Цифрлық әдіскер

ақпараттық-білім беру ортасы

ДӘРІСТЕР ЖИНАҒЫ

Алгоритмдеу және бағдарламалау негіздері

Алгоритмдік ойлаудың негізі: ақпараттық жүйелер, деректерді өңдеу кезеңдері және бағдарламалау парадигмалары.

- Ақпараттық жүйелер
- Алгоритм құру принциптері
- Блок-схемалар
- Басқару құрылымдары

Пән санаты

Бағдарламалау

adisker.kz

Әдістемелік қор

ДӘРІС ЖИНАҒЫ

№ 1 тақырып

КІРІСПЕ. АЛГОРИТМДЕУ ЖӘНЕ ДЕРЕКТЕР ҚҰРЫЛЫМЫ ЭЕМ-ҒА ПРОГРАММАЛАР ҚҰРУҒА БАҒЫТТАЛҒАН ПӘН. ҚАЗІРГІ УАҚЫТТАҒЫ МАМАНҒА КОМПЬЮТЕРДІҢ ЖҰМЫСЫ ПРИНЦИПТЕРІН ЖӘНЕ ОНЫ ПРОГРАММАЛАРМЕН ҚАМТАМАСЫЗ ЕТУ МҮМКІНДІКТЕРІН БІЛУ ҚАЖЕТ

Жоспары:

1. Ақпараттық жүйелердегі компьютерлік технологияның рөлі.
2. Әр түрлі салаларда ақпараттық жүйелерді пайдалану мысалдары.

1. Ақпараттық жүйелердегі компьютерлік технологияның рөлі Ақпараттық жүйе - нақты тапсырманы шешу үшін ақпаратты сақтау, өңдеу және шығару үшін қолданылатын құралдар, әдістер мен қызметкерлердің өзара байланысы. Ақпараттық жүйенің қазіргі заманғы түсінуі компьютерді ақпаратты өңдеудің негізгі техникалық құралы ретінде пайдалануды қарастырады. Арнайы бағдарламалық қамтамасыз етумен жабдықталған компьютерлер ақпараттық жүйенің техникалық базасы мен құралы болып табылады. Ақпараттық жүйенің жұмысында келесі кезеңдерді бөліп алуға болады:
2. Деректердің пайда болуы - белгілі бір операциялардың нәтижелерін, объектілердің қасиеттерін және басқару субъектілерінің қасиеттерін, процестердің параметрлерін, нормативтік-құқықтық актілердің мазмұнын және т.б. сақтайтын негізгі хабарламаларды қалыптастыру.
3. Деректерді жинақтау және жүйелеу - қажетті ақпаратты жылдам іздестіруді және таңдауды, деректерді әдістемелік жаңартуды, бұрмалаудан, жоғалтудан, адалдықтың деформациялануынан қорғауды қамтамасыз ететін осындай орналастыруды ұйымдастырады.
4. Деректерді өңдеу - бұған бұрын жинақталған деректер негізінде жаңа деректер түрлері қалыптастырылады: қорытындылау, талдау, кеңес беру, болжау. Деректер алынған деректерді де жиі жалпыланған ақпаратты алуға болады. Деректерді көрсету - оларды адамның қабылдауына қолайлы нысанда көрсету. Ең алдымен, бұл басып шығару, яғни қатты (қағаз) медиа бойынша құжаттарды жасау. Графикалық иллюстрациялық материалдарды (графиктер, диаграммалар) және дыбыстық сигналдарды қалыптастыру үшін кеңінен қолданылады. Бірінші кезеңде пайда болатын хабарлар қарапайым қағаз құжаты, «машина нысанында» немесе екеуінде де хабар болуы мүмкін. Заманауи ақпараттық жүйелерде бұқаралық хабарламалар көбінесе «машина көрінісі» бар. Осы жағдайда қолданылатын жабдықтар негізгі ақпаратты жазатын құрылғы деп аталады. Екінші және үшінші кезеңдердің қажеттіліктері қазіргі заманғы ақпараттық жүйелерде негізінен компьютерлік технологиялар арқылы қанағаттандырылады. Адамдар үшін ақпараттың болуын қамтамасыз ететін құралдар, яғни деректерді көрсету құралы компьютерлік техниканың құрамдас бөлігі болып табылады. Ақпараттық жүйелердің көпшілігі пайдаланушымен диалог режимінде жұмыс істейді. Ақпараттық жүйелердің үлгілі бағдарламалық құрал компоненттері, деректер өңдеу ішкі жүйесі қолдану логикалық ішкі жүйесі бақылау логикалық деректерді диалог логикасын іске асырады интерактивті кіріс-шығыс кіші қамтиды. Желі ақпараттық жүйелер үшін жалпы рений тапсырмаға өзара іс-қимыл желі түйіндерін қамтамасыз байланыс қызметінің маңызды элементі болып табылады. ақпараттық жүйелердің функционалдық бөлігі жүйесі бағдарламалық салып: операциялық жүйелер, жүйелік кітапханалар және дамыту құралдары әзірлейді. Сонымен қатар бағдарламалық компонент ақпараттық жүйелер, құрылымын анықтайтын маңызды рөл ақпараттық компоненті атқарады атрибуттары, және деректер түрлері, сондай-ақ тығыз деректерді басқару логика байланысты.
5. Әр түрлі салаларда ақпараттық жүйелерді қолдану мысалдары Қазақстандағы ақпараттық технологиялар саласындағы жылдам жетістіктер дәуірінде қазіргі заманғы компьютерлердің есептік қуаты тәжірибелік қызметтің барлық салаларында адам еңбегін жеңілдету үшін жиі пайдаланылады. Үздіксіз дамып келе жатқан математика және бағдарламалық қамтамасыз ету, өндірісте қолданылатын компьютерлік технологиялар, кеңселер, оқу орындары

ұштастыра отырып, адамға күнделікті жұмысына айтарлықтай көмектесетін қуатты құрал болды. Ақпараттық технологияларды қолданудың маңызды бағыттарын атап өтейік:

6. Дамудың маңызды стратегиялық факторы болып табылатын қоғамның ақпараттық ресурстарын белсенді және тиімді пайдалануға бағыттау. Ақпараттық технологияларды (ғылыми білімдер, өнертабыстар, озық тәжірибелер) жандандыру, тарату және тиімді пайдалану түрлі ресурстарды (шикізат, энергетика, минералдар, материалдар мен жабдықтар, адам ресурстары, уақыт) үнемдеуге мүмкіндік береді.
7. Ақпараттық процестерді оңтайландыру және автоматтандыру. Өркениеттің дамуы ақпараттық қоғамның қалыптасуы бағытында орын алады, онда жұмыспен қамтылған халықтың көпшілігінің еңбегі мен нәтижесі материалдық құндылықтар емес, негізінен ақпараттық және ғылыми білімдер болып табылады. Жұмыс күшінің көпшілігі ақпаратты дайындау, сақтау, өңдеу және беру үдерістерімен байланысты немесе одан да көп, сондықтан осы ақпараттық технологияларды үйренуге және іс жүзінде қолдануға тиіс.
8. Өндірістік және әлеуметтік технологияларды енгізу. Сонымен қатар, әдетте, осы технологиялардың «интеллектуалды» функциялары жүзеге асырылады: өнеркәсіптік өнімдердің компьютерлік дизайны, икемді автоматтандырылған және робототехникалық өндіріс, автоматтандырылған процестерді басқару жүйесі және т.б.
9. Бұқаралық ақпараттарды дайындау және тарату жүйелерінде адамдар арасындағы ақпараттық өзара іс-қимылды қамтамасыз ету. әлеуметтік секторда (мысалы, телефон, телеграф, радио және теледидар) байланыс дәстүрлі құралдарын қоса кеңінен қолданылатын электрондық байланыс жүйесі (е-пошта мен компьютерлік басқа да байланыс түрлері) болып табылады. Бұл қаражат адамдар әлемдік қауымдастықтың жаһандану және интеграция процестерін туындаған көптеген өндірістік, әлеуметтік және жеке проблемаларын, ішкі және халықаралық экономикалық және мәдени қарым-қатынастарды, халықтың көші-қон және планетаның оның барған сайын динамикалық қозғалыс кеңейту алып тастау, үлкен ыңғайлы жасауға мүмкіндік береді.
10. Қоғамды зияткерлік ету, білім мен мәдениетті дамыту ақпараттық технология оқыту пайдалану тиімді білім беру жүйесін әдісін, сондай-ақ оқу және қайта даярлау жүйесі үшін дәлелдеді. Білім берудегі ақпараттық технологиялар (ББАТ) білім беру қызметіндегі эволюциялық өзгерістердің үдеткіші; оқыту әдістерін және ұйымдастыру формаларын жетілдіру жолдары, білім беру сапасын арттыру; оқу, оқудан тыс, әдістемелік, әкімшілік, ғылыми қызметті автоматтандыру және т.б.
11. Жаңа білім алу және жинақтау процесіне қосу. Ғылыми зерттеулерді ақпараттық қамтамасыз етудің дәстүрлі әдістері ғылыми-техникалық ақпараттың жинақталуы мен таратылуымен, фундаменталды және қолданбалы ғылымның мүмкіндіктерін ақпаратқа негізделген жаңа әдістермен ауыстырылады. Ең алдымен, ғылымда зерттелген үрдістер мен құбылыстарды ақпараттық модельдеу әдістері ғалымға «компьютерлік эксперимент» түрін жүргізуге мүмкіндік береді. Бұл жағдайда эксперименттік жағдайлар практикада олардың күрделілігі, жоғары құны немесе тәжірибешіге қауіп төндіретіндіктен қиын немесе мүмкін емес болуы мүмкін. Жасанды интеллект әдістері нашар ресімделмейтін мәселелерге, сондай-ақ толық емес ақпаратпен және анық емес бастапқы деректермен байланысты проблемаларды шешуге мүмкіндік береді.
12. Жаһандық проблемаларды және, ең алдымен, өркениеттің әлемдік дағдарыс тәжірибелі халықаралық қоғамдастықты жеңу қажеттілігімен байланысты мәселелерді шешуде көмек. әсіресе мониторингі бойынша ғарыш негізделген ақпаратты әдістермен ұштастыра жаһандық процестерді ақпараттық модельдеу әдістері, қоғамға жоғары қауіп төндіретін, табиғи апаттар мен ірі технологиялық авариялардың аудандарда, өсіп, әлеуметтік және саяси шиеленістің аймақтарда, сондай-ақ экологиялық апат аймақтарында көптеген дағдарысты жағдайлардың болжам қамтамасыз ете алады. Ақпараттық технологиялар қандай бағыттарда пайдаланылғанын олар әрдайым ақпаратты өңдеумен байланыстырады. Бақылау сұрақтары 1.Программалау тілі дегеніміз не ? 2.Программалау тілдерінің даму эволюциясы ? 3.Қандай программалау тілдері бар?

№2 тақырып Алгоритмдер. Алгоритмдерді өңдеу принциптері. Қойылған есептерді шығарудың негізгі сатылары. Алгоритмдер, деректер құрылымы және программалау тілдерінің негізгі түсініктері.

Жоспары:

- Алгоритм анықтамасы
- Алгоритм қасиеттері
- Сызықтық немесе тізбекті алгоритм.
- Тармақталу алгоритмдері.

- Циклдік алгоритмдер.

- Алгоритм анықтамасы Алгоритм - берілген тапсырманы шешу үшін орындалуы қажет әрекеттер тізбегі. XX ғасырдың ең үлкен жетістігі алгоритмдер теориясы - жаңа математикалық пән. Электрондық компьютерлер теориясы және бағдарламалау тәжірибесі онсыз мүмкін емес. Атауы айтып отырғандай - алгоритмдер теориясы - оның негізі - алгоритмдер. Кез келген компьютерлік мәселені шешкен кезде, кейбір ақпарат бағдарлама деп аталатын алдын ала құрастырылған нұсқаулыққа сәйкес өңделеді. Компьютерде есептеу экспериментінің негізі жүйелілік: модель - алгоритм - бағдарлама. «Алгоритм» сөзі өзбек математикы Мохаммед әл-Хорезмидің (Хорезм - Өзбекстан аумағында көне мемлекеттің атауы) есімі болып табылады, олар 9 ғасырда Ондық сандар жүйесіндегі сандар бойынша төрт арифметикалық операцияға арналған ережелер әзірленген. Алгоритм түсінігі компьютерлік ғылымның негізгі ұғымдарының бірі болып табылады. Қазіргі кезде, алгоритмдерге деген қызығушылық әсіресе инженерлік, экономика, ғылыми зерттеулер және т.б. компьютерлерді қолдану мүмкіндігіне байланысты. Мұнда нүкте компьютер жұмыс істеп тұрған бағдарламаны орындайды және бағдарлама арнайы белгілерде жазылған кейбір алгоритм болып табылады. Тиісті белгілеу жүйесі бағдарламалау тілі деп аталады. Дәлірек айтқанда, бағдарламалау тілі - компьютерге ыңғайлы түрде алгоритмді ұсынуға арналған құралдар мен ережелер жиынтығы. Әрбір алгоритм бастапқы деректерден қажетті нәтижеге жететін тізбектегі әрекеттерді көрсететін ереже болып табылады. Мұндай әрекеттер тізбегі алгоритмдік процесс деп аталады, және әр іс-әрекет оның қадамы деп аталады. Кез келген мәселені шешу бастапқы деректер, яғни нәтиже алуды талап етеді, біз алгоритмі нәтижелері бастапқы деректерді қайта тізбекті процесін сипаттайды деп айтуға болады. Мәселені шешуге арналған алгоритмді әзірлеу - оны орындайтын тізбекті қадамдарға бөлу қажеттілігін туындатады. Есепті шешу үрдісін сипаттайтын айқын құрастырылған іс әрекеттің тізбектілігі алгоритм болып табылады. Бағдарлама кейбір алгоритмді іске асыратын, пайдаланушыға түсінікті тілде жазылған командалар жиынтығын көрсетеді. Біздің жағдайымызда, компьютер орындаушы болып табылады, ал бағдарламалау тілі ретінде жоғары деңгейлі тіл Python бағдарламалау тілі қарастырылады. Өкінішке орай, кез келген жоғары деңгейлі тіл бағдарлама жазушыға немесе бағдарламаны өңдеушіге ыңғайлы, бірақ компьютерге мүлдем түсініксіз болып келеді. Бұл тілдегі бағдарлама бастапқы мәтін деп аталады және .pas кеңейтімі бар шығыстық файлда сақталады. Орындаушы-компьютерге түсінікті төменгі деңгейлі тілге бағдарламаны аудару үшін арнайы аудармашы-бағдарлама – компиляторлар бар. Компилятор жұмысының нәтижесі (басқаша айтқанда, компиляция процессінің нәтижесі) .exe кеңейтілімді файлға жазылатын орындалушы код болып табылады. Шешімге тапсырманы дайындау тәртібі. Мәселені қарапайым әрекеттер деңгейіне дейін жеткізу Компьютерде әртүрлі мәселелерді шешуге болады, мысалы: ғылыми және инженерлік; бағдарламалық қамтамасыз етуді әзірлеу; білім беру; өндірістік процестерді бақылау және т.б. ЭЕМде ғылыми инженерлік есептерді дайындау және шешу барысын келесі кезеңдерге бөлуге болады:
- есептің қойылымы;
- есептің математикалық сипаты;
- есепті шешу әдістерін таңдау мен негіздеу;
- есептеу үрдісін алгоритмдеу;
- бағдарлама құру;
- бағдарлама отладкасы; 7.ЭЕМде есепті шешу және нәтижені талдау. Басқа кластың міндеттерінде кейбір сатылар болмауы мүмкін, мысалы, жүйелік бағдарламалық қамтаманы құрастыру есептерінде математикалық сипаттамалар болмауы мүмкін. Аталған этаптар бір бірімен байланыста. Мысалы, нәтижелерді талдау бағдарламаға, алгоритмге немесе есептің қойылымына өзгерістер енгізу қажеттілігін көрсетуі мүмкін. Осындай өзгерістердің санын азайту үшін мүмкіндігінше келесі кезеңдердің талаптарын ескеру қажет. Кейбір жағдайларда, әртүрлі сатылар арасындағы байланыс, мәселен, мәселенің тұжырымдамасы мен шешім әдісін таңдау арасындағы алгоритм мен бағдарламалауды құру арасындағы байланыс соншалықты жақын болуы соншалықты, бөліну қиынға соғуы мүмкін. Есептің қойылымы. Бұл кезеңде есепті шешудің мақсаты қалыптастырылады және оның мазмұны толық сипатталады. Есептеуде қолданылатын барлық шамалардың сипаты мен маңызы талданады, және есепті шешетін шарттар қарастырылады. Есеп қойылымының айқындылығы маңызды сәт болып табылады, себебі одан қандай да бір дәрежеде басқа кезеңдер тәуелді болады. Есептің математикалық сипаты. Бұл кезең нәтижені анықтайтын шамалардың қатынасымен есептің математикалық формализациясымен сипатталады, математикалық формулалар арқылы анықталады. Сондықтан құбылыстың математикалық моделі нақты дәлдікпен, жорамалдар мен шектеулермен қалыптасады. Бұл жағдайда шешілетін мәселелердің ерекшеліктеріне байланысты математика және басқа пәндердің әртүрлі бөлімдері пайдаланылуы мүмкін. Математикалық модель кем дегенде екі талапты қанағаттандыруы керек: реализм және іске асыру. Реализм - зерттелетін құбылыстың ең маңызды ерекшеліктерінің үлгісі арқылы дұрыс көрініс беру. Жүргізілген шешімді белгілі бір шешімге байланысты проблеманы азайту үшін ақылға қонымды абстракциямен, кішігірім бөлшектерден абстракция арқылы қол жеткізіледі. Өткізудің шарты қажетті ресурстардың бар шығындарымен бөлінген уақытқа қажетті есептеулерді іс жүзінде жүзеге асыру мүмкіндігі. Шешім

әдісін таңдау және негіздеу. Мәселені шешуге арналған модель, оның ерекшеліктерін ескере отырып, шешімді нақты әдістердің көмегімен шешуге тиіс. Өз кезегінде, мәселенің математикалық сипаттамасы көбінесе машина тіліне аудару қиын. Мәселені шешу әдісін таңдау және пайдалану мәселенің шешімін нақты машина операцияларына жеткізуге мүмкіндік береді. Әдістің таңдауын негіздеу кезінде әр түрлі факторлар мен жағдайларды, соның ішінде есептеулердің дәлдігін, компьютердегі мәселені шешу уақытын, қажетті жад көлемін және т.б. ескеру қажет. Бір және сол мәселені әр түрлі әдістермен шешуге болады, ал әр әдіс бойынша әр түрлі алгоритмдер жасалуы мүмкін. Есептеу процесін алгоритмдеу. Бұл кезеңде шешімнің таңдалған әдісімен анықталған әрекеттерге сәйкес проблеманы шешу үшін алгоритм жасалады. Деректерді өңдеу процесі бөлек салыстырмалы тәуелсіз блокта бөлінеді және блоктарды орындау реті белгіленеді. Алгоритмнің блок схемасы әзірленеді. Бағдарламаны құрастыру. Бағдарламаны құрастырған кезде мәселені шешу алгоритмі нақты бағдарламалау тіліне аударылады. Бағдарламалау үшін жоғары деңгейлі тілдер әдетте пайдаланылады, сондықтан компиляцияланған бағдарлама компьютердің компьютер тіліне аудармасын қажет етеді. Осындай аудармадан кейін тиісті машина бағдарламасы орындалады. Бағдарламаны түзету. Отладка бағдарламасында синтаксистік және логикалық қателерді табу және жою болып табылады. Бағдарламаның синтаксистік бақылауында аудармашы осы тілде салу немесе жазу ережелерінің тұрғысынан рұқсат етілмейтін рәміздердің құрылымдары мен комбинацияларын анықтайды. Компьютер қателерді бағдарламашыға хабарлайды, ал мұндай хабарламалардың түрі мен нысаны тілдің түріне және пайдаланылатын аудармашы нұсқасына байланысты болады. Синтаксистік қателерді жойғаннан кейін, бағдарламаның жұмыс істеу логикасы оны орындау кезінде нақты бастапқы деректермен тексеріледі. Ол үшін арнайы әдістер пайдаланылады, мысалы, бағдарлама аралық нәтижелерді қолмен есептейтін бақылау нүктелерін таңдайды. Бұл нәтижелер отладталған бағдарламаны орындағанда осы нүктелерде компьютермен алынған мәндерден тексеріледі. Бұған қоса, отладқылар түзету кезеңінде арнайы әрекеттерді орындайтын қателерді іздеу үшін пайдаланылуы мүмкін, мысалы, бағдарламаның бөлек нұсқауларын немесе толық фрагменттерін жою, ауыстыру немесе кірістіру, көрсетілген айнымалылардың мәндерін шығару немесе өзгерту. Мәселені компьютерде шешу және нәтижелерді талдау. Бағдарламаны түзеткеннен кейін оны қолданбалы мәселені шешу үшін пайдалануға болады. Бұл жағдайда, әдетте, компьютерлік мәселенің көптеген шешімі әдеттегі деректердің әр түрлі жиынтықтары үшін орындалады. Нәтижелер тапсырманы қойған маман немесе пайдаланушы тарапынан талданады және талдайды. Ұзақ мерзімді пайдаланудың дамыған бағдарламасы компьютерге орнатылады, әдетте орындау үшін дайын бағдарлама түрінде. Бағдарлама пайдаланушыға арналған нұсқауларды қоса, құжаттамамен бірге жүреді. Бағдарламаны кейінірек пайдалану үшін дискіге орнатқанда көбіне орындалатын кодтары бар файлдарға қосымша көмекші бағдарламалар, (утилиталар, каталогтар, баптаушылар және т.б.), сондай-ақ бағдарламалармен жұмыс істеу үшін қажетті мәтін, графика орнатылады. Есептеу экспериментінің негізі жүйелілік: модель - алгоритм - цикл бірізді кезеңге бөлінген бағдарлама. Компьютерді пайдалана отырып, проблемаларды шешу процесі бірнеше кезеңге бөлінеді:

- Мәселе туралы мәлімдеме (нәтижелердің сипаттамасы және қажетті бастапқы деректер).
- Үлгіні құру (бастапқы деректер мен нәтижелер арасындағы қатынастардың сипаттамасы).
- Компьютерді пайдалана отырып, проблеманы шешудің дайын әдісін әзірлеу немесе таңдау (компьютерде компьютерде қолдануға болатын операциялардың кезектілігінің түрінде шешімді анықтау үдерісін ұсыну).
- Мәселені шешуде алгоритм жасау (проблеманы шешу үшін іс-әрекеттердің ретін - ауызша сипаттама немесе блок-схема ретінде анықтау).
- Компьютерде алгоритмді енгізу (арнайы жұмыс түрлерінің жүйеленген ретін көрсету және оларды жүзеге асыру).
- Алынған нәтижелерді талдау (нәтижелерді перспективамен салыстыру және сәйкессіздіктер туындаған жағдайда олардың себептерін анықтау). Кез келген мәселені шешуге арналған алгоритмді құру маңызды және маңызды сәт болып табылады, себебі ол компьютермен орындалатын әрекеттердің реттілігін анықтайтын алгоритм. Алгоритмді құрастыру кезінде жасалған қателер, әдетте, есептеу үдерісінің дұрыс емес бағытына және, тиісінше, дұрыс емес нәтижеге әкеледі. Алгоритм - бұл мәселені шешу үшін орындалатын әрекеттердің дәйектілігінің нақты сипаттамасы. Себебі кез-келген мәселені шешуде кез-келген бастапқы деректермен нәтиже алу қажет, алгоритм түпнұсқалық деректердің нәтижеге өзгертулер тізбегін сипаттайды. Алгоритмнің дамуы мәселенің шешілуін қадамдар мен қадамдардың соңғы санына бөлу болып табылады. Жеке қадамдарды орындау кезінде алынған нәтижелер бастапқы деректер ретінде келесі кезеңдерде қолданылуы мүмкін. Алгоритмді әзірлеу кезінде әр кезеңнің мазмұны және оларды орындау реті міндетті түрде көрсетілуі керек. Жазу алгоритмдерінің формасы: жазудың ауызша әдісі алгоритмдер; алгоритмдердің графикалық әдісі Алгоритмдердің әртүрлі формалары бар, олардың бастысы алгоритмнің ауызша сипаттамасы (ауызша нысаны), графикалық әдіс (блоктық схема түрінде) және арнайы алгоритмдік тілде (бағдарламалау) алгоритмдерді іске қосылған мазмұнын жазу ауызша әдісі дәйекті өңдеу кезеңдері көрсетуі және еркін ұсыну табиғи тілде берілген. Бұл әдіс жалпыға қолжетімді болып табылады және егжей-тегжейлі кез келген дәрежесі, алгоритм сипаттамасы мүмкіндік береді. Екінші жағынан, бұл, өйткені табиғи синонимдер қасиеттерін туындамаса тілі, омонимов, көп мағыналылығымен қабылдау бірдей дегенде ресімделеді. алгоритмдерді

