнен тұрады: Boolean шарттары білдіру (1) есептеу; (2) аппараттық деңгейде мәртебесі жалаушалар ло-

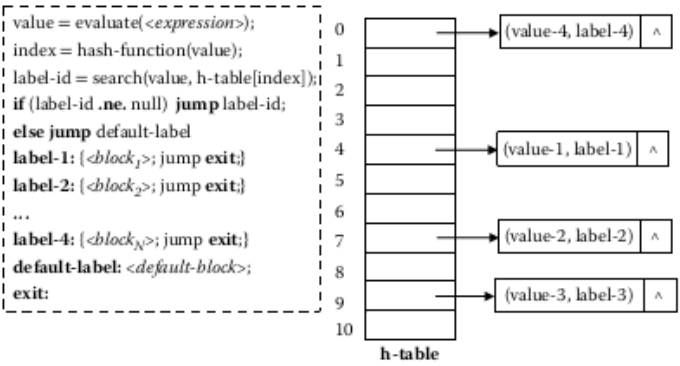
гикалық өрнектерді есептеу негізінде; және (3) өтпелі басқа бөліктеріндегі бір, содан кейін-бөлігі немесе бірлігі операторлары Нұсқаудың терімінің орындауға жасауға, жалаулар мемлекеттік филиалдарының құнының негізінде. Сурет 5.2 төмен деңгейлі аудару үшін шартты оператор және керекті схемаларын бақылау логика диаграммасын көрсетеді.
Диаграмма бақылау логика жазық екі өлшемді сан. Бұл бір өлшемді екенін коды айырбастауға тиіс. мақсатында өзара тастауды <содан блок> және <қалғанының-блок> қамтамасыз ету және <, әйтпесе блогында> арқылы жандандыру болдырмау үшін, оператор көшу кейін талап етіледі <содан бұғаттау>.
<Содан блогында> арқылы бақылау логикасын өткеннен кейін, өтпелі оператор <қалғанының-блок> айналып және <қалғанының-блогында> кейін ұйымдастырылуын бақылау үшін жүзеге асырылады. жағдай жалған болса, басқару, <қалғанының-блок> өтеді. шарттар дұрыс болса, бақылау автоматты түрде, <содан блок> енеді, өйткені алға бағытта әдепкі мәні бойынша бақылау логика жүріп жылдан бастап көшу, <қалғанының-блогында> жылжыту үшін ғана қажет. Бұл үшін бас тарту шарттарын тексеріледі. жағдай бас тарту шынайы болса, онда бақылау <қалғанының-блок> өтеді. Әйтпесе, бақылау <содан блок> Әдепкі арқылы өтеді.

5.2.4 Нұсқау операторының аудармасы

Операторлар нұсқасы болса-содан есептілігін ретін пайдалана отырып үлгіленуі мүмкін - <содан бұғаттау>, төменде көрсетілгендей, өзара эксклюзивті опциялардың бірін орындауға:

```
1. result = evaluate(<expression>);
2. if (result == value1) then {<block1>; jump-to exit;}
3. if (result == value2) then {<block2>; jump-to exit;}
   …
   If (result == valueN) then {<blockN>; jump-to exit;}
   <default-block>;
   <exit>:
```

Result – **нәтиже;** evaluate – **бағалау;** expression – **білдіру;** if – **егер;** value – **құны;** then – **содан кейін;** block – **блок;** jump-to exit – **шығысқа қарай секіру;** default-block – **әдепкі блок;** exit – **шығу.**



Value – **құны;** evaluate – **бағалау;** expression – **білдіру;** index- **көрсеткіш;** hash – **хэш;** function – **функция;** label – **заттаңба**, search – **іздеу;** table – **кесте;** if – **егер;** null – **нөл;** jump – **секіру;** else – **тағы басқа;** default – **әдепкі;** block – **блок;** exit – **шығу.**
5.3 сурет. хэш-кестесі арқылы ауысу операторының нұсқасы.

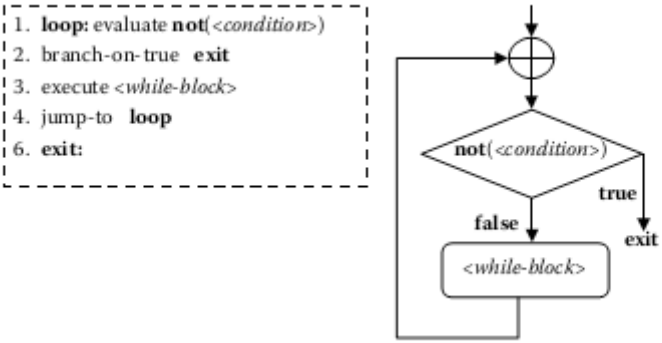
Сонымен қатар, ол бақылау суретте 5,3 көрсетілгендей, <өрнек> есептеу аяқтауға болады орындарда жиымын сақтайтын хеш-кестені пайдаланып жүзеге асырылуы мүмкін. Өрнекті бағалау кейін нәтижесі хэш кесте болып табылады. Хеш-кестесін үш есе нысаны (күтілуде мәні, келесі триплетті үшін сілтегіш, белгісі-бөлме) бар. Күтілетін мәні туынды құны сәйкес келеді, онда ол тиісті белгіні қайтарады; әйтпесе, ол «нөл» бар мәнін қайтарады, және бақылау Әдепкі белгі барады. күтілетін құнына тиісті блок бекітілгеннен кейін, басқару Марк басу арқылы басқа блоктар айланып «шығу».

5.2.5 Итерациялық конструкцияның аудармасы

Итерациялық жобалау аударма шартты құрылыстар пайдалана түрлендіріледі. Егер біз шартты мәлімдеме пайдалану арқылы циклын аударсақ, сондай-ақ мәлімдемеде мынадай болады:

```
1. Initialize the variables;
2. loop: if (evaluate(not <expression>) = true) {
                3. jump-to exit;
          4. else {execute <while-block>
                5. jump-to loop;}
          }
6. exit:
```

1.айнымалылар инициализациясы;
2.**цикл**: **егер** (есептеу(**жоқ** <*өрнек*>) = шындық) {
3.шығысқа бару;
басқалай{орындау< *блок кезінде* >
4.циклге өту;}}
5.**шығу:**
біз аударуға {<өрнек>, содан кейін {<блок болып табылады> болса, онда; Шартты сөйлемнің аудармасы алдын ала білімдерін пайдалана отырып, өтпелі-дейін} цикл, содан кейін аударым суретте 5.4 сияқты болады. (<Өрнек>) бірінші емес есептейді. (<Өрнек>) шын орнатылған жоқ болса, басқару әзірге, ілмектер тыс. Әйтпесе, бақылау <блогын дейін> өзгертпелі, және әлі, ол артқы цикл басына барып орындайды.



1.**контур:бағаламау (<шарт>) 2. Дұрыс және шығу 3. Орындау <блок кезінде> 4. Контурға секіру 6. Шығу.**
Not (<conditional>)-**жоқ (<шарт>);** false – **қате;** true – **дұрыс;** exit – **шығу; <**while-block> - **блок кезінде.**
5.4 сурет. Төмен деңгейлі реферат есепте аударым циклын қозғаушы

5.2.5.1 Контурға аудару
Контур үшін құрылыс келесі түрде:

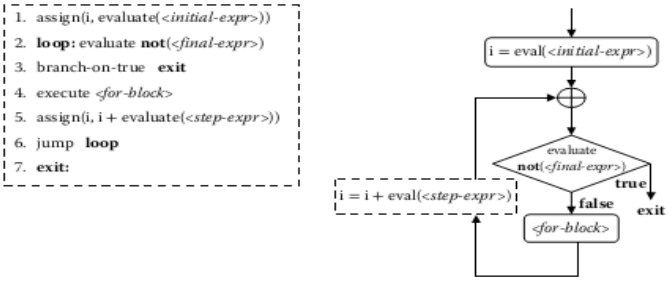
```
for (i = <initial-expression>';'<final-expression>';'<step-expression>)
        <for-loop-block>';',
```

For-loop-ты модельдеу үшін while-loop-ты қолданады, ол жерде соңғы шарт ол (in evaluate (<final-expression>)). While-loop-ты қолданып for-loop-ты аудару үшін келесі жағдайлар болады:

```
1) assign(i, evaluate(<initial-expression>);
2) while (not evaluate(<final-expression>))
       {3. execute <for-loop-block>;
        4. assign(i, i + evaluate(<step-expression>));
       }
```

Итеративтік құрастыру білімін қолданып, 5.5 суретінде контурдын аударымы корсетіліп тұр.

Using the knowledge of iterative constructs, the for-loop, is translated as shown in Figure 5.5.



1. Тапсыру (I, бағалау (<бастапқы білдірме>)) 2. Контур: бағаламау (<соңғы білідрме>) 3. Дұрыс тармағы және шығу 4. Орындау <блок үшін> 5. Тапсыру (I,i+бағалау (<қадам білдірмесі>) 6. Секіру контурға 7. Шығу; i=eval(initial-expr) - I, бағалау (<бастапқы білдірме>); evaluate not (<final expr>) - бағаламау (<соңғы білідрме>); true – дұрыс; false – қате; <for-block> - блок үшін; I=I+eval(<step-expr>) – I=i+бағалау (<қадам білдірмесі>).
5.5 сурет Төмен деңгейлі абстрактілі машинаға арналған циклінің аудару үшін схемасы.

Барлық 5.5 суретте аударма екі есептілікте қоспағанда цикл-болып ұқсас:. (1) индекстелген айнымалы баптандыру үшін команданы тағайындау, және (2) пайдаланушыға көрінбейтін индекстелген айнымалы тәсілі жаңарту. Инкременте блок индексі пайдаланушы цикл-дейін басында қадам мөлшері қоспағанда өрнектер көрсете есептілігінің блок шегінде біртіндеп индекстелген айнымалы ешқандай бақылау жоқ екенін көрсету үшін сызықты сызықтар пайдаланып көрсетілді.
Мұндай аударым ілмек «біраз уақыт» енгізілген және студенттер үшін жаттығулар ретінде қалдыруға болады. Итерация цикл болып табыла-

ды және үшін цикліне өте ұқсас. Айырмашылық тек қана кездейсоқ теріпалу элементтерінің бір жиынтығы арқылы өтеді, яғни, және цикл басында, олар жиынтығында соңына жеткендігі тексереді.

5.3 Статикалық үлестіру

Қазіргі заманғы тілдер рекурсивті тәртібін, қайта пайдалану жад рекурсивті динамикалық деректер құрылымы және деректер нысандарын қолдайды. Осылайша, гибридті моделін таңдайды. Алайда, кейбір статикалық жады бөлу ұғымдар дестесін және гибридті тәсіл негізінде жүзеге асыруды түсіну пайдалы болып табылады. Мысалын алайық Fortan-66, өйткені Fortan одан кейінгі нұсқасы стек негізінде бөлуді қолдайды.

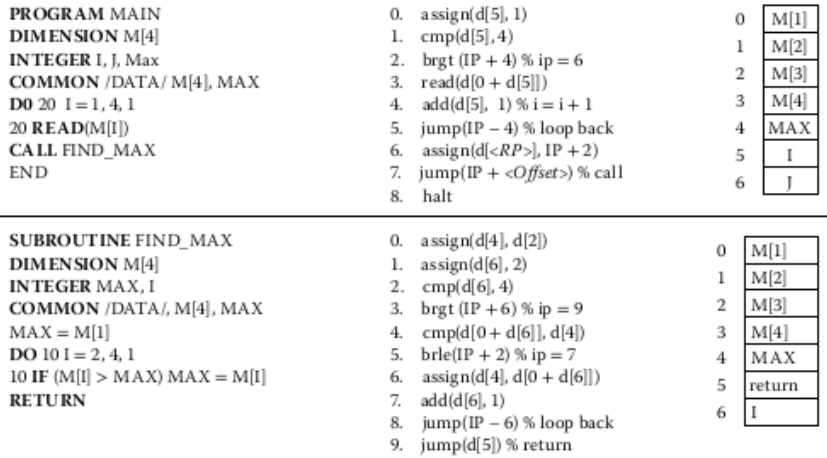
Деректер облысы түрінде берілген массив, I ұяшықта жад облысына қол жеткізу үшін сондай-ақ d [i] деп белгілеу жазбасын пайдаланылады. Алғашында ағымдағы жоспарлы тапсырмасы қоңырау кейін келесі тапсырмасы қайтару мекенжайын сақтау үшін жады жері кейін ортақ жад ұяшықтарды бөлінеді. Әр кезде бағдарлама басқа орындардан шақырылған сайын оның мағынасы өзгеретіндіктен кері мекен-жайы деректер саласында сақталады. Бірге қолданылмайтын барлық жергілікті айнымалылар, программа белсендірушілер жазбасында жергілікті бөлінеді.

Әр түрлі бағдарлама бірліктерін бөлек жасауға болады. Құрастырушы кодтары пайдаланушы белгіленген тәртіппен бір-бірімен байланысты. Деректер аудандары мен қала коды пайдаланушы белгіленген тәртіппен байланысты, және аралас түрлі кодтары туралы символы кестеде жинақталған мәліметтер қол жетімді болып табылады. Олар бөлек құрастырылған болса шақырушының бағдарламасының ауданы бойынша кез келген ақпарат және шақырылған аймақтың коды жайында деректер болмайды; ақпараттық кестеде нышандар ғана біріктіру кейін қол жетімді болады. Символ кестелерден алынған ақпарат қамтиды: (1) ақпараттық объектілердің әкелу және әкету туралы ақпаратты, (2 Программаны шақыру кезінде қайда өзгеріс жасауға болатыны туралы ақпаратты, (3) ағымдағы программа аяқталғаннан кейін қайтару үшін қайда бару туралы ақпарат. Компиляция кезінде, әрбір бағдарлама жетіспейтін мәліметтердің жиынтығы символы кестеде арқылы жалғастыру кезінде оған бағдарламаның тиісті бірлік қамтамасыз етіледі деп болжанады.

Байланысу барысында әртүрлі кодтан құрастырылған деректер ауданы жалпы жарнам блогі бірге, содан кейін кері мекен-жай, содан кейін жергілікті айнымалылар бірігетіндей болып құрастырылады. Құрастырылған кодтар аймағы (C, R_I (I > 1), L_I (I > 1)) делік, бұл жерде C – ортақ блок, RI – I құрастырылған код аймағындағы жад ұяшығының

кері мекен-жайы, LI- і деректер аумағындағы жергілікті ауыспаллардың жиынын білдіоеді, деректер аймағын біріктіргеннен кейін қолданушы көрсеткен тәртіппен келісідей болады (C, L1, R2, L2,..., RI, LI,... RN, LN). Түрлі кодтардан жасалған осы аттас барлық ортақ блоктар ортақ бірлік ретінде бірге біріктіріледі және бірлескен деректер саласындағы басында орналастырылады. Осы салаларда бірігуі нәтижесінде байланысу процесінің барысында кері мекен-жайларда жад адресі мен жергілікті хабарландырулар біріктіріледі. Байлау — екі сатылы процесс: (1) бірінші кезеңде, барлық деректер аудандар біріктіріледі,бағдарлама үшін есептелген қадам және бағдарлама қайтып, деректер облыстардың түрлі бағдарлама бірлік ауыстыру есептеу, ал (2) екінші кезеңде, тиісті бос коды облысында енгізілген.
Шақыру бағдарлама қамтиды : (1) бағдарламаларды кері мекен-жаймен шақыратын клесі орындалатын команданы жад ұяшығында сақтау , (2) мүмкіндіктері туралы ақпарат алмасу , (3) шақырылған бағдарламаның бірінші тапсырмасына көшу.