айқындық жоғары дәрежесін сипаттайтын, және алгоритм логикалық-математикалық құрылымын бейнесі болып табылады үшін графикалық әдісі. деректерді өңдеу Барлық жеке қадамдар (кезеңдері) іс-қимыл сипаты сәйкес, сондай-ақ анықталған конфигурациясын бар түрлі геометриялық фигуралар (символы блок) ұсынылған, және қадамдар (қадамдар тізбегі) арасындағы байланыстар осы көрсеткіштерді байланыстыратын көрсеткі арқылы көрсетіледі. Алгоритмнің графикалық бейнесі тізбек деп аталады. Схема алгоритм құрылымын бейнелеуге мүмкіндік береді. Атап айтқанда, диаграмма анық көрінетін циклдар: соңғы кейін қадамдардың Бұл дәйектілігі жүйелілігінің басына қайтып асырылатын (схемасы олар жабық бағыттарын білдіреді). Схемалар әдетте алгоритмнің аралық нұсқаларын бейнелеу үшін қолданылады. соңғы нұсқасы ойыншы-компьютер (бағдарлама) арналған, ол алгоритмдік тілде жазылуы тиіс. Алгоритмдер негізгі құрылымы - блоктар және типтік жұмыс үрдістері олардың қосылу стандартты әдістерін шектеулі диапазоны. Бұрын айтылғандай, әзірленген алгоритм бірнеше жолмен белгіленуі мүмкін: табиғи тілде (ауызша); графикалық әдіс (блок диаграммалары түрінде); арнайы алгоритмдік тілде (бағдарлама). Мәселелерді шешуге арналған бағдарламалар жазғанда, блоктық диаграмма түрінде алгоритмді алдын-ала ұсыну ұсынылады. Блок-схемадағы стандартты белгілер. Блок-схема, яғни алгоритмді іске асыру кезіндегі әрекеттердің дәйектілігінің графикалық көрінісі әртүрлі әрекеттердің белгілі бір мағынасы бар әртүрлі конфигурациялардың геометриялық фигуралары түріндегі элементтерден тұрады (3-сурет). Бұл элементтер бір-бірімен көрсеткілермен (ағын сызығы) бір операциядан (операциядан) екіншісіне ауысудың бағыттарын көрсетумен байланыстырылған. Ең жиі пайдаланылатын блок-символдар: № Блоктар кескіндемесі Блоктар қызметі 1

Алгоритм басы 2

Алгоритм соңы 3

Есептеуіш әрекет 4

Мәліметтерді енгізу, мәліметтерді шығару 5

шарт жоқ иә

Шартты тексеру 6

Циклдың басы 7

Көмекші алгоритмді шақыру 8

Байланыс бағытын көрсету

Сурет 3 – Неігзгі блок символдар

- Алгоритм қасиеттері
- Алгоритмнің айқын, дәл өрнектелу қасиеті. Алгоритмде келтірілген барлық әрекеттердің мағынасы айқын, нықты анықталған болу керек. Онда қандай қадам көрсетілсе тек солар ғана орындалуы қажет. Есеп шығаруға керектің бәрі анықталуы және орындаушыға түсінікті әрі нақты болуы тиіс.
- Алгоритмнің үзіктілік қасиеті. Алгоритмнің үзік модульдерге бөлінуі, яғни алгоритмді бірнеше кішкене алгоритмдерге жіктеу мүмкін болу керек. Бұл қасиеті бойынша алгоритм аралық нәтиже беретіндей бірнеше ықшам бөліктерге, ал олар одан кіші қадамдарға бөлінеді, яғни мәселені шешу процесінің тізбегі жеке-жеке әрекеттер жіктеледі. Сондықтан алгоритмді, екі-үш бқлікке бөліп, оларды жеке қабылдай алатын дәрежеде жұмыс істелінуі қажет.
- Алгоритмнің нәтижелік қасиеті. Кез-келген алгоритмнің нәтижесі болу керек. Әрекеттердің шектеулі санынан кейін белгілі бір уақытта қорытынды нәтиже алуымыз қажет.
- Алгоритмнің жалпылық немесе ортақтық қасиеті. Алгоритм құрғанда белгілі бір жеке проблемаға қарсы ғана арналмай, осы тәріздес мәселелер шешуін толық қамтуға мүмкіндік беретіндей етіп құрылуы қажет.
- Алгоритмнің формальды орындалуы. Алгоритмді орындағанда орындаушы оныәр командасының мағынасын түсінуі де, түсінбеуі де мүмкін. Бірақ алгоритмнің әр командасы орындаушының нақты бір әрекетті орындауын талап етеді. Алгоритм құрылымдарының алуан түрлері Алгоритмдердің негізгі құрылымдары блоктардың шектеулі жиынтығы және оларды әдеттегі әрекеттер тізбектерін орындау үшін оларды қосудың стандартты тәсілдері болып табылады. Алгоритмдер блоктардың өзара байланысуына қарай үш құрылымға – сызықтық тармақтық және циклдік түрлерге бөлінеді.
- Сызықтық немесе тізбекті алгоритм. Сызықтық алгоритм тізбектеле орналасқан командалардан, ал блок-схемалар бір сызық бойына орналасқан тізбекті блоктардан тұрады. Әрекеттердің тізбектей орындалуы – сызықтық алгоритм деп аталады. Мысалы: алг үй тапсырмасын орындау басы күнделікті алу, тиісті бетін ашу, үй тапсырмасын анықтау үй тапсырмасын орындау күнделікті орнына қою соңы Сызықтық алгоритм командалары осында көрсетілген рет

бойынша орындалатын тізбектеле орналасқан командалардан (блоктардан) тұрады. Амалдардың бұлай бірінен соң бірі реттеліп орындалу тәртібін табиғи атқарылу дейді. Мысалы, төменде көрсетілген Z функциясының сандық мәнін есептеп шығару алгоритмін жасау керек болсын. Z =(ax + b)2 + cos (ax + b)2 – tg (ax + b)2 Бұл функцияның мәнін табу үшін алдымен жақшада тұрған (ax + b)2 көпмүшелігін жеке есептеп алу қажет, себебі ол тізбек үш рет есептеліп, орындаушы машина оған уақытты көп кетіреді. Есептеліп болған Z функциясының мәні қағазға не экранға басылып шығуы тиіс. Жалпы компьютер жадына a, b, x мәндері алдын – ала енгізілуі керек. алг Z функциясын есептеу (нақ a, b, x, z) арг a, b, x нәт z басы a, b, x енгізу t : = ax2 + b z : = t + cost + tgt x, z шығару соңы Сонымен қарастырылған алгоритм қарапайым сызықтық алгоритмнің мысалы болып табылады. Мұндағы 2-блок - a, b, x мәндерін пернелерден программаға енгізу блогы, 3-блок t - ның, ал 4-блок Z функциясының мәндерін есептейді. 5-блок x айнымаласының және Z функциясының нәтижесін қағазға басып шығарады.

- Тармақталу алгоритмдері. Тармақталу алгоритмінде көбінесе арифметикалық теңсіздік түрінде берілген логикалық шарт тексеріледі. Егер орындалса, онда алгоритм бір тармақпен жүзеге асырылады да, соңында екі тармақ қайта бірігеді. Мұндай алгоритмде шартты тексеру тармақталу командасы деп аталады. Оны алгоритмдік алгоритмдік тілде өрнектелгенде егер, онда, әйтпеес, бітті түйінді сөздері пайдаланылады. Орынду тәсіліне байланысты тармақталу командасы «таңдау»(толымды) және «аттап өту» (толымсыз) болып екі түрге бөлінеді.
- Циклдік алгоритмдер. Көптеген алгоритмдерде белгілі бір әрекеттер тізбегі бірнеше рет қайталанып орындалып отырады. Математикада есеп шығару кезінде бір теңдеуді пайдаланып, ондағы айнымалы мәнінің өзгеруіне байланысты оны бірнеше рет қайталап есептеуге тура келеді. Осындай есептеу процесі бөліктерінің қайталап орындалуы цикл деп атайды, ал қайталанатын бөлігі бар алгоритдер тобы циклдік алгоритмдер жатады. Қайталану командасын алгоритмдік жазу үшін әзірше (әзір), цикл бар (цб), және цикл соңы (цс) түйінді сөздер қолданылады.

Бақылау сұрақтары

- Алгоритм дегеніміз не?
- Алгоритмнің қанша қасиеті бар?
- Алгоритм жазудың неше түрі бар?
- Алгоритмнің графиктік түрде кескінделуін не деп атаймыз?
- Алгоритмдік тіл дегеніміз не?
- Алгоритмдердің құрылымы қандай?

№ 3 тақырып

АЛГОРИТМДЕР АНАЛИЗІ. АЛГОРИТМДЕРДІҢ КҮРДЕЛІЛІГІН БАҒАЛАУ

АЛГОРИТМДЕР, ДЕРЕКТЕР ҚҰРЫЛЫМЫ ЖӘНЕ ПРОГРАММАЛАУ ТІЛДЕРІНІҢ НЕГІЗГІ ТҮСІНІКТЕРІ.

Жоспары:

- Алгоритмдерді талдау әдістері
- Python бағдарламалау тілі Мазмұны
- Алгоритмдерді талдау әдістері Талдау - алгоритмдерді практикалық мәселелерге тиімді қолдануға жеткілікті деңгейде түсінудің кілті. Әрбір бағдарламаны терең талдау және терең математикалық талдау жүргізу мүмкіндігіміз болмаса да, әртүрлі алгоритмдерді салыстыру және оларды практикалық мақсаттарда қолдану үшін біздің алгоритмдеріміздің негізгі сипаттамаларын зерттеуге көмектесу үшін эмпирикалық тестілеуді және болжалды талдауды қолдана аламыз. Бір есепті шешуге арналған әртүрлі алгоритмдерді салыстыру үшін программаның орындалу уақытын жуық мөлшерде анықтау үшін алгоритмдердің математикалық анализі қолданылады. Қолданбалы есептерді шешу процесінде есеп шығарудың неғұрлым жақын алгоритмін таңдау керек болады. Мұнда алгоритм келесі талаптарды қанағаттандыруы керек:
- Түсінуге жеңіл, программалық кодқа және орындауға, аударуға жеңіл болуға тиіс.
- Компьютер ресурстарын тиімді пайдалану және орындалу жылдамдығы неғұрлым тез болуы керек. Программаның орындалу уақытына келесі факторлар әсер етеді:
- Алғашқы ақпаратты программаға енгізу.
- Орындалып жатқан программаның компилияцияланған кодының сапасы.
- Программаның орындалуында пайдаланылып жатқан жатқан машиналық инструкциялар.

- Сәйкес программа алгоритмінің уақытша күрделілігі. Алгоритмнің Уақыт бойынша күрделілігі дегеніміз – жоспарланған нәтижеге жету үшін алгоритмге орындау қажет болатын қадамдармен өлшенетін алгоритмнің орындалу уақыты. Бұл терминнің синонимі ретінде, көбінесе алгоритмнің орындау уақыты сөзі пайдаланылады. Программаның орындалу уақыты алғашқы мәліметтердің мөлшеріне тәуелді. Мысалға, массивтегі ең кіші элементті табу бұл мәліметтер элементтерін салыстыруға негізгі операция немесе амал. N элементтерін массиві үшін алгоритмге N-1 салыстыру қажет болады және оның орындалу уақыты T-ге пропорционал. Алгоритмдердің уақыт бойынша күрделілігін 0 символика деген қолданылады. Мысалы, егер қандай да бір программаның орындалу уақыты T(n) O(n2) ретке ие болса, бұл дегеніміз қандай да бір C әлде n0 константалар болып, n0-дан үлкен немесе тең болатын n константалар үшін T(n)<=cn теңсіздігі орындалады. Яғни, орындалу уақыты ең аз дәрежеде өсетін программаларды таңдап алуы керек. Бұл дегеніміз, неғұрлым өсу дәрежесі аз болса, компьютердің шығарып отырған есептің өлшемі соғұрлым көп болады деген сөз. Программаның орындалу уақытын теориялық түрде табу күрделі математикалық есеп, оның шешілу үшін бірнеше базалық принциптерді білу керек. Алдымен 3 маңызды ережелерді қарастырамыз:
- Тұрақты көбейткіштер уақыт бойынша күрделі реттік анықтау үшін маңызды емес, яғни O(k*f)=0.
- Көбейткіштер ережесі: екі функцияның уақыт бойнша күрделі реттің көбейткіштері олардың күрделіліктің O(f*g)=O(f)*O(g).
- Қосындылар ережесі: функцияның уақыт бойынша күрдел реттің қосындылары қосылғыштардың ең үлкенінің ретіне тең болады. O(n+n+n)=O(n).
- Алгоритмдер, деректер құрылымы және программалау тілдерінің негізгі түсініктері. Электронды-есептеу техникасының қоғам дамуында қолдану саласының орасан көбеюіне байланысты, алгоритм ұғымы тұрмыста кәдімгі ұғымдардың бірі болып келе жатқандығы белгілі. Алгоритм информатиканың негізгі ұғымдарының бірі. Тиімді алгоритм құрастыру – ЭЕМ-да есеп шығару процесінің қажетті кезеңі. «Алгоритм» сөзіне келетін болсақ, кейбір сөз тіркестерінде мәні жағынан нұсқау, жарлық, қағида, рецепт, ереже, тәртіп, заң, жоба, жол, норма сөздеріне синоним болып келеді. Алгоритм сөзі IX ғасырдағы өзбек ғалымы Әл-Хорезмидің атымен байланысты шыққан. Ол «Арифметикалық трактат» деген еңбегінде арифметикалық амалдарды орындау тәртібін ұсынған. Сөйтіп арифметикалық амалдардың орындалу ережесі, геометриялық фигуралардың салыну ережесі, сөздердің жазылуының грамматикалық ережесі т.с. сияқтылар алгоритм деп аталып кеткен. Алгоритм бағалылығы сол алгоритмді орындаушының (атқарушының) мүмкіндігіне тікелей байланысты. Сондықтан алгоритмді құрастырғанда алдын-ала анықталған орындаушының мүмкіндігін ескеру қажет, яғни орындаушы құрастырылған алгоритмді орындай алатын болуы керек.

1. Python бағдарламалау тілі Python-бір мезгілде қарапайым және қуатты объектілі-бағытталған бағдарламалау тілі болып табылады. Ол, жоғары деңгейдегі деректер құрылымын қамтамасыз ететін, талғампаздық синтаксисі бар және динамикалық теруді пайдаланады, ол түрлі қосымшалар арқылы бірнеше платформаларында жұмыс істеу үшін арналған тамаша тіл. Python - бүкіл әлем бойынша түрлі мақсаттар -деректер базасын және сөз өңдеу үшін кең таралған әмбебап тіл,ойындарға интерпретатор қосу,және де GUI-ді бағдарламалау және жылдам прототип құру (RAD)үшін арналған тіл. Сонымен қатар Python-INTERNETжәне WEB қосымшаларын бағдарламалау үшін пайдаланылады. Python бай стандартты кітапханадан, және модульдердің бай жиынтығынан тұрады.Python мен қосымшалар ең танымал және үлкен фирмалар пайдаланып жазылған, мысалға алып қарайтын болсақ: IBM, Yahoo, Google.com, Hewlett Packard, Infoseek, HACA, Red Hat, CBS MarketWatch, Microsoft. Бұл тілде: · Mailman - тарату тізімдерінің менеджері (Тарату тізімін басқару), жоба адресаттар тізімдерінің (GNU)ресми менеджері болған. · Медуза - HTTP, FTP, NNTP, XML-RPC секілді сенімді өнімділігі жоғары TCP / IP серверлер үшін арналған архитектура. · Zope - кең танымалдылыққа ие болған бағдарлама-Web қосымшалар сервері (Web бағдарлама сервері). Python - бұл сізге керек. Python қарапайым, бірақ ол құрылымдау және басқаға қарағанда үлкен бағдарламалар бойынша қолдау үшін әлдеқайда нақты программалау тілі болып табылады. Екінші жағынан, қателерді өңдеу үшін жақсы және өте жоғары стандарт тілдік табылатын, икемді массивтер және сөздіктер ретінде кіріктірілген жоғары деңгейдегі деректер түрлері бар бағдарлама.Көптеген нәрселер Python-да жасалады. Басқа қосымшаларда пайдалануға болады,модульдер ішінде бағдарламаны бөлуге мүмкіндік береді. Python-ды сіз өз бағдарламаларыңыз үшін негіз ретінде, немесе тілді зерттеу мысалдар ретінде пайдалануға болады. Стандартты модульдер түрлі графикалық кітапхана файлдар, жүйе қоңыраулар, желілерге қосылу, тіпті интерфейстердің жұмыс істеу үшін құралдар ұсынады. Python -уақытты айтарлықтай аз жұмсау үшін берілген тіл.

PYTHON БАҒДАРЛАМАЛАУ ТІЛІНІҢ ТАРИХЫ

Python программалау тілі 1980 жылы ойластырылған, және оның құру Нидерландыда математика және информатика орталығында Гидо ван Россумның көмегімен 1989жылдың желтоқсанынан бастады. Python тілі операциялық жүйені ерекшелеп өңдеу және өзара іс-қимыл қабілетті бағдарламалау тілінде Ван Rossum негізгі авторы Python-мен осы күнге дейін тілді дамытуға қатысты шешім қабылдауда маңызды рөл атқаруын жалғастыруда. Python 2.0 нұсқасы 16 қазан, 2000 жылғы шығарды, және көптеген жаңа ірі мүмкіндіктерді қамтитын болды,осындай толық қоқыс жинау және Unicode қолдау ретінде қолданылды. Алайда, барлық өзгерістер ең маңызды тіл дамыту және оның құру неғұрлым мөлдір процесіне көшу процесінде өзгеруі болды.Ал Python-ның 3.0 алғашқы нұсқасы тестілеуден ұзақ уақыт өткеннен кейін 2008 жылы і желтоқсанда шығарылды. Бұл жаңа редакцияда мүмкіндіктердің көбі Python 2.6 және Python 2.7 сыйысымды. Негізгі және аралық нұсқалардың уақыты: · Python 1.0-Қаңтар 1994 · Python 1.5 - 31 желтоқсан 1997 · Python 1.6-5 қыркүйек 2000 · Python 2.0 - 16 қазан 2000 · Python 2.1 - 17 сәуір 2001 · Python 2.2 - 21 желтоқсан 2001 · Python 2.3-29 шілде, 2003 · Python 2.4 - 30 қараша 2004 · Python 2.5-19 қыркүйек, 2006 · Python 2.6-1 қазан, 2008 · Python 2.7-3 шілде, 2010 · Python 3.0-3 желтоқсан, 2008 · Python 3.1-27 маусым 2009 · Python 3.2-20 ақпан, 2011 · Python 3.3-29 қыркүйек, 2012 · Python 3.4-16 наурыз, 2014

PYTHON-НЫҢ БАСҚА ТІЛДЕРГЕ ӘСЕРІ

Салыстырмалы түрде кеш пайда болған Python программалау тілдерінің түрлі әсерінен құрылды: · ABC - операторлар тобы үшін шегініс, жоғары деңгейдегі деректер құрылымын (картасы) (Python шын мәнінде ABC жобалау кезінде жасалған қателерді түзету әрекеті ретінде құрылды); · Modula-3 - пакеттер, модульдер басқа функцияларынды пайдалана отырып, бірлесіп әрекет ету.(бұл да Common Lisp әсер) ; · C, C ++ - кейбір синтаксистік конструкциялар (Гидо ван Rossum жазғандай - Python үшін C бағдарламашылар арасында наразылық туғызып қалмас үшін ол, C конструкциясыныңбасқаша дизайнын пайдаланды); · Smalltalk - объектілі-бағытталған бағдарламалау; · Lisp - функционалдық бағдарламалау кейбір ерекшеліктері (lambda, map, reduce, filter және баскалары); · Fortran - күрделі арифметикалық массивтер тілімі; · Miranda - тізім-өрнек; · Java - logging, unittest, threading модульдері (модуль бастапқы мүмкіндіктерінің бөлігі іске асырылмаған) xml.sax стандартты кітапхана және ерекшеліктерді іске асыру,және де @ декораторын қолдану үшін. Бейімделген Python және барлық дерлік белгілі платформа жұмыс істейді - ККП-дан мэйнфреймов. Microsoft Windows үшін порттары, барлық дерлік нұсқалары (FreeBSD және Linux қоса алғанда) UNIX, Plan 9, Mac OS және Mac OS X, Iphone OS 2.0 немесе одан жоғары, Palm OS, OS / 2, Amiga, HaikuOS, AS / 400, тіпті OS бар / 390, Windows Mobile, Symbian және Android. Платформаның ескіруі тілге көмек беруін тоқтатады. Мысалы, 2,6 Windows 95, Windows 98 және Windows ME [18] қолдау төмендеді. Алайда, бұл платформаларында, сіз Python алдыңғы нұсқасын пайдалануға болады - қазіргі уақытта қатты (олардың жүзеге түзетулер үшін) 2,3-ден Python нұсқасын қолдайды. Осылайша, барлық негізгі платформалардан көптеген жүйелердің айырмашылығы Python осы нақты платформа технологияларды (мысалы, Microsoft COM / DCOM) қолдау бар. Сонымен қатар, Python B.M. Java арнайы нұсқасы бар - Jython, Java қолдайтын кез келген жүйесінде іске қосу үшін интерпретаторы беретін, Java сыныптар осылайша Python тікелей пайдаланылуы үшін тіпті жазылуы мүмкін. IronPython және Python.Net -Сондай-ақ, кейбір жобалар платформасында Microsoft .NET, негізгі біріктіруді қамтамасыз етеді.

PYTHON-НЫҢ ТҮРЛЕРІ МЕН ДЕРЕКТЕР ҚҰРЫЛЫМЫ

Python-ның айнымалы түрі тек бағдарламаны орындау кезінде анықталады, динамикалық теруді қолдайды. Сондықтан оның орнына «айнымалы тағайындау» шамамен айтуға жақсы «кейбір атымен міндетті күші құндылықтар». Python-ның кірістірілген түрлері: логикалық, жолды, Unicode-жолды, бүтін, еркін-дәлдігін, қалқымалы нүктелі нөмірін, комплекс санды, және басқалар. Python-ның жаңа түрін қосу, сіз класс (класс) жазыңыз, немесе (мысалы, C -жазылған) кеңейту модулінің жаңа түрін анықтауға болады. Сыныптар мұралық (бір және бірнеше) және метобағдарламалық әрекетті қолдайды.Ең кіріктірілген түрлері мұралық болып есептелінеді. Барлық нысандар сілтеме және атом болып бөлінеді. Атом бойынша INT және long күрделі және кейбір басқалар болады. Атом заттарды тағайындау кезінде ғана объектіге анықтамалық көрсеткіш үшін көшіріледі, ал олардың мәні көшіріледі, сондықтан бірдей мәнді тағайындау кезінде екі айнымалы қолданылады. Анықтамалық нысандар өтпелі және даусыз болып табылады. Айнымалы - Мысалы, жолдар және луын өзгермейтін және тізімдер, сөздіктер және басқа да көптеген нысандар болып табылады. Python жылы Tuple, шын мәнінде, өзгеріссіз тізімі болып табылады. Көптеген жағдайларда, луын жылдам тізімдері сондықтан сіз ретін өзгертуді жоспарлап болмаса, оларды пайдалану үшін үздік болып табылады.

PYTHON-НЫҢ МҮМКІНДІКТЕРІ

Lisp және Прологта сияқты, Python интерпретаторы операторлар пернетақтадан енгізіледі, онда интерактивті режим, дереу орындалады бар, және нәтижесі (REPL) көрсетіледі. Бұл режим бастаушы ғана емес, қызықты, бірақ сондай-ақ, сіз басты бағдарлама оны пайдаланар алдында интерактивті кез келген аймақ кодын тексеру, немесе жай ғана функцияларын үлкен жиынтығы бар калькулятор ретінде пайдалануға болады деп есептейді тәжірибелі бағдарламашылар. Интерактивті режимде Python-дағы байланыс осылай көрсетіледі:

```
>>> 2 ** 100 # возведение 2 в степень 100
```

1267650600228229401496703205376L

```
>>> from math import * # импорт математических функций
>>> sin(pi * 0.5) # вычисление синуса от половины пи
```

1.0

```
>>> help(sorted) # помощь по функцииsorted
```

Help on built-infunction sorted in module __builtin__:

```
sorted(...)
sorted(iterable, cmp=None, key=None,reverse=False) --> new sorted list
```

Интерактивті режимде PDB () (көмек үшін деп аталатын) көмек жүйесі қол жетімді. Анықтамалық жүйесі олар құжаттама жолдың қамтамасыз етілді, тек егер модульдермен функциялар үшін жұмыс істейді.

ҚОСЫМША(208-218)

TRACEBACK- объектілермен жұмыс істеуге арналған модуль.Бұл модуль traceback обьектілері үшін нысандардар мен деректерді, өндіру және пішімдеу үшін арналған. Ол сізге толық қабықтың әрекетін модельдеуге мүмкіндік береді және орындалуын бақылау жоғалтпай қате туралы хабарлар үшін пайдалы болуы мүмкін. Traceback модулі келесі функцияларды анықтайды:

```
print_tb(traceback [, limit [, file]])
```

traceback обьектінен шыққан ақпаратты limit дәрежесіне дейін жеткізеді. Егер limit аргументі None аргументіне тең немесе одан төмен болса, онда мұндай жағдай да sys.tracebacklimit үшін ақпаратты көрсетеді. Барлық ақпарат file -да көрсетіледі. (әдіс объектісі write() болуы керек) немесе стандартты шығару файлында көрсетіледі.

```
print_exception(type, value, traceback [, limit [, file]])
```

Ол) (функциясы print_tb сияқты жұмыс істейді, бірақ traceback объектінің шығуы ақпаратқа қосымша мынадай әрекеттерді орындайды: · •Егер traceback аргументі None-ға тең болмаса, онда'Traceback (most recent call last):'деген жазу шығады. · Болған жағдай туралы ақпарат шығарады (type и value;болған жағдайдың түрін және мағынасын сипаттайтын обьект болуы тиіс) · Егер type is SyntaxError форматы бар болса, символы '^' бар синтаксистік қате аудармашының қате орнын көрсете отырып оның желісін көрсетеді. Осылайша, аудармашының қате туралы хабарламасы көрсетіледі.

```
print_exc([limit [, file]])
```

Бұл функцияның шакырылуы ‘print_exception(*(sys.exc_info() + (limit, file))) осы шақыруға эквивалентті.

```
print_last([limit [, file]])
```

Бұл функцияның шакырылуы ‘print_exception(sys.last_type, sys.last_value, sys.last_traceback, limit, file)’ осы шақыруға эквивалентті.

```
print_stack([frame [, limit [, file]]])
```

Мониторлар және бағдарламаның (модуль __main__) түбірі кадрдың стека кадрдың стека жолын көрсетеді. Дәлел print_stack жұмыс істеуі стек кадры блоктар тең болуы үшін, жоқ болса немесе жақтау None болса () деп аталды. Қосымша дәлелдер) (файлды шектеу және функциясы print_exception сол сияқты мағынасы бар.

```
extract_tb(traceback [, limit])
```

'(filename, line_number, function_name, мәтін) «(әр элемент келесі ендірілген нысан traceback тиісті стек кадры болып табылады) тізіміне, нысан traceback қамтылған жол ақпаратты қайтарады, онда файл аты - filename, line_number - жолдың рет нөмірі file, function_name - функцияның атауы және мәтіндік - басында және аяғында, немесе бірде-бір кем бос (бос) қысқартуға көзі желісі. Қосымша дәлел) (функциясы print_exception бірдей шекті мәні бар.

```
extract_stack([frame [, limit]])
```

Бағдарламаның түбіріне көрсетілген немесе ағымдағы кадрдың стека жолды сипаттайтын ақпаратты қайтарады. Қайтар-мағына) extract_tb (қайтарылған құны бірдей пішімі бар. Қосымша дәлелдер жақтау мен шегі (функциясы print_stack сияқты бірдей мәні бар)

```
format_list(list)
```

Пішімдеуден алынған дайын өнімнің тізімі (мысалы, writelines файл объектісі) желілерінің тізімін қайтарады - нәтиже функциясы extract_tb () немесе extract_stack (). Әрбір жол дәлел ұқсас көрсеткішімен тізіміне элементі үшін сәйкес оралады. Тиісті мәндер (көзі желісі) соңғы элементі None емес, егер әрбір жолы, жолдың аяқталады, және бірнеше таңбаларды жаңа жолды болуы мүмкін.

```
format_exception_only(type, value)
```

(writelines файл объектісі) шығу үшін дайындалған ақпаратты жолдарының тізімін қайтарады. Қайтарылған тізімінде Әрбір жол аяқталғаннан кейін тоқтатылады. type мен value табылansy нысандар болуы керек. Әдетте, тізім тек бір жолды қамтиды. Алайда, SyntaxError ерекше тізіміне үшін қайда синтаксистік қате анықтау туралы бастапқы коды және ақпарат фрагментке қоса алғанда, құрамында бірнеше жолдар, тұратын болады. Ерекшелікті сипаттайтын жол әрқашан соңғы тізімделген болып келеді.

```
format_exception(type, value, traceback [, limit])
```

Ерекшелік ақпарат және ақпараттық қамтамасыз нысан (writelines файл объектісі, мысалы) шығару үшін дайындалған, жолдар тізімін қайтарады. Дәлелдер) (функциясы print_exception тиісті дәлелдер ретінде бірдей мағынасы бар. Қайтарылған тізімінде Әрбір жол жаңа жолға аяқталады және жолдың бірнеше таңбалар болады. Осы желілерін (мысалы, файл нысан writelines әдісімен) шығару) (функциясы қоңырау print_exception бірдей нәтиже береді.

```
format_tb(traceback [, limit])
```

Бұл функцияны шақыру ‘format_list(extract_tb(traceback, limit))’осы функцияға эквивалентті. format_stack(frame [, limit])Бұл функцияны шақыру ‘format_list(extract_stack(frame, limit))’осы функцияға эквивалентті.

```
tb_lineno(traceback)
```

Ағымдағы жолдың нөмірі, нысан traceback. өкілдігі қайтарады. Әдетте, қайтаратын мағына traceback.tb_lineno болып табылады. Алайда, оңтайландыру, ал функция tb_lineno () tb_lineno атрибут, дұрыс емес жаңартылады қосулы кезде әрқашан дұрыс мәнді қайтарады. Келесі мысал қарапайым интерактивті интерпретатордың жүзеге асыруын көрсетеді:

```
import sys, traceback
def run_user_code(envdir):
source = raw_input(">>> ") try:
```

exec source in envdir except: print "исключение в пользовательском коде:" print ’-’*60

```
traceback.print_exc(file=sys.stdout)
```

print ’-’*60

```
envdir = {}
```

while 1:

```
run_user_code(envdir)
```

Неғұрлым күрделі интерактивті интерпретаторы жасау үшін модуль коды анықталған сыныптар пайдаланған дұрыс. Маңызды хабарлама: traceback модулі өңдеуге болатын барлық файлдар мазмұнын сақтайды , пайдаланады. (Мысалы, интерактивті пәрмендерін енгізу арқылы) кез келген тәсілмен пайдаланушы, файлдар түрлі кодты орындауға бұл жағымсыз пайдалану жадтың үлкен көлемін алып келуі мүмкіндік береді. Оны болдырмау үшін,linecache.clearcache пайдаланып кэшін тазалау керек.

```
IMP— import инструкциясы арқылы жүзеге асырылатын операцияларға қол нұсқаулық.
```

Бұл модуль импортын жүзеге асыру үшін пайдаланылатын операциялардың интерфейспен қамтамасыз етеді. Ол мынадай функцияларды анықтайды:

```
get_magic()
```

Байткомпиляторланған файлдар кезінде пайдаланылатын сиқырлы жол мәнін қайтарады. Бұл мән аудармашының түрлі нұсқаларын әр түрлі болуы мүмкін.

```
get_suffixes()
```

Жинақтарын тізімін қайтарады, ‘(suffix, mode, type)’ және модульдер белгілі бір түрін сипаттайды, suffix-жол,файл атын қалыптастыру модулі атауы қосылады.

```
find_module(name [, path])
```

Path жолында аты аталатын модуліді табуға тырысады. Get_suffixes қайтарылған жұрнақтар кез келген файлдарды іздеу (), өз кезегінде, жолын тізімделген каталогтар шығады. Жарамсыз каталог атаулары үнсіз елемейді, бірақ барлық тізім элементтері жолдар болуы тиіс. Дәлел жоқ болса немесе жол None болса, іздеу ресурс (PY_RESOURCE) Macintosh операциялық жүйелерде, немесе тізбе Windows тізімделген файлдар қоса алғанда (C_BUILTIN) енгізілген және салынған (PY_FROZEN) модульдер, арасында бірінші болып, содан кейін каталогтар sys.path тізімделеді. Басында орнатылған индексімен файлды оқу үшін ашылады, Модуль файлында жоқ болса, файл None, жолы болып табылады - модуль атауы және сипаттамасы бос жол (жұрнақтар және режимі) болып табылады. Модуль, ерекшелік ImportError табылған жоқ болса. Басқа ерекшеліктер дәлелдер немесе қоршаған ортаға проблемалар көрсетеді. Бұл функция иерархиялық модуль аттары өңдемейді. P.M табу үшін, бұл, модуль M пакеті P болып табылады, онда П .__ path__ тең дәлел жолы бар () find_module пайдалануға, модуль P тауып жұмыс істеуге болады.

```
new_module(name)
```

name атымен жаңа обьект-модульді қайтарады. Бұл нысан sys.modules тізіміне қосылмайды. Модульдің түрін сипаттау мынадай бүтін тұрақты мәндерді анықтайды: PY_SOURCE Python тіліндегі бастапқы файл. PY_COMPILED Байт компиляторланған файл. C_EXTENSION Динамикалық жүктелетін кітапхана. PY_RESOURCE Macintosh операциялық жүйесіндегі ресурс. PKG_DIRECTORY Пакет каталогы. C_BUILTIN Кірістірілген модуль.

```
PY_FROZEN Кірістірілген модуль (Python тіліндегі модуль,интерпретаторға қосылған).
```

Аудармашы модуль алдыңғы нұсқаларымен сыйысымдылық үшін, сондай-ақ ескірген тұрақтылар мен функцияларын санын анықтайды. Және Rexec - осы құжатта сипатталған мүмкіндіктердің пайдалану Жақсы мысалдар стандартты модульдер (ол стандартты интерфейс қарастырылмауға тиіс осы модуль тек мысал ретінде қазіргі болып табылады) табуға болады. PPRINT - неғұрлым тартымды жолмен деректердің шығарылуы. Бұл модуль Python-да еркін объектілерін пішімделген өкілдігін алуға мүмкіндік береді. Сол сияқты, кіріктірілген repr (), қайтар- мағына әдетте) кіріктірілген функциясы дәлел ретінде пайдалануға болады. Енгізілген нысандар үлкен рұқсат етілген мәндер өкілдігінің ұзындығы болса, әрбір объектінің ұсыну бөлек жолда орналасқан болады. Модуль тек қана бір классты анықтайды:

```
PrettyPrinter([keyword_list])
```

Бұл конструктор анықталған дәлелдерді (keyword_list) қабылдайды.Шығу ағыны (әдісі жазу (бар объект болуы керек)) дәлел пайдаланып орнатуға болады. Шығу ағыны, sys.stdout көрсетілмеген болса. Дәлелдер аттары абзац, тереңдігі және ені пішімін анықтайды. Келесі деңгейде объектілерін ұсыну ауыстырылады '...' - шегініс ойыс (кеңістіктің саны), кірістірілген объектілерін әрбір кейінгі деңгейін анықтайды, әдепкі 1. тереңдігі ұсынылатын болады. Енгізілген нысандардың ең жоғары деңгейін айқындауы болып табылады. Аттас дәлел көрсетілген немесе оның мәні өтірік емес болса, деңгейлері санына шектеулер жоқ. тұсаукесеріне жеке желілерінің ұзындығы ені (әдепкі 80 таңбадан) атындағы дәлел құнының шектелетін болады.

```
>>> import pprint, sys
>>> stuff = sys.path[:]
>>> stuff.insert(0, stuff[:])
>>> pp = pprint.PrettyPrinter(indent=4)
>>> pp.pprint(stuff)
[ [ '',
```

'/usr/local/lib/python1.5', '/usr/local/lib/python1.5/test', '/usr/local/lib/python1.5/sunos5', '/usr/local/lib/python1.5/sharedmodules', '/usr/local/lib/python1.5/tkinter'], '/usr/local/lib/python1.5', '/usr/local/lib/python1.5/test', '/usr/local/lib/python1.5/sunos5', '/usr/local/lib/python1.5/sharedmodules', '/usr/local/lib/python1.5/tkinter']

```
>>> >>> import parser
>>> suite = parser.suite(open('pprint.py').read())
```

```
>>> tup = parser.ast2tuple(suite)[1][1][1]
>>> pp = pprint.PrettyPrinter(depth=6)
>>> pp.pprint(tup) (266, (267, (307, (287, (288, (...))))))
```

Пайдалану модуль жеңілдету үшін, сондай-ақ, әдепкі пішімді пайдаланып бірқатар функцияларды анықтайды:

```
pformat(object)
```

Нысанның жол пішімделген көрсетілімін қайтарады.

```
pprint(object [, stream])Файл ағымын көрсетеді (жолдың сонында символмен) объект объектінің өкілдігі
пішімделеді. Ағыны дәлел, sys.stdout көрсетілмеген болса. Бұл функция мәндерін бақылау үшін интерактивті
пайдаланылуы мүмкін.
>>> stuff = sys.path[:]
>>> stuff.insert(0, stuff)
>>> pprint.pprint(stuff)
[<Recursion on list with id=869440>,
```

'', '/usr/local/lib/python1.5', '/usr/local/lib/python1.5/test', '/usr/local/lib/python1.5/sunos5', '/usr/local/lib/python1.5/sharedmodules', '/usr/local/lib/python1.5/tkinter']

```
isreadable(object)
```

Объектінің нысан ұсынады.Eval функциясын пайдаланып, оны қалпына келтіру үшін пайдаланылуы мүмкін, егер олай болмаған жағдайда ол рекурсивті объектілер үшін 0 қайтарады,Әрқашан 0 қайтарады.

```
>>> pprint.isreadable(stuff)
```

0

```
isrecursive(object)
```

Обьект рекурсивті сілтемелер құраса онда бұл жағдайда 1 қайтарады,ал басқа жағдайларда 0 болып қала береді.

```
saferepr(object)
```

Форматталмаған обьекттіні рекурсивті сілтememен шығарамыз. '<Recursion on type with id=id>', бұл жерде type және id — обьекттің түрі мен идентификаторы. PrettyPrinter экземпляр класстары келесі іс әрекеттерді көрсетеді:

```
pformat(object)
```

Форматталған көрініспен обьектті қайтарады.

```
pprint(object)
```

Көрсетілген конструтивті экземплярды форматталған көрініспен обьектті көрсетеді.

```
isrecursive(object)
```

Обьект рекурсивті сілтемелер құраса онда бұл жағдайда 1 қайтарады,ал басқа жағдайларда 0 болып қала береді.

```
REPR — альтернативті функцияның реализациясы repr()
```

Бұл модуль кіріктірілген функциясы repr үшін объектілерді идеясы ұқсас алуға мүмкіндік береді, бірақ желісі мөлшерін шектейді. Бұл мүмкіндіктер Python бағдарламасы үшін пайдаланылады. Модуль келесі аттарды анықтайды:

```
Repr()
```

Кіріктірілген ұқсас функцияларды іске асыру үшін пайдаланылатын функция, (). Тым үлкен идеяларды генерациялау болдырмау үшін объектілерді әр түрлі типтері үшін шекті мөлшерін қамтиды. aRepr Repr данасы. Төменде сипатталған функция repr (), іске асыру үшін пайдаланылады. Бұл нысанның атрибуттарын өзгертуді, тиісінше,жіберуші функциясы (repr) пайдаланылған мөлшері лимиттерді әсер ете алады.

```
repr(object)
```

Бұл әдісі repr () aRepr данасына жатады. Кіріктірілген функциясы repr қайтарады, бірақ мөлшері шектелген. Repr данасының әр түрлі объектілердегі өкілдіктері қайтару әдістерін мөлшері шектеулер жасап пайдаланылуы мүмкін, бірнеше деректер атрибуттарын қамтамасыз етеді: maxlevel Обьект құрудың ең жоғарғы деңгейі,көрініске қосылады.оның әдеттегі мөлшері 6 ға тең. maxdict maxlist maxtuple Элементтердің ең көп саны, тиісінше, сөздіктер, тізімдер мен жинақтарын ұсыну енгізілуі үшін. Әдепкіде, сөздіктер мен тізімдер алты элементтер мен жинақтарын төрт элементтердің көрінісін қамтиды. maxlong Ұзақ бүтін ескере отырып таңбалар санының ең көбі. Цифрлар ортасынан түседі. Әдепкі мөлшері 40 болып табылады. maxstring Жол ұсыну таңбалар саны ең көбі. Бастапқы функциясы ретінде стандартты көрінісін пайдаланады екенін ескеріңіз. Бастапқы қорытындыда көрініс бақылау жүйесі бар болса, олар бұрмаланған болуы мүмкін. Әдепкіде, таңбалар саны ең көбі 30 болады. maxother Басқа түрлерде көрінетін символдардың максималды саны,оның мөлшері 20 ға тең.

```
repr_typename(object, level)
```

Белгілі бір типті объектілердің ұсыну үшін арналған әдістері. Әдіс атауы кеңістігін префикс 'repr_' қосу арқылы, астыңғы ауыстырылуы онда түрі атынан салынды. Объектілердің өкілдігінің нақты түріне арналған әдістері, (әдістері repr1 шақыру) өздері дәлел деңгейде, бір аз салынған нысандар үшін repr1 () шақыру керек. Сіз объектілерін басқа түрлері қолдау қосу немесе қолдау түрлерінің мінез-өзгертуге болатын Repr алынған сынып, анықтаймыз. Келесі мысал файл объектілерін қолдау іске асыру жолын көрсетеді:

```
import repr
import sys
class MyRepr(repr.Repr):
def repr_file(self, obj, level):
if obj.name in [’<stdin>’,
’<stdout>’,
’<stderr>’]:
return obj.name
else:
return ‘obj‘
aRepr = MyRepr()
print aRepr.repr(sys.stdin)
# выводит ’<stdin>’;
```

Бағдарламалау тілдері төменгі деңгейден жылдар өте келе жоғарғы деңгейлі бағдарламалау тілдеріне дейін дамыған. Осы даму барысында не жаңадан тілдер ойланып табылып немесе бұрын болған тілдерді жетілдіру арқылы жаңа бағдарламалау тілдерін жасаған. ''Бағдарламалар кітапханасы (Библиотека программ; program library)''— 1) бағдарламалардың белгілі бір жүйемен топтастырылған жиынтығы; 2) жүйеде жұмыс істейтін кез келген адамға қатынас құруға болатын стандартты бағдарламалардың жиынтығы. Бағдарламалар кітапханасы бағдарламашылардың қандай да нақты бір мәселені шешуге арналған қажетті дайын алгоритмдер мен бағдарламаларды таңдауына мүмкіндік жасап,

олардың қызметін жеңілдетеді.Python бай стандартты кітапханадан, және модульдердің бай жиынтығынан тұрады.Python мен қосымшалар ең танымал және үлкен фирмалар пайдаланып жазылған, мысалға алып қарайтын болсақ: IBM, Yahoo, Google.com, Hewlett Packard, Infoseek, HACA, Red Hat, CBS MarketWatch, Microsoft.

Бақылау сұрақтары

- Бағдарламалаудың қандай әдісі бағдарламалау процесін жақсартады?
- Құрылымдық бағдарламалаудың қанша бөлігі бар?
- Алгоритмдік тілдерге қандай талаптар қойылады?
- Объекті-бағытталған бағдарламалау түсініктері?
- Процедураға бағытталған тілдер қандай тілдер?
- Компьютермен пайдаланушының диалогын қалай ұйымдастырамыз?

№ 4 тақырып Функцияның өсуі. Стрилинг формуласы. Процедуралық және құрылымдық программалауға кіріспе. Параметрсіз процедуралар. Процедура-функциялар.

Жоспары:

- Функцияның өсуі.
- Стрилинг формуласы

Стирлинг формуласы – n! = 1•2•…•n факториалдарының шамасына жуықталғандығын және n-нің үлкен мәніндегі түрдегі гамма-функцияларды, мұндағы анықтайтын асимптотикалық теңдік. Стирлинг формуласы бойынша факториалды табу қателігі. Басқаша айтқанда, асимптотикалық теңдіктің орнына

Есептеу кезінде салыстырмалы қате n! кем e1 / 12n - 1, сондықтан нөлге ұмтылады шексіз өсуі n. Мысалы, n = 10 Stirling формуласы үшін n береді! ≈ 3 598 700, ал дәлдік 10-ді құрайды! = 3,628,800. Бұл жағдайда салыстырмалы қате 1% -дан аз. Формула Дж. Стирлинга (1730) деп аталды, ол алдымен қарастырылып жатқан өрнектер алынған гамма функциясының, Стирлинг сериясының логарифмін асимптотикалық кеңейтуге берді. Относительная ошибка в данном случае составляет менее 1 %. Формула названа по имени Дж. Стирлинга (1730), который впервые дал асимптотическое разложение логарифма гамма-функции, так называемый ряд Стирлинга, из которого получаются рассматриваемые выражения. Дж. Стирлингтен тәуелсіз осындай түрлендірулерді А.Муавр да алды (1730). N санының факторийі факториялық әрпімен белгіленеді n! Санның факториалы сан бүтін және теріс емес болса ғана маңызды болады.

Элементтері жоқ бос жиынты бір жолмен реттелуі мүмкін, сондықтан нөлдің факторигі бір. Жалпы алғанда, форекория сандар теориясы, ықтималдықтар теориясы, функционалдық талдау, комбинаториканы кеңінен қолдануды, сондай-ақ Тейлор сериясындағы функцияларды кеңейтуді табады. 10-нан аз табиғи сандар үшін фактруктураны есептеу аса күрделі емес, бірақ функцияның найзағайының өсуі сандардың көбеюі сияқты факторларды есептеуді қиындатады.

Компьютерлік есептеулерде негізгі қиындық - бұл n функциясын есептеу нәтижесін көрсету және сақтау. Stirling формуласы Stirling формуласы мұнда n санымен және тұрақты коэффициенттермен жұмыс істейтін кез-келген санның факториалының шамамен шамасын есептеуге мүмкіндік береді. Бұл формула үлкен аралық есептеулерден аулақ. Мәннің дәл есептеуі үшін Stirling формуласы 7 терминді қамтиды, бірақ көбіне бұл терминдер жоқ, ал факторийлік жақындаған.

Бақылау сұрақтары 1.Көмекші программалар дегеніміз не? 2.Көмекші программалардың қандай түрлері бар? 3.Функциялар мен процедуралардың негізгі айырмашылықтары?

№ 5 тақырып Қарапайым рекурсиялар. Есептеудің негізгі эффективті схемалары. Рекурсивті қосалқы программаларды ұйымдастыру. Параметрлер-массивтер және параметр-жолдар. Процедуралық типтер. Параметрлер-функциялар және параметр-процедуралар.

Жоспары:

1. Процедуралық және құрылымдық программалауға кіріспе. Параметрсіз процедуралар. Процедура-функциялар.