Мысал 5.1
5.6 Сурет статикалық бөлу схемасын пайдаланып іске асыруды көрсетеді. Бағдарламаның негізгі және кіші ТАБУ_MAKS бөлек құрастырылған деп есептейік.
Негізгі бағдарлама және бағдарлама ортақ жалпы блок тұратын айнымалы МАКС және тіркелген массив өлшемімен 4. Негізгі емес пайдаланылатын бірлесіп айнымалы I және –J болып табылады. Бағдарлама НЕГІЗГІ деректер жалпы алқабындағы M оқиды, содан кейін ТАБУ _ МАКС бағдарламасы тудырады. Бағдарлама ТАБУ _МАКС максимум массивінің M табады және жалпы блок бірлесіп пайдалану үшін маңызы бар айнымалы МАКС пайдаланады.Төмендәрежелі коды жадқа тура рұқсат көрсеткендей, d[] пайдаланады, және ығысу қол жеткізу үшін элемент массиві әдісін қолданады.
Негізгі бағдарламада компиляциядан кейін, айнымалы мекен-жайы M [1]-M [4] дейін болып табылады тиісінше d [0] - ге d [3] дейін, ал айнымалы МАКС ұяшық жады d [4] сәйкес келеді, және жергілікті айнымалылар I және J сәйкесінше d [5], d [6] жад ұяшықтарына сәйкес келеді. ТАБУ_МАКС бағдарламасында, мекенжай элементтері айнымалы деректер M[1] M [4] дейін сәйкесінше d[0] - d[3]ге дейін сәйкес келеді.



Program main – **негізгі бағдарлама;** dimension – **өлшем;** integer – **бүтін;** common – **жалпы;** data – **мәлімет;** max – **максимум;** do – **орындау;** read – **оқу;** call find max – **максимумды табу шақырылым;** end – **соңы;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** subroutine find max – **кіші жосарлы әдістерінің максимумын табу;** if – **егер;** return – **қайтару;** loop back – **контурды қайтару;** offset – **офсеттік;** call – **шақыру.**

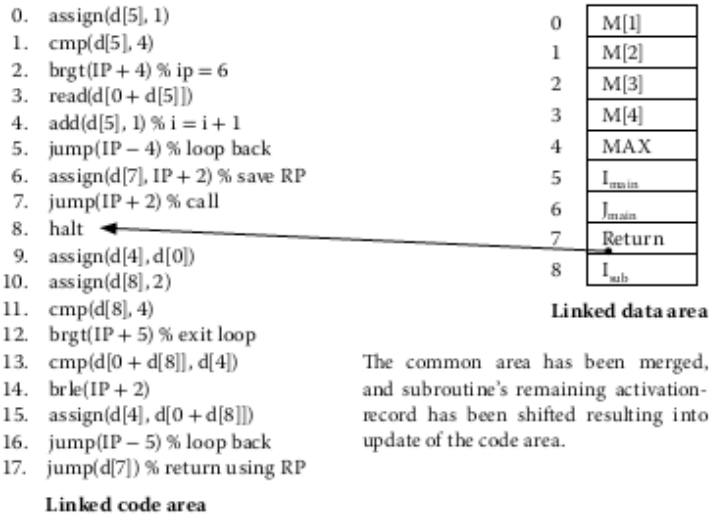
Сурет 5.6 Код аймағы және деректер аймағы үшін статикалық орындау кейін компиляциясы

Бұл бағдарлама find_max және тезистерді екі негізгі бақылау режимдері қоңырау шалу үшін төмен деңгейлі көшу көрсетеді: Цикл және шартты есептілігін (цикл үшін бір түрі) «болып табылады». Transfer Цикл негізгі бағдарлама [0] ұялы ауданы коды басталады және [5] аяқталады жылы «болып табылады». A [0] жад ұяшық 1 D мәні орнатады бекіту [5], айнымалы I. Қабылдау болып табылатын [1] мәні D салыстырады [5] UI + 4 үшін [2] филиалдарының тұрақты 4. Бала асырап алу бар, құны, егер D [5] асатын 4. Ескерту циклының бақылау жою үшін «IM + 4» пайдаланады. Егер меңзер тапсырмасы ығысу қосу әдісі бағдарламасы компьютерде іске жүктеледі жад аудандардың төмен деңгейлі коды тәуелсіз етеді. Ал [3] жад ұяшықты M [I] қатынау үшін ығыстыру әдісін қолданады нұсқаулары: 0 S базалық мекен-жайы болып табылады [0] және D [5] [4] мәні D арттырады бар айнымалы I. Нұсқаулар мәні сақтайды [5] 1 және оқыту [5] Артқа [1], топтамасының келесі итерация басталады,

онда бақылау қалыптастырады. беттегі нұсқауларды [6] және [7] қоңырау find_max әдістерге байланысты. [6] жад «IM + 2» туралы нұсқаулары - осы нүктесінде белгісіз жоғары деңгейлі кіші бағдарламасы қоңырау кейін келесі оқыту мекенжай find_max кіші мекенжайын кері белгісіз болғандықтан, жаңа символы символы кестеде енгізіледі. қайтару жады мекенжай меңзердің байланысу барысында символы кестеде іздеу жүргізіледі. [7] деп аталатын кіші find_max бірінші орындалатын туралы есепте ағымдағы тапсырмасы көшуді ығысу нұсқаулары бірге пайдаланады. Алайда, ығысу түптеу және сілтеме уақытта есептеледі дейін, сондай-ақ белгісіз.

Құрастырғандар программа find_max төмен деңгейлі код-DO рамалы [8] үшін [1] жады орындала алады, және Шартты сөйлемнің аудармасы, ілмектер «, ал» кірістірілген, [6] дейін [4] ұяшықтарын алады. [1] бетте 2. нұсқауларын Мен индекстелген айнымалы баптандыру тең туралы нұсқаулық [2] Мен индекстелген айнымалы 4. нұсқаулары артық болса, [3] циклі-DO бақылау алады отырып 4. нұсқаулары жоғарғы лимиті бар индекстелген айнымалы мәнін салыстырады M [I] кем немесе айнымалы MAX құнына тең болса, [4], [7], айнымалы жоласты M айнымалы MAX мәнімен [I] және табыстардың құнына салыстырады. Әйтпесе, S [6] M [I] MAX айнымалы мәні орнатады. 6 [8] 1. Ескерту [7] мәні D арттырады үшін нұсқаулары [6] (жергілікті айнымалы I) [2] тапсырмасы, және топтамасының келесі итерация өту үшін IP нұсқаулық меңзерді алынады. [9] жад ұяшық кері мекен-жайы сақталады мекенжайын таңдайды, және шақырушының тәртіппен келесі тапсырмасына бақылауды қайтарады. Соңғы мәлімдемеде күнделікті аймақ кодын түкпір-түкпірінен бірнеше рет деп атауға болады болғандықтан, одан кейін жад ұяшығының мекен-жайы спам-сақталатын қайтару әрбір уақыт әр түрлі болады.

5.7 суретте көрсетілгендей, ортақ айнымалыларға міндетті кейін M M [1] бұғаттау [4] және айнымалы MAX [4] D үшін [0] аралас және жад ұялы D орналастырылған. Естеліктер мекенжайын және мен find_max тәртіппен деректер саласындағы негізгі бағдарлама үшін таңдалған соңғы жадындағы кейін орналастырылады жергілікті айнымалыны қайтарады. кері мекенжай [7] D пайда болады және Isub [8] D көрсетіледі жергілікті айнымалы. find_max тәртібін бөлісу жоқ естеліктер (негізгі бағдарламаның іске қосу жазу) өлшемі бойынша жылжыту болып табылады - мөлшері (кіші бағдарлама find_max жалпы блок) 7 = - Бұл ығысу = 2. 5 кіші компилируемый код жад ұяшықтарының қосылады.



max – **максимум;** read – **оқу;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** return – **қайтару;** loop back – **контурды қайтару;** call – **шақыру;** save RP – **RP-ны сақтау;** exit loop – **контурдан шығу;** return using RP – **қолданған RP-ны қайтару;** linked code area – **қосылған код орыны;** linked data area – **қосылған мәліметтер орыны;** the common area has been merged and subroutine's remaining activation-record has been shifted resulting into update of the code area – **жалпы орын біріктіріл-ді және кіші әдетті белсендіру рекорды жылжып кетті, нәтижесінде код орыны жаңартылды.**

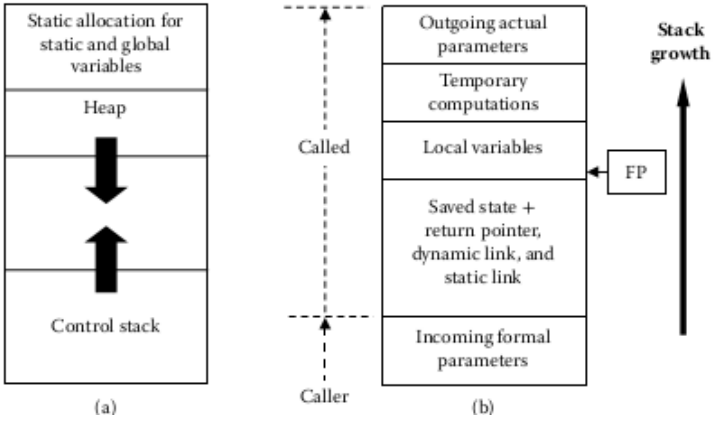
Сурет 5.7 Коды және деректер облыстардың байланыстыру бағыттары. *Байланысу барысында NAYTI_MAKS байланған кодты жасау.Symbol <R> C [6] 7 ауыстырылады - қайтару жады мекен-жайы меңзердің - және символы <S> бірінші санатшасы бағдарламасы нұсқауларға жылжыту үшін 2 офсеттік мәні ауыстырылады.*

5.4 Гибридті үлестіру

Блок құрылымына тілдер, суретте 5.8 көрсетілгендей, басқару дестесін, үйіндісін және статикалық бөлу тұратын гибридті бөлу, пайдалану, ре-курсивті тәртібін немесе динамикалық объектілерді қолдайды. инте-грацияланған моделін деректер ауданы нысаны (статикалық облысы,

стек басқару, үйілген) бір триплетті болып табылады. Статикалық жады бөлу компиляция кезінде бекітіледі және статикалық айнымалылар және жаһандық айнымалы үшін пайдаланылады. Стек бағдарламасы барысымен қарқынды басқару мен өзгерістер көп. Түрлі бір және сол бағдарламаны қоңырау үшін бастапқы деректерді байланысты әр түрлі өлшемдері мен үйме басқару дестесін талап етуі мүмкін, өйткені стек және үйінді басқару қарама-қарсы бағытта өседі. Бақылау стек мөлшері бірнеше рет үлкен, онда үйменің кеңейту үшін қосымша жад қажет болады. Үйме жинақ биіктігі сақтай отырып және бір-біріне қарай екі ұшын бақылау кезінде, бақылау стек және үйме ұшы пайдалану тұрғысынан анықталады, және жад аймағы барынша пайдаланылады.

Үймеге орналастырылған объектінің бірінші сілтегіші, кадрдың немесе тіркелу тәртіппен қалады. Үйме бөлу келесі тарауда егжей-тегжейлі талқыланды. Алайда бұл тарайда үйінді өту параметрлерін талқылау үшін нысандар үйіндіде бөлінген кезде аталған болатын. Бағдарламада аталатын әр кадр көп мағұлматты қамтиды: (1) кірітен нақты параметрлер үшін жад ұяшығы, (2) шақырылғвһан бағдарлама үшін сақталған жағдай, (3) (қоңырау шалушының жақтаудың және емес жергілікті айнымалылар қоса алғанда) бақылау стек кіру үшін және есептеу кіші жай-күйін қалпына келтіру үшін әр түрлі көрсеткіштер деп аталады, (4) жергілікті айнымалы жад, (5) дөрекі есептеу үшін жады, және (6) жад шығыс нақты параметрлері.



Static allocations for static and global variables - **статикалық және жаһандық айнымалыларға статикалық бөлу;** heap – **үйінді;** control stack - **бақылау стек;** called – **шақырылды;** caller – **шақырушы;** outgoing

actual parameters - **шығыс нақты параметрлері;** temporary computations - **уақытша есептеулер;** local variables – **жергілікті айнымалылар;** saved state + return pointer, dynamic link, and static link – **сақталған түрі + көрсеткіш, динамикалық байланыс және статикалық байланытардың оралуы;** incoming formal parameters - **Кіріс ресми параметрлері;** FP-**FP;** Stack growth – **стек өсуі.**

5.8 сурет. Блок-құрылымдалған тіл үшін Аудан деректер. (A) жалпы деректер аймағы (B) жақтау.

Бағдарлама шақырылған кезде, шақырылған бағдарлама өзгерген кезде есептудің кейбір режимі сақталады. Сақталған есептеу режимі шақырушының ескі өзгерту тізілімдері (1)шақырылған бағдарламадағы регистрлердің мәні мәндерін, (2) PSW, (3) өріс кодын және кіші бағдарлама қоңырау деректер аймағын кіру үшін пайдаланылады түрлі көрсеткіштер. Әртүрлі индекстері қол жеткізу үшін құрылған: (1) бағдарлама коды және деректер аумағы негізгі мекен-жайы нақты параметрлерін (2) шақыруы бойынша сілтеме жады ұяшықтарына нақты параметрлердің базалық мекен шеңберінде шақырылатын бағдарлама, егер параметрлері бойынша беріледі (3) кадрлар рәсімдерді кезінде енгізілуде ағымдағы рәсімі. Әрбір локальдық айнымалы ұяшық бөлінеді. Айнымалы деректердің түрлі көлемдегі нысандарымен байланысты болуы мүмкін. Мысалы символы ғана бір байттан алады, және бүтін 32 биттік компьютерде 4 байт. Жоғарғы бөлігінде кадр кейбір бар жад ұяшық, олар сақтау үшін пайдаланылады және ішінара нәтижелерін есептеу кезінде есептеу өрнектерді, және, ақырында, оның құрамында нақты параметрлері, олар шақырылатын бағдарламаға берілуі тиіс
Нақты параметр өрнек есептейді, және нәтижесінде құны уақытша жадында шығыс параметрлері ауданында жаңа ұяшықты сақталады. Жабылмайтын облысы бар динамикалық айнымалылар жергілікті жады неғұрлым тиімді пайдалану үшін бірдей жад ұяшықтарының салыстырылатын болуы мүмкін. Жоспарлы аяқталғаннан кейін, жақтау кіші алынып тасталады, жиынтығы көрсеткіштер шақырушының бағдарламаға кіру үшін, сондай-ақ стек меңзер жоғарғы бірден шақырушының бағдарламасының блогында кейін жад орынға көрсету үшін қалпына келтіріледі. Осылайша, шақыратын бағдарлама арқылы қолданылады барлық жад болашақта қайта пайдалану үшін босатылады.