Біз Бейсиктегі қосалқы программаларды қолдануды еске түсірейік. Соған ұқсас, Паскальда да сондай құрылым беруге болады. Олар функциялар мен процедуралар түрінде болады. Екі құрылым да жеке үзінді түрінде бейнеленеді. Функциялар. Функцияның ерекшелігі, оған қатынасуға болатын атының болуында және осы ат (осы функцияның орындалуы нәтижесінде) қорытынды мәнге ие. Басқаша айтқанда, функция аты айнымалы ретінде қызмет атқарады. Мұнда функция бірнеше параметрлерге тәуелді болғанмен, ал оның нәтижесі-бір ғана сан екендігін астын сызып айтамыз(әзірше сандық функция туралы). Функцияны сипаттау функцияның аты, ол тәуелді болатын параметрлер тізімі және типі көрсетілген function қызметші сөзімен орындалады:

```
function Функция_аты (var параметрлер тізімі: 1–тип): 2–тип;
```

Мұнда функция аты әдеттегідей (бірінші символы әріп болатын латынның 63-ке дейінгі әрпінен және цифрлардан тұратын символдар жиынтығы), параметрлер тізімі –функция тәуелді болатын(1-тип) типтері көрсетілген айнымалылар тізімі, 2-тип- функция типі(мұнда функцияның қорытынды мәні көрсетіледі) беріледі. Бұдан бөлек, функция бұл да программа сияқты, онда тақыры-бынан басқа, оның ішінде қолданылатын айнымалылардың сипат-тамасы және begin, end қызметші сөздері бола алады. Оның ішінде Паскальдың кезкелген операторларын қолдануға болады. Соңғы оператордың функцияның қорытынды мәнін функция атына меншіктеуі маңызды. Функцияны сипаттау негізгі программа басталғанға дейін, яғни негізгі программаның айнымалыларын сипаттаудан кейін, бірақ негізгі программадағы begin-ге дейін орындалуы керек. Мысал 1. Қабырғалары мен екі диагоналының ұзындықтары белгілі бесбұрыштың ауданын есептейтін программа құрастырамыз. Шешімі: Бесбұрыштың ауданы қабырғалары белгілі үшбұрыш аудандарының қосындысына тең. Әр үшбұрыштың ауданын есептеу үшін Герон формуласын қолданамыз (қабырғаларын шартты түрде x,y,z деп белгілейік). ,

```
Мұнда p=(x+y+z)/2.
```

Паскаль тіліндегі программасы: Program mys10_1; Uses crt;

```
var a,b,c,d,f,g,h,s1,s2,s3,s,sum:real;
function yshb(x,y,z:real):real;
var p:real;
```

begin

```
p:=(x+y+z)/2; yshb:=sqrt(p*(p–x)*(p–y)*(p–z))
```

end; begin clrscr;

```
write('a,b,c,d,f,g,h–қа мән бер:');readln(a,b,c,d,f,g,h);
s1:=yshb(a,b,h);
s2:=yshb(h,c,g);
s3:=yshb(f,g,d);
ssum:=s1+s2+s3;
writeln('Бесбұрыш ауданы=',ssum);
```

readkey; end. Осы программа бойынша есептеу жүргізсек, онда a =1,5, b =1, c =2, d =2,5, f =1, g =2,5, h =2 болғанда, бесбұрыш ауданы =3.90 (шамамен) болады. Процедуралар. Процедураның функциядан басты айырмашылығы, процедура бірнеше параметрлерге тәуелді болып қана қоймай, оның нәтижелік мәні де бірнешеу болуы мүмкін. Сондықтан да параметрлер тізімі шартты түрде мән-параметрлер және айнымалы-параметрлер деген екі типке бөлінеді. Бұлардың бір-бірінен айырмасы мынада, мән-параметрлер мәліметтерді «оған»(процедураға) жеткізуге қызмет етсе, ал айнымалы-параметрлер мәліметтерді «оған» да және «кері» де жеткізуге қызмет жасайды. Сонымен мән-параметрлер берілген мәліметтерді процедураға жеткізуге, ал айнымалы-параметрлер процедура жұмысының нәтижесін негізгі программаға жеткізуге арналған. Негізгі программадан процедураға қатынас оның аты бойынша жасалады. Процедура да негізгі программа басталғанға дейін көрсе-тіледі. Процедураның тақырыбы былай жазылады:

```
Procedure Процедура_аты(var 1–параметрлер: 1–тип; var 2–параметрлер: 2–тип);
Мұнда 1–параметрлер – мән–параметрлер, ал 2–параметрлер – айнымалы–параметрлер, ол міндетті түрде var қызметші
сөзінен басталады. Параметрлердің әрбір түрінің типі(1–тип және 2–тип) көрсетіледі.
```

Мысал 2. . Үш үштік: a1,b1,c1; a2,b2,c2; a3,b3,c3 сандары берілген. Әр үштіктің ең үлкені мен ең кішісін, сонан соң табылған үлкендері мен кішілерінің ішінен олардың ең үлкені және ең кішісін жеке-жеке табыңыз. Паскаль тіліндегі программасы: Program mys10_2; Uses crt;

```
var a1,b1,c1,a2,b2,c2,a3,b3,c3:real;
var max1,max2,max3,min1,min2,min3,max4,min4,max5,min5:real;
procedure maxmin(x,y,z:real; var max,min:real);
```

begin

```
if (x>=y) and (x>=z) then max:=x;
if (x<=y) and (x<=z) then min:=x;
if (y>=x) and (y>=z) then max:=y;
if (y<=x) and (y<=z) then min:=y;
if (z>=x) and (z>=y) then max:=z;
if (z<=x) and (z<=y) then min:=z;
```

end; begin clrscr;

```
write('a1,b1,c1-ді енгіз:');readln(a1,b1,c1);
write('a2,b2,c2-ні енгіз:');readln(a2,b2,c2);
write('a3,b3,c3-ті енгіз:');readln(a3,b3,c3);
maxmin(a1,b1,c1,max1,min1);
maxmin(a2,b2,c2,max2,min2);
maxmin(a3,b3,c3,max3,min3);
maxmin(max1,max2,max3,max4,min4);
maxmin(min1,min2,min3,max5,min5);
writeln('1-үштіктің ең үлкені-',max1,'ең кішісі-',min1);
writeln('2-үштіктің ең үлкені-',max2,'ең кішісі-',min2);
writeln('3-үштіктің ең үлкені-',max3,'ең кішісі-',min3);
writeln('Үлкендерінің ең үлкені-',max4,'ең кішісі-',min4);
writeln('Кішілерінің ең үлкені-',max5,'ең кішісі-',min5);
```

readkey; end. Программада бірмезгілде бірнеше функциялар мен процедура-ларды сипаттауға(қолдануға да) болады. Оларды әдеттегідей бірі-нен кейін бірін тізбектеп жазып орналастырады. Бақылау сұрақтары 1.Рекурсия және рекурсивті функция деген не? 2.Итерация деген не? 3.Динамикалық бағдарламалуда рекурсияның алатын орны?

№ 6 тақырып Деректер типтері. Деректердің құрылымдық және базалық типтері. Қарапайым, реттелген, бүтін, логикалық, символдық, санаулы, тип диапазон, нақты типтер. Олармен жұмыс істеуге арналған стандартты математикалық функциялар

Жоспары:

- Деректер типтері. Деректердің құрылымдық және базалық типтері
- Жиындар. Жиындарға қолданылатын амалдар
- Тізімдер. Ағаштар Мазмұны
- Деректер типтері. Деректердің құрылымдық және базалық типтері Соңғы 20-30 жылда құрылымдалған программалау идеялардың кең таралуы программалаудың теориясы мен практикасына үлкен әсер етіп, сәйкес алгоримдер мен программаларды жасаудағы мәліметтердің типі мен құрлымдарының ролі қайта қарауға алып келді. Осыған байланысты соңғы он жылдықта электронды есептеуіш машиналарды (ЭЕМ) программалық қамтамасыз ету саласында мамандар даярлайтын оқу орындарында мәліметтердің типтері мен құрылымдары пәні пайда болды. Бұл пән ақпараттық технологиялар саласында соған сәйкес программалық қамтамасыз етуге жоғары сапалы мамандар даярлауға мүмкіндік беретін компьютер ғылымы информатиканың фундаментальды (негізгі, түпкі) бөлімі болып саналады. Бұл курстың идеялық мазмұны программалық тілге тәуеді емес, бірақ мәліметтер құрылымын көрнекі түрде белгілеуге болатын Турбо Паскаль тілі қолданылады. Мәліметтер құрылымын кез келген программаның немесе программалық бөлімнің бөлінбес саласы болып саналады. Программаны жасауда нақтылы есепті сипаттайтын мәліметтер жиынын анықтап, есепті шешудің алгоритмін құру керек. Программаның мақсатына қарай мәліметтер әртүрлі күрделі деңгейде болуы мүмкін. Яғни жай типтерден бастап, жеткілікті типтегі күрделі құрылымдарға дейін. Мәліметтер – бұл белгілі бір пішімде берілген және ары қарай пайдалануға арналған қандайда бір жүйені, құбылысты, үрдісті немесе объекті сипаттайтын мағлұматтар. Ақпарат және мәліметтер ұғымдарының арасындағы арақатынас тұрғысынан алып қарағанда: мәліметтер – бұл ақпараттың мазмұнын көрсететін нақты пішім; мәліметтер – бұл

пайдаланушы үшін маңызды және онымен қандай да бір есептерді шешуге қолданылады. Мәліметтерге бірнеше классификациялық мінездемелер меншіктеледі. Олардың ішіндегі ең маңыздысы мәліметтер типі болып табылады. Мәліметтер типі мүмкін мәндер жиынын; олардың өңделу ережелерін (түрлендіру); сақтау кезінде ОСҚ және ССҚ-да олардың орналасу ретін; оларға қатынау ретін анықтайды. Негізгі мәліметтер типтері: стандарттық, қарапайым, шектелген, күрделі құрылымдық және т.б. Мәліметтер типтерінің концепциясы пайдаланушыға мәліметтерді беру, сақтау, қолдану есептерін тиімді шешуде көп мүмкіндіктер береді; мәліметтердің жинақтылығына әсер етеді. Сонымен қатар мәліметтер элементарлық (қарапайым) құрылымданбаған және құрылымданған (күрделі) болып жіктеледі. Элементарлық мәліметтерге символдар, сандар және логикалық мәліметтер жатады, олардың әрқайсысы бір мәнмен және атпен анықталады. Мәліметтерді және олардың арасындағы байланыстарды біріктіретін ақпараттық массивті құрылымданған мәліметтер деп атайды. Біріктірілетін қарапайым мәліметтердің тізімі, олардың мінездемелері, сонымен қатар арасындағы байланыстардың ерекшеліктері мәліметтер құрылымын анықтайды. Күрделі мәліметтер де мәнімен және атымен анықталады. Өңдеу үрдісінде мәндерінің өзгеруіне байланысты (қарапайым және құрылымдық мәліметтер үшін) мәліметтерді айнымалы және тұрақты деп бөледі. Өңдеудің қай кезеңінде қолданылуына байланысты мәліметтерді алғашқы(енетін), аралық, қорытынды(шығатын) мәліметтер деп бөледі. Сақтауда және өңдеуде мәліметтерді беру үш негізгі есепті шешуді қажет етеді: элементарлық мәліметтерді беру әдістерін анықтау; мәліметтердің құрылымға бірігу әдістерін анықтау; мәліметтердің материалдық тасуышта орналасуын анықтау. Мәліметтерді берудің үш деңгейін анықтайды: концептуалдық, логикалық және физикалық. Концептуалдық деңгейде ақпараттық массивтің жалпы құрылымы анықталады, ол мәліметтер моделі деп аталады. Мәліметтер моделдерінің негізгілері: иерархиялық, желілік, реляциялық, объектілі-бағытталған. Қатынастарының мінездемелеріне байланысты мәліметтер құрылымдарының бірнеше жіктелу белгілерін анықтауға болады: мәліметтердің реттері бойынша – реттелген және реттелмеген, қатынастарының мінездемелері бойынша – сызықтық және сызықтық емес, құрылымдағы мәліметтер типтеріне байланысты – біртекті және біртексіз, мәліметтерге қатынау әдістеріне байланысты – тура жететін және тізбектік жететін, ақпараттық массивтің өлшемінің өзгеру мүмкіндігіне байланысты – статикалық және динамикалық. Негізгі күрделі құрылымдар: массив, жиын, кезек, стек, ағаш (екілік ағаш), граф, тізімдер (логикалық жазба). Мәліметтерді ұйымдастыруда үш типті қолданады: сызықтық(тізбектік), кестелік, иерархиялық. Мәліметтер дегеніміз - дербес компьютерде өңдеуге болатын, кез келген ақпаратты сипаттайтын мәнді немесе мәндер жиыны. Мәліметтердің маңызды мінездемесін беретін тип болып саналады. Ол мәліметтерді компьютер жадысына беруде қолданылады және осы мәндерге орындауға болатын амалдар жиыны мен мәліметтер мәндері жиынын анықтайды. Мәліметтерді ұйымдастырудың логикалық және математикалық моделі Мәліметтер құрылымы деп аталады. Мәліметтердің логикалық құрылымы және физикалық құрылымы ұғымы бар. Мәліметтердің логикалық құрылымы дегеніміз - сәйес құрылымның моделі, берілу схемасы (екі өлшемді массив, тік бұрышты матрица). Мұнда әрбір элементте индекс болады. Мәліметтердің физикалық құрылымы дегеніміз - құрылымды компьютер жадысына орналастыру немесе сақтау схемасы (жады ұяшықтарының тізбегі). Мәліметтердің барлық жиынын екіге бөлуге болады: Статикалық және динамикалық құрылым мәліметтері. Статикалық құрылым мәліметтері – жай және күрделі болуы мүмкін. Олар қандай да бір заңдылық бойынша жай құрылымдарда қалыптасады. Құрылым элементтерінің өзара орналасуы мен өзара байланысы әрқашан тұрақты болып қалады. Динамикалық құрылым мәліметтері дегеніміз – ішкі жасалуы қандай да бір заң бойынша қалыптасқан, бірақ элементтер саны олардың өзара орналасуы, өзара байланысы программаның орындалу барысында динамикалық түрде өзгере алатын мәліметтер.

Мәліметтердің типтерін сызықтық құрылымды мәліметтер типтері және сызықтық емес мәліметтер типтері деп бөлуге болады. Сызықтық құрылым мәліметтер типтері – орналасуы бойынша реттелген элементтер тізімін анықтайды. Сызықтық емес құрылым мәліметтер типтері – позициялық реттеусіз элементтерді анықтайды. Тікелей кіру дегеніміз – элементті тікелей таңдау және тізімдегі алдыңғы элементтерге байланыссыз таңдауды айтады

Массив дегеніміз – мәліметтердің бүтін санды индекс көмегімен кіруге болатын бір ғана типтен тұратын құрылымы.

```
A[0], A[1], A[2], …, A[n]
```

Массивтермен жұмыс жасауда элементтерге кіруді ұйымдастыру маңызды әрекет болып саналады. Элементке логикалық деңгейде кіру үшін – массив аты мен элемент индексі керек. Ал физикалық деңгейде жұмыс жасауда элемент адресін массив атымен ізделіп отырған элемент индексі мен оның ұзындығына қарай есептеп шығару қажет болады. Индексті кіру мәліметтері типтері үшін – жазуға кіруге қолданылатын қандай да бір кілт байланыстырады. FLASH кесте – кілтпен байланысты мәліметтерді сақтайды. Кілт – мәліметтерді табу үшін қолданылатын бүтін санды индекске трансформацияланады. Кілт бүтін болуы мүмкін емес. Сөздік деп аталатын мәліметтер құрылымы ассоциация деп

аталатын кілт және мән жұптарының жиынынан тұрады. Мысалға, кілт сөз болуы мүмкін, ал мән сөздің анықтамасын көрсететін сөйлем болуы мүмкін. Сызықтық құрылым мәліметтер типтері – мәліметтерге тізбектей кіруді пайдаланғанда мәліметтердің динамикалық құрылымы болып табылады да, келесі қасиеттерге ие болады:

1. Өлшемнің тұрақсыздығы ;
2. Жадыдағы элементтер құрылымдарының физикалық реттілігінің жоқтығы. Сызықтық тізім – элементтердің кез келген санына ие болады, тізімнің өлшемі тізімдегі элементтерді қосуға және алып тастауға байланысты өзгеріп отырады. Бірінші элементі басында орналасады, соңғы элементі аяғында орналасады және әрбір елемент (соңғысынан басқа) алдында бір элементке ие болады. Реттелген сызықтық тізімде – мәліметтер бір-біріне қатысты реттеліп орналасады. Стек дегеніміз – элементтері тізімнің төбесі деп аталатын бір жағынан ғана қосылатын немесе өшірілетін сызықтық тізімді айтамыз (соңғысы келсе біріншісі кетеді). Шірет (очередь) дегеніміз – тізімнің басынан немесе аяғынан ғана кіруге болатын сызықтық тізім. Элементтер тізімнің соңына қойылады да басынан өшіріледі. Очередь приоритеті дегеніміз – очередь немесе шірет типті тізім, тізімдегі объектіні өшірген кезде ең жоғары приоритетке ие объект анықталады. Файл дегеніміз – байттар тізімі. Ол бір ағымға теңеледі, яғни, бір құрылғыдан екінші құрылғыға ауысып отыратын байттар тізбегі. Тек қана дисктік файлға тікелей кіруді жүзеге асыруға болады. Иррархиялық құрылым – бұл деңгейлері бойынша бөлінетін элементтер жиынындағы, яғни, бір деңгейдегі элемендеңгейде бірнеше ұрпағы болуы мүмкін. Тармақ (дерево) – бұл түпкі немесе тамыр деп аталатын бір ғана көзден тарайтын элементтері бар мәліметтер құрылымы. Пирамида дегеніміз – тармақтың ерекше түрі. Мұнда ең үлкен элемент үнемі түпкі деңгейде тұрады. Топ (группа) дегеніміз – элементтері ешқандай реттеусіз орналасқан сызықтық емес құрылым. Жиын дегеніміз – мәліметтер реттеусіз болғанда және мәліметтердің әрбір элементі өз алдына қайталанбайтын болғанда қолданылатын мәліметтер құрылымы. Граф дегеніміз – төбелер жиынынан және сол төбелерді қосатын байланыстар жиынынан тұратын мәліметтер құрылымы. Желі (сеть) дегеніміз – графтың ерекше формасы. Мұнда әрбір байланыстың өз салмағы болады.
3. Жиындар. Жиындарға қолданылатын амалдар Қандай да бір элементтер тобын жиын деп түсінетінбіз. Мысалы, жазықтықтағы фигуралар (төртбұрыш, шеңбер, ромб, квадрат) жиыны. Жиындарға келесі амалдар қолданатындығы бізге белгілі:
4. жиындарды біріктіру (c=ab). С жиынының әрбір элементі не a жиынының элементі, не b жиынының элементі болады;
5. жиындардың қиылысуы (c=ab). С жиынының әрбір элементі бірмезгілде a мен b жиынының элементі болады;
6. екі жиынның айырмасы (c=a\b) c жиынының әрбір элементі a жиынының элементі болады, b жиынының элементі болмайды;
7. {шеңбер, ромб}{ шеңбер, квадрат}={шеңбер, ромб, квадрат};
8. {шеңбер}{ шеңбер, ромб, квадрат}={шеңбер};
9. {шеңбер, ромб, квадрат}\{шеңбер, квадрат}={ромб} Ескерту. Жиынның элементтері ретсіз орналаса береді, сондықтан, мысалы {2, 10, 9},{9, 10, 2},{2, 9, 10} т.с.с. жиындар бірдей болып есептелінеді. Паскаль тілінде ЖИЫН – деп, бір типтегі ретсіз орналасқан әртүрлі элементтердің шектелген тобын айтады. Жиынға енетін элементтердің типі ретінде стандартты (нақты типтен басқа), саналатын және шектелген типтерді пайдалануға болады. Жиындарды сипаттау былай жүзеге асырылады:
10. var <жиын аты>:set of <негізгі тип>; Мысалы, var жыл:set of 1880..2000; c:set of char;
11. type <тип аты>=set of <негізгі тип>; var <жиын аты>:<тип аты>; Мысалы, Элементтері КОМПЛЕКТ типіндегі мәліметтер болып табылатын КОМПЬЮТЕР жиынын сипаттайық: Type КОМПЛЕКТ (ДИСК, КЛАВИАТУРА, МОНИТОР, МЫШЬ, КОЛОНКА, ПРОЦЕССОР, ВЕНЕЛЯТОР); КОМПЬЮТЕР=SET OF КОМПЛЕКТ; Енді осы типтегі айнымалыларды былай сипаттауға болады: VAR ПРИНТЕР, СКАНЕР, МОДЕМ:КОМПЬЮТЕР; Жиын тұрақтылары мен айнымалыларының мәндері операторлар бөлімінде конструктор көмегімен беріледі. Конструктор – квадрат жақша ішіне орналасқан негізгі типтің элементтері тізімінен тұрады. Мысалы, ФИГУРА:=[РОМБ]; Немесе ФИГУРА:=[ШЕҢБЕР, РОМБ, КВАДРАТ]; M1:=['a', 'b', 'c']; M2:=[1, 3, 2, 5]; M1:=[]; {Бос жиын} Паскаль тілінде жиындарға мынадай амалдар қолданылады: +жиындарды біріктіру; *жиындардың қиылысуы;
12. жиындардың айырмасы; =, <> жиындардың теңдігін, тең еместігін тексеру. Егер a жиынының әрбір элементі b жиынының элементі болса және керісінше b–ның әрбір элементі a жиынының элементі болса, онда a жиыны b жиынымен тең болады. Басқа жағдайда a және b жиындары тең болмайды. <=, >= жиындардың ішкі жиынға енуін тексеру. Егер a жиынының барлық элементтері бір мезгілде b жиынының элементтері болып табылса, онда a жиыны b жиынына ішкі жиын болады, яғни b жиынына енеді дейміз. Бұны былай сипаттаймыз: a<= b (немесе b>=a) in - элементтің жиынға жататындығын тексеру. Бұл амал (c in a) c негізгі типтің элементін a жиынына жататындығын тексереді. 1-есеп. Символдық типтегі үш жиын өз конструкторларымен берілген: y1=['a', 'b', 'd', 'r', m']; y2=['r', 'a', 'h', 'd']; y3=['a', 'r']. Мына формуламен анықталатын жаңа жиын құрыңдар: x=(y1\y2) (y1\y2) Алынған жиынды жауапқа

шығарыңдар және y3 жиыны x жиынының ішкі жиыны болатындығын анықтаңдар. program prog_1; var y1, y2, y3, x : set of char; c : char; begin y1:=['a', 'b', 'd', 'r', m']; y2:=['r', 'a', 'h', 'd']; y3:=['a', 'r']; {Формула бойынша жиын құру және жауапқа шығару} x=(y1*y2)+(y1-y2); write('x жиыны ='); for c:='a' to 'r' do if c in x then write(c); writeln; {y3 жиыны x жиынының ішкі жиыны екендігін тексеру} if y3<=x then write('y3 жиыны x жиынының ішкі жиыны') else write('y3 жиыны x жиынының ішкі жиыны емес') end. Жауабы: x жиыны = abdmr

y3 жиыны x жиынының ішкі жиыны 2-есеп. 1..20 бүтін сандар жиынынан: 6-ға қалдықсыз бөлінетін сандар жиынын; 2-ге немесе 3-ке қалдықсыз бөлінетін сандар жиынын ажыратып алыңдар. program prog_2;

```
const   n=20;
var n2, n3, n6, n23 : set of char;
```

k : integer; begin

```
    n2:=[];
    n3:=[];
```

for k:=1 to n do begin

```
        if   k mod 2 = 0    then n2:=n2+[k];
        if   k mod 3 = 0    then n3:=n3+[k];
```

end.

```
    n6:=n2*n3;
    n23:=n2+n3;
writeln('6-ға бөлінетін сандар: ');
```

for k:=1 to n do

```
        if   k  in  n6   then   write(k);
```

writeln;

```
writeln('2-ге немесе 3-ке бөлінетін сандар: ');
```

for k:=1 to n do

```
            if  k  in  n23   then   write(k);
```

end. Жауабы:

6-ға бөлінетін сандар: 12 18 2-ге немесе 3-ке бөлінетін сандар: 2 3 4 6 8 9 10 12 14 15 16 18 20

1. Тізімдер. Ағаштар Байланған сызықтық тізімдер. Тізім дегеніміз – сандары өзгеруі мүмкін элементтердің жиынтығы. Ол шынжырдағы шығыршық іспеттес. Тізімнің ұзындығы жаңа шығыршықтар қосу арқылы көбеюі мүмкін. Жаңа элементтер тізімге тізімнің бір жерін үзіп, соған жаңа элемент қосып, қайтандан жалғау арқылы кіргізуі мүмкін. Элементтер тізіміне сол сияқты тізімді үзіп, бір шығыршықты алып қайта жалғап қойған сияқты алынып тасталады. Тізімдер үш түрлі болады:
2. Бір байланысты.
3. Циклық

4. Екі байланысты Бір байланысты сызықтық тізім. Бір байланысты сызықтық тізім екі әдіспен жүзеге асуы мүмкін: 1-массивтер негізінде 2-көрсеткіштер көмегімен Тізімді массив негізінде жүзеге асыруда тізім элементтері массивінің қатарлас ұяшықтарында орналасады. Бұлай берілуі тізімнің мазмұнын жеңіл қарап, оның аяғына жаңа элементтер қосуға мүмкіндік береді. Бірақ жаңа элементті тізімнің ортасына қосу қажет болғанда оған арнап кейінгі элементтердің барлығын массивтің аяғына қарай бір позиция жылжытып, бір орын ашу керек. Сол сияқты массив элементін алып тастау үшін де босаған ұяшықты жою үшін элементтің барлығын солға қарай жылжыту керек болады. Бірақ бұлай істеудің бірнеше кемшіліктері бар:
5. Тізімді массив негізінде жүзеге асыру орындалмас бұрын тізімнің максималды өлшемін көрсетуді талап етеді.
6. Массив негізіндегі тізімдерді жүзеге асыру компьютер жадысының үлкен көлемін қажет етеді. Тізімді жүзеге асыру үшін тізімнің ең үлкен мүмкін мөлшеріне жететіндей етіп жады көлемін бөлу қажет. Көрсеткіштер көмегімен жүзеге асыру. Тізімді сақтау үшін жадының үзіліссіз аймағын пайдаланудан босатады. Сондықтан бұл жағдайда элементтерді қосқанда немесе өшіргенде элементтер ары-бері жылжытуды қажет етпейді. Алайда, бұл жағдайда көрсеткіштерді сақтайтын қосымша жады керек болады. Тізімнің әрбір бөлігі – мәліметтер өрісінен және тізімдегі келесі элементтерді көрсететін көрсеткіштен тұрады. Байланған тізімнің құрылымы келесі түрде берілуі мүмкін:

Мұндағы тізбектегі соңғы элементтің көрсеткіш өрісі 0 деген мәнге ие болады. Тізімдә сипаттау үшін үш түрлі көрсеткіш пайдаланған. Тізбектің бірінші элементінің көрсеткіші - head. Тізбектің соңғы элементінің көрсеткіші – rear. Тізбектің ағымдағы элементінің көрсеткіші – ptr. Head деген көрсеткіші бар элементсіз тізім 0 деген мәнге ие болады, себебі тізбектің кез келген элементіне кіру үшін басынан бастап қарау қажет. Тізімдермен жүргізілетін операциялар. Тізімді қалыптастыру. Бұл жағдайда тізімге элементті қосу аяғынан басталады немесе аяғында жүргізіледі.