5.4.1 Әр түрлі нұсқаулар рөлі

Өңдеу үшін қажетті негізгі бес көрсеткіштер бар. Бағдарлама шақыру

кадрға шақыру шалушы және қатынас сілтемелерді есептеу күйін сақтаудан кейін, басқару шақыратын бағдарламаның бірінші мәлімдемесіне өтеді. Стек арқылы бөлу бағдарлама орындау барысында бес көрсеткіштер талап етеді: кадлық меңзер, қылмыстық кодекс деп аталады және қол жеткізу мүмкіндік береді: (1) қазіргі бағдарламалауда уақытта жұмыс жасап тұрған деректер аумағы, (2) орналасу кіріс параметрлері, (3) Шақырушы бағдарлама туралы сақталатын ақпаратты. Блок меңзер сақталған мемлекет кейінгі алғашқы жады орнын көрсетеді; және жад қол жеткізу барлық басқа ұяшықтарға жақтау меңзер сақталған офсеттік базасы мекенжайын қосу арқылы жүзеге асырылады. *Көрсеткіш шыңы стек*, КШС ретінде белгіленеді , және келесі бос ұяшыққа жад дестесін басқаруды көрсетеді. *Динамикалық байланыстырушы*: ДБ мен жад үшін шақырушының жақтау меңзерді аталатын және абоненттің жақтау меңзерді қалпына келтіру үшін пайдаланылады. *Индекс қайтару*, бағдарламаны шақыратын мәлімдемесінен кейін ИҚ мен жад үшін шақырушының бағдарламасының кезекті тапсырмасы мекенжайын көрсетеді. *Статикалық сілтеме,* СС деп аталатын, және қазіргі уақытта бағдарлама инвестицияланатын бағдарламаның кадрына бағдарламаны қатынау үшін мекенжайын сақтайды. Бұл жергілікті айнымалылар қол жеткізу үшін қолданылады. ИҚ және КШС сияқты көптеген көрсеткіштер неғұрлым жылдам қатынасқа қол жеткізу үшін тіркелімдерінде сақталады.

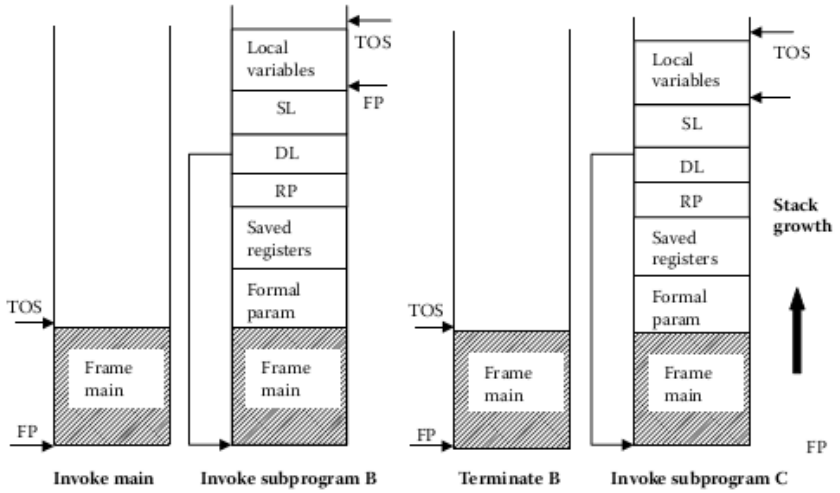
5.4.2 Шақырылатын бағдарламалар

Шақыру кезінде көрсеткіштегі өзгерістер келесі тәртәппен болады:

1. Шығыс ауқымы трансмиссиялық параметр түрі мен параметрлерін санына негізделіп орнатылады. Осы уақытта, КШС нақты параметрлерін мәндерін итеру арқасында жаңартуларды алуын жалғастырады.
2. Шақырылған бағдарламадағы келесі нұсқаулықтағы жад ұяшығы шақырылған қайтару бағдарламасының сілтеулігінде сақталады.
3. Динамикалық байланыс шақырылған бағдарламаның меңзеркадрына тең болып орынатылады.
4. Статикалық байланыс енгізілген бағдарламаның кадрға көрсететіндей етіп орнатылады.
5. Шақырылған бағдарламаның кадрын көрсету үшін көрсеткіш стек шыңының қазіргі мәні кадрлік көрсеткішке көшіріледі.
6. Нұсқаулық меңзер деп аталатын бағдарламаның бірінші тапсырмасын еске орналасқан жері бойынша орнатылады.
7. Стектегі жергілікті динамикалық айналмалыларын бөлу үшін шақырылған бағдарламаның КШС жергілікті ортаның мөлшеріне артады.

Осы көрсеткіштер орнатылған рет маңызды болып саналады. Мысалы, егер біз КК-ні КК-нің ескі мәнін динамикалық сілтеме көшіруді алдында өзгертсек, онда шаұырылған бағдарламаның кадрына өол жеткізу көзі жойылады. Сол сияқты егер бірінші КШС орнатсаңыз, сіз жаңа КК орнату мүмкін емес.

Көрсеткіштер орнатылған тәртіп жүзеге асыруға байланысты. Бұл жерде біз КК ығысуы 3, динамикалық сілтеме ығысуы - 2, және статикалық сілтеме ығысуы (бар болса)- 1 деп болжаймыз. Егер статикалық сілтеме болмаған жағдайда тіл енгізілген рәсімдерді қолдамайды, өйткені онда КК 2 және ДБ ығысуы 1 болады. Осылайша, қол жеткізу көрсеткішке қайтару арқылы жүргізіледі d[КК-3] (немесе D[КК-2], егер SL болмаған жағдайда). Біз сондай-ақ КК және КШС тиімді еске сақтау үшін бағдарламаның тізілімінде сақталады деп санаймыз.



С бағдарламасын шақыру
5.9 Сурет Шақыру рәсімі кезінде кадрдың қозғалысы.

Мысал 5.2
Сценарий қарайық. Негізгі бағдарлама А программасын шақыралы. А программасы В программасын шақырады. В программа жұмысты орындайды және аяқтайды. Содан кейін, негізгі бағдарлама С бағдар-

ламасын шақырады. 5.9a суретте стектің бақаруы B бағдарламасында болғандағы бақылау бумасының суретін көрсетеді атайды. 5.9b сурет бақылау программасы B болғандағы стек басқаруын көрсетеді. 5.9c суретт B аяқталған кезде, бақылау элементі негізіг программаға қайтқан кездегі стек басқарун көрсетеді. 5.9 d -суретте C бағдарлама негізгі бағдарламамен шақырылатын кездегі стек басқару көрсетілді. B программасымен қолданылған жадтың кері қолданылып жатқанына назар аударыңыз.

...5.4.3 Деректер аймағын және кодты жасау

Сақтау кіші бағдарлама қамтиды: (1) жаһандық айнымалыны құрылған жад, (2) жергілікті емес айнымаламен құрылған жад, (3) аөпаратты жіберу жолымен құрылған жад , (4) жергілікті айнымалармен құрылан жад. Ғаламдық айнымалылар және статикалық айнымалылар статикалық деректер саласындағы басында бөлінді және жад қол кез келген меңзерді пайдаланбай жүзеге асырылады. Жергілікті айнымалы қатынау бар бақылау дестесін жақтау сілтегіш жылжуын қосу арқылы жүзеге асырылады. Мысалы, D [КК + 3] Корпус көздегішінде көзделген мекенжай бойынша үш жасушаларында болып жады ұяшыққа, қол жеткізуге мүмкіндік береді.

Статикалық сілтеме (СС) сыртқы келесі деңгейлі кіші мекен-жайы, және жергілікті емес айнымалы базалық жақтау салынған тәртіппен жоғарыда N (N≥ 1) деңгейі ретінде мәлімделген болып сақтайды. Статикалық сілтемелер (1) тізбегі, және тізілімдер (2) тіркеуді көрсету: жергілікті емес айнымалыларға қатынау үшін екі әдістері қолданылды.

Статикалық сілтемелердің тізбегін қозғаушы статикалық сілтеме ығысуы белгілеген, бұл жағдайда орналасқан жады ұяшығында, сақталады деп болжайды. Бұл әдіс шеңберінде сақталған статикалық қосылым деректерін пайдалана отырып, дәйекті өтулер пайдаланады. Айнымалы емес, жергілікті «M» деп жарияланған басқа деңгейге кірістірілген болса, деректерге қол жетімділікті төменгі деңгейі D (м, ығысу) ретінде жазуға болады, және төмендегідей айнымалы қол жеткізу үшін емес, жергілікті алгоритм болып табылады:

```
while (m > 0) {
...     m = m - 1; frame-base = d[SL];
...     SL = frame-base − 1;}
...     value = d[frame-base + offset]
```

Статикалық сілтемелер мақсаты қарапайым әдіс болып табылады және енгізу деңгейі тым жоғары болса үстеме жад-қа бірнеше қатысудан зардап шегеді. . Екінші әдіс дисплей тізілімдері пайдаланады. Дисплей тізілімдер процессор немесе тікелей адресті емес жергілікті айнымалылар бар кэш тізілімдері болып табылады. Дисплей тіркелу деп аталатын рәсімін іске толтырылған, және емес жергілікті айнымалы қатынау тиімді сілтегіш тізбегін өткеннен жүзеге асырылады.
Параметрлері сілтеме бойынша өтіп кезде қоңырау шалушының аясында айнымалылар деп аталады. Шақыруда сілтеме бойынша формальды параметр болып табылады көрсеткішімен бірінші ұяшықты жад ақпараттық объектіге сілтеме беріледі. Егер абстракция деректер болып табылады құрамдас ақпараттық объектісі болса, онда қол жеткізу әр түрлі атрибуттар арқылы жүргізіледі қосу ығысу атрибута бірінші ұяшықты жад ақпараттық объект болады. Сол сияқты, абстракция жеке деректер мен элементтеріне D [сілтеме байланысты] ретінде есептеледі деректер нысаны жад ұяшық і мөлшерін көбейту арқылы индексін ақпараттық объектілердің жинағы, ығысу есептелінеді + і өлшемін (жеке-тармағы деректерді) және мәнді * егер D ретінде қол жетімді жад [D [сілтеме байланыс] + і өлшемін (жеке элементі деректер) *].
Шақырушы күнделікті тек жақтау меңзерді ығысуды пайдаланып, жергілікті шеңберінде жад ұяшықты қатынасады. Бағдарлама іске қосылғанда бағдарлама арқылы шақырылған кадр итерілсе онда кадрдегі қолданылып отырған жад программа аяқталғаннан кейін қайта пайдалануға шақырылады. Бағдарлама қоңырау үшін нұсқаулар тізбегі төмендегідей болады:

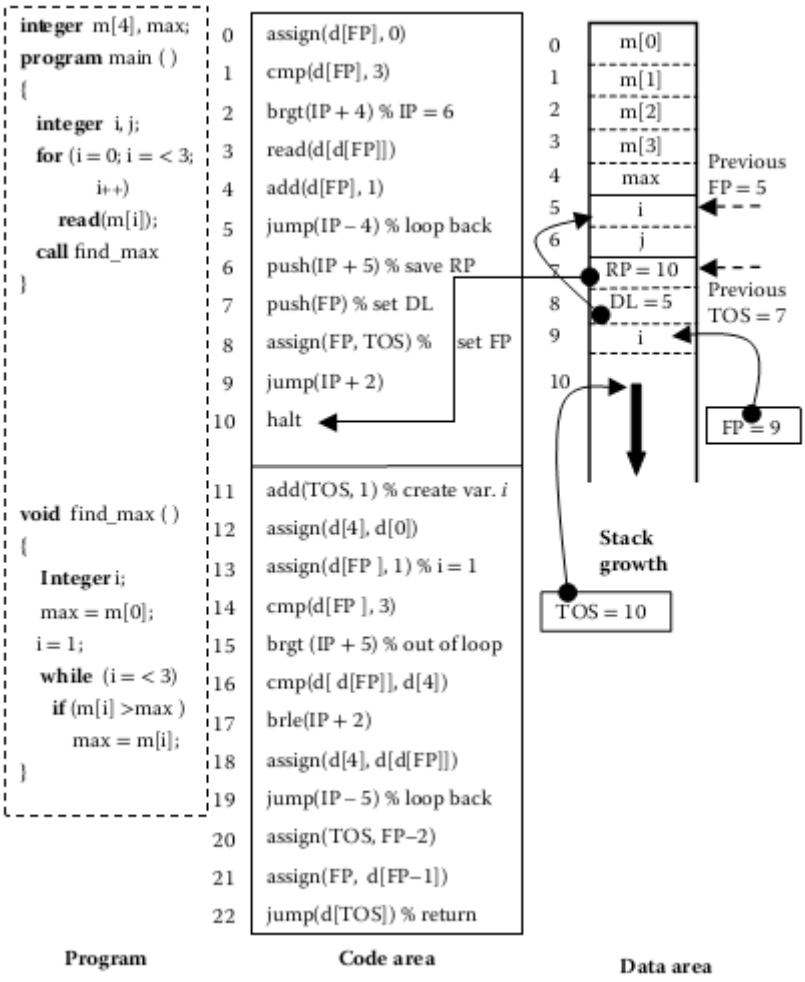
1.Нақты параметрлері өрнекті бағалау және шығу параметрлері мәндерін көшіріңіз.
2.Шақыратын кадрға режимін сақтау үшін PSW және қоңырау шалушының тіркелімдер сақтаңыз. Программаны сақтау кезінде өзгертуге ғана тізілімдері сақталған.
3.Бағдарламаны басқарудан оралғаннан кейін шақырушының бағдарламасының келесі нұсқау көрсету үшін қайтару меңзерді жаңартыңыз.
4.Бағдаламамен шақырылған динамикалық сілтемеге кодекске көрсеткішін көшіріңіз.
5.Егер жергілікті емес айнымалылар тілмен байланысса, статикалық сілтемені жаңартыңыз.
6.Жаңа кодекс көрсеткіші мәні ретіндеКШС көшіру арқылы жақтау меңзерді жаңартыңыз.
7.Көрсеткіші қайтару үшін ол нұсқаған болатын нұсқаулығы, келесі

шақырылатын бағдарлама үшін жаңарту
8.Шақырылатын бағдарламада бірінші нұсқаулыққа көшу.
9.Коды бойынша бағдарлама, бірінші нұсқаулық сілтегіш-шыңдары дестесін (КШС) бағдарламасының кадр өлшемін қосу арқылы кіші бағдарлама шақыратын жергілікті қоршаған ортаны бөледі.
Программаны бастап қайтару ортаны қалпына келтіру үшін мынадай операцияларды орындайды және сақтау суб-бағдарламаларды тудырады:

1. Сақталған регистрлер және PSW қоңырау есептеу кіші дәрежесін қалпына келтіру үшін, қайтып тиісті тізілімдер көшіріледі.
2. Бағдарлама шақыру КШС шақыратын программа пайдаланатын жадты қалпына келтіру үшін қалпына келтірілді. Шегеру жолымен мөлшерін кіріс параметрлерінің сақталған жай-күйі және әр түрлі көрсеткіштердің бірі көрсеткіштің кадр шақырылатын программа КСШ шақырылатын кіші қалпына келтіріледі.
3. Шақырылған программаның кадрын қалпына келдіру үшін ДБ динамическая құрастыру ҚК көшіріледі.
4. КҚ қайтару индексі шақырушының бағдарламасының бақылауды жіберу үшін КИ көшіріледі.
Бағдарлама оралғаннан кейін, сондай-ақ мынадай нұсқауларды өткізбей тұрып, ресми параметрлерін нәтижелері ұяшығы нақты параметрлерін жадында жады ұяшықтары шығу параметрлерін көшіріледі. Қалған біздің талқылауға біз бағдарламаны жүзеге асыруға қажетті ақпараттың ешқайсысын жоғалтпас үшін PSW мен тіркелімдерінің сақталуын қамтымаймыз.

Мысал 5.3
5.10 сурет блок-құрылымдалған бағдарламасын пайдалана отырып тұжырымдаманы суреттейді. Меңзер нұсқаулар (МН), кадрдың көрсеткіші (КК) және стек сілтегіш (СС), тиімді қол жеткізуін қамтамасыз ету үшін процессордың тізілімдері сақталады. Бағдарлама арнайы түрлі абстракция басқару, деректер жинау және статикалық бөлу көрсету үшін әзірленген болатын. Негізгі бағдарлама деректердің төрт элементтерді делінген [4] және Табу_maks бағдарламасы шақырады. Суретті бақылау ішіндегі nayti_maks программасы болып табылатын сценарий көрсетеді. Brgt (неғұрлым-кем филиалы), секіру (шартсыз филиалы), brle (аз-сәл-тең филиалы), және CMP (салыстыру) нұсқаулары интуитивті болып табылады. Нұсқаулары жаңарту бірінші дәлел қосу үшін, екінші дәлел сақталған мәнді қосады. Мысалы, нұсқау «қосу (КСШ 1)» «КСШ

= КСШ+ 1.» баламасы болып табылады. Көптеген абстактілі нұсқаулықтар нұсқаулықты жинақталу деңгейіне көшірген уақытта нұсқаулықтың жал,асымен теңестіріледі. Мысалы, «қосу (d[КК], КИ + 2)» нұсқаулығы "қосу (КИ , 2, R1), присвоить(d[КК], R1)." Мен тең келеді. Сол сияқты, оқу операторы оқу үшін операциялық жүйесін іске қосады. «Итеру» нұсқаулығы стек басқаруындағы аргументке кедергі жасап, *КШС*-ні бірінен соң бірін арттырады.



Program main – **негізгі бағдарлама;** integer – **бүтін;** data area– **мәлімет орны;** max – **максимум;** read – **оқу;** call find max – **максимумды табу шақырылым;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** if – **егер;** return – **қайтару;** loop back – **контурды қайтару;** call – **шақыру;** for – **үшін;** void find max – **максимумды табуға жарамсыз;** while – **сол кезде;** program – **бағдарлама;** push – **басу;** save RP – **RP-ны сақтау;** set – **орнату;** create var – **айнымалыларды жасау;** out of loop – **контурдын сыртында;** previous – **алдыңғы;** stack growth – **стек өсуі;** code area – **код орыны;** data area – **мәлімет орыны.**
5.10 сурет. Генерация кода и данных в реализации на основе стека.

Деректер аумағы үш бөлікке бөлінеді: жаһандық айнымалы үшін статикалық ауданы, бағдарламаның басты кадрлары, және шақырылатын find_max кадры. Жахандық айнымалылар массивті m[4] пен айнымалы max *тұрады. Жаһандық айнымалылар статикалық болып бөлінеді: m*[0] →d[0], m[1] → d[1], m[2] → d[2], m[3] → d[3], және max → d[4]. Негізгі бағдарламаның кадрі d[5] мен d[6] арасында орналасқан, ал ал шақырылатын «алмасу» d[7] мен d[9] арасында. Негізгі программа кадрында қайтару көрсеткіші мен динамикалық байланыс жоқ. d[7] және d[8] ұяшықтары басқару элементін негізгі программа, а қайтару үшін қайтару көрсеткіші мен динамикалық байланысты сақтайды.
Негізгі программаның құрамында екі жергілікті айнымалы үшін жад ұяшықтары бар: *i* →d[5] және *j* → d[6]. find_max программасының кадрында құрамында қайтару көзсеткіші (ҚК) → d[7], динамикалық байланыс (ДБ) → d[8] және локалдық айнымалы *i* → d[9] бар. ҚК d[9] ұяшығын көрсетеді, КШС d[10],ал ДБ негізгі программаның кадр базасын көрсетеді, яғни d[5]. ҚК (ҚК) код аумағындағы c[11] ұяшығын көрсетеді, ол орындалатын “halt”командасына тең келеді.
Аудан коды қазіргі орындау процесінің шеңберінде жад орындарды кіру үшін кадр көрсеткішін пайдаланады. [0] және [5] арасындағы нұсқаулары орындау «for» цикліне сәйкес келеді.

[0] нұсқаулық КК көрсеткен жад ұяшығындағы 0 мәнін қосады, бұл «*i* = 0» өрнегінің төмендеггейлі аудармасы болып табылады; [1] нұсқаулық d[КК] мәнін 3 константасымен салыстырады. Осы мәлімдеме орындау PSW бойынша орындалады, және [2] командасы PSW жалаулар пайдаланып, while (әзірге) циклінен шығу үшін [6] операторына өтеді. [6] нұсқаулығы төменгі индекс мәнін *m*[*i*] жадқа екі рет жүгінумен есептейді: біріншісі d[КК] (мәні *i*) мәнін оқиды, ал одан кейін идестелген жадты оқиды. Оқу жүйедегі шақыру деңгейін белсендіреді. [5] нұсқау-

лары қайтып[1] нұсқаулығына – «for» циклінің басқы нүктесі, өту үшін нұсқаулық көрсеткішінен 4-ті шегеріп тастайды.
[6] және [9] ұяшықтары бар коды саласындағы нұсқаулар түрлі көрсеткіштерін орнату және кіші бірінші тапсырмасы көшу үшін пайдаланылады. [6] орналасқан нұсқаулығы, в стеке жоғарғы жағында «КИ + 5» итеріп кезде (D [7]) қайтару меңзерді белгілейді, және КШС 1 ұлғайды. [7] орналасқан нұсқаулығы, КК мәнін стектің жоғарғы жағына (D [8]) итеріп, динамикалық байланыс орнатқан кезде КШС мәні 1 ұлғайды. Регистрлер мәні өзгеріс кезінде сондай-ақ стекке итеріледі. [8]-дағы нұсқаулық КК-ны find_max программасын қол жеткізу үшін КК-ны КШС-ке қосу арқылы орнатады. Шақыратын бағдарламаларды есептеу алғашқы күйін сақтау үшін қажетті бақылау жад ұяшықтарының ығысу; бұл регистрлер, PSW және статикалық сілтемені ғана сақтайтын болса өзгереді. [9] нұсқаулық символы кестеде ығысу іздеу арқылы бірінші санатшасы кіші бағдарлама нұсқаулар find_max үшін бақылауды өтеді: бұл жағдайда ығысу 2-ге тең. [10] нұсқаулығындағы соңғы нұсқаулық жоғары деңгейлі «STOP» нұсқаулығына сәйкес келеді.

[11] нұсқаулығы жадқа жергілікті айналымдарға ұяшық қосу арқылы КШС-ті белгілейді: бұл жағдайда өлшемін 1. D → Max [4] және м [0] → D [0]: беттегі нұсқауларды [12] жоғары деңгейдегі тапсырмасы «Max = м [0]» сәйкес келеді. Мен D [КК] →: беттегі нұсқауларды [13] жоғары деңгейдегі нұсқаулар «I = 1» сәйкес келеді. [14] және [19] ұяшықтары арасындағы нұсқаулары болып табылады цикл сәйкес, сондай-ақ [20] және [22] ұяшықтары арасындағы нұсқаулары find_max рәсімдерді қайтару сәйкес келеді. [14] ұяшықтағы нұсқаулық тұрақты і 3 айнымалыны салыстырады. Ілмектер жүзеге бар ұяшықта нұсқаулары, ал, і айнымалы мән 3 артық болса [15] ұяшығындағы нұсқаулық «STOP» циклінан шығады. [16] ұяшықтағы нұсқаулық m [i] > макс: i → d[КК] жоғары деңгейдегі нұсқаулар *макс* → d[4] және d[d[УК]] → *m* [*i*] сәйкес келеді. [17] ұяшықтағы нұсқаулары басқаша-шартты бір нұқсанды бөлігіне сәйкес келеді және [19] ұяшыққа оператордың басқару нұсқауларды алады. [18] ұяшықтағы нұсқаулар жоғары деңгейдегі нұсқаулар *макс = m* [*i*] тең. [19] ұяшықтағы нұсқаулықтар болып циклдің басына кері өту үшін «КИ» 5 бастап шегеріледі.
Қайтару тәртібіне тастайды көрсеткіштер мынадай тәртіппен орындалады:

1.[20] ұяшықтағы нұсқаулық кадрдың find_max программасында пайдаланылатын жадты босату үшін КК-2 мәнді КШС көшірмелейді.

1.[21] ұяшықтағы нұсқаулық (d[УК-1]) динамикалық нұсқаулығын КК-ге көшіру арқылы негізгі программаның көрсету кадрын қалпына келтіреді.
1.[22] ұяшықтағы нұсқаулық КБ-дегі (d[КШС]) КИ адресті көшіру арқылы басқаруды негізгі программаға кері береді.
Қол жеткізу үшін айнымалы алаптың жады кіру үшін қажет: бірінші индекстелген айнымалы мәнін алу үшін, екінші тиісті массив элементінің жад ұяшық мәнін алу үшін. Бұл ақпаратқа қосымша жад компиляторын қысқаратады, сақтау жолымен тіркелімдерінде индекстелетін айнымалы. Кэш-жады пайдалану, сондай-ақ үстеме шығындар қол жеткізуді жақсартады.

5.5 Параметрлерді жіберуді үлестіру

Ақпарат алмасу тиісті іске асыру шеңберінде параметр өту тетіктерін түріне қарай өзгереді. Мысалы, құны арқылы шақыру үшін ресми параметрдің жад ұяшықта білдіру айырысу мәнін көшіру талап етеді. Сілтеме бойынша шақыру үшін. ресми параметрдің жад ұяшықта нақты параметр базалық мекенжайын сақтау керек.
Бағдарлама аясында ресми көрсеткіштер бойынша нақты шығыс параметрі кеңістік деп аталатын шақыратын бағдарлама параметрлері және жад орналасу үшін кеңістік жады кіріс параметр кеңістігі болып табылады.Шығыс параметрдің кеңістік бағдарлама шақыратын кадрға тиесілі, және кіріс параметр кеңістік бағдарлама деп аталатын кадрға жатады. Алайда, кіріс және шығыс параметрлерінің ауданы бір бірінің үстіне біріктіпіледі, және шақыратын және шақырылатын программалар қол жеткізе алатындай болу керек. Шақырылатын бағдарлама дереу параметр кеңістік енгізгеннен кейін сақталған алғашқы күйін ақпаратты есептеу алғашқы күйін сақтайды. КК сақталған күйі жайындағы ақпаратты көрсетеді. Ресми параметрдің ығысуы программаның көрсету кодексіне қатысты теріс болып табылады. Алайда, қалған жергілікті айнымалы есепке көрсету кодексіне қатысты оң болып табылады.

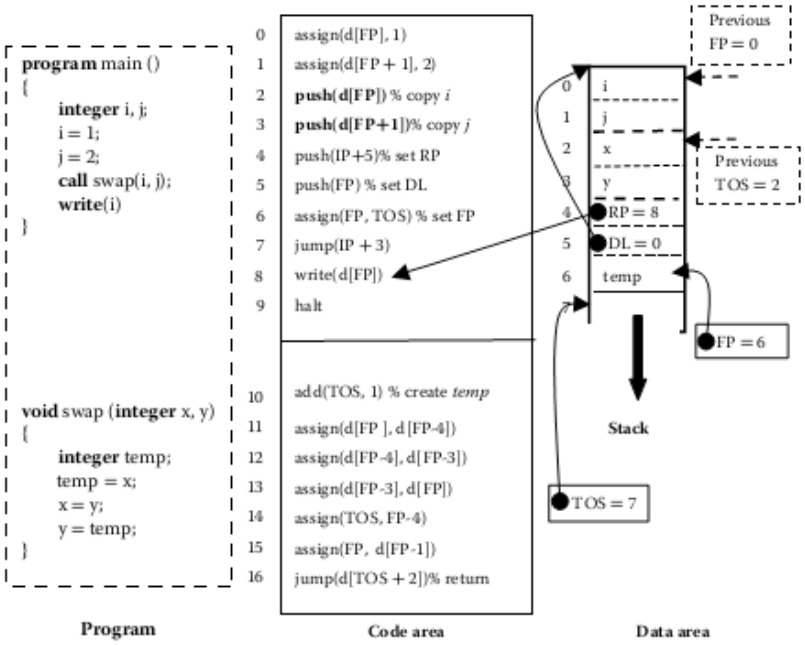
5.5.1 Мағынасы бойынша шақыруды үлестіру

Білдіру нақты параметр мәндерінің құнына шақыру барысында шығыс параметрлерін кеңістікте бағаланады және сақталады. Шақырылған программа PSW-ті сақтайды, және шақыратын бағдарлама тізілімдер және әр түрлі көрсеткіштер өзгерді.Кіретін параметрлерге қол жету жеке ақпараттық объектілер үшін d[КК – ығысу] арқылы және массивті ақпараттқ элементтерге d[КК- база-адрес-ығысу + d[КК + индекс- ығысу]] арқылы жүзеге асырылады, бұл жерде база-адресс массивтағы бірінші

элементтің ығысуы, ал ығысу- индекс ол массивтегі элементке қол жету үшін і айнымалыс арқылы индекстелетін ығысу болып табылады. Үйінді композициялық нысандарына қол жеткізу, сондай-ақ ресми параметр жады объект сілтемені көшіру арқылы жүзеге асырылады.

Мысал 5.4

Сурет 5.11 мәні бойынша шақыру дерексіз орындалуын көрсетеді. Негізгі бағдарлама маңызы бар екі айнымалы і және j тиісті формальды параметрлер x және y көшіреді. Sub-алмасу x және y ресми параметрлері мәндерін өзгертеді. Есептеу нәтижесі бағдарламасында кері алмасу берілмейді.



Program main – **негізгі бағдарлама;** integer – **бүтін;** data area– **мәлімет орны;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** return – **қайтару;** call swap – **шақыру алмастыру;** program – **бағдарлама;** push – **басу;** set – **орнату;** create temp – **уақытша жасау;** previous – **алдыңғы;** stack – **стек;** code area – **код орыны;** writer – **жазушы;** temp – **уақытша;** copy - **көшіру**

Сурет 5.11 Нысанды шақырулар сұлбасы

Қадыр негізгі орналасу d[0] - d[3] ке дейін тұрады, ал шақырылатын кадр программасы d[2]- d[6]дейін, тұрады. D [2] және D [3] жабатын ұяшықтар x және y формальды параметрлеріне сәйкес келеді. [2] және D [3] ұяшықтары негізгі бағдарламамен шығыс параметрінің кеңістігі ретінде қарастырлады, ал шақыртатын процедура «айырбас» кіріс параметр кеңістік ретінде қарастырылады. [4] және [5] ұяшықтарында КК мен ДБ-ның көрсеткіштерін сақтайды: d[4] ұяшығында write(i) (жазу) командасы сақталады, ал d[5] ұяшығында динамикалық байланыс сақталады. «temp» айнымалысы d[6] ұяшығында салыстырылады.

Шақырылатын «swap» (алмасу) процедурасына дейін басқару элементі негізгі программада болады, КК d[0] ұяшығын көрсетеді, КШС көрсеткіші d[2] ұяшығын көрсетеді. «swap» программасының КК көрсеткіші d[6] ұяшығын көрсетеді. D[КК-1] динамикалық байланыс ұясы болып табылады, және d[ҚК-2] келесі нұсқаулықтың кері-мекенін сақталады, ол негізгі бағдарламада орындалуы керек. Статикалақ байланыс болмайды.

Кеңістік шығыс көшірмесі параметрі үшін параметрлерді жіберу КК көрсеткішін қолданады. [2] және [3] ұяшықтарындағы *иперу* D [КК + 1] және *иперу (*D [КК]) нұсқаларын пайдална отырып нақты параметрлер мәнін шығыс параметрлер ұяшықтарына d[КК] и d[КК+1] көшіреді негізгі бағдарламасына қалған нұсқаулар. түрлі көрсеткіштерді орнату үшін және кіші бағдарлама алмасу көшу,пайдаланылады, олар мысалда 5.3 талқыланды.

d[ҚК-4] және d[ҚК-3] жазбалары x және y формальды параметрлеріне қол жеткізу үшін тиісінше қолданылады.4 және 3 ығысуы белгілі мөлшерін сақталған ақпарат жағдайын мөлшері мен кіріс параметрлері арқылы есептеледі. [11] ұяшығындағы нұсқаулық көшіреді де, мәні x айнымалы мәнін айналмалы шаблонның жад ұяшығына көшіреді. [12] ұяшығындағы нұсқаулық айналмалы y мәнін x-ке, ал [13] ұяшығындағы нұсқаулық шаблон айналымындағы мәнді y жад ұяшығына көшіреді. [14] және [16] ұяшықтары арасындағы үш ұяшық басқару элементін кері шақырылған программаға беру нұсқалығы болып табылады, бұны біз 5.3 мысалда қарастырдық. [16] нұсқаулығында қайтару көрсеткіш ұяшығын кіріс параметр ұяшығының өлшемін білген жағдайда, компиляция кезінде анықтауға болады.