1. Байланған тізімнен өту.
2. Тізімді тазалау.
3. Элементті енгізу схемасы. Реттелген тізімді жасау. Көптеген қосымшаларда мәліметтердің реттелген тізімін қолдану қажет болады. Оның элементтері өспелі немесе кемімелі ретте орналасқан болуы тиіс. Жаңа элементтің дұрыс орнын анықтау үшін қыстыып қою алгоритмі тізімді сканерлеп өтеді. Одан кейін барып қою жүзеге асады. Мысалы: 60,65,74,82 тізімі берілген.

осы тізімге 50,70,90 сандарын қою керек.

1. 50 санын тізімге енгіземіз.

Тізімдегі бірінші элемент – 60. Оның мәні 50-ден үлкен, сондықтан жаңа элемент тізімнің басына қойылады.

1. 70 санын тізімге енгізу керек. Мұнда 74 саны 70-тен үлкен тізімдегі бірінші элемент. Сондықтан қыстырып қою мына брйынша жүзеге асырылады:

1. 90 санын тізімге енгіземіз. Тағы барлық тізім тексеріледі. Онда элементтің ішінде мәні 90-нан кіші элемент жоқ. Сондықтан 90 тізімнің ең соңына қоыслады.

Екі байланысты сызықтық тізім. Кейбір есептерде тізім бойымен екі бағытты қозғалыс жасау керек болады. Бұл дегеніміз – тізімнің әрбір эелементіне бір өрістің орнына екі байланыс өрісін енгізгенде мүмкін болады. Бұл өрістерде жаңағы элемент алдыңғы және артқы элементтердің адрестері болады. Сызықтық тізім екі байланысы бар немесе екі байланысты сызықтық тізім деп аталады. Егер бұл тізімнің әрбір элементі екі көрсеткішіне ие болса, яғни алдыңғы және артқы элементке арналған көрсеткіштер. Екі байланысты тізімді графиктік түрде былай көрсетуге болады.

2 байланысты тізімнен элементті алып тастау схемасы:

Циклдық тізімдер. Осы уақытқа дейін біз тізімнің соңғы элементін көрсеткіш мәнін 0 деген сызықтық тізімді қарастырдық. Егер тізімнің соңғы элементінің көрсеткіш мәні тізімнің басының адресіне ауыстыратын болсақ, циклдық тізім аламыз. Көптеген кәсіби программистер байланған тізімдерді жүзеге асыру үшін келесі циклдік модельді қолданады:

Циклдық тізімнің бір жетістігі бұл элементтерге кез келген жерде элементтерге кіруге болатындығында. Ал кемшілігі тізіммен жүрген кезде байқамаса ақырсыз циклге кіріп кетуге болады. Циклдық тізім сызықтық тізімге қарағанда құрылымы неғұрлым иілгіш болып келеді. Ол тізімде жүруді кез келген жағдайда бастауға мүмкіндік береді және оны бастапқы позицияға дейін жалғастыруға болады. Ағаштар Тармақтар. Тармақ дегеніміз – иеархиялық құрылым құратын элементтер мен қатынастардың жиынтығы. Тармақтың элементтері түйін деп аталады. Ағаш тәрізді құрылым түпкі деп аталатын алғашқы бір түйіннен басталатын көптеген түйіндер жиынтығын құрайды.

Мұнда ағаштың түбірі дегеніміз – ары қарай түбірі болмайтын төбесі, яғни мұндағы А. Әрбір тармақтың бір ғана түбірі болады. Тармақтың ең төменгі түйіндері E, F, G, H жапырақ деп аталады. Егер тармақтың элементі жапырақ болмаса, онда оны тармақтың ішкі түйіні немесе төбесі деп аталады. Бос тармақ дегеніміз – ешқандай төбесі жоқ тармақты атаймыз. Балалық төбесі деп – тармақта орналасқан өзінің ата-аналық төбесінен төмен қарай тұрған және онымен тікелей байланысқан төбені атаймыз. Мысалға, егер і төбесі F төбесіне қатысты балалық төбесі болса, онда F төбесі I төбесіне қатысты ата-аналық төбе деп аталады. Балалық түйіндер және олардың балалық түйіндері ұрпақ деп аталады. Мысалға, E, F және I, J B – түйінінің ұрпақтары. Тектері бұл түйіннің ата-аналары немесе ата-әжелері. Тармақтың әрбір түйіні осы түйіннен және барлық ұрпақтарынан тұратын бұтақтың түбірі болып табылады. Мысалы, B төбесі E,F,I,J деңгейінің түйіні. Мысалға, F түйіні F,I,J түйіндерінен тұратын бұтақтың түбірі болып табылады. G түйіні ұрпақсыз бұтақтың түйіні. А түйіні өзі тармақ болып табылатын бұтақтың түбірі. Кез келген түйіннен түйіннен бастап оның ұрпақтарына қарай жүру белгілі бір жол бойымен жүзеге асады. Мысалға, А түбірінен бастап F түйініне дейінгі жол A,B,F төбелерінен өтеді. Кез келген түйіннен оның ұрпақтарына баратын жол жалғыз ғана болады. Түйіннің деңгейі дегеніміз – ол түбірден осы осы түйінге дейінгі жолдың ұзындығы. Мысалға, А түбірінің деңгейі 0-ға тең. Түбірдің әрбір баласының деңгейі 1-ге тең. Мысалға F түйіні 2 деңгейдегі ұзындығы 2-ге тең түйін болып табылады. Тармақтың тереңдігі дегеніміз – оның кез келген түйінінің максимальды деңгейі немесе тармақтың тереңдігі дегеніміз ол түбірден бастап түйінге дейінгі ең ұзақ жолдың ұзындығы. Тармақ тәрізді құрылымды беру әдістері. Тармақтарды берудің 4 түрлі әдісі бар:

1. графтар
2. кірістірілген жақшалар
3. кірістірілген жиындар
4. кесінді тізбектер. Екілік ағаш. Толықтырылған екілік ағаш деп келесі қасиеттері бар ағашты есептейміз. Екілік ағаштардың қасиеттері:
5. Ағаштың барлық парақтық элементтері төменгі немесе жоғарғы сатыларда орналасады;
6. Сатыдағы барлық парақтар сатыны сол жақтан бастап толтырады;
7. Барлық сатылар элементтермен толық түрде толықтырылады.

Сол жақтан оң жаққа қарай толықтырылған ағаш:

Толықтырылған ағаштың ішіндегі элементтерінің нумерациясы Түйіннің атасының номері мұнда:

Массивте аталарды және балаларды табу мысалысы

Бақылау сұрақтары 1.Көмекші программалар дегеніміз не? 2.Көмекші программалардың қандай түрлері бар? 3.Функциялар мен процедуралардың негізгі айырмашылықтары?

№ 7 тақырып Көрсеткіштер. Жадының динамикалық таралуы. Динамикалық жады түсінігі. Көрсеткіштерді баяндау.

Жоспары:

- Көрсеткіштер және бағдарламалаудағы оларды қолдану
- Екі байланысты сызықтық тізім
- Циклдық тізімдер
- Көрсеткіштер мен тұрақтылар

- Көрсеткіштер және бағдарламалаудағы оларды қолдану Тізімді сақтау үшін жадының үзіліссіз аймағын пайдаланудан босатады. Сондықтан бұл жағдайда элементтерді қосқанда немесе өшіргенде элементтер ары-бері жылжытуды қажет етпейді. Алайда, бұл жағдайда көрсеткіштерді сақтайтын қосымша жады керек болады. Тізімнің әрбір бөлігі – мәліметтер өрісінен және тізімдегі келесі элементтерді көрсететін көрсеткіштен тұрады. Байланған тізімнің құрылымы келесі түрде берілуі мүмкін: head ptr rear NULL Dann1 Next

Dann2 Next

Dann N-1 next

dannN next

Мұндағы тізбектегі соңғы элементтің көрсеткіш өрісі 0 деген мәнге ие болады. Тізімдә сипаттау үшін үш түрлі көрсеткіш пайдаланған. Тізбектің бірінші элементінің көрсеткіші - head. Тізбектің соңғы элементінің көрсеткіші – rear. Тізбектің ағымдағы элементінің көрсеткіші – ptr. Head деген көрсеткіші бар элементсіз тізім 0 деген мәнге ие болады, себебі тізбектің кез келген элементіне кіру үшін басынан бастап қарау қажет. Тізімдермен жүргізілетін операциялар. Тізімді қалыптастыру. Бұл жағдайда тізімге элементті қосу аяғынан басталады немесе аяғында жүргізіледі.

- Байланған тізімнен өту.
- Тізімді тазалау.
- Элементті енгізу схемасы. Реттелген тізімді жасау. Көптеген қосымшаларда мәліметтердің реттелген тізімін қолдану қажет болады. Оның элементтері өспелі немесе кемімелі ретте орналасқан болуы тиіс. Жаңа элементтің дұрыс орнын анықтау үшін қыстыып қою алгоритмі тізімді сканерлеп өтеді. Одан кейін барып қою жүзеге асады. Мысалы: 60,65,74,82 тізімі берілген.

осы тізімге 50,70,90 сандарын қою керек.

1. 50 санын тізімге енгіземіз.

Тізімдегі бірінші элемент – 60. Оның мәні 50-ден үлкен, сондықтан жаңа элемент тізімнің басына қойылады.

1. 70 санын тізімге енгізу керек. Мұнда 74 саны 70-тен үлкен тізімдегі бірінші элемент. Сондықтан қыстырып қою мына брйынша жүзеге асырылады:

1. 90 санын тізімге енгіземіз. Тағы барлық тізім тексеріледі. Онда элементтің ішінде мәні 90-нан кіші элемент жоқ. Сондықтан 90 тізімнің ең соңына қоыслады.

Екі байланысты сызықтық тізім.

---

2 ЕКІ БАЙЛАНЫСТЫ СЫЗЫҚТЫҚ ТІЗІМ

Кейбір есептерде тізім бойымен екі бағытты қозғалыс жасау керек болады. Бұл дегеніміз – тізімнің әрбір әелементіне бір өрістің орнына екі байланыс өрісін енгізгенде мүмкін болады. Бұл өрістерде жаңағы элемент алдыңғы және артқы элементтердің адрестері болады. Сызықтық тізім екі байланысы бар немесе екі байланысты сызықтық тізім деп аталады. Егер бұл тізімнің әрбір элементі екі көрсеткішіне ие болса, яғни алдыңғы және артқы элементке арналған көрсеткіштер. Екі байланысты тізімді графиктік түрде былай көрсетуге болады.

2 байланысты тізімнен элементті алып тастау схемасы:

Циклдық тізімдер.

---

3 ЦИКЛДЫҚ ТІЗІМДЕР

Осы уақытқа дейін біз тізімнің соңғы элементін көрсеткіш мәнін 0 деген сызықтық тізімді қарастырдық. Егер тізімнің соңғы элементінің көрсеткіш мәні тізімнің басының адресіне ауыстыратын болсақ, циклдық тізім аламыз. Көптеген кәсіби программистер байланған тізімдерді жүзеге асыру үшін келесі циклдік модельді қолданады:

Циклдық тізімнің бір жетістігі бұл элементтерге кез келген жерде элементтерге кіруге болатындығында. Ал кемшілігі тізіммен жүрген кезде байқамаса ақырсыз циклге кіріп кетуге болады. Циклдық тізім сызықтық тізімге қарағанда құрылымы неғұрлым иілгіш болып келеді. Ол тізімде жүруді кез келген жағдайда бастауға мүмкіндік береді және оны бастапқы позицияға дейін жалғастыруға болады. Көрсеткіш туралы мәлімдеме келесі синтаксисті қамтиды:

```
<type> * <identifier> [= <initializer>];
```

Көрсеткіш базаның мәндеріне, санақ түріне, құрылымына, бірлестікке, функцияға, меңзерге нұсқай алады.

int *pi; // int көрсеткіші char *ppc; // char көрсеткішіне көрсеткіш int* p, s; // жарнаманың нашар стилі, s көрсеткіш емес! int *p, s; // s – көрсеткіш еместігі айқын int *p, *s; // екі көрсеткіш

```
char *names[] = {"John", "Anna"};
```

// көрсеткіштер жиыны

- Соңғы декларацияда екі оператор: * және [] түрін жасау үшін пайдаланылады, олардың біреуі атаудың алдында, ал екіншісі кейін. Бекіту декларацияларын пайдалану әлдеқайда жеңіл болар еді, олардың барлығы префикс немесе суффикстер. Алайда *, [] және () өрнектердегі олардың мағыналарын көрсету үшін жасалған. Осылайша, * префикс, ал [] және () - жұрнақтар. Жұрналды операторлар префикстен басқа атаумен «тығыз байланысты». Сондықтан, [names] кез келген объектілерге көрсеткіш массивін және «функция көрсеткіші» сияқты түрлерін көрсетуді білдіреді, жақшаларды қолданыңыз. Көрсеткіштермен жұмыс істеу үшін маңызды екі операция бар. Бұл операциялар:
- адресті алу (адрестеу) жұмыс істеуі;
- Мекен-жайы бойынша құндылықтарды қабылдау (жанама немесе ауызша сөйлеу) *. int a, *p;

```
p  = &a;
```

// p айнымалысына a айнымалысының адресі меншіктеледі

```
*p = 0;
// Айнымалы мәндегі мекенжайдағы мән (яғни ауыспалы a мәні) 0 болады
```

Мекенжай арифметикасы Көрсеткіштердің үстінде келесі әрекеттерді орындай аласыз:

- көрсеткіш пен бүтін сан қосу;
- көрсеткіштен бүтін сандарды алып тастаңыз;
- көрсеткішті көрсеткіштен алып тастаңыз - бүтін сан алынады. Бұл жағдайда қосылған / шығарылған немесе алынған бүтін сан байттардың санын емес, көрсеткішпен көрсетілген түрдегі элементтердің санын көрсетеді, бұл сан көбейтіледі немесе үлгінің өлшемімен бөлінеді. Көрсеткіштерді бір-бірінен алып тастау, екі көрсеткіш бірдей массивтің элементтеріне нұсқайтын болса ғана анықталады (бірақ бұл тіл осылай болғанын тез тексеруге мүмкіндік бермейді). Бір көрсеткішті екіншісінен шығару нәтижесі осы көрсеткіштер арасындағы массив элементтерінің (бүтін) саны. Көрсеткіштің бүтін санының қосылуы және көрсеткіштен бүтін санның алынуы нәтижесінде жаңа көрсеткіш алынады. Егер осылайша алынған көрсеткіш бірдей массивтің элементін көрсетпесе (немесе соңғы элементтен кейінгі элементке) бастапқы көрсеткіш ретінде көрсетілсе, онда оны пайдалану нәтижесі анықталмаған. Көрсеткіштер массивтерді өңдеу үшін пайдаланылуы мүмкін.

```
int a[100], n, *end, *p;

end = a + n;
```

// n - a массивіндегі элементтердің саны. Массивтің атауы - оның бастапқы мекенжайы. // олай болса, end – массивтің соңғы элементінің адресі

```
for (p = a; p < end; p++)

  printf("%4d", *p);
```

Көрсеткіштен бүтін сандарды шегеру мүмкін болғандықтан, операциядағы теріс сандарды пайдалануға болады [ ].

```
int a[N];
int *endA = a + N - 1, i;

for (i = 0; i < N; i++)
  printf("%4d", endA[-i]);
```

Функция көрсеткіштері Функциямен сіз тек екі нәрсені жасай аласыз: оны шақырыңыз және оның мекен-жайын алыңыз. Функция мекенжайын алу арқылы алынған көрсеткіш функцияны шақыру үшін пайдаланылуы мүмкін.

```
void f(int x) { ... }

void (*pf)(int);
```

// функция көрсеткіштері. Жақшалар қою міндетті!

```
void g()

 { pf = &f;
```

// f фнукциясын нұсқайтын pf

```
   pf(0);
```

// pf көрсеткіші арқылы f функциясын шақыру }

Компилятор pf - көрсеткіш болып табылады және ол көрсететін функцияны шақырады. Яғни, функцияның * көрсеткішінің функциясымен айырмашылығы міндетті емес. Сол сияқты, функция мекен-жайды алу үшін & операторды пайдаланудың қажеті жоқ. Функцияға көрсеткіш параметрлері функцияның параметрлері сияқты бірдей түрде жарияланады. Берілген кезде функциялардың түрлері дәл сәйкес болуы керек.

```
typedef void (*PF)(int);
// Функция көрсеткіш түрін жариялау үшін typedef декларациясын пайдалануға болады
```

PF pf; // Біз функцияны өзіндік түрде алдын ала анықталған түрде жариялаймыз

```
void f1(int  x) { ... }
int  f2(int  x) { ... }
void f3(char x) { ... }

void f ()

 { pf = &f1;
```

// дұрыс

```
   pf = &f2;
```

// қате – қайтарылатын тип емес

```
   pf = &f3;
```

// қате – қайтарылатын параметр емес }

Көрсеткіш арқылы функцияларды шақыру кезінде параметрлерді беру ережелері тікелей функциялық қоңырауларға ұқсас.

- Көрсеткіштер мен тұрақтылар Көрсеткіштермен операциялар кезінде екі нысан қатысады: көрсеткіштің өзі және ол сілтейтін нысан. Көрсеткіш туралы мәлімдеме жасамай тұрып, кілт сөзді константаны қою көрсеткіш емес, тұрақты нысанды жасайды. Көрсеткіштің өзін тұрақты деп жариялау үшін, * * const * операторы операторын пайдаланыңыз. void f(char *p)

```
{ char s[] = "const";

  const char *p1 = s;
```

// тұрақты көрсеткіші

```
  p1[3] = 'r';
```

// қателік - p1 көрсеткіші тұрақтыны меңзеп тұр

```
  p1 = p;
```

// дұрыс

```
  char * const p2 = s;
// Констаталық көрсеткіш (инициализациялау қажетті)
  p2[3] = 'r';
```

// дұрыс

```
  p1 = p;
```

// қателік - p2 тұрақты болып табылады

```
  const char * const p3 = s;
```

// тұрақтыға тұрақты көрсеткіш

```
  p3[3] = 'r';
```

// қателік - p3 тұрақтысына көрсеткіш

```
  p3 = p;
```

// қателік - p3 тұрақты болып табылады }

Айнымалы адресті көрсеткішке тұрақты мән бере аласыз, себебі бұл зиянсыз әрекет. Дегенмен, еркін көрсеткішке тұрақты мекен-жайды тағайындауға болмайды, себебі бұл жағдайда тұрақты нысанның мәнін өзгерте аласыз. Ноль Нөл түрі int. Стандартты түрлендірулердің арқасында нөл кез келген интегралдық типті, өзгермелі нүкте түрін және көрсеткішті қолдануға болады. Нөл түрі мәтінмәнмен анықталады. Нөлдік, әдетте (бірақ әрқашан емес), тиісті ұзындығы нөлдік биттердің реті ретінде физикалық түрде ұсынылады. Нөлдік мекен-жайы жоқ объектілердің жоқтығына кепілдік беріледі. Сондықтан нөлге тең көрсеткішті ештеңеге сілтеме жасайтын көрсеткіш ретінде түсіндіруге болады. B-да, NULL макрос әдетте нөлдік көрсеткішті көрсету үшін анықталған. Себебі C ++ түрлерінде қатаң түрде тексерілсе, NULL орнына қарапайым нөлді пайдалану аз проблемаларға алып келеді.

Бақылау сұрақтары:

- Көрсеткіштер және олармен жұмыс.
- Стекті көрсеткіштер көмегімен көрсету.
- Стекке қолданылатын операциялар.

№ 8 тақырып Деректер құрылымы. Сызықтық және сызықтық емес құрылымдар. Файл түсінігі. Файлдың ерекше сипаттамалары. FILE типін сипаттаудың жалпы түрі.

Жоспары:

- Мәліметтер құрылымдары және олардың ұйымдастырылуы
- Мәтіндік файлдар

- Мәліметтер құрылымдары және олардың ұйымдастырылуы Соңғы 20-30 жылда құрылымдалған программалау идеялардың кең таралуы программалаудың теориясы мен практикасына үлкен әсер етіп, сәйкес алгоритмдер мен программаларды жасаудағы мәліметтердің типі мен құрлымдарының ролі қайта қарауға алып келді. Осыған байланысты соңғы он жылдықта электронды есептеуіш машиналарды (ЭЕМ) программалық қамтамасыз ету саласында мамандар даярлайтын оқу орындарында мәліметтердің типтері мен құрылымдары пәні пайда болды. Бұл пән ақпараттық технологиялар саласында соған сәйкес программалық қамтамасыз етуге жоғары сапалы мамандар даярлауға мүмкіндік беретін компьютер ғылымы информатиканың фундаментальды (негізгі, түпкі) бөлімі болып саналады. Бұл курстың идеялық мазмұны программалық тілге тәуеді емес, бірақ мәліметтер құрылымын көрнекі түрде белгілеуге болатын Турбо Паскаль тілі қолданылады. Мәліметтер құрылымын кез келген программаның немесе программалық бөлімнің бөлінбес саласы болып саналады. Программаны жасауда нақтылы есепті сипаттайтын мәліметтер жиынын анықтап, есепті шешудің алгоритмін құру керек. Программаның мақсатына қарай мәліметтер әртүрлі күрделі деңгейде болуы мүмкін. Яғни жай типтерден бастап, жеткілікті типтегі күрделі құрылымдарға дейін. Мәліметтер – бұл белгілі бір пішімде берілген және ары қарай пайдалануға арналған қандайда бір жүйені, құбылысты, үрдісті немесе объекті сипаттайтын мағлұматтар. Ақпарат және мәліметтер ұғымдарының арасындағы арақатынас тұрғысынан алып қарағанда: мәліметтер – бұл ақпараттың мазмұнын көрсететін нақты пішім; мәліметтер – бұл пайдаланушы үшін маңызды және онымен қандай да бір есептерді шешуге қолданылады. Мәліметтерге бірнеше классификациялық мінездемелер меншіктеледі. Олардың ішіндегі ең маңыздысы мәліметтер типі болып табылады. Мәліметтер типі мүмкін мәндер жиынын; олардың өңделу ережелерін (түрлендіру); сақтау кезінде ОСҚ және ССҚ-да олардың орналасу ретін; оларға қатынау ретін анықтайды. Негізгі мәліметтер типтері: стандарттық, қарапайым, шектелген, күрделі құрылымдық және т.б. Мәліметтер типтерінің концепциясы пайдаланушыға мәліметтерді беру, сақтау, қолдану есептерін тиімді шешуде көп мүмкіндіктер береді; мәліметтердің жинақтылығына әсер етеді. Сонымен қатар мәліметтер элементарлық (қарапайым) құрылымданбаған және құрылымданған (күрделі) болып жіктеледі. Элементарлық мәліметтерге символдар, сандар және логикалық мәліметтер жатады, олардың әрқайсысы бір мәнмен және атпен анықталады. Мәліметтерді және олардың арасындағы байланыстарды біріктіретін ақпараттық массивті құрылымданған мәліметтер деп атайды. Біріктірілетін қарапайым мәліметтердің тізімі, олардың мінездемелері, сонымен қатар арасындағы байланыстардың ерекшеліктері мәліметтер құрылымын анықтайды. Күрделі мәліметтер де мәнімен және атымен анықталады. Өңдеу үрдісінде мәндерінің өзгеруіне байланысты (қарапайым және құрылымдық мәліметтер үшін) мәліметтерді айнымалы және тұрақты деп бөледі. Өңдеудің қай кезеңінде қолданылуына байланысты мәліметтерді алғашқы(енетін), аралық, қорытынды(шығатын) мәліметтер деп бөледі. Сақтауда және өңдеуде мәліметтерді беру үш негізгі есепті шешуді қажет етеді: элементарлық мәліметтерді беру әдістерін анықтау; мәліметтердің құрылымға бірігу әдістерін анықтау; мәліметтердің материалдық тасуышта орналасуын анықтау. Мәліметтерді берудің үш деңгейін анықтайды: концептуалдық, логикалық және физикалық. Концептуалдық деңгейде ақпараттық массивтің жалпы құрылымы анықталады, ол мәліметтер моделі деп аталады. Мәліметтер моделдерінің негізгілері: иерархиялық, желілік, реляциялық, объектілі-бағытталған. Қатынастарының мінездемелеріне байланысты мәліметтер құрылымдарының бірнеше жіктелу белгілерін анықтауға болады: мәліметтердің реттері бойынша – реттелген және реттелмеген, қатынастарының мінездемелері бойынша – сызықтық және сызықтық емес, құрылымдағы мәліметтер типтеріне байланысты – біртекті және біртексіз, мәліметтерге қатынау әдістеріне байланысты – тура жететін және тізбектік жететін, ақпараттық массивтің өлшемінің өзгеру мүмкіндігіне байланысты – статикалық және динамикалық. Негізгі күрделі құрылымдар: массив, жиын, кезек, стек, ағаш (екілік ағаш), граф, тізімдер (логикалық жазба). Мәліметтерді ұйымдастыруда үш типті қолданады: сызықтық(тізбектік), кестелік, иерархиялық. Мәліметтер дегеніміз - дербес компьютерде өңдеуге болатын, кез келген ақпаратты сипаттайтын мәнді немесе мәндер жиыны. Мәліметтердің маңызды мінездемесін беретін тип болып саналады. Ол мәліметтерді компьютер жадысына беруде қолданылады және осы мәндерге орындауға болатын амалдар жиыны мен мәліметтер мәндері жиынын анықтайды. Мәліметтерді ұйымдастырудың логикалық және математикалық моделі Мәліметтер құрылымы деп аталады. Мәліметтердің логикалық құрылымы және физикалық құрылымы ұғымы бар. Мәліметтердің логикалық құрылымы дегеніміз - сәйес құрылымның моделі, берілу схемасы (екі өлшемді массив, тік бұрышты матрица). Мұнда әрбір элементте индекс болады. Мәліметтердің физикалық құрылымы дегеніміз - құрылымды компьютер жадысына орналастыру немесе сақтау схемасы (жады ұяшықтарының тізбегі). Мәліметтердің барлық жиынын екіге бөлуге болады: Статикалық және динамикалық құрылым мәліметтері. Статикалық құрылым

мәліметтері – жай және күрделі болуы мүмкін. Олар қандай да бір заңдылық бойынша жай құрылымдарда қалыптасады. Құрылым элементтерінің өзара орналасуы мен өзара байланысы әрқашан тұрақты болып қалады. Динамикалық құрылым мәліметтері дегеніміз – ішкі жасалуы қандай да бір заң бойынша қалыптасқан, бірақ элементтер саны олардың өзара орналасуы, өзара байланысы программаның орындалу барысында динамикалық түрде өзгере алатын мәліметтер.