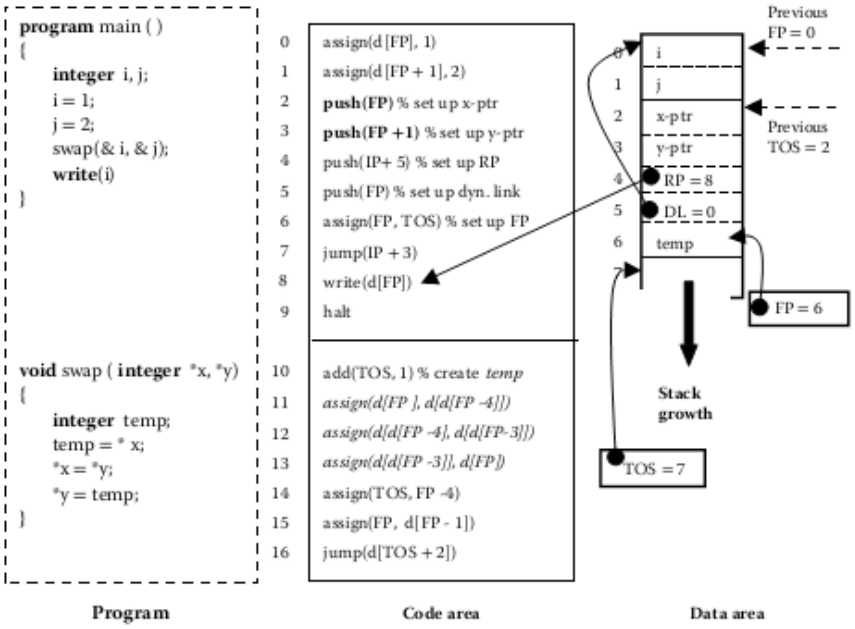
5.5.2 Сілтеме арқылы шақыруды жүзеге асыру

Сілтеме бойынша шақыру кезінде шақырылатын программа ақпараттық объектінің негізгі адресін шығыс параметр кеңістігінде сақтайды, ол шақырылған программа үшін нақты кіріс параметрі кеңістігі болып

табылады. Мөлшер сілтемесіне қол жету d[d [КК – параметр-ығысу] + индекс-ығысу] арқылы жүзеге асады, бұл жерде ығысу-параметрі бұл шақырылатын программаның нақты кіріс параметрінің негізгі адресінің ығысуы болып табылады, ал индекс ығысуы шақырылған программаның кадрындағы жергілікті айнымалының индексінің ығысуы болып табылады. Индексі айнымалы жергілікті айнымалы басқа болса, екінші элементі тиісті, механизмін шешуге алмастыра алады. Қатып келу кезін-де шақырылған программаның есебінен айтарлықтай еш қимыл жасап қажет емес, себебі сілтеме бойынша шақыру физикалық айналымға үздіксіз мутацияға мүмкіндік береді. Сондықтан анық түрде қандай да болсын нәтижелерді жіберіп қажет емес.

Мысал 5.5

5.12-суретте x және y нақты көрсеткіштердің жад ұяшықтарында сәйкесінше *i* және *j* айналымдардың жад ұяшығының адресі сақталады. x және y *i* және *j-тің нақты параметрлерін сақтайтындықтан параметрлерді* [2] ұяшығына жіберу r-мәнінің орнына d[КК], КК жад адресін шығарады, ал [2] ұяшығына жіберу r-мәнінің орнына d[КК+1], КК +1 жад адресін шығарады.

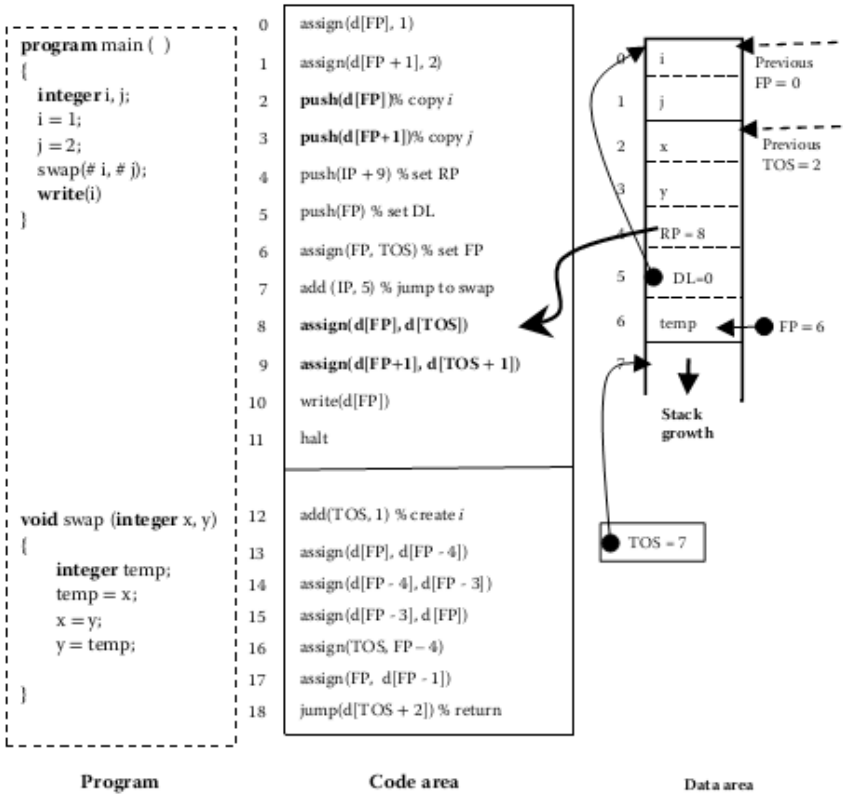


Program main – **негізгі бағдарлама;** integer – **бүтін;** data area– **мәлімет орны;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** swap – **алмастыру;** program – **бағдарлама;** push – **басу;** set up– **орнату;** dyn. Link – **динамикалық байланыс;** create temp – **уақытша жасау;** previous – **алдыңғы;** stack growth – **стек өсуі;** code area – **код орыны;** write – **жазу;** temp – **уақытша;** void swap – **жарамсыз алмасу;** integer temp – **уақытша бүтін**
СУРЕТ 5.12 Сілтеме бойынша шақыруды жүзеге асыру схемасы

[11], [12] және [13] ұяшықтары нұсқаулары нақты параметрге қол жеткізу үшін формалды х және у параметрлерін қолданады. Олар жадқа екі сұранысқа иеленеді: айнымалы і нақты параметр мәнін кіру үшін d[d [УК-4]] , және айнымалы *j* нақты параметр мәнін кіру үшін d[d[УК-3]]. d[d [УК-4]] нақты параметр і жад мекенжайын қамтамасыз екенін, ал d[d[УК-3]] нақты параметр *j* жад мекенжайын қамтамасыз екенін ескеріңіз. Бұрын талқыланған басқа нұсқалар нәтижесә бойынша шақыруға ұқсайды.

5.5.3 Нәтиженің мағынасы бойынша шақыруды үлестіру

Нәтижесінде мәні бойынша шақыру барысында программа нақты параметрлері мен сақтау шығыс параметрлерін кеңістікте тиісті жад ұяшықтарының нәтижесінде мәні үшін өрнектердің мәні бағаланып шақырады. Программа аяқталғаннан кейін, нәтижесі, қажет болған жағдайда, ұяшықтың тиісті айнымалы қоңырау жоспарлы кеңістіктен шығыс параметр қайтып көшіріледі. Кеңістік опция шығыс шақыру және программа кеңістігі қолданылатын параметрлері енгізілген сәттен бастап, нәтижесі автоматты шақырушының программасына қайтып беріледі.



Program main – **негізгі бағдарлама;** integer – **бүтін;** data area– **мәлімет орны;** assign – **тапсыру;** add – **қосу;** jump – **секіру;** halt – **тоқтату;** swap – **алмастыру;** program – **бағдарлама;** push – **басу;** set – **орнату;** create – **жасау;** previous – **алдыңғы;** stack growth – **стек өсуі;** code area – **код орыны;** write – **жазу;** temp – **уақытша;** void swap – **жарамсыз алмасу;** integer temp – **уақытша бүтін;** return – **қайтару;** copy - **көшіру**

5.13 сурет. Мән нәтижесінде шақыруды жүзеге асыру схема

Нәтижесі бойынша шақыру және ресми параметрлер жергілікті айнымалылар ретінде қарастырылады, және жеке ақпараттық объектілер үшін d[КК – ығысу] арқылы қол жетімді, ал массивті элементтер үшін

d[КК- базовый-адрес-ығысу + d[КК + индекс-ығысу]] арқылы қол жетімді. Бұл жерде ығысу-параметрі бұл шақырылатын программаның нақты кіріс параметрінің негізгі адресінің ығысуы болып табылады, ал индекс ығысуы шақырылған программаның кадрындағы жергілікті айнымалының индексінің ығысуы болып табылады.

Мысал 5.6

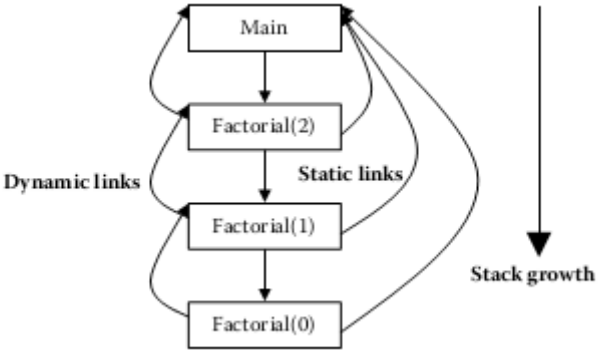
5.13-суретте көрсетілген, сол бағдарлама ауысуына параметрді шақыру мәні бойынша нәтиже. Нақты параметрлері мәні бойынша шақыру «#» таңбасы көмегімен таңбаланған. Деректер облысы мәні бойынша шақырудағыдай болып табылады. Код аймағында тағы екі косымша нұсқаулар бар: [8] және с [9] ұяшықтарындағы нұсқаулық нәтижелерді шығыс параметрлер кеңістігінен х және у айнымалыларын нақты жад ұяшығына көшіреді. Бақылау аймақ кодын қайтарылады кейін, ресми параметр нәтижесі келесі жоғары деңгейдегі тапсырмасы бұрын нақты жад орынға қайтып көшіріледі. Бұл шығыс параметр кеңістігін бірнеше рет пайдаланылатынына назар аударыңыз, әртүрлі программалар қайта-қайта шақырулар арқылы белгілі айнымалылар суреті ретінде пайдаланылуы мүмкін емес екенін ескеріңіз. Барлық басқа нұсқаулар 5.5.1 бөлімінде талқыланды және шақыру мәні ұқсас болды.

5.6 Рекрусивті процедуралардың төмендеңгейлі мінез-құлқы

Рекурсивті процедура әр рекурсивті шақыру үшін жаңа енген белсендіруді тудырады. Сол орын кодын аймағын және статикалық сілтеме кадр аясында мекенжайы әрбір дананың балл оралу рекурсивная процедураның әр данасы рекурсивті рәсімі енгізілген, оған сәйкес, сол тәртіппен жақтауын көрсетеді. Алайда, динамикалық сатыдағы байланыс mго (m > 2) базалық кадрдың (m − 1) мекен-жайы көрсетілуге ші данасын және рекурсивті тәртібін бірінші сатыдағы рекурсивті тәртібін туындаған жақтау тәртібін көрсетеді.

Мысал 5.7

Рекурсивті тәртіппен статистикалық және динамикалық шақыру қарым-қатынастар арасындағы айырмашылық факторлық кіші шақырушының тәртібі үлгіні көрсете отырып, суретте 5.14 түсіндіріледі – факториалды бағдарлама- бұл негізгі бағдарлама арқылы шақырылатын бағдарлама.



Main – **Негізгі;** Factorial – **факториалдық;** dynamic links – **динамикалық байланыстар;** static links – **статикалық байланыстар;** stack growth – **стек өсуі**

5.14 сурет. Рекрусивті процедуралардың динамикалық және статикалық байланысы

Бағдарламаның негізгі факторлық (2) болып табылады. Факторлық (2) (1) факторлықты шақырады және Факториалық (1) мәнді күтіп өзін тоқтата тұрады. Динамикалық сілтеме факторлық (1) жақтау базасы Факториалық (2) басталғанын көрсетеді. Сол сияқты, факторлық (1) (0) факторлық тудырады. Факториалық үшін динамикалық сілтеме тәртібі шақыруды жақтау (0) факторлық базалық жақтауын (1) көрсетеді. Алайда, факторлық (1) факторлық (2) және барлық факторлық шақыру салдарынан статикалық аясында (0) факторлық бағдарламасы енгізілген, оған сәйкес бағдарламаның, негізгі базалық жақтауын көрсетеді.

5.7 Ерекшеліктер өңдегңшінің іске асуы

Ерекшелік өңдегіштері көшу амалдағыштарды пайдаланып орындалады. Бағдарлама ерекшелік өңдегіштер бар немесе шақырылушының рәсімінде жылғы оператор орындаудан қайтарңан кезде, «табысты аяқтауды.» туын орнату нұсқаулық кез келген қоспағанда сәтті орындалған болса, онда бақылау оператор филиалы пайдаланып, ерекше өідеушіден кейін мәлімдемесінде барады. Әйтпесе, бақылау ерекшелік өңдеушілерді арқылы өтеді. Ол әрбір ерекше өідеуші шарттарын тексереді және шарт шын болса, тәртіппен тиісті ұстанымын ұялы нұсқауларды таңдайды. Ерекшелік мәлімделген болса, онда активтендіру жазба орындалып тұрған бағдарлама орнын сақтайды жақтау тәртіппен түсу болып табылады. Ерекшелік өідеуінен кейін, келесі ерекше шартты тек-

серу үшін басқару элементі кері қайтарылад.
Ерекшелік өңдегіші сәтті орындалды болса, онда бақылау қазіргі уақытта орындалып жатқан программада қалады. Әйтпесе ерекшелік өңдеуінің ешқайсысы ерекшелік жағдайын сәйкес келмесе, онда салалық нұсқаулық ретпен оралу үшін бақылауды алады, айрықшалықтарды тексеріліп шақырушының бақылау программасына қайтару құсбелгісін қойып, процесс қайталанады.
Жүзеге асыру үшін тағы бір тәсіл ерекшелік стек деп аталатын жеке стек ерекшелік қамтамасыз ету болып табылады. Бірінші нұсқаулық ерекше өңдеу жолдауы және жоғарғы жақтауын қосылу орындау стеке итеріледі. ерекшелік өңдеуі табысты орындалғаннан кейін, табысты ерекшелік өңдеушілерді жоғарыда жақтаудың аясында барлық стек бақылау жойылады.