Мәліметтердің типтерін сызықтық құрылымды мәліметтер типтері және сызықтық емес мәліметтер типтері деп бөлуге болады. Сызықтық құрылым мәліметтер типтері – орналасуы бойынша реттелген элементтер тізімін анықтайды. Сызықтық емес құрылым мәліметтер типтері – позициялық реттеусіз элементтерді анықтайды. Тікелей кіру дегеніміз – элементті тікелей таңдау және тізімдегі алдыңғы элементтерге байланыссыз таңдауды айтады

Массив дегеніміз – мәліметтердің бүтін санды индекс көмегімен кіруге болатын бір ғана типтен тұратын құрылымы.

```
A[0], A[1], A[2], …, A[n]
```

Массивтермен жұмыс жасауда элементтерге кіруді ұйымдастыру маңызды әрекет болып саналады. Элементке логикалық деңгейде кіру үшін – массив аты мен элемент индексі керек. Ал физикалық деңгейде жұмыс жасауда элемент адресін массив атымен ізделіп отырған элемент индексі мен оның ұзындығына қарай есептеп шығару қажет болады. Индексті кіру мәліметтері типтері үшін – жазуға кіруге қолданылатын қандай да бір кілт байланыстырады. FLASH кесте – кілтпен байланысты мәліметтерді сақтайды. Кілт – мәліметтерді табу үшін қолданылатын бүтін санды индекске трансформацияланады. Кілт бүтін болуы мүмкін емес. Сөздік деп аталатын мәліметтер құрылымы ассоциация деп аталатын кілт және мән жұптарының жиынынан тұрады. Мысалға, кілт сөз болуы мүмкін, ал мән сөздің анықтамасын көрсететін сөйлем болуы мүмкін. Сызықтық құрылым мәліметтер типтері – мәліметтерге тізбектей кіруді пайдаланғанда мәліметтердің динамикалық құрылымы болып табылады да, келесі қасиеттерге ие болады:

1. Өлшемнің тұрақсыздығы ;
2. Жадыдағы элементтер құрылымдарының физикалық реттілігінің жоқтығы. Сызықтық тізім – элементтердің кез келген санына ие болады, тізімнің өлшемі тізімдегі элементтерді қосуға және алып тастауға байланысты өзгеріп отырады. Бірінші элементі басында орналасады, соңғы элементі аяғында орналасады және әрбір елемент (соңғысынан басқа) алдында бір элементке ие болады. Реттелген сызықтық тізімде – мәліметтер бір-біріне қатысты реттеліп орналасады. Стек дегеніміз – элементтері тізімнің төбесі деп аталатын бір жағынан ғана қосылатын немесе өшірілетін сызықтық тізімді айтамыз (соңғысы келсе біріншісі кетеді). Шірет (очередь) дегеніміз – тізімнің басынан немесе аяғынан ғана кіруге болатын сызықтық тізім. Элементтер тізімнің соңына қойылады да басынан өшіріледі. Очередь приоритеті дегеніміз – очередь немесе шірет типті тізім, тізімдегі объектіні өшірген кезде ең жоғары приоритетке ие объект анықталады. Файл дегеніміз – байттар тізімі. Ол бір ағымға теңеледі, яғни, бір құрылғыдан екінші құрылғыға ауысып отыратын байттар тізбегі. Тек қана дисктік файлға тікелей кіруді жүзеге асыруға болады. Иррархиялық құрылым – бұл деңгейлері бойынша бөлінетін элементтер жиынындағы, яғни, бір деңгейдегі элемендеңгейде бірнеше ұрпағы болуы мүмкін. Тармақ (дерево) – бұл түпкі немесе тамыр деп аталатын бір ғана көзден тарайтын элементтері бар мәліметтер құрылымы. Пирамида дегеніміз – тармақтың ерекше түрі. Мұнда ең үлкен элемент үнемі түпкі деңгейде тұрады. Топ (группа) дегеніміз – элементтері ешқандай реттеусіз орналасқан сызықтық емес құрылым. Жиын дегеніміз – мәліметтер реттеусіз болғанда және мәліметтердің әрбір элементі өз алдына қайталанбайтын болғанда қолданылатын мәліметтер құрылымы. Граф дегеніміз – төбелер жиынынан және сол төбелерді қосатын байланыстар жиынынан тұратын мәліметтер құрылымы. Желі (сеть) дегеніміз – графтың ерекше формасы. Мұнда әрбір байланыстың өз салмағы болады. Файл – бір типті компоненттер тізбегі. Файл компоненті ретінде кез келген жай типті немесе жазуды алуға болады. Тек файл компоненті ретінде файл болуы мүмкін емес. Сонымен файлдың бір өлшемді массивтен негізгі айырмашылығы:
3. сыртқы тасығыштарға; яғни магниттік дискіге тиеледі;
4. компоненттер саны бастапқыда анықталмайды. Турбо Паскальда файлдар үш түрге бөлінеді: текстік файлдар, типтік файлдар, типсіз файлдар Файлдармен жұмыстағы негізгі ұғым – файлдық айнымалы. Файлдық айнымалы <ФА> немесе f-деп белгіленсе, оның сипатталыну тәсілдері:
5. VAR < ФА – аты >:text; {текстік файл}
6. VAR < ФА – аты >: File of < компонент типі>; {типтік файл}
7. VAR < ФА – аты >: File; {типсіз файл} Файлды жазу не оқу үшін мынадай процедуралар пайдаланылады: Assign(<ФА>,’<файлға жол сілтеу>\<сыртқы файл аты>’). Бұл процедура әдетте бірінші жазылады да, файлдық айнымалыға <ФА> сыртқы файл атауын меншіктейді. Жаңадан құрылатын файл Rewrite(<ФА>) операторымен

ашылады. Ал мазмұны әрмен қарай қолданылатын файлдың компоненттерін оқу Reset(<ФА>) операторымен ашылады. Файл компоненттерін оқу үшін Readln, Read және BlockRead процедуралары, ал компоненттерді файлға жазу үшін Writeln, Write және BlockRead процедуралы пайдаланылады. Файлдармен жұмыс соңында файлды жабу - Close(f) процедурасымен, файлды жою - Erase(f), ал файлдың атын өзгерту - Rename(f, <файлдың жаңа аты>) процедурасымен орындалады. Жазылып біткен файл соңына жаңа компонент қосылған файл соңын білдіретін белгі (маркер) қойылады. Егер файл бос болса немесе соңғы компонент – маркер оқылған соң Eof(f) функциясының мәні TRUE болады. 2.Мәтіндік файлдар Текстік файлдардың әр компоненті қатар соңын білдіретін Eoln маркерімен аяқталатын жолдық қатардан тұрады. Әр жолдық қатардың ұзындығы 255-тен кіші әр түрлі ұзындықта болуы мүмкін және оның 2 байтында маркер орналасады. Бір уақытта текстік файлды оқуға және жазуға болмайды, яғни алдымен оны ашып бір режимде жұмыс істеу керек. Текстік файл компоненттерін тізбекті түрде ғана өңдеуге болады. Тек Append(<ФА>) процедурасы ғана бұрынғы текстік файлдың соңына жаңа текст қоса алады. Текстік файлмен жұмыста енгізу, шығару тізімінде тек жолдық қатарлар, символдық, сандық айнымалылар болуы мүмкін; ал структуралы-айнымалыларды (жазу, массив, жиын, файл) көрсетуге болмайды. Бірақ, текстік файл копоненттерінің қажетті бөлігін негізгі текстен бөліп алып, оларды массив элементтері, жазу өрістерінің мәндері ретінде пайдалануға болады. 1-мысал. Тестік файлда Ank.dat жолдық қатар түрінде студенттердің анкеалық деректері жазылған: аты-жөні (20 символ), туған жылы, ұлты және тағы басқалар.Байттар мазмұны: Айтбаев Т. 1981 Қазақ Абдураимов В. 1980 Өзбек Шаджиков Ш. 1979 Әзірбайжан Файлдағы қазақтар саны анықтау керек. әр жолдық қатардағы 26 – 35 дейінгі ұлттың мәнін ғана тексеретін болсақ та, текті файл тізбекті түрде оқылатын болғандықтан бастапқы мәндерге сәйкес айнымалыларды анықтау керек: Var Fam: String[20]; Ut: string[9]; X: string[1]; N,TJ:Word; F: text; Begin n:=0; Assign(f,'c:\tp7\ank.dat'); Reset(f); {файлды оқу үшін ашылды} While not eof(f) do {файл біткенше қайалау} Begin Readln(f,Fam, TJ, X, Ult); {анкетаның 3 өрісінің оқылуы} {х-ке 25-ші байт оқылады} If ult='қазақ' then N:=N+1 end; Close(f); Writeln('қазақтар саны:' N:4) End. Программада файлдың әр қатарының 3 өрісінің мәні енгізіледе де, қалған өрістер маркерімен қоса ескерілмейді. Келесі қатарды оқуға өтеді. Қатардың ішіндегі сандық айнымалы мәндері бос орынмен яқталады (Қатардың 25-ші позициясы). 2-мысал. Kitap – деп аталатын текстік файлдағы текстi беттерге бөліп, арасына бет нөмерін және бос қатарлар қосу керек. Есептің шартына сәйкес қатарлар қосылған файлды X-деп атап, құралған файлды жауып, Kitap – файлы жойылады. Ал X файлының атауын Kitap – деп, өзгертіледі. Әдеттегідей кітап бетіндегі текстің әр қатарында шамамен 57-60 символ, ал қатар саны 40 деп алынсын. Program Example2; Var f,k:integer; F1,F2:текст {екі файлдық айнымалы анықталады} S: string[60]; {қатар} Bet: string[5]; {бет номері} Begin Assign (f1,'c:\Kitap'); Assign(f2,'c:\x'); Reset(f1); Rewrite(f2); While Not Eof(f1) do Begin j:=1; k:=1; {к – бет саны есептегіші} {файлдан файлға қатар көшіру циклі} Repeat Readln(f1,s); Writeln(f2,s); j:=j+1; Until(j>40) or EOF(f1); If Not Eof(f1) Then begin for j:=1 to 5 do Writeln(f2); {бос 5 қатар} k:=k+1; Bet:=Str(K); For I:=1 to 60 do s[i]:=''; {бір қатарды тазалау} Instr(Bet,s,55); {55 позициядан бастап бет нөмірін қосу} Writeln(f2,s); {бет нөмірін файлға жазу} End; Close (f1); Close (f2); Erase(f1); {Kitap файлын жою} Rename(f2,'kitap'); {жаңа файлдың атын өзгерту} End. 3-мысал. Текстік файлды оқып, ондағы тыныс белілерді жою программасы: Program esep3; Label 1; Var F,F1:Text; S,S1,par1,par2: String; N,I,I: Integer; Begin writeln(':'); readln(par1); Writeln(':'); readln(par2); Assign(F,par1); Assign(F1,par2); Reset(F); Rewrite(F1); s1:=''; clrscr; gotoxy(15,2); While not eof(F) do Begin Readln(f,S); for i:=1 to length(s) do if s[i] in ['-',',',':','?','!','.'] then delete(s,I,1); writeln(f1,s); writeln(s); end; close(f); close(f1); end. 4-мысал. Текстік файлды оқып, артық бос орындарды жойып, сөз санын есептеу программасы: Program sjatie; Label 1; Var F,F1,F2:Text; S,S1,par1:String; K,N,i,I: integer; Begin Writeln('өңделетін файл атын енгіз:'); Readln(par); Writeln('нәтиже-файл атын енгіз:'); Readln(par1); Assign(F,par); Assign(F1,par1); Reset(f); Rewrite(F1); I:=0; s1:='';I:=0; clrscr; gotoxy(15,2); While not Eof(f) do Begin Readln(f,S); Repeat N:=Pos(' ',S); If n<>1 then begin i:=i+1; s1:=s1+copy(s,1,n); end; Delete(s,1,n); Until n=0; s1:=s1+s+' '; Writeln(f1,s1); Writeln(s1); I:=I+1; s1:=''; End; Writeln('текстегі сөз саны = ',i); Close(f); close(f1); End. 5-мысал. Текстік файлды оқып, сөздер жиілігін экранга шығар және жиілік сөздер файлын құр. Program juilik_sozdik; Label 1,2; Var F,f1,f2: Text; c:char; S,S1,slov,n2,par1,par2: String; K,N,I,I,n1,m: Integer; Begin Writeln('өңделетін файл атын енгіз:'); Readln(par1); Writeln('нәтиже-файл атын енгіз:'); Readln)par2); Assign(F,par1); Assign(F1,par2); Assign(F2,'nn2.TXT'); Rewrite(f1); rewrite(f2); c reset(f); M:=0; While not eof(f) do Begin Slov:=''; readln(f,S); n:=pos(' ',s); slov:=copy(s,1,n-1); Delete(s,1,n); I:=length(slov); Repeat 1:n1:=pos(slov,s); If n1<>0 then begin k:=k+1; delete(s,n1,I+1); goto 1; end; If s<>'' then writeln(f2,s); readln(f,s); Until (eof(f)) and (s=''); Str(k,n2); s1:=slov+' '+n2; writeln(f1,s1); gotoxy(12,3+m); Writeln(slov,n2:10-length(n2)); m:=m+1; Close(f); close(f2); Erase(f); rename(f1,par1); Assign(f,par1); Assign(f2,'nn2.txt'); Reset(f); rewrite(f2); end; end. 6-мысал. Текстік файл түрінде сөздік берілген. Сөздікті пайдаланып, сөзді тура және кері бағытта аударатын программа құр. Program audarma; Label 1; Var f:text; S,s1,slov:string; C,C1:char; kk,n:integer; Procedure delprobel(var s:string); {қатардың басы мен соңындағы бос орындарды жояды} Var k:byte; Begin Repeat k:=length(s); if s[1]=' ' then delete(s,1,1); if s[k]=' ' then delete(s,k,1); k:=length(s); until (s[1]<>' ') and (s[k]<>' '); end; begin assign(f,'s1.txt'); reset(f); clrscr; write('қандай тілде сөз енгізесіз?'); write('а-ағылшынша, к-қазақша: '); readln©; write('сөзді енгіз:'); read(slov); kk:=0; while not eof(f) do begin Readln(f,s); n:=pos('#',s); {"#"-белгісі сөздікте сөз бен оның аудармасы арасындағы белгі} If c='a' then begin s1:=copy(s,1,n-1);

delete(s,1,n); end Else Begin s1:=copy(s,n+1,length(s)); delete(s,n,length(s)); end; Delprobel(s1); If slov=s1 then begin writeln(‘ ‘,s); kk:=kk+1 end; End; If kk=0 then writeln(‘Сөздікте мұндай сөз жоқ ‘); End; end.

Бақылау сұрақтары

1. Бағдарламаларды ұйымдстыру кезінде қандай ережелерді ескеру қажет?
2. Бағдарламалардың кодын қандай әдіспен жазу керек?
3. Программаның қателерін болдырмау үшін не істеу керек?

№ 9 тақырып Типтелген және типтелмеген файлдар. Файлдармен жұмыс істеуге араналған стандартты процедуралар. Мәтіндік файлдар. Жоспары:

---

1 ТИПТІК ФАЙЛДАР

---

2 ТИПСІЗ ФАЙЛДАР

---

1 ТИПТІК ФАЙЛДАР

Тізбекті ену файлы. Паскаль тілінде программа көмегімен сыртқы есте сақтау құрылғыларына (қатты диск немесе винчестер, дискета) түрлі информацияны файл атауына беріп, жазып қоюға болады, оны берілгендер файлы не дискілік файл деп атайды. Типтік файл – типі анықталған компоненттердің нөмірленген тізбегінен тұратын дискілік файл. Мысалы, мынадай құрылыммен талапкерлердің мәліметтік тізімі берілсін: аты-жөні, туылған жылы, таңдаған мамандық шифры, тестілеуден жинаған ұпайы (балл). Осы құрылыммен өнделетін информацияны сыртқы файлға жазу керек болса, Паскаль тілінде файл типін сипаттаудың жалпы форматы мынадай:

```
Type<тип атауы>=file of <элементтер типі>;
Var<айнымалы атауы>=<тип атауы>;
```

Жоғарыдағы мысалдағы файлдың сипатталуы: Type abit=Record

```
Fam: string [15];
```

TJ: integer; Shifr: integer; Ball: integer; End;

```
Varv:abit; f: file of abit;
```

Мұндағы abit- жазу типі атауы. Fam,TJ, Shifr, Ball- жазу өрісі атаулары. F- файлдық айнымалы, ал V- жазу аты. Файлды жазу, оқу үшін bf – берілгендер файлының атауы меншіктелетін айнымалы, V – файлдық элемент (жазу), ал V.Fam, V.Tj, V.Shifr, V.ball- жазу өрістері айнымалылары деп белгіленіп, сыртқы ЕСҚ-ға жазылады. Турбо Паскальда берілгендер файлын жазу үшін мынадай процедуралар пайдаланылады. Assign(f,bf) – F файлдық айнымалыға bf- сыртқы файл аты тағайындалады. Rewrite(f)- атауы Assign процедурасындағы анықталған файлды жазу мақсатында сыртқы файлды ашу. Ол bf үшін сыртқы дискіден арнайы орын бөліп, көрсеткішті осы бөлінген орынның басына жазуға дайындап орналастырады. Write(f,V)- f үшін бөлінген сыртқы файлдағы орынға V – элементін жазу. Close(f)- файлды жабу. Reset(f)- файлды оқу мақсатында ашу. Read(f,v)- файл элементін оқу. While not eof (f) do – файл элементтерін соңына дейін оқу үшін орындалатын цикл басы. Файл көрсеткіші дискілік файлдың соңына жеткенде Eof(f) логикалық функциясының мәні True болады. Write(V.Fam,V.Tj,V.Shifr, V.ball) – файл элементі өрістерін экранға шығару. әдетте бұл оператордың параметрлеріне арнайы орындар бөліп, файлды экранға оқуға ыңғайлы түрде шығарады. Мысалы, V.Fam:15, V.Shifr:4, V.ball:3 параметрлеріндегі String типті айнымалы мәнін бағанның сол жағынан бастап шығарған дұрыс. Яғни, V.Fam:15 параметрінің орнына V.Fam,’ ‘:15-length(V.Fam); параметрлерін алса болғаны. Бұл кезде адамның аты-жөні еніне 15 символ

сиятын бағанның сол жағынан бастап орналастырылады да, ‘ ’:15-length(V.Fam) параметрі артық қалған жерді бос тастап кетеді. 7-мысал. Құрылымы – Fam, Tj, Shifr, ball болатын файл құрып, оны оқып, жинаған ұпай саны 45 асқандарын конкурсқа жіберетін программа құру керек. Program Example3; Label 1, 2; Type abit=Record

```
Fam: string [5];
```

Shifr: integer; Ball: integer; End;

```
Var f:file of abit; v:abit; bf:string [6];
Begin Write (‘Құрылатын файл аты–?’);
Readln (bf); Assign (f,bf); Rewrite (f);
```

With v do Begin

```
2: Write(‘Фам:’); Readln(Fam); If Fam=’***’ Then Goto 1;
Write(‘Шифрб балл:’); Readln (Shifr, ball); Write (f,v); goto 2;
```

End;

```
1:Close(f);
{файлды оқу};
Reset(f);
While noteof (f) Do
Begin        Read(f,v);
If v.ball>45 Then Writeln(V.fam,’ ‘,15–length(V.Fam),v.Shifr:5, V.ball:3)
End;  Close(f)   End.
```

8-мысал. Алдыңғы тақырыптағы 1-мысал қарастырылсын. Ондағы ANK.dat типтік файл түрінде сипатталады. Pogram Example4;

```
Type jaz=Record Fam: String[20] {аты–жөні өрісі}
Tj:Word;  {туған жылы}
Ult: string[9]       {ұлты}
```

End;

```
Var f:File of Jaz;    {типтік файл сипатталуы}
Stud:Jaz;  {компонент оқылатын айнымалы}
```

Begin

```
N:=0;    Assign (f,’c:\Ank1.dat’);
Reset(f);    {файлды оқу үшін}
```

Repeat

```
Read(f,stud);  {файлдың ағымдық компонент мәнін Stud–қа берілуі}
If Stud.ult=’қазақ’ Then n:=n+1
Until Eof(f);
Close (f); Writeln(‘қаақтар саны’,N=4);
```

End. 9-мысал. 1-мысалдағы файл компонентін қатар түрінде сипаттап қарастыратын программа.

```
Type Katar=string[10];
```

Var f:file of katar;

```
N:word; S:katar;
Begin N:=0;  Assign (f,'c:\Ank2.dat');
```

Repeat

```
Read(f,s);    {файл компонентінің s–ке енгізу}
If Copy (s,26,9)='қазақ'      {қатар бөлігі мәнін тексеру}
Then N:=N+1;
Until Eof(f);
Close (f); writeln ('қатар саны',N=4);
```

End. Программада 1980 жылғы туғандардың санын есептеу керек болса, ол сандық мәнді қатардан бөліп алу үшін алдымен орын дайындап (z:word;), Val(Copy(S,21,4),Z, m) процедурасымен сандық түрге айналады. Егер 21-24 позицияларында қате символдар болмаса, түрлендіру барысында Z-ке сандық мән беріледі, m-параметрінің мәні 0-ге тең болады. Еркін ену файлдары. Типтік файлда компоненттер ұзындықтары бірдей болғандықтан, әрбір компоненттің позициясын есептеп алуға болады. Файлдағы компоненттер саны FileSize(f) функциясы арқылы анықталады. Типтік файлды оқуға Read, жазуға Write процедуралары, ал көрсетілген компонентке өтуге Seek процедурасы қолданылады. Seek – процедурасы компоненттермен ешқандай әрекет орындамайды, тек көрсетілген компонент нөмірінде ағымдық позицияны жылжытады. Тізбекті ену тәсілімен файлды жазу, оқу файл элементтерін басынан бастап ретімен жазу, оқу арқылы жүргізіледі. Дискілік есте сақтау құрылғысы (винчестер) бар компьютерде жазуды бірден іздеп табу үшін программаға Seek(f,k-1) процедурасын, содан кейін Read(f,v) немесе Write(f,v) процедуралары қолданған жөн. 10-мысал. 3-мысалдағы құрылыммен берілген файл дискіде жазулы болса, оған компоненттерді жалғастырып жазатын программа: Program Example6; Label 1,2; Type abit=Record

```
Fam: string [15];
```

Shifr: integer; Ball: integer; End; Var f: file of abit;

```
v: abit;  bf: string [6]; k: integer;
Begin Write ('Жалғастырылатын файл аты–?');
Readln (bf); Assign (f,bf); Reset(f);
K:=filesize(f); {компоненттер санын анықтау}
  Seek(f,k); {k+1 компонентке өту}
2: Write('Фам:'); readln(v.Fam);
If v.Fam='****' Then Goto 1;
Write('Шифр, балл:'); Readln (v.Shifr, v.ball); Write (f,v); goto 2;
1:Close(f); end.
```

11-мысал. 3-мысалдағы құрылыммен берілген файл дискіде жазулы болса, оның К-сыншы компонентті өзгертіп жазатын программа:

```
Program Example7;    {файлды өзгерту}
```

Label 1,2; Type abit=Record

```
Fam: string [15];
```

Shifr: integer; Ball: integer; End; Var f: file of abit;

```
v: abit; bf: string [6]; k: integer;
Begin Write (‘редакцияланатын файл аты–?’);
Readln (bf); Assign (f,bf); reset(f);
Writeln(‘өзгертілетін компонент нөмерін енгіз:’); Readln(k);
seek(f,k–1);      {компонентке өту}
Write(‘Фам:’); Readln(v.Fam); {К–шы компоненттің жаңа}
Write(‘Шифр балы:’); Readln (v.Shifr, v.ball); {мәнін енгізу}
Write (f,v);  Close(f);
```

End.

---

2 ТИПСІЗ ФАЙЛДАР

Типсіз файлдар информацияны көп өлшемде оқу (жазу) қажет болғанда қолданылады. Типтік файлдарда оқу (жазу) әр компонентті тізбектеп оқу (жазу) арқылы іске асады. Ал файлдармен жұмыста информацияны көп өлшемде оқу (жазу) тиімді болғандықтан, логикалық жазуларды (файл компоненттері) физикалық деп аталатын топқа біріктіріп, бұл топтауда блоктау деп, блоктардағы жазу санын блоктау коэффициенті (КБ) деп атайды. Физикалық жазу өлшемі екі тәсілмен есептелінеді:

```
a) КБ * SizeOf(<логикалық жазу аты>) өрнегі арқылы;
```

b) Физикалық жазу өлшемі арқылы логикалық жазу санын көрсету. Шындығында программаға берілгендер бөліктерінің оқылатын не жазылатын өлшемі ғана белгілі болады. Типсіз файлдармен жұмыста екі тәсіл де қолданылады. Reset және Rewrite процедуралары типсіз файлдарды ашқанда параметр ретінде файл атауы және физикалық жазу өлшемі (“a” тәсілі) немесе (“b ” тәсілі) логикалық жазу өлшемі көрсетіліп, ал Read (Write) процедурасының орнына BlockRead (BlockWrite) процедурасы қолданылады: BlockRead(<ФА>, <Буфер>, <саны>, <нақты саны>) соңғы параметрдің болуы міндетті емес. Мұндағы <Буфер>- файлдан оқылған жазулардың мәндері жазылатын айнымалы (әдетте массив түрінде беріледі); <саны>- керекті жазу саны және <нақты саны> - нақты жазу саны. BlockWrite процедурасының құрылымы да осындай. Файлдың соңғы позициясы оқылғанда (жазылғанда) әдетте 3-ші параметрде көрсетілген мәннен жазу саны кем болуы мүмкін. Бұл жағдайда 4-ші параметр міндетті түрде көрсетілуі тиіс. Типтік файлдармен жұмыстағы басқа процедуралар типсіз файлдарға да қолданылады. Тек Seek процедурасында : ”a” тәсілінде физикалық жазу нөмірі көрсетіледі. 12-мысал. Логикалық жазу өлшемін 35, ал оқылған жазу санын 4 деп алдыңғы тақырыптардағы Ank.dat файлын тисіз файл түрінде сипаттап, BlockRead процедурасының қолданылуымен құрылған программа жазу керек: Program Example8;

```
Type Str=String[35];
```

Var i,n,m: integer;

```
S:array [1..39] of str; {s–буфер}
fp : file;  {Типсіз файл сипаттамасы}
```

Begin

```
n:=0; Assign `fp,’c:\ank.dat’); Reset(fp,35);
While not eof(fp) do
```

Begin

```
Blockread (fp,S,39,M);  {M=39}
```

For i:=1 to M do

```
If copy(s[i],26,9)=’қазақ’ then n:=n+1;
End;  Close `fp`;
Writeln (‘қазақтар саны’,n:4)
```

End.

Бақылау сұрақтары 1.Бағдарламалаудың қандай әдістемелер бар? 2.Модульдік программалау дегеніміз не? 3.Құрылымдық программалаудың негізгі концепциялары? 4.Құрылымдық программаладың неше құраушысы бар?

№ 10 тақырып Ақпараттық құрылымдарды өңдеу алгоритмдері Абстракты деректер типі. Тізімдерді өңдеу алгоритмдері.

Жоспары:

1. Деректер құрылымын өңдеу алгоритмдерінің мысалдары 2.Векторлар 3.Массивтер
2. Енгізілген деректер түрлері

1. Деректер құрылымын өңдеу алгоритмдерінің мысалдары Деректер құрылымдары деректер элементтерінің жиынтығы және олардың арасындағы қатынастар. Бұл жағдайда деректер элементтері қарапайым деректер мен деректер құрылымын білдіре алады. Деректер арасындағы қатынастар олардың арасындағы деректер мен функционалдық байланыстарды білдіреді, олар осы деректер орналасқан. Деректер құрылымының элементінің графикалық көрінісі.

Қатынас элементі - қарастырылатын құрылымды басқа деректер элементтерімен барлық элементтердің қатынастарының жиынтығы.

```
S:=(D,R)
```

S - деректер құрылымы, D - деректер, ал R - бұл қатынас. Дегенмен, деректер құрылымы күрделі, ол ақыр соңында қарапайым деректерден тұрады (төмендегі суреттерді қараңыз).

Машинаның есімі кіретін ақпаратты өңдейтін миллиондаған флип-флоптан тұрады. Біз ақпаратты компьютерге енгізген кезде, біз оны деректерге тапсырыс беріп, мағынасын беретін қандай да бір пішінде ұсынамыз. Құрылғы кіріс ақпарат үшін өрісті алға айналдырып, оған бірнеше мекен-жай береді. Осылайша, деректер дерексіз түрде, логикалық деңгейде өңделеді және машина оны физикалық деңгейде жасайды. Логикалық ұйымнан жеке тұлғаларға өтетін өтулердің тізбесі төмендегі суретте көрсетілген.

Деректер құрылымдарының жіктелуі Деректер құрылымдары:

1. Құрылымдағы деректермен байланысты:
2. егер құрылымдағы деректер өте нашар қосылған болса, онда мұндай құрылымдар (vector, array, row, stacks)
3. Егер құрылымдағы деректер байланысқан болса, онда мұндай құрылымдар қосылған (байланысқан тізімдер)
4. Құрылымның уақыттың немесе бағдарламаның орындалу процесінің өзгермелілігі бойынша:
5. статикалық құрылымдар - бағдарламаны орындау аяқталғанша өзгертпейтін құрылымдар (жазбалар, массивтер, сызықтар, векторлар)
6. жартылай статикалық құрылымдар (стектер, палубалар, кезектер)
   - динамикалық құрылымдар - бағдарламаны орындаудың толық өзгерісі бар
7. Құрылымның тәртібі туралы:
8. Сызықтық (векторлар, массивтер, стакалар, палубалар, жазбалар)
9. Сызықсыз (көбейтуге байланысты тізімдер, ағаш құрылымдары, графиктер) Ең маңызды ерекшелігі - уақыт құрылымының өзгермелілігі. Статикалық деректер құрылымдары

- Векторлар Ең қарапайым статикалық құрылым - бұл вектор. Вектор - бұл таза сызықты реттелген құрылым, оның элементтері арасындағы байланыс қатаң құрылымдардың элементтерінің дәйектілігі (төмендегі суретті қараңыз).

Вектордың әрбір элементі өзінің индексі бар, ол вектордағы берілген элементтің орнын анықтайды. Индекстер бүтін сандар болғандықтан, олар бойынша операциялар орындалуы мүмкін және, осылайша, логикалық қол жеткізу деңгейінде құрылымдағы элементтің орнын анықтайды. Вектор элементіне кіру үшін, вектордың (элементтің) атауын және оның

индексін ғана көрсетіңіз. Бұл элементке қол жеткізу үшін, мекенжай функциясы пайдаланылады, ол бастапқы элементінің мәні орналасқан индекс мәнінен слот адресін қалыптастырады. Бағдарламада векторды жариялау үшін оның атауын, элементтер санын және олардың түрін (деректер түрі) көрсету керек. Мысал: var

```
M1: Array [1..100] of integer;
M2: Array [1..10] of real;
```

Вектор толығымен бірдей деректерден тұрады және олардың саны қатаң анықталған.

3.Массивтер
Жалпы, массивтің элементі вектордың элементі болып табылады, ол өзі де құрылымның элементі болып табылады (төмендегі суретті қараңыз).

Екі өлшемді массивтің элементіне қол жеткізу үшін индекс жұбының мәндері (жолдың нөмірі және элементтің қиылысындағы бағанның нөмірі) қажет. Физикалық деңгейде, екі өлшемді массивдер бірөлшемді (векторлық) массивге ұқсас, аудармашылар массивтер ретінде немесе жолдар немесе бағандар ретінде көрінеді.

1. Енгізілген деректер түрлері Жоғарыда айтылғандай, Pythondегі барлық деректер нысандармен ұсынылған. Аттар - бұл тек осы нысандарға сілтеме жасайды және жүктің түрін декларацияда жүктемейді. Кірістірілген түрлердің мәндері тілдің синтаксисінде арнайы қолдау көрсетеді: әріптік жолды, санды, тізімді, кілтсөзді, сөздікті (және олардың сорттарын) жаза аласыз. Кіріктірілген түрдегі операцияларға арналған синтаксистік қолдау пайдаланушылар сыныптары анықтаған нысандар үшін оңай қол жетімді болады. Сондай-ақ объектілер өзгермейтін және өзгермелі болуы мүмкін екенін атап өткен жөн. Мысалы, Python жолдары өзгермейді, сондықтан жолдардағы операциялар жаңа жолдарды жасайды. Кіріктірілген түрлердің картасы (осы түрлерден мұраға алу үшін қалаған түрге және сынып атауларына арналған функциялардың аттары бар):
2. Арнайы типтер: None, NotImplemented и Ellipsis ;
3. сандар;
4. бүтіндер
5. қарапайым бүтін int
6. еркін дәлдік бүтіндігі long
7. логикалық bool
8. өзгермелі нүктелі сан float
9. комплекстік сан complex
10. тізбектілік;
11. өзгермелі:
12. қатар str ;
13. Unicode-қатар unicode ;
14. кортеж tuple ;
15. өзгермелі:
16. тізім list ;
17. көрсету:
18. сөздік dict
19. шақыруға болатын объектілер:
20. функциялар (қолданбалы және кіріктірілген);
21. функция-генераторлар;
22. (қолданбалы және кіріктірілген) әдістер;
23. класстар (жаңа және "классикалық");
24. класстар экземпляры (__call__әдісі бар болса);
25. модульдер;
26. класстар;
27. класстар экземпляры;

28. файлдар file ;
29. көмекші типтер buffer, slice. type() кіріктірілген функциясының көмегімен кез келген объект типін анықтауға болады. int және long типтері Екі түрлі: int (бүтін сандар) және ұзын (еркін дәлдіктің бүтін сандары) бүтін сандарды ұсыну үшін үлгі болып табылады. Біріншісі C қолданылған архитектураға арналған C компиляторындағы ұзын түрге сәйкес келеді. Сандық литералдар 8, 10 немесе 16 базалық жүйелері арқылы жазылуы мүмкін: # Бұл литералдарда 10 нөмірі жазылған print 10, 012, 0xA, 10L Сандар бойынша операциялардың жиынтығы семантикамен қатар белгілер бойынша өте стандартты: >>> print 1 + 1, 3 - 2, 2*2, 7/4, 5%3 2 1 4 1 2 >>> print 2L ** 1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376 >>> print 3 < 4 < 6, 3 >= 5, 4 == 4, 4 != 4 # салыстыру True False True False >>> print 1 << 8, 4 >> 2, ~4 # биттік ауысым және инверсия 256 1 -5 >>> for i, j in (0, 0), (0, 1), (1, 0), (1, 1): ... print i, j, ":", i & j, i | j, i ^ j # биттік операциялар ... 0 0 : 0 0 0 0 1 : 0 1 1 1 0 : 0 1 1 1 1 : 1 1 0 Int типіндегі мәндер -2147483648-ден 2147483647 дейінгі ауқымды қамтуы тиіс, ал бүкіл дәлдік дәлдігінің дәлдігі қол жетімді жад көлеміне байланысты болады. Айта кету керек, егер операция нәтижесі рұқсат етілген мәннен асатын мән болса, онда int типі толықтай ұзынырақ болуы мүмкін: >>> type(-2147483648) <type 'int'> >>> type(-2147483649) <type 'long'> Сондай-ақ тұрақты мәндерді жазу кезінде абай болу керек. Нөлдер санның басында сегіздік сан жоқ жүйенің сегіздік белгісі болып табылады: >>> 008 File "<stdin>", line 1 008 ^ SyntaxError: invalid token Float типі Қолданылатын архитектура үшін C типті екі есеге сәйкес келеді. Ол әбден дәстүрлі түрде немесе нүкте арқылы немесе көрсеткішпен белгіленеді: >>> pi = 3.1415926535897931 >>> pi ** 40 7.6912142205156999e+19 Арифметикалық операциялардан басқа, сіз математикалық модульден операцияларды пайдалана аласыз. Пайдалы функциялардың ішінен сіз round (), abs () функциясын есте аласыз. Complex типі Ойнатылған бөліктің кәдімгі сөзі j-ды жұрнау ретінде қосу арқылы беріледі (үнсіздік бірлік көбейтіледі):>>> -1j * -1j (-1-0j)
30. Түрі нақты негізінде жүзеге асырылады. Арифметикалық операциялардан басқа, сіз cmath модулінен операцияларды пайдалана аласыз. bool типі «Канондық» логикалық шамаларды белгілеу үшін бүтін түрінің ішкі түрі. Бұл екі мән шын және жалған, бұл осы түріне жатады. Жоғарыда айтылғандай, кез-келген Python нысаны шындыққа ие, қисынды әрекеттерді логикалық түрімен көрсету мүмкін: >>> for i in (False, True): ... for j in (False, True): ... print i, j, ":", i and j, i or j, not i ... ... False False : False False True False True : False True True True False : False True False True True : True True False Айта кету керек, Python екінші операндасын немесе операцияны есептемейді, егер оның шығу бірінші операндадан анық болмаса. Осылайша, егер бірінші операнд шын болса, ол нәтижесінде немесе қайтарылады, әйтпесе екінші операнд қайтарылады. and үшін операция бәрінде бірдей. Жол түрі мен Unicode түрі Python-да екі түрлі жолдар бар: тұрақты және Unicode жолдары. Шын мәнінде, жол - символдар тізбегінен (дәстүрлі желілерін жағдайда «байт ретін» айтуға болады). Тұрақты жолдар жолдың литералы көмегімен бағдарламада көрсетілуі мүмкін. литералдардан үшін бірдей (\) апострофтардан үш есе немесе жол литералдары ішінде Sequences қиғыш сызығы арқылы анықталады тырнақшаға үш есеге дәйекше (') ретінде пайдаланылатын, және көп салалы литералдардан дәстүрлі қос тырнақшаға («) пайдалануға болады жол литералдары жазбаша мысалдары ...: s1 = "строка1" s2 = 'строка2\nc переводом строки внутри' s3 = """строка3 с переводом строки внутри""" u1 = u'\u043f\u0440\u0438\u0432\u0435\u0442' # привет u2 = u'Еще пример' # не забудьте про coding! Жолдар үшін тағы бір түрі бар: өрескел жолақтар. Бұл литералдарда кері қиғаш сызық және келесі таңбалар арнайы таңбалар ретінде түсіндірілмейді, бірақ «дәл сол сияқты» жолына енгізіледі: my_re = r"(\d)=\1" Әдетте бұл жолдар тұрақты өрнектерді жазуды талап етеді (мәтіндік ақпаратты өңдеу туралы лекцияда талқыланады). Жолдардағы операциялар жиынтығы «+» тіркесімін, «*» қайталануын, «%» пішімін қамтиды. Сондай-ақ, жолдарда көптеген әдістер бар, олардың кейбіреулері төменде көрсетілген. Әдістердің толық жиынтығын (және олардың қосымша дәлелдерін) Python құжаттамасынан табуға болады. >>> "A" + "B" 'AB' >>> "A"*10 'AAAAAAAAAA' >>> "%s %i" % ("abc", 12) 'abc 12'
31. Мәтінді өңдеу бойынша лекцияда жол объектілерінің кейбір әдістері қарастырылады. Tuple типі Тұрақты тізбектілігін көрсету үшін (түрлі емес) нысандарды пайдаланады. Кәдімгі әмиян әдетте жақшаларда жазылған, бірақ ешқандай белгісіз болса, онсыз жазуға болады. Жолақ жазбаларының мысалдары: p = (1.2, 3.4, 0.9) # точка в трехмерном пространстве for s in "one", "two", "three": # цикл по значениям кортежа print s one_item = (1,) empty = () p1 = 1, 3, 9 # без скобок p2 = 3, 8, 5, # запятая в конце игнорируется Сондай-ақ, тағайындау туралы мәлімдеменің сол жағындағы кесінді синтаксисін пайдалануға болады. Бұл жағдайда оң жақта есептелетін мәндерге негізделіп, сол жақта аттармен бір-бірімен байланыстырылады. Сондықтан құндылықтармен алмасу өте талғамайды: a, b = b, a list типі «Таза» Python-та ерікті элемент түрі бар массивтер жоқ. Оның орнына тізімдер қолданылады. Олар төртбұрыш жақшаларда жазылған литералдармен немесе тізімдерді қосу арқылы көрсетілуі мүмкін. Тізімді орнату параметрлері келесідей: lst1 = [1, 2, 3,] lst2 = [x**2 for x in range(10) if x % 2 == 1] lst3 = list("abcde") Тізімдермен жұмыс істеу үшін, өзгермелі

тізбегі бар адамдарға қосымша бірнеше әдіс бар. Олардың барлығы тізімін өзгертуге байланысты. Кездейсоқтар Төмендегі негізгі әдістер төменде келтірілген. Естеріңізге сала кетейік, дәйектілік өзгермейді және өзгермейді. Соңғы әдістер біршама үлкен. Синтаксис Семантика len(s) s тізбегінің ұзындығы x in s Кезектілік элементін тексеру. Python жаңа нұсқаларында, ішкі жолдың жол екенін тексеруге болады. True немесе False қайтарады x not in s = not x in s s + s1 Тізбекті біріктіру s*n немесе n*s N қайталанатын реті. Егер n <0 болса, бос тізбекті қайтарады. s[i] I <0 болса, i-ші элементін немесе len (s) + i-th элементін қайтарады s[i:j:d] Төменде i-дан бастап d-ға дейінгі қадамдардағы үзіліс төменде қарастырылады min(s) Ең кіші элемент s max(s) Ең үлкен элемент s Айнымалы тізбектерге арналған қосымша конструкциялар: s[i] = x Айнымалы тізбектерге арналған қосымша конструкциялар: s[i:j:d] = t I-дан j-ге дейінгі аралық (d-қадаммен) ауыстырылады (тізім) t del s[i:j:d] Қайтару элементтерінен жою •

32. Бірізділікпен жұмыс істеудің кейбір әдістері
33. Кестеде айнымалы тізбектердің бірнеше әдістері көрсетілген (мысалы, тізімдер).

әдіс Сипаттама

```
.append(x)
```

Кезектің соңына элементті қосады

```
.count(x)
```

Элементтердің санын x санына теңестіреді

```
.extend(s)
```

s тізбегін тізбектің соңына қосады

```
.index(x)
```

S [i] == x сияқты ең кіші i қайтарады. Егер x табылмаса, ValueError ерекше жағдайын көтереді

```
.insert(i, x)
```

I-ші аралықта x элементін кірістіреді

```
.pop(i)
```

I-ші элементін қайтарады, оны тізбектен шығарады

```
.reverse(s)
```

кері S туралы элементтердің ретін өзгертеді

```
.sort([cmpfunc])
```

S элементтерін сұрыптайды. Cmpfunc салыстырудың өз функциясын көрсетуге болады • Элементті индекс пен секциялар бойынша қабылдау Мұнда индекстерді индекстеу және индекстерге қатысты субстраттардың (және, жалпы алғанда,) бөлінуі туралы бірнеше сөз айту керек. Тізбектің жеке элементін алу үшін тік жақшалар пайдаланылады, онда индексті беретін өрнек бар. Python жүйесінде сеанстық индекстер нөлден басталады. Теріс индекстер элементтерді тізбектің соңынан санау үшін қызмет етеді (-1 - соңғы элемент). Мысал бұл мәселені түсіндіреді:

```
>>> s = [0, 1, 2, 3, 4]
>>> print s[0], s[-1], s[3]
```

0 4 3

```
>>> s[2] = -2
>>> print s
[0, 1, -2, 3, 4]
>>> del s[2]
>>> print s
[0, 1, 3, 4]
```

Айырықтарға қатысты жағдай біршама қызықты. Фирма Python жүйелі қисық сызықты қабылдағанда, элементтерді емес, олардың арасындағы бос орындарды аудару әдеттегідей. Бастапқыда бұл әдеттен тыс көрінеді, дегенмен, еркін кесектерді көрсету үшін өте ыңғайлы. Нөлге дейін (индекс бойынша) кезектілік элементіне дейін интервалда 0 саны бар, одан кейін - 1 және т.б. Теріс мәндер сызықтың соңынан бос орындарды санайды. Б іктерді жазу шін келесі синтаксисті пайдаланы ыз: Ал енді кесектермен жұмыс істеудің мысалы:

```
>>> s = range(10)
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s[0:3]
[0, 1, 2]
>>> s[-1:]
[9]
>>> s[::3]
[0, 3, 6, 9]
>>> s[0:0] = [-1, -1, -1]
>>> s
[-1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del s[:3]
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Осы мысалдан көріп тұрғандай, кез келген ішкі жолды, тіпті нөлдік ұзындығы болса да, элементтерді жою үшін де, қатаң анықталған орынға кірістіру үшін де тілдерді пайдалану ыңғайлы. Тип dict Сөздік (хэш, ассоциативті массив) - кілт-мән жұптарын сақтау үшін өзгермейтін деректер құрылымы, мұндағы мән кілтпен анықталады. Кілт өзгермейтін деректер түрін (сан, жол, тупле және т.б.) болуы мүмкін. Кілт-мән жұптарының тәртібі ерікті болып табылады. Төменде сөздікке арналған сөздік және сөздікпен жұмыс істеу мысалы келтірілген: d = {1: 'one', 2: 'two', 3: 'three', 4: 'four'} d0 = {0: 'zero'} print d[1] # мән кілт бойынша қабылданады d0[0] = 0 # мән кілт бойынша меншіктеледі del d0[0] # осы түйменің кілт-мәндік жұбы жойылады print d for key, val in d.items(): # бүкіл сөздік бойынша цикл print key, val for key in d.keys(): # бүкіл сөздік бойынша цикл print key, d[key] for val in d.values(): # сөздік мәндері бойынша циклды print val d.update(d0) # сөздік басқа жолмен толығады print len(d) # сөздіктегі жұп саны file типі Осы типтегі нысандар сыртқы деректермен жұмыс істеуге арналған. Қарапайым жағдайда, бұл дискідегі файл. Файл нысандары негізгі әдістерді қолдауы керек: read(), write(), readline(), readlines(), seek(), tell(), close() и т.п. Келесі мысал файлды қалай көшіру керектігін көрсетеді: f1 = open("file1.txt", "r") f2 = open("file2.txt", "w") for line in f1.readlines(): f2.write(line) f2.close() f1.close() Айта кету керек, Python-дің нақты файлдарынан басқа, файлға ұқсас нысандар да пайдаланылады. Көптеген функцияларда, егер ол бірдей әдістерге (және сол мағынада) ие болса, оған типті файлдың немесе басқа түрдің нысанын тапсыру маңызды емес. Мысалы, мазмұнды (URL) file2.txt файлына көшіруге бірінші жолды ауыстыру арқылы қол жеткізуге болады import urllib f1 = urllib.urlopen("http://python.onego.ru") Өрнектер Қазіргі заманғы программалау тілдерінде деректерді өңдеудің көптеген түрлерін өрнектерде жасау дәстүрге айналады. Көптеген бағдарламалау тілдерінде өрнек синтаксисі шамамен бірдей. Python өрнектерінің синтаксисі бағдарламашыға жаңа нәрсемен таңданбайды. (Егер тізбекті салыстыру мүмкін болмаса). Операциялардың басымдылығы келесі кестеде көрсетілген (ұлғайту тәртібімен). Біртекті операциялар үшін x - операнды білдіреді. Python операцияларының ассоциациясы солнан оңға қарай, солдан оңға қарай ассоциативті күшке (**) көтерілу операциясын қоспағанда.

Операция Атауы lambda лямбда-өрнек or логикалық НЕМЕСЕ and логикалық ЖӘНЕ not x логикалық ЕМЕС in, not in тиістілігін тексеру is, is not жеке басын тексеру