ҚЫСҚАША КОРЫТЫНДЫ

Бұл тарауда біз императивті бағдарламалау тілдерінің абстрактілі моделін іске асыруды оқыдық. Нақты іске асыру егжей-тегжейлі болып табылады. Алайда, бұл дебат бізге жақсы іске асыруды түсіну және деректер, аралық деңгейде абстракция басқару әр түрлі қол жеткізуге мүмкіндік береді.
Бұл тарауда, ауданда тұрғысынан деректер мен нұсқаулары сілтегіш тізіліміне басқаруды абстрактілі машинаның және оның мінез-құлық түсінігі талқыладық. Деректер ауданы бірінші класты заттарды қолдау, тілдерді қоспағанда, тіркелген кодты пайдаланып, үздіксіз өзгереді. статикалық өріс, бақылау дестесін және yymeler: деректер ауданы үш бөлікке бөлінген. Статикалық облысы компиляция кезінде бекітіледі және орындау кезінде ұлғаймайды немесе азаяды.Стек және үйменің орындау кезінде өседі және қысу мүмкін. Стек сызықтық өсуде. Үйінді - барлық кіші және деректер ұяшықтарға көрінетін еске жалпы ауданы кез келген уақытта талап ету бойынша бөлінуі мүмкін. Көрсеткіштер тізбектерінің арқылы байланыстырылған үйіндісіне дене жасушалары логикалық деректер құрылымы.
Статикалық бөлу жад ұяшықтарының қол жеткізу кез-келген меңзерді пайдаланбай тікелей жүзеге асырылады, сондықтан компиляция кезінде жады бөлетін тұрақты жады бөлу схемасы болып табылады. Статикалық бөлу орындау кезінде жады өзгертпей картаға негізгі жады талап ақпарат объектілері үшін пайдаланылады. Жаһандық және статикалық айнымалылар статикалық бөліндіге бөлінеді. Олар жарияланған бағдарлама бірлік немесе бірлікте шектеулі қызмет ету мерзімі динамикалық айнымалылар мен деректер нысандар жергілікті, стек бақылау бойынша

бөлінеді. Рекурсивті деректер құрылымдар мен динамикалық деректер нысандардың қызмет көрсету мерзімі бағдарлама блоктардан ұзағырақ, және олар үйінді бойынша анықталады. Логикалық бақылау диаграммасы абстрактілі басқару деңгейі нұсқауларға жоғары деңгейдегі абстракцияның аудару үшін пайдаланылады.Мұндай brlt, breq, brne, brgt, brge және brne сияқты шартты филиалы тапсырмасы, кейін шартты білдіру бас тарту, Шартты сөйлемнің құрылысына <қалғанының-мәлімдемесінде> беруге басқару және итерациялық конструкцияларын циклын шығу үшін пайдаланылады.

Статикалық асыру статикалық бөлу және деректерге тікелей қол жету үшін пайдаланады. Келесі орындалатын нұсқау мекенжайын жадын сақтау және аталатын программаны бақылау үшін филиал нұсқауды пайдалана көшу жасауға процедураны шақырады. Барлық нұсқауларды аяқтағаннан кейін, аталатын программалар келесі тапсырмасына қайтып беруге бақылау сілтегіш тапсырмасы сілтегіш қайтару үшін мәнді тағайындайтын программаларды шақырады.

Стек негізделген іске асыру әрбір программаны шақыру үшін бірнеше көрсеткіштер, және жеке жақтауларды пайдаланады. Кадр көрсеткіші ығысумен бірге жадқа немесе деректер объектінің жергілікті айнымалыны кіру үшін пайдаланылады. Динамикалық сілтеме программа аяқталғаннан кейін шақырушының программа кадрға қайтып алу үшін пайдаланылады. Статикалық сілтеме немесе дисплей регистрлер жергілікті емес айнымалылар кіру үшін пайдаланылады. Программа параметрлерін хабарласпас бұрын программа енгізу параметрлерін бос орын болып шығыс параметр кеңістікте беріледі. PSW программа шақырушылар индекстер мен статус сөздер сақталады, меңзер шақырылған программалардың кадрына қол жеткізу үшін өзгереді, ал бақылау шақырылған программаның бірінші нұсқаулығына өтеді. Программа аяқталғаннан кейін, кері процесс шақырушының регламентін, кейін қайта тапсырмасы параметрлерін және аудару бақылау трансмиссиялық тетігінің байланысты, шақырушының бағдарламасының ортасын қалпына келтіру үшін жіберу параметрлерін пайдаланылады. Үйіндісіне таратылатын композитті нысанды, ортақ пайдалану үшін, үйінді деректер объектісіне қол жетуге мүмкіндік беретін мәні бойынша қоңырау пайдаланыңыз. Сілтеме бойынша шақыру кезінде, нақты параметр сілтегіш ұяшықтың ресми параметрі сақталады. Нақты параметр жад ұяшыққа қатынау үшін бірінші, R-мәнге қол жеткізу, екінші R мәнін табу үшін екі жад кірме талап етеді. Нақты параметрлері нәтижелері деп аталатын параметр енгізу параметр аймақ болып аумақты, шығатын мән көшірмелерімен шақырыңыз. программа оралғаннан кейін, шығыс

параметр саласындағы нәтижелері нақты айнымалы жад ұяшыққа қайтып көшіріледі.
Рекурсивті функциялары әр шақыруға жаңа кадрды жасайды.. Статикалық және динамикалық байланыс сілтеме түрлі жолдармен рекурсивті функцияны көрсетеді: Шақыру базалық кадрға динамикалық сілтеме балл, және қоңыраулар статикалық қосылу ендірілген рекурсивті функция оған сәйкес, базалық жақтау тәртібін көрсетеді.
Ерекшелік өңдейтін екі әдістің бірін пайдаланып жүзеге асырылады: (1) программа кадларында ерекшелік өңдеу процедураларын орынды сақтау немесе (2) жеке ерекшелік стек. Бірінші әдіс, егер барлық өңдеуіш ерекшеліктер ақау берсе, онда басқару элементі кері шақырылады, және процесс қайталанады. Екінші әдісте егер өңдеуші ерекшеліктер сәтті орындалатын болса, онда жоғары табысты кадрдың барлық мамандары стектен ерекшелігінен жойылады.

5.9 Бағалау

5.9.1 Тұжырымдамалар мен анықтаулар

...Абстрактілі нұсқаулар; абстрактілі машина; белсенді кеңістігі; мәні бойынша шақыру; сілтеме бойынша шақыру; мән нәтижесі бойынша шақыру; қала коды; бақылау стек; деректер ауданы; дисплей динамикалық сілтемені тіркейді; ерекшелікті өңдеу; стек ерекшелік кадр; кадр меңзері; байламы; күтіп ауданы; рекурсивті деректер құрылымын; рекурсивті тәртібі; Индекс оралу; стек негізделген бөлу; Статикалық тарату; SECD машина; статикалық сілтеме; стек нұсқауышы; із стек, WAM (Уоррен Abstract Machine).

...5.9.2 Тапсырманы шешу

1.X=Y+ 2 + 3 + 5 есептілігін X = Y + + 3 2 + 5 жүзеге асыру үшін төмен деңгейлі код ретін жазыңыз.

2.Оператордың іске асыру үшін бақылау логика суретін салыңыз, содан кейін оператордың параметрін іске асыру үшін абстрактілі төмен деңгейлі кодты жазу.

3. Статикалық бөлу пайдалана отырып, Fortran 66 бағдарламасы сияқты, коды ауданы және келесі деректер аймағын жазу.

```
PROGRAM MAIN
INTEGER I, J, K[20]
COMMON/B1/K[20]
DO 20 I = 1, 20, 1
20  READ(K[I])
CALL SORT
DO 30 I = 1, 20, 1
30 WRITE(K[I])
END
```

```
SUBROUTINE SORT
INTEGER M[20], I, J
COMMON/B1/M[20]
DO 10 I = 1, 20, 1
J = 20 - I + 1
10 CALL MAX
RETURN
```

```
SUBROUTINE MAX
INTEGER M[20], I, J, K
COMMON/B1/M[20]
DO 30 I = 1, J, 1
IF (M[I].GT. M[I + 1])
    K = M[I]
    M[I] = M[I + 1]
    M[I + 1] = K
ENDIF
RETURN
```

Program main – **негізгі бағдарлама;** integer – **бүтін;** common – **жалпы;** read – **оқу;** call sort – **сұрыптау шақыру;** do – **орындау;** write – **жазу;** end – **соңы;** subroutine sort – **кіші әдетті сұрыптау;** call max – **максимумды шақыру;** return – **қайтару;** subroutine max – **кіші әдеттің максимумы;** if – **егер;** end if – **соңы егер**

4. (2) факториалын табады функциясын жазу (N) және факториалын есептеу үшін бақылау стек көрсету, бақылау шақыруының болғанда, факториалын (0) есептейді.
5.Келесі блок-құрылымдалған тілі үшін стек негізделген бөлуді пайдалана отырып келесі бағдарлама бойынша код аймағы мен деректер аймағын жазу. Блок-құрылымдалған тілі трансмиссия параметрлерін үш түрін қолдайды деп есептейік: мәні бойынша шақыру, сілтеме бойынша шақыру және нәтиже мәні бойынша шақыру. Нақты параметрде сілтеме бойынша шақыру «&» деп , ал нақты параметрде мәні бойынша шақыру «#» белгізімен белгіленеді. Шақырылған программада нақты параметрлер алдында нақты параметрдәі мәніні қол жету «*» белгісімен белгіленеді.
6. Көпіршікті сұрыптау үшін қарапайым бағдарламаны жазу, бұнда онда find-Max айырбас көрші мәндерді пайдалану орнына уақытша max сақтау рәсім болып табылады. Ол сілтеме бойынша шақыруды пайдаланып жиымын тасымалдайды. Деректер аймағын және кодын көрсетіп, және код параметрлерін таңдаңыз. Бақылау негізгі бағдарламада кезде, және ол процесс find_max болғанда көрсеткіштер анық анықтау. Массив өлшемі 5 делік.

5.9.3 Толық жауап

7. Статикалық бөлудің негізгі кемшіліктері қандай? Статикалық бөлу тұратын гибридті тізбегін бөлу қалай негізінде в стеке тарату және үймелер ретінде шешеді?

8.Стек негізінде іске асыруға кадрларды рөлін түсіндіріңіз. Неліктен ең бастысы жергілікті кадрмен шектелген көрініст сақтау керек? Түсіндіріңіз.
9.Кіріс және шығыс параметрлерін араластыру артықшылықтары қандай? Схемаларын пайдалану арқылы түсіндіріңіз.
10. Егер сізде стек және үйіндіге негізделген басқару болса, параметрді ауыстыру механизмін әзірлеңіз. Бұл шақыру программаларын орындау кезінде тек оқуға арналған сілтеме бойынша шақыру ретінде жұмыс жасайды. мән қазіргі деп аталатын рәсімін орындау кезінде өзгерсе нақты параметр жаңарту шақырды рәсімін соңында нақты параметр қандай да бір өзгерістер оқиды және тасымалдайды. Сілтемені бойынша шақыру мен нәтиже бойынша шақырумен салыстырғанда, осы схеманың артықшылықтары мен кемшіліктері талқылаңыз.
11. Итератор жүзеге асыру үшін әдебиетті қарап, итераторды жүзеге асырудың түрлі әдістері мен мәселелері жазыңыз.
1.Ерекшелік өңдегіштері дегеніміз не және стек негізінде олармен қалай жұмыс жасауғу болады?
2.Ada-дан шығатын бірнеше шартты дизайнын таңдау үшін бақылау логика суретін салыңыз, және дизайн үшін төмен деңгейлі кодты жазыңыз.

Қосымша әдебиет

1.Брент Р. «Динамикалық жад бөлу үшін бірінші тиісті Стратегиясын тиімді жүзеге асыру». *Бағдарламалау тілдері және жүйелер ACM мәмілелер*. 11(3). Июль 1989 года. 388-403.
2.Диль, Стефан, Хартель, Питер и Сештофт, Питер. «Абстрактілі машинаның іске асыру үшін тілдерді программалау». *Компьютерлік жүйелер болашақ ұрпақ*, 16.2000. 739-751.
3.Хансон, Дэвид Р. «Өмір бойы жедел жады бөлу және қозғалысы.» *ОпытиПрактикаПрограммного Қамтамасыз Ету*, 20(1). Январь 1990 года. 5-12.
4.Джонс, Ричард, Хоскинг, Антоний и Мосс, Элиот. *Күл-қоқыс коллекциялар анықтамалығы* CRC Пресс / Группа Тейлор и Фрэнсис. 2012. 481.
5.Джонс,РичардиЛинс,РафаэльД. *Коллекция Мусора: Автоматты Динамикалық жады басқару алгоритмдері*.Нью-Йорк:ДжонУайли.1996.
6.Уилсон, Пол Р., Джонстон, Марк С., Нили, Майкл и Боулз, Дэвид. «Динамикалық сақтау тарату: Қарау және сыни талдау.» *Жұмыс IWM Естелік басқару жөніндегі халықаралық семинар '95 материалдары*. Спрингер Ферлаг. 1 – 116.

Қосымша I
Тілдердегі қолдайтын парадигмалар

C^T - міндет негіздеріндегі конкуренттік желі; C^P - деректер параллелизмі: C^D – таратылды; C^S - синхронды параллелизма; E - Ic-шара негізінде; F – функционалдық; I – императивтік; L – логикалық; M – мультимедиалық; O^F - жалпақ заттармен бағдарланған нысан; V – визуалдық; W – веб.