```
<,<=,>,>=,!=,==
```

салыстыру | биттік НЕМЕСЕ ^ биттік НЕМЕСЕ & биттік ЖӘНЕ

```
<<, >>
```

биттік ығысу +, - қосу және азайту *, /, % көбейту, бөлу, қалдықтар +x, -x унарнлы плюс және таңбаны ауыстыру ~x биттік ЕМЕС ** дәрежеге шығару x.атрибут атрибутқа сілтеме

```
x[индекс]
```

индексі бойынша элемент алу

```
x[от:до]
кесуді таңдау (оның ішінде)
f(аргумент,...)
```

функция шақыру

```
( ... )
```

жақшалар немесе мәндер жолы

```
[ ... ]
```

тізім немесе тізімге қосу

```
{кл:зн, ...}
```

кілт-мән жұптарының сөздігі `выражения`

```
жолға қайта айналдыру (repr)
```

Осылайша, операндтарды есептеу тәртібі осындай ережелермен анықталады:

1. Экспоненциалды қоспағанда, сол жақтағы операнда барлық екілік операцияларда оң жақтан операнда алдында есептеледі.
2. <b <c ... y <z түріндегі конгруэнциялар тізбегі іс жүзінде баламалы: (a <b) және (b <c) және ... және (y <z).
3. Операцияның нақты орындалуына дейін оған қажетті операндалар есептеледі. Ең екілік операциялар бұрын операнда (бірінші сол жақ), бірақ нәтиже алу үшін жеткілікті болып табылады операндтар сомасын есептеу операциялары мен немесе мен, және салыстыру тізбек екі есептелген. Өрнек түсініксіз бөлігінде бұл жағдайда тіпті белгісіз атаулар болуы мүмкін. Бұл жанама әсерлері бар функцияларды қолданатындығын ескеру маңызды.
4. Функциялардың дәлелдері, тізімдер үшін өрнектер, кассеталар, сөздіктер және т.б. өрнектегі тәртіп бойынша солдан оңға қарай бағаланады. Приоритеттердің анықталмаған жағдайда, жақшаларды пайдалану керек. Әр түрлі операциялар үшін сол таңбаларды қолдануға болатынына қарамастан, операциялардың басымдықтары өзгермейді. Мәселен,% -ы * сияқты бірдей басымдыққа ие, сондықтан келесі мысалда жақшалар көбейту әрекетін пішімдеу жұмысына дейін орындауға қажет: print "%i" % (i*j) Өрнектер көптеген Python мәлімдемесінде, тіпті тәуелсіз

операторда да пайда болуы мүмкін. Ешбір -. (Өрнек жанама әсерлерін үшін бағаланады) кейбір жағдайларда, бұл нәтиже «нәрсе» болуы мүмкін, дегенмен білдіру қатар, әрқашан нәтижесі болып табылады
Әдетте жиіліктер тағайындалған немесе кеңейтілген тапсырма операторының оң жағында. Python жылы (қарсы деп, айталық, C) синтаксис = белгісі ғана Identifier Code, тілімдерін, жоғарыда немесе төлсипат реттелуіне (тізім) қол тұра алады бұрын жерде тағайындау операторы болып табылады, сондықтан. (Құжаттағы мәліметтер). Аттар Аттар бірнеше рет айтылған, дегенмен, оларды Python тілінде қолдану туралы бірнеше сөз айту керек. Атаудың латынша хатпен (кез-келген тіркелімнен) немесе астын сызудан басталып, сандарды пайдалану рұқсат етіледі. Тіл кілт сөздерін идентификаторлар ретінде пайдалана алмайсыз және енгізілген аттарды елемеуді қаламайсыз. Кілт сөздердің тізімі келесідей: >>> import keyword >>> keyword.kwlist ['and', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'yield'] Астыңғы сызықпен немесе екі астыңғы сызықпен басталатын атаулар ерекше мәнге ие. Жалғыз астын сызу бағдарламалаушыға аттың жергілікті бағдарлама екенін және модульден тыс қолданылмауы керектігін айтады. Бастапқыда және соңында қосарланған астын сызу әдетте арнайы атрибут аттары берілген - бұл объективті бағдарланған бағдарламалау бойынша лекцияда талқыланады. Бағдарламаның әрбір нүктесінде аудармашы үш атауды «көреді»: жергілікті, жаһандық және кіріктірілген. Аттар кеңістігі - атаулардан объектілерге салыстыру.
Python айнымалы мәнді қалай табатындығын білу үшін код блогы түсінігін енгізу керек. Python-де, код блогы - бұл бір нысан ретінде орындалады, мысалы, функцияның анықтамасы, сынып немесе модуль.Локальные имена - имена, которым присвоено значение в данном блоке кода. Жаһандық атаулар модуль анықтау кодының блоктық деңгейінде немесе жаһандық мәлімдемені анық көрсететін атаулар болып табылады. Енгізілген атаулар __ салынған __ сөздіктің аттары болып табылады. Аттар кеңістіктері бір-біріне кірістірілуі мүмкін, мысалы, шақырылатын функцияның ішінде, қоңырау кода анықталған атаулар көрінеді. Код блокында пайдаланылатын, бірақ кодтан тыс мәнге қатысты айнымалылар бос айнымалылар деп аталады. Айнымалысы блокта кез келген жерде нысанмен байланыстырылуы мүмкін болғандықтан, ол бұған дейін бұл орын алуы маңызды, әйтпесе, NameError ерекшелігі көтеріледі. Есімдердің мәндерге байланысы функцияның функционалдық дәлелдерінде, функцияларды немесе сыныпты анықтауда, операторлардан басқа операторлардың екінші бөлігінің екінші параметрінде тағайындау операторларында, импорттан тұрады.

5. Көрінетін жерлер мен атауды байланыстыратын аймақтар бар, олар жақсы құжатталған көптеген нюанстар бар. Бағдарламалар осындай нюанстарға тәуелді емес, бірақ бұл үшін келесі ережелерді сақтау жеткілікті:Всегда следует связывать переменную со значением (текстуально) до ее использования.
6. Сіз жаһандық айнымалы мәндерден аулақ болуыңыз және барлық параметрлер ретінде өтуіңіз керек. Модульдік атаулар, сынып атаулары мен функциялары тек модульдік деңгейде қалады.
7. Сіз импорт * модулінен ешқашан пайдаланбауыңыз керек - бұл басқа модульдерден көлеңкелі аттардың пайда болуына әкелуі мүмкін, бірақ функцияның анықтамасына тек тыйым салынады. Жақсы жаһандық айнымалы мәнді пайдаланбастан, код қайта жазылған. Әрине, бір модульден тұратын бағдарламалар үшін бұл өте маңызды емес: өйткені модуль деңгейінде анықталған барлық айнымалылар жаһандық болып табылады. Del сөзін пайдаланып, атаудың атауын объектімен бірге жоюға болады. Бұл жағдайда, егер объект оған басқа сілтемелер бермесе, ол жойылады. Python-дағы жадты басқару үшін, (reference counting) сілтеме санау пайдаланылады және қоқыс жинағы пайдаланылған (garbage collection) сілтемелермен объектілер жиынтығын жою үшін пайдаланылады.

Бақылау сұрақтары 1.Объектілер дегеніміз не? 2.Функционалдық - логикалық программалаудың ерекшеліктері? 3.Логикалық программалау тілдері қайсылар? 4.Логикалық программалакдың қолданылу облыстары?

№ 11 тақырып Рекурсивті алгоритмдер. Бірігу алгоритмдері. Іштей сорттау алгоритмдері. Сырттай сорттау алгоритмдері. Рекурсивті функциялармен жұмыс.

Жоспары:

1 РЕКУРСИЯЛАР, РЕКУРРЕНТТІКТЕР ЖӘНЕ ИТЕРАЦИЯЛАР

2 СҰРЫПТАУ АЛГОРИТМДЕРІ. МАССИВТЕРДІ СҰРЫПТАУ (ІШКІ СҰРЫПТАУ)

3 ІШКІ СҰРЫПТАУ ӘДІСТЕРІ

a1 a2 a3 a4 a5 a6 Орындалатын операциялар қадам1 15 33 42 07 12 19 салыстыру 19 және 12, ауыстыру жоқ

15 33 42 07 12 19 салыстыру 12 және 07, ауыстыру жоқ

15 33 07 42 12 19 салыстыру 07 және 42, орындарын ауыстырамыз

15 07 33 42 12 19 салыстыру 07 және 33, орындарын ауыстырамыз

07 15 33 42 12 19 салыстыру 07 және 15, орындарын ауыстырамыз; 07 – минималды қадам2 07 15 33 42 12 19 салыстыру 19 және 12, ауыстыру жоқ

07 15 33 12 42 19 салыстыру 12 және 42, орындарын ауыстырамыз

07 15 12 33 42 19 салыстыру 12 және 33, орындарын ауыстырамыз

07 12 15 33 42 19 салыстыру 12 және 15, орындарын ауыстырамыз, 12 –екінші минималды. қадам3 07 12 15 33 19 42 салыстыру 19 және 42, орындарын ауыстырамыз

07 12 15 19 33 42 салыстыру 19 және 33, орындарын ауыстырамыз

07 12 15 19 33 42 салыстыру 19 және 15, ауыстыру жоқ, 15 – үшінші минималды. қадам4 07 12 15 19 33 42 салыстыру 42 және 33, ауыстыру жоқ

07 12 15 19 33 42 салыстыру 33 және 19, ауыстыру жоқ, 19 – төртінші элемент. қадам5 07 12 15 19 33 42 салыстыру 42 және 33, ауыстыру жоқ, сұрыптау аяқталды

07 12 15 19 33 42

Нәтижесінде, алты элемент үшін 5+4+3+2+1 = 15 салыстыру және 8 орын ауыстырулар өткізілді. Жалпы жағдайда, әр n-1 қадамда орташа n/2 салыстырулар өткізіледі, сондықтан салыстырулар саны үшін бағалау n(n-1)/2 жазбасымен бейнеленеді, яғни берілген әдіс O(n2) классына байланысты. Аналогті түрде, орын ауыстырулар саны да n2 мәніне пропорционалды. Бұл әдіс ең тиімді емес сұрыптау әдісі болып табылады. Мысалы, 1000 элементтер үшін салыстырулар саны 500 мыңға жуық болады. Бағдарламалық түрде іске асыру екі циклдан тұрады: сыртқы нәтижі –алгоритмнің негізгі қадамдары, ішкі – элементтерді салыстырады және орындарын массивтің соңынан бастап ауыстырады. for i := 2 to n do begin for j := n downto i do

```
        if  a[j-1] > a[j]  then
           begin  temp := a[j-1];  a[j-1] := a[j];  a[j] := temp;  end;
```

end; Таңдау көмегімен сұрыптау. A массивінде мәліметтердің n элементі сақталған және бұл массив бойынша n-1 жүріс етеді. 0-ші жүрісте ең кіші элемент таңдалады. Ол кейіннен A0 элементпен айырбасталады. Келесі жүрісте тізімнің A1 элементінен бастап реттелмеген бөлігі қарастырылады. Мұнда ең кіші элемент тауып алынады да A1-де сақталады. Ары қарай A2 ... An-1 тізіміндегі ең кіші элемент ізделеді. Табылған мән A2-мен ауысады. Осылайша, n-1 жүріс өтеді. Соңында тізімнің реттелмеген аяғы 1 элементке дейін қысқарады. Сол элемент ең үлкен болып табылады. Мысалға, 50,20,40,75,35 массив берілген.

```
0-жүріс. 20-ны таңдаймыз. Оны A0-мен ауыстырамыз (A0=50).
```

20,50,40,75,35. 1-жүріс. 35 таңдаймыз. Оны A1-мен орын ауыстырамыз. 20,35,40,75,50. 2-жүріс. 40 таңдаймыз. Оны A2-мен орын ауыстырамыз. 20,35,40,75,50. 3-жүріс. 50 таңдаймыз. Оны A3-пен орын ауыстырамыз. 20,35,40,50,75. Соңғы қалған 75 саны ең үлкен элемент сұрыпталып шыққанда: 20,35,40,50,75. Сұрыптау – массив өлшеміне ғана тәуелді салыстырулардың белгіленген санына ие болуы керек. i-ші жүрісте (A... A)-ге дейінгі элементтердің салыстырулар саны (n-1-(i+1)+1)=n-i-1 тең болады. Салыстырулардың жалпы саны мына формуламен анықталады:

```
Алгоритмнің күрделілігі – O(n) тең болады.
```

Есептеу күрделілігі. Массив реттелген болған жағдайда бүкіл тізім бойынша 1 ғана жүріс өтеді. Мұнда тиімділігі O(n)-ға тең. Ал ең тиімсіз жағдайда i-1 жүріс орындалады және i-ші жүрісте n-i-1 салыстыру жүргізіледі. Ең тиімсіз жағдайда тиімділігі O(n2) тең. Жалпы жағдайда таңдау арқылы сұрыптау көбікше арқылы сұрыптауға қарағанда ауыстырылатын сан аздығымен тиімді болады. Шейкер сұрыптау алгоритмі элементтердің барлығы немесе көпшілігі сұрыпталған жағдайда пайдаланған тиімді. Қою арқылы сұрыптау. Қою арқылы сұрыптау – келесі процеске ұқсас. Карточкаларға аттарды жазып, карточкаларды алфавит бойынша өзіне керекті орынға қыстырып қою арқылы реттеу. Мысалға: 50,20,40,75,35 массивін қыстыру арқылы сұрыптау керек. 50 элементінен бастаймыз. 20-ны 0 позициясына қыстыру, 50-ді 1 позициясына жылжыту. 40-ты 1 позициясына қыстыру, 50-ді 2 позицияға жылжыту. 75-ті 3 позициясына қыстыру. 35-ті 1 позициясына қыстыру, қалғандарын оңға қарай жылжыту. Есептеу күрделілігі: жалпы салыстырулар саны -ге тең. Ең жақсы жағдайда, яғни тізім реттелген жағдайда күрделілігі O(n)-ге , ал жаман жағдайда O(n2)-ге тең. Бинарлық қоюлар арқылы сұрыптау. Бұл жай енгізулермен сұрыптаудың жақсартылған варианты. Жаңа элементті қосуға қажетті дайын тізбек реттелген болып, енгізу орнын неғұрлым жылдам табуға негізделген. Ол үшін дайын тізбектің ортаңғы элементі ізделіп, ары қарай ортасынан бөлу қашан енгізу орыны анықталғанша жалғаса беретін бинарлық іздеу жүргізіледі. Есептеу күрделілігі: Енгізу орыны табылады егер, al<=item<=ar болса. Соңында іздеу интервалы 1 ге тең болуы керек; бұл дегеніміз I элементтерден тұратын интервал ортасынан (log2i) рет бөлінеді деген сөз.

Салыстырулардың минимальды саны барлық элементтер кері ретпен орналасқанда, ал максимальды саны олардың осы кезде реттелген болғанында талап етіледі. Тез сұрыптау. Тез сұрыптау әдісі – күнделікті тәжірибеден алынған. Мысалы: Аттары бойынша алфавиттік карточкалар жиынын қандай да бір әріпке қатысты, мысалы К, екі кіші жиынға бөлуге болады. Барлық К-дан кіші немесе тең тең болатын бір жиынға, К-дан үлкен болатын бір жиынға бөлеміз. Одан кейін әрбір жиын тағы да екіге бөлінеді т.с.с. Тез сұрыптау алгоритмінде орталық элементті анықтап, сол арқылы бөлу әдісі қолданылады. Яғни, алғашқы массив екіге бөлінеді. Орталық элементтен үлкен және орталық элементтен кіші. Тура осы әрекет алынған массивтің 2 бөлігіне де жүргізіледі. Осылайша бөліне береді. Әрбір бөлікте бір ғана қалғанша жалғастырамыз. Сұрыптау принципі: Массивтің орталық элементі таңдап алынады. Массивтің барлық элементтері солдан оңға және оңнан солға қарай қарап өтіледі. I. Солдан оңға қарай қозғалғанда A[scan up] деген элементті іздейміз және бұл элемент орталық элементтен үлкен болуы керек, оның позициясын есімізге сақтап аламыз. Оңнан солға қарай қозғалғанда A[scan down] деген элементті іздейміз. Ол элемент орталық элементтен кіші немесе тең болады. Позициясын есте сақтаймыз. Табылған элементтердің орындарын ауыстырамыз және scan up және scan down индекстері қиылысқанша іздеуді жалғастырамыз. 1-этапты орындап болғаннан кейін алғашқы масситің элементтері орталық элементке бөлінеді. 2-этапта 1-этаптың әрекеттері массивтің оң жақ және сол жақ бөліктері үшін жеке-жеке орындалады. 3-этапта осы әрекеттердің барлығы 4 бөлігі үшін жеке-жеке орындалады. 4-этапта 4 бөлігі жеке-жеке орындалады. Есептеу күрделілігі. Тиімділік анализі кей жағдайда ғана мүмкін болады. Айталық, массив элементтер саны n=2 (K=log2n) 1-сканерлеуде n-1 салыстыру жүргізіледі. Оның нәтижесінің өлшемі өлшемді ішкі тізім пайда болады. Өңдеудің келесі фазасында әрбір ішкі тізім үшін n/2 салыстыру қажет болады. Осылайша, бөлу процесі табылған ішкі тізімдер тек бір ғана элементтер тұрғанша K жүрістен кейін аяқталады. Мұндағы салыстырулардың жалпы саны мына формуламен анықталады: n*k=n*log2n. Жалпы түрдегі тізім үшін есептеу күрделілігі – O(n log2n) тең болады. Ал ең нашар жағдайда орталық элемент ең кіші элемент болғанда есептеу күрделігі O(n2) тең болады.

Бақылау сұрақтары

- Деректердің ағаш түріндегі құрылымдары. Негізгі түсініктер.
- Ағаштарды ЭЕМ-да көрсету.
- Ағаштың құрылу процедуралары.
- Бинарлы ағашты айналып өту.

№ 12 тақырып Іздеу алгоритмдері. Сызықтық және екілік іздеу. Іздеу алгоритмдері.

Жоспары:

- Деректерді іздеу алгоритмі

- Аса бұтақталатын ағаштар

1. Деректерді іздеу алгоритмі Екілік ағашты айналу Ағашты айналу есебін тұтас тізбекті процесс деп қарастырсақ, онда түйіндерге белгілі бір ретпен қатысады және сызықты орналасқан деп есептеле алады. Бинарлы ағашты айналу үшін әдетте үш тәсілдің біреуі қолданылады:
2. жоғарыдан төменге айналу - A, B, C (ішкі ағаштарға дейін түбірді қарап шығу);
3. солдан оңға қарай айналу - B, A, C;
4. төменнен жоғарыға айналу - B, C, A (ішкі ағаштардан кейін түбірді қарап шығу). Осындай үш процедураның әрбіреуінде әртүрлі реттегі келесі қадамдар бар:
5. ішкі ағаштың түбірін қарап шығу,
6. сол ішкі ағашты айналу,
7. оң ішкі ағашты айналу. Біздің ағашты айнала отырып, және түйіндерде орналасқан символдарды олар қандай ретте кездеседі, сол ретте жазып ала отырып, келесі тізбектерді аламыз:

1) жоғарыдан төменге: *+a/bc-d*ef ;

```
(1)
```

2) солдан оңға: a + b / c * - e * f;

```
(2)
```

3) төменнен жоғарыға: abc/+def*-*;

```
(3)
```

Өрнектерді жазудың үш формасын алдық, олар

- – префиксті жазба; (2)- инфиксті жазба (бізге үйреншікті форма, бірақ операциялардың орындалуының ретін анықтауға арналған жақшаларсыз болса да);
- – постфиксті жазба. Енді барлық осы үш айналу әдісін ағашты білдіретін нақты Т параметрі бар және әрбір түйінде орындалатын операцияны білдіретін нақты емес P(T) параметрі бар процедура ретінде бейнелеуге болады. Анықтамалар енгіземізы: TYPE REF = ^NODE; NODE = RECORD ..... LEFT, RIGHT: REF END; Айналып өтудің аталып өткен үш әдісін рекурсивтік процедуралар ретінде көрсеткен оңай. Олар деректердің рекурсивтік анықталған құрылымдары рекурсивтік алгоритмдермен сипатталатындығының бейнесі болып келеді. Жоғарыдан төмен қарай: PROCEDURE PREFIX(T: REF); BEGIN IF T <> Nil THEN BEGIN P(T); { түйінді өңдеу} PREFIX(T^.LEFT); PREFIX(T^.RIGHT); END; END; Солдан оңға қарай: PROCEDURE INFDC(T: REF); BEGIN IF T<>Nil THEN BEGIN INFIX(T^.LEFT); P(T); INFIX(T^.RIGHT); END; END; Төменнен жоғары қарай: PROCEDURE POSTFIX(T; REF); BEGIN IF T <> Nil THEN BEGIN POSTFIX(T^.LEFT); POSTFIX(T^.RIGHT); P(T); END; END; Осы процедуралардағы Т сілтемесі параметр-мән ретінде жіберілетіндігін атап өтейік. Яғни мұнда мәні сілтеме болып табылатын және
- параметр-айнымалы болып жіберілетін жағдайда мәнді өзгерте алатын айнымалы емес, қарастырылатын бағыныңқы ағаштың сілтемесі маңызды болып табылады. Ағашты айналып өтудің мысалы – әрбір деңгейді ерекшелейтін сәйкес жылжуы бар ағаштың түйіндерін қағазға шығару бағдарламасы. PROCEDURE PRINTTREE(T: REF, H: INTEGER); { T – шыңға сілтеме, H – қағазға шығару кезіндегі жылжу } VAR I: INTEGER; BEGIN { T ағашын H жылжумен қағазға шығару} IF T <> Nil THEN WHITF T^ DO BEGIN PRINTTREE(LEFT, H+1); { сол жақ бұтақты айналып өту } FOR I := 1 TO H DO WRITEC (‘ '); { жылжу } WRITELN(KEY); { кілтті қағазға шығару } PRINTTREE(RIGHT, H+1) { оң жақ бұтақты айналып өту } END; END; Шақырушы бағдарлама: BEGIN

ROOT := TREE(N); { ТАМЫРҒА СІЛТЕМЕ } PRINTTREE(ROOT,0) END.

Іздеу мәселесі Келесі міндетті жиі орындауға тура келеді: кілттің керекті мәні бар элементті табу. Бұл міндетті массивтерді, тізім, ағаштарды қолданып орындау керек. Бұл мәселе элементтер кілттерін өсу ретімен реттелген және реттелмеген жағдайларда әртүрлі шешіледі. Соңғы жағдайда барлық элементтерді қарастырмай-ақ кілті талап етілгеннен үлкен элементке дейін ғана баруға болады. Іздеудің екі альтернативті әдісі бар : тізбектелген және екілік. Тізбектелген тізімде сілтемелердің тізбегі ретімен тізбекті іздеу ғана қолданылады. Екілік ағашта екілік іздеу орынды болады. Массивте алгоритмдердің екеуі де қолданыла алады. Міндетті былай қояйық: кілт мәні X-ке тең компоненттің ең кіші индексін (сілтемесін) табу. Массивтегі тізбектелген іздеу Массив мынау болсын:

```
VAR A: ARRAY[1..N] OF INTEGER;
```

X: INTEGER;

```
FUNCTION LOC(X: INTEGER): INTEGER;
```

VAR I: INTEGER; BEGIN

```
I:=0;
```

REPEAT I:=I+ 1

```
UNTIL(A[I]=X) OR(I=N);
IFA[I]<>X THEN LOC:=O
```

ELSE LOC := I END; Процедура реттелген және реттелмеген массивтерде де жұмыс істейді. Функцияның қолайсыздығы аяқталу шартын тексерудің күрделілігі. Бағдарламаны «бөгет» қойып тездетуге болады, яғни массивті бір элементке ұлғайтып ізделінетін кілттің мәнін соған