Ескертпе: тілдердің кейбірлері ешқандай айқын стандартталған нұсқалары жоқ. **Соңғы компилятор дамыту негізінде, кейбір күндер шамамен алынған.**

Тіл	Түзету	Парадигмалар
Algol W	1966	I
ALICE	2000	I, E, M, V, білім беру мультимедиалық тіл
Ada	2012	I, C^T, O^I
C	1999	I, C^T (кітапхана), C^P (ауыткуылар)
C++	2011	I, C^T (кітапхана), O^I, V (ауыткуылар)
C#	2010	I, E, C^P, M, O^I, V, W (.Net-пен)
Chapel	Дамуда	I, C^P, O^I, (жаппай параллель есептеу үшін)
Clair	2009	I, F, L, O^I, W
Clojure	2011	F, C^T, сценарийлер тілі
COBOL	2002	I, O^I, бизнес бағдарлама тілі
ECL[1]PS[e]	1997	L, тұрақты логикалық бағдарлама тілі
Emerald	1994	I, C^D, O^F, таратылған есептеу тілі
Estrel	1991	I, C^S, синхронды аппараттық модельдеу тілі
F#	2005	I, F, O^I, E, V, W
FORTRAN	2008	I, O^I, C^P (HPF ретінде ауыткуылар)

Тіл	Түзету	Парадигмалар
Haskell	2010	F, C^T
Java	Үздіксіз	I, E, O^I, C^T, M, V, W
Javascript	1995	I, E, O^P, W, веб сценарий тілі
Lisp family	1994	I (шектеулі), F (негізгі), O^I (ауытқуылар), C (шектеулі)
Lua	2006	I, F, O, C^T, W (ойын сценарий тілі)
ML	1998	I (шектеулі қолдау), F (негізгі)
Modula-3	1991	I, O^I, C^T
Oz/Mozart	2007	I, F, L, O^I, C^T
Perl	Әр қалай	I, F (шектеулі), O^I, сценарийлер тілі
PhP	2004	I, O^I, W, веб сценарий тілі
Prolog	Әр қалай	I, O^I (кітапхана), V (ауытқуылар), C^T (ауытқуылар)
Python	Үздіксіз	I, F, O^I, сценарийді жасауға арналған тіл
Ruby	2012	I, F, O^I, интеграцияланған мультипарадигма
Scala /EScala	2012	I, F, O^I, C^T, интеграцияланған мультипарадигма
SMIL	2008	I, M, W (веб есептеуге)
VRML /X3D	1996/2005	I, M, W, 3D модельдеу тілі
X_{10}	Дамуда	I, C^P, O^P, жаппай параллель есептеулерді үшін
XML/HTML based	Әр қалай	M, V, W, веб негізіндегі есептеулер үшін

Қосымша II
Абстракцияның деректер жиынтығы

1. Отыратын деректер тұлғасы
2. Отырмайтын деректер тұлғасы
3. Тәуелсіз айқын көрсеткіштер / анықтамалар
4. Қосымша дәлдігімен негізгі түрлері
5. Жолдар
6. Массивтер / индекстелген реттіктер
7. Хеш кесте / карталар / негізгі құндылық кесте
8. Луын / жазба / құрылым аттары
9. Реттік түрі
10. Жиынтықтар
11. Рекурсивті деректер түрлері / тізімдер
12. Әмбебап полиморфизм / түр_тарауы
13. Заттар
14. Класс және мұра (бір немесе бірнеше мұралар)
15. Модульдер / пакет/ есім аясы

• Кіріктірілген / кіріктірілген кітапханасында; A - сілтеме ретінде қолданылатын лақап аттар; Δ - шектеулі; D - қарқынды терілуі; L – кітапхана арқылы; M – медиа заттар; R – көрсеткіштің орнында анықтама; ρ - диапазон; S - басқа деректер құрылымымен үлгіленуі; V - тілдік қолдаудың ауытқуы; X – орында жоқтығы

Languages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ALICE	▪	X	X	X	▪	▪	X	X	X	X	▪	X	M	▪	X
Ada 2005	▪	Δ	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
C	▪	X	▪	▪	▪	▪	X	▪	▪	S	▪	X	X	X	▪
C++	▪	X	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
C#	▪	X	▪	Δ	▪	▪	▪	▪	▪	S	▪	▪	▪	▪	W
Chapel	▪	X	X	▪	▪	▪	▪	▪	▪	▪	X	▪	▪	▪	▪
Claire	▪	▪	X	X	▪	▪	X	▪	ρ	▪	▪	▪	▪	▪	▪
Clojure	L	▪	R	L	▪	▪	▪	▪	ρ	▪	▪	▪	▪	▪	▪
ECLiPSe	X	▪	▪	D	▪	▪	▪	▪	S	▪	▪	▪	▪	▪	▪
Emerald	▪	▪	X	X	▪	▪	X	▪	▪	▪	▪	▪	▪	X	▪
Estrel	▪	X	X	X	X	X	X	X	X	X	X	X	X	X	▪
F#	▪	▪	R	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
Fortran 2008	▪	X	▪	▪	▪	▪	X	X	▪	X	▪	▪	▪	▪	▪
Haskell	X	▪	X	▪	▪	▪	S	▪	▪	▪	▪	▪	▪	▪	▪
Java	▪	X	R	▪	▪	▪	▪	X	▪	▪	S	▪	▪	▪	▪
Javascript	▪	Δ	X	D	▪	▪	▪	X	S	X	X	X	▪	X	X
Lisp	▪	▪	X	D	▪	▪	▪	S	V	S	▪	▪	V	V	X
Lua	▪	▪	X	X	▪	▪	▪	X	S	S	▪	V	▪	S	▪
ML	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	X	▪
Modula-3	▪	X	▪	▪	▪	▪		▪	▪	▪	▪	▪	▪	▪	▪
Perl	▪	X	L	L	▪	▪	▪	S	▪	S	S	▪	▪	▪	▪
PhP	▪	X	X	X	▪	▪	▪	X	X	X	X	L	▪	▪	▪
Python	▪	▪	A	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
Ruby	▪	▪	R	D	▪	▪	▪	X	ρ	▪	ρ	▪	▪	▪	▪
Scala	▪	▪	X	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
SMIL	▪	X	X	X	▪	X	X	X	X	X	X	X	M	X	▪
X10	▪	▪	X	▪	▪	▪	X	▪	X	L	▪	▪	▪	▪	▪

Languages - тілдер

Қосымша III

1. Destructive assignment-деструктивті иелендіру
2. Multiple/parallel assignment – көп түрлі/параллельді иелендіру
3. If-then-else – Егер-содан кейін- егер-онда-басқа
4. Case/switch statement – жағдай/қосу иелендірулері
5. For loop/definite iteration –түйінге/нақты итерация
6. While-loop/indefinite loop – аралық түйін/тәуелсіз түйін
7. Do-while (repeat-until) loop – орындау-аралық (қайталау-дейін) түйіні
8. Iterators - итераторлар
9. Functions/subprograms-функциялар/қосымша бағдарламалар
10. Guards-тежеуіштер
11. Exception handling – ерекшеліктерді қабылдау
12. User-defined threads/tasks – пайдалуншы анықтаған қауіптер/тапсырмалар
13. Monitors/mutual exclusion using locks/protected objects/synchronized methods – синхрондаған әдістерді /объектілерді қорғалған тежеуіштерді ұолдана отырып мониторинг жүргізу/өзара тізімнен шығару
14. Other concurrency constructs – басқа паралелльді құрылымдар
15. Distributed computing – таралған есептеу

—built-in support;-қоса орнатылған қолдау
C—cluster-based computing; - кластерлерге негізделген есептеу
D—general loop and exit; - негізгі түйін және шығыс
L—interfaced with library; - кітапхана интерфейсі бар
LT—temporal loop; - уақытша түйін
M—external message passing library like PVM or MPI; - PVM немесе MPI кітапханалары сияқты қосымша хабарлама
N—not part of current specification; - ағымдағы техникалық сипаттамалардың бөлігі емес
R—simulated using tail-recursion; - соңындағы рекурсияны қолдануды

имитациялау
S—can be simulated using existing constructs; - әрекеттегі келісім-шарттарды арқылы имитацияланады
T—as a trait; - соңғы
U—until-loop; -түйінге дейін
V—language - тіл
variations support; - қолдау түрлері
W—web programming support; - веб бағдарламалық қолдау
X—not supported in standards – стандартпен қолдау таппаған

	Control Abstractions														
Language	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ALICE	▪	X	▪	X	▪	▪	X	X	▪	X	X	▪	X	▪	X
Ada	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	V
C	▪	X	▪	▪	▪	▪	▪	X	▪	X	X	L	L	V	M
C++	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	▪	L	L	X	X
C#	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
Chapel	▪	X	▪	▪	▪	▪	▪	▪	▪	X	N	▪	▪	▪	C
Claire	▪	X	▪	▪	▪	▪	U	▪	▪	X	▪	X	X	▪	X
Clojure	▪	▪	▪	▪	▪	▪	X	▪	▪	X	▪	▪	▪	▪	▪
ECLiPs[e]	▪	▪	▪	X	▪	R	R	▪	▪	X	Δ	Δ	▪	X	▪
Emerald	▪	X	▪	X	▪	▪	D	X	▪	X	X	▪	▪	X	▪
Estrel	▪	X	▪	▪	▪	▪	▪	X	▪	X	▪	X	X	▪	X
F# 3.0	▪	X	▪	▪	▪	▪		▪	▪	X	▪	▪	▪	▪	W
Fortran 2008	▪	X	▪	▪	D	D	D	X	▪	X	Δ	V	V	▪	M
Haskell	X	X	▪	▪	▪	S	S	S	▪	▪	▪	▪	▪	▪	▪
Java	▪	Δ	▪	▪	▪	▪	▪	▪	▪	X	▪	▪	▪	X	▪
Javascript	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	▪	X	X	X	W
Lisp	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	▪	X	X	▪	X
Lua	▪	▪	▪	X	▪	▪	▪	▪	▪	X	▪	L	L	▪	W
ML	▪	▪	▪	▪	S	▪	X	▪	▪	V	▪	▪	▪	▪	▪
Modula-3	▪	X	▪	▪	▪	▪	▪	X	▪	X	▪	▪	▪	X	▪
Perl	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	▪	▪	X	▪	V
PhP	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	▪	X	X	X	W
Python	▪	▪	▪	▪	▪	▪	X	▪	▪	X	▪	▪	▪	X	▪
Ruby	▪	▪	▪	▪	▪	▪	U	▪	▪	X	▪	▪	▪	X	▪
Scala	▪	▪	▪	▪	▪	▪	X	T	▪	▪	▪	▪	▪	▪	W
SMIL	▪	X	X	X	▪	X	X	X	X	X	X	X	X	▪	W
X10	▪	X	▪	▪	▪	▪	▪	X	▪	L	▪	▪	▪	▪	C

Language-тіл
Control abstractions – бақылау абстракциялары

Қосымша IV

Language	Websites
Alice	http://www.alice.org/
Ada	http://www.sigada.org/education/ and http://www.ada-auth.org/arm.html
C	Multiple available on the Internet
C++	Multiple available on the Internet including Microsoft Visual Studio
C#	Multiple available on the Internet including Microsoft Visual Studio
Chapel	http://chapel.cray.com/
Claire	http://www.claire-language.com/
Clojure	http://clojure.org/
ECLiPSe	http://www.eclipseclp.org/
Emerald	http://www.emeraldprogramminglanguage.org
Esterel	http://www-sop.inria.fr/meije/esterel/esterel-eng.html
F#	http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/ http://msdn.microsoft.com/en-us/library/dd233154.aspx
FORTRAN	http://j3-fortran.org/
Haskell	http://www.haskell.org
Java	http://www.java.com and http://www.oracle.com/us/technologies/java/overview/index.html
Javascript	http://msdn.microsoft.com/en-us/library/ie/d1et7k7c(v = vs.94).aspx
Lisp	Multiple including GNU Lisp site
Lua	http://www.lua.org
ML	http://www.smlnj.org
Modula-3	http://www.modula3.org
Perl	Multiple available on the Internet
PhP	http://www.php.net/
Prolog	http://www.sics.se/software; http://www.gprolog.org; http://www.swi-Prolog.org
Python	http://www.python.org/
Ruby	http://www.ruby-lang.org
Scala	http://www.scala-lang.org
SMIL	http://www.w3.org/AudioVideo/ and http://www.w3.org/TR/2008/REC-SMIL3-20081201/
X10	http://x10-lang.org/

Language-тіл
Website-вебсайт

Қосымша V
Локалдылық принципі

Бағдарлама шамалы уақыт аралығында жергілікті қоршаған ортаның шегінде орындалады. Бұл қазіргі таңда жергілікті қоршаған ортаның белсенді қатысуы телім деп аталады. Жергілікті қағидасы бағдарламаның жақын арадағы болашақта сол телімді қолданатынын бекітеді. Орналасу жерінің басқару ағымымен өзгеруі: (1) мәліметтердің үлкен құрылымының әр түрлі бөліктерін өңдейді, (2) үлкен бағдарламаның әр түрлі бөліктерін алмастырады немесе (3) басқа бағдарламаны шақырады.

Жергілікті қағидасының тиімді орындауы және бағдарламаны орындауда кеңістікті тиімді пайдалану үшін маңызды мағынасы бар, себебі қазіргі заманғы операциялық жүйелер бірнеше үдерістерді орындайды — бағдарламаның белсенді бөлігін бір уақытта — және әр белсенді үдеріс үшін жедел жадының шекті көлемін бөледі. Жедел жадыдағы орынның шектілігі үшін бағдарламаның шағын бөлігі белсенді жадыға ауыстырылды және белсенді жадыменүздіксіз басқарылуы тиіс. Өзгерістерді орындау орыны тәрізді, кеңістіктің жаңа бөліктері айқындалған пайдаланушының кеңістігіне келтірілді, оның нәтижесінде ақпараттың жоғары бөлігін, қатты диск тәрізді екінші құралдарынан ақпараттар нәтижесінде сақтау. Бағдарламашы қисынды кеңістіктің бағдарламасын жазады. Алайда, шын мәнінде, төменгі деңгейде, бағдарламалар және мәліметтер, бағдарламалардың бөліктері және файлдар қатты дискіде сақталғанда, жедел жадыда сақталады. Жадының тағы екі кеңістігі бар: бағдарламашының жадыдағы қисынды кезектілік ойлары виртуалды жады және әр түрлі физикалық жады кеңістігі деп аталады, мұнда нақты бағдарламалар және міліметтер сақталады. Виртуалды жады операциялық жүйенің физикалық жадысында айқындалады. Жады блогы не ауыспалы көлемде, немесе бекітілген көлемде, ол операциялық жүйеге байланысты. Ауыспалының көлемінің блоктары сегменттер деп аталады және туындататын қосалқы бағдарламалардың көлемі болады. Блок-

тардың бекітілген көлемдері беттер деп аталады.
Беттер (немесе сегмент) жедел жадының екінші реет сақталуынан келтірілген, егер де мекен-жайы RAMнан тиімділікті жүзеге асырса жедел жады жетіспейді. Жетіспейтін беттердің RAM қатты дискілерінен импорт үдерісі беттер қатесі деп аталады және маңызды мәліметтерді жіберуді ұстауы бар. Беттердің істен шығуының саны ұлғайған тәрізді, және пайдаланудың нақты үдерісі азаяды. Жоспарлаушы пайдалануды жүктеуді жақсарту үшін көбірек үдерістерді келтіреді. Бірнеше үдерістерді орындау беттердің көбірек істен шығуын туындатады және пайдалану жүктемесін азайтады. Ақырында, процессор пайдалы жұмысты орындауды тоқтатады. Бұл құбылыс үндемеу деп аталады, және оны болдырмау жөн.

Қосымша VI
Виртуалды жад және парақшадағы қателер

Бағдарламаны жазушы, бағдарламаны барлық деректер кеңістігі және код кеңістігі үздіксіз болғандықтан логикалық кеңістікте есептеу арқылы жазады. Алайда, негізінде, төменгі деңгейде, бағдарлама мен жинақталған деректер RAM-да және қатқыл дискте сақталады: RAM-да бағдарламаның белсенді бөлігі сақталса, бағдарламаның қалған бөліктері және деректері қатқыл дискте сақталған. Негізінде екі жад кеңістігі бар: жад кеңістігінің логикалық үздіксіз бағдарлама жазушының көз-қарасы виртуалды жад деп аталады, негізгі бағдарлама және деректер сақталатын, алмасқан физикалық жад кеңістігін атайды. Виртуалды жад физикалық жадқа операциялық жүйе арқылы салынған. Жад блогы операциялық жүйенің қолднауына байланысты өзгермелі көлемді немесе тұрақты көлемді болуы мүмкін. Өзгермелі көлемді блогтардың сегменттері деп аталады, және қосымша бағдарлама деп аталатын көлемде болады. Тұрақты көлемді блогтар беттер деп аталады.

Виртуалды жадта RAM-мен рұқсат етілген мекенжай RAM-да болмаса бет (немесе сегмент) RAM-ға қосалқы қордан алынады. Жоғалған беттерді қатқыл дисктен RAM-ға алып келу үдерісі беттегі қате деп аталады, және бұл үдеріс деректердің алмасуын талап етеді. Беттегі қателердің көбеюіне байланысты негізгі CPU өңделуі азаяды. Үдеріс жоспарлаушысы CPU өңделуін оңтайландыру үшін одан да көп үдерістер жасайды. Көп үдерістерді орындау беттерде көп қателердің болуын тудырып CPU өңделуін одан бетер азайтады. Соңында, CPU қандай да бір пайдалы әрекеттер жасауды тоқтатады. Бұл жадтың толып кету салдарынан жүйенің ақаулығын туғызатын феномен тоқтап қалу деп аталады. Бұндай жағдайды болдырмау керек.

Қосымша VII
Бағдарлама жұмысының дұрыстығы және тығыздығы

Бағдарлама дұрыс болап табылады, егер де шешімдер жиынтығы осымәселені шешу үшін тиімді барлық мүмкін шешімдерді қарастырса. Бағдарлама аяқталған болып табылады, егер де, мәселені шешудің жиынтығы шешімдер жиынтығына кірсе. Бағдарламада қате шешім мәселесі болуы мүмкін, ол бағдарламаның аяқталуы туралы. Ұқсас тәсілмен, егер де бағдарлама тек дұрыс шешімдерді қабылдауға тырысса, бағдарламаның толық емес болуы мүмкін деген болжам бар. біз бағдарламаны жазған кезде, біз шешімнің дұрыстығын және толықтығын іздеуіміз керек. Нақты өмірде дұрыс және толық болып табылатын бағдарламалық қамтамассыз етуді жазу қиын. Бағдарламалық қамтамассыз етудің көптеген қисынды қателері бар, олар бағдарламалық қамтамассыз ету қателері деп аталады. Ол қателердің саны бағдарламалық қамтамассыз ету көлемімен ұлғаяды.

Бағдарламалық қамтамассыз етудің қателерін жоюдың бірнеше тәсілі бар: (1) бағдарламашы қисыннан қайта қайта өтеді (2) бағдарламашы әр түрлі кіріс мәліметтерінің үлгілерімен, бағдарлама қажетті шығыс мағынасын қайтармаған жағдайда, қателерді айқындау және түзету үшін, бағдарламаны іске қосады және (3) дұрыс бағдарламаларды алу үшін автоматтандырылған бағдарламаларды талдау үшін құралдар қолданылады. 1 және 2 әдістер кең таралған болып табылады. Алайда, 3 әдіс бағдарламалық қамтамассыз ету үшін сирек болып табылады, себебі тартылған есептеу техникасын талап етеді.

Қосымша VIII
Алгоритмдер қиындығы

Төменгі деңгейдегі аралық кодтың жоғары деңгейлі құрылымын аудару уақытысында және бағдарламаларды орындау уақытысында бағдарламалардың тиімділік мәселесі шешілуі тиіс. Бағдарламалардың тиімділігі келесідей көптеген факторларға байланысты: (1) құрылымдық мәліметтердің орналасуы;
(2) Қатты дискіде сақталатын мәліметтерді шығару жиілігі; (3) тиімді алгоритмдердің болмауы; және (4) үстеме шығындардың мәліметтері, әсіресе, қиын тапсырманы шешу үшін бірнеше үдерістерді қолдану кезінде.
Орындаудың тиімділігін оқудың тәсілдерінің бірі болып, шығыс көлемдерінің ұлғаюы кезіндегі жағдайды орындау тиімділігін оқыту тәсілі табылады. Мысалы сызықтық қиындықты алгоритмде орындау уақытысы шығыс мәліметтерінің көлемінің ұлғаюымен өседі. Алгоритмнің төртбұрышты сыйымдылығы бар, егер де бағдарламаны орындау уақыты O(m2) баламасын ұлғайтса. Егер шығыс мәліметтерінің көлемі m баламасына ұлғайса. Кубтық алгоритмдерде, кіріс мәліметтерінің көлемі m рет ұлғайғанда, орындау уақытысы O(m3) ұлғаяды. Кубтық алгоритмдерден бөлек, алгоримдер үлкен көлемдегі мәліметтерде қолдануға арналмаған және өз тапсырмаларын орындауда өте баяу болады. Қуат баламасы үстіңгі шек тәрізді тұрақты шамамен шектелген мұндай алгоритмдер Полиномиалды алгоритмдер деп аталады, және тәжірибеде куаттылық баламасының мағынасы > 4 жақсы шешім болып табылмайды. Тиімді алгоритмдер класстары тұрақты қиындылық, лог(лог(N)) қиындылық, қиындылық лог(N) және сызықтық қиындылық(O(N)), уақытты ұлғайту екпіні аз немесе кіріс мәліметтерінің көлеміне тең. лог(N)қиындылықтың кейбір мысалдары іздеудің екілік алгоритмдері және тұрақты уақытта әдістерді хэштеу алгоритмдері болып табылады.
Алгоритмдердің қызықты классы экспоненциалды алгоритмдерді қарастырады, онда уақыт кіріс мәліметтерінің көлемінің ұлғаюымен

экспоненциалды ұлғаяды. Кіріс мәліметтерінің көлемінің ұлғаюы кезінде, экспоненциалды алгоритмдер, кез келген компьютерде орындау үшін орындау уақыты тым үлкен деңгейге қол жеткізеді. Егер де полиномиалды қиындықта және осындай НР мәселелері осы мәселеге қайта құрылуы мүмкін екендігін біз білмесек, онда мәселе НР-толық (анықталмаған полиномиалды толық) деп аталады. Бағдарламалау тілдерін үлестіруде біз, осы уақытқа дейін тиімсіз болып табылатындығы мәлім болған, НР-толық мәселелерінің алдын алғанды қалаймыз.

Қазіргі таңда НР-толық алгоритмдері аппроксимацияны, эвристиканы (математикалық функцияның негізінде немесе танымал шешімдердің негізінде зияткерлік табу) және кіріс мәліметтерін және алгоритмдерді орындау тиімділігін арттыру үшін параметрлерін шектейді.

Уақыт тиімділігінің мәселесі айтарлықтай күрделі болып табылады және алгоримдік қиындықтардың модулденген зерттеулері ғана бола алмайды. Қарауға арналған басқа да сұрақтар бар: (1) Қатты дискілерден мәліметтерді шығарудың жасанды шығындары ОЗУ, (2) елді мекен қағидасын қолдану, (3) жинау мәліметтерін жинау каналы және (4) мәліметтерді жіберудің жасанды шығындары. Бұл мәселелер операциялық жүйелерде қарастырады.

Мазмұны

1 Кіріспе……5
1.1 Пәндер салаларының жиынтығы……6
1.2 Мотивация……8
1.3 Оқыту нәтижелері……10
1.4 Компоненттер мен бағдарламалар……11
1.4.1 Бағдарламалардағы абстракциялар……13
1.4.2 Қолсараптамасы (бағдарламаны түсіну) және Ауысулар……18
1.4.3 Бағдарламалардың орындалуы……21
1.5 Бағдарламалау тілдерінің сәйкестілігі……28
1.6 Бағдарламалық қамтамасыз етуді жетілдіру циклі……28
1.7 Дұрыс бағдарламалау тілінің критерийлері……32
1.8 Парадигмалар мен бағдарламалау тілдерінің тарихы……34
1.8.1 Императивті бағдарламалау .парадигмасы……34
1.8.2 Декларативті бағдарламалау парадигмасы……36
1.8.3 Объектіге бағытталған бағдарламалау парадигмасы……38
1.8.4 Параллель бағдарламалау парадигмасы……40
1.8.5 Визуалды бағдарламалау парадигмасы……41
1.8.6 Мультимедиялық бағдарламалау парадигмасы……42
1.8.7 Веб-бағдарламалау парадигмасы……42
1.8.8 Оқиға-бағдарланған бағдарламалау парадигмасы……43
1.8.9 Бағдарламалау парадигмаларын интерграциялау……44
1.9 Тілдердің жіктелуі……44
1.9.1 Бағдарламалау парадигмаларды негізіндегі жіктеу……45
1.9.2 Қолдану негізінде жіктеу……45
1.9.3 Басқа жіктеу……47
1.10 Қысқаша қорытындылар……48
1.11 Бағалау……49
1.11.1 Тұжырымдамасы мен анықтамалар……49

2 Алғысөз және Негізгі түсініктер……52
2.1. Фон нейман машинасы……52
2.1.1 Адрес механизмдері……54
2.2 Үзілісті құрылымдардың тұжырымдамасы……57
2.2.1 Жиындармен операциялар……57
2.2.1.1 Декарттың шығармасы……58
2.2.1.2 Бейнелеу……58
2.2.1.3 Изоморфизм……59
2.2.1.4 Функциялар……59
2.2.2 Буль логикасы және предикаттарды есептеу……61
2.2.2.1 Бірінші қатардың предиктін есептеу……62
2.2.2.2 Қатынастар……63
2.2.3 Рекурсия……65
2.2.3.1 Құйрықтық Рекурсия және Итерация……65
2.2.3.2 Сызықтық рекурсия және итерация……66
2.2.4 Соңғы автоматтар……67
2.3 Деректер құрылымының тұжырымдамасы……69
2.3.1 Әрекеттер тізбегі……69

2.3.2 Стектар мен Тізімдер.....70
2.3.2.1Стек.....71
2.3.2.2 Тізім.....72
2.3.3 Референттік механизмдер.....73
2.3.4 Рекурсивті мәліметтер құрылымы.....74
2.3.5 Ағаш.....75
2.3.6 Бағаналар.....76
2.3.7 Іріктеп алу әдісі.....79
2.3.7.1 Тереңінен іздеу.....79
2.3.7.2 Енінен іздеу.....81
2.3.8 Бірөлшемді жадыдағы мәліметтер құрылымының бейнесі.....83
2.3.9 Хэш-кестелер.....85
2.4 Есептеу барысында абстрактілі ұғымдар қолда-ну.....86
2.4.1 Өзгергіш айнымалылар өзгермейтін айнымалыларға қарсы.....89
2.4.2 Көріну облысын байланыстыру және ережесі.....89
2.4.3 Айнымалылар типі.....91
2.4.4 Қабықша және Сақтаушы.....93
2.4.5 Функциялар мен Үдерістер.....95
2.4.6 Бағдарламаны орындауды дерексіздендіру.....96
2.4.6.1 Үштік сияқты есептеу күйі (Қабықша, Сақтаушы, Дамп).....96
2.4.7 Процестер және Ағындар.....99
2.4.8 Буферлік салалар.....100
2.5 Қысқаша қорытынды.....102
2.6 Бағалау.....105
2.6.1 «Тұжырымдамалар мен анықтамалар» деп өзгерту.....105
2.6.2 Тапсырма Шығару.....105
2.6.3 Толық жауап.....107

3 Синтаксис және Семантика.....109
3.1 Синтаксис және семантиканы енгізу.....109
3.2 Грамматика.....111
3.2.1 Грамматика түрлері.....113
3.2.2 Бэкус-Наур формасын пайдалана отырып, грамматиканы ұсыну.....115
3.2.3 Бэкус — Наур кеңейтілген пішіні (РБНП).....117
3.2.4 Атрибутивті грамматикалар.....121
3.2.5 Гиперқағида мен Мета-анықтамалар.....123
3.2.6 Абстрактылы синтаксис.....124
3.3 Синтаксистік диаграммалар.....126
3.3.1 Синтаксистік диаграммадағы синтаксистік қағидаларды ау-дару.....129
3.3.2 Синтаксистік диаграммаларды синтаксистік қағидаларға ау-дару.....134
3.4 Сөйлем құрылымын тексеру.....134
3.4.3 Грамматикалық әр түрлі мәнді жою.....136
3.4.3.1 Енгізілген құрылымдардың әр түрлі мәнділігі.....140
3.4.5 Автоматтандырылған синтаксистік талдау.....141
3.5 Семантика.....143
3.5.1 Операциялық семантика.....143
3.5.2 Аксиомалық семантика.....146
3.5.3 Денотациялық семантика.....148

4 Бағдарламалардағы абстракция және ақпаратпен ауысу……158
4.1 Деректер абстракциясы……160
4.1.1 Деректердің бірлік элементтері……162
4.1.2 Деректердің құрама элементтері……162
4.1.3 Деректер элементтерінің жиыны……165
4.1.4 Кеңейтілетін деректер элементтері……167
4.2 Басқару абстракциясы……168
4.2.1 Тағайындау және командалар жүйелілігі……169
4.2.2 Шартты операторлар……172
4.4 Параметрлерді беру……175
4.4.1 Параметрлерді мәні бойынша беру және оның түрлері……177
4.4.1.1 Күрделі объектілерді бөлу үшін параметрлерді мәні бойынша беру……179
4.4.3 Нәтиже бойынша шақыру……183
4.4.4 Нәтиже мәні бойынша шақыру……184
4.4.5 Атауы бойынша шақыру……186
4.4.6 Қажеттілігі бойынша шақыру……189
4.4.7 Параметрлер түріндегі ішкі бағдарламаларды беру……190
4.4.8 Бөлінген есептеулерге арналған параметрлерді беру……190
4.5 Жанама әсерлер……191
4.5.2 Жанама әсерлерді реттеу……195
4.5.3 Маңызды мысал……196
4.6 Ерекше жағдайларды өңдеу……198
4.7 Анықталмаған есептеулер……201
4.7.1 Қорғалған командалар……203
4.7.2 Бағдарламаны біртіндеп құру……205
4.8 Мәлімет тәрізді бағдарламалар……206
4.8.1 Бірінші деңгейдегі нысан ретіндегі функциялар……207
4.8.2 Метабағдарламалау және рефлексивтілік……207
4.9 Бағдарламалық қамтамассыз етуді қайталама қолдану……207
4.9.1 Тағы да өзара әрекетте мүмкінділік туралы……208
4.10. КЕЙС-ӘДІС……209
4.10.1 Бағдарламалау тіліндегі мәліметтер абстаркциясы……210
4.10.1.1 Бірыңғай Абстракция……210
4.10.1.2 Құрамалы абстракция……211
4.10.1.3 Ақпараттық Нысандардың Бірыңғайлығы……214
4.10.1.4 Кеңейтілген ақпараттық нысандар……217
4.10.2 Бағдарламалау тілдерінде абстракцияларды басқару ……219
4.10.2.1 Мутация……219
4.10.2.2 Шартты бекітулер……219
4.10.2.3 Итерациялар және Итераторлар……219
4.10.3 Бағдарламалау тіліндегі мәліметтермен алмасу……220
4.11 Қысқаша қорытындылар……222
4.12 Баға……225
4.12.1 Концепциялар және Анықтамалар……225
4.12.2 Тапсырманы шешу……226
4.12.3 Толық жауап……227

5 Императивті тілдердегі үлгілерді үлестіру……230
5.1 Абстрактілі есептеу машиналары……232

5.2.1 Айқындамалардың аудармасы.....237
5.2.2 Иелену операторының аудармасы.....237
5.2.3 Шартты операторлар құрылымының аудармасы.....238
5.2.4 Нұсқау операторының аудармасы.....239
5.2.5 Итерациялық конструкцияның аудармасы.....240
5.2.5.1 Контурға аудару.....241
5.3 Статикалық үлестіру.....243
5.4 Гибридті үлестіру.....247
5.4.1 Әр түрлі нұсқаулар рөлі.....249
5.4.2 Шақырылатын бағдарламалар.....250
5.5 Параметрлерді жіберуді үлестіру.....258
5.5.1 Мағынасы бойынша шақыруды үлестіру.....258
5.5.2 Сілтеме арқылы шақыруды жүзеге асыру.....260
5.5.3 Нәтиженің мағынасы бойынша шақыруды үлестіру.....262
5.6 Рекрусивті процедуралардың төмендеңгейлі мінез-құлқы.....264
5.7 Ерекшеліктер өңдегңшінің іске асуы.....265
5.9 Бағалау.....268
5.9.1 Тұжырымдамалар мен анықтаулар.....268
5.9.3 Толық жауап.....269
Қосымша I
Тілдердегі қолдайтын парадигмалар.....271
Қосымша II
Абстракцияның деректер жиын-тығы.....273
Қосымша III.....275
Қосымша IV.....277
Қосымша V
Локалдылық принципі.....278
Қосымша VI
Виртуалды жад және парақшадағы қателер.....280
Қосымша VII
Бағдарлама жұмысының дұры-стығы және тығыздығы.....281
Қосымша VIII
Алгоритмдер қиындығы.....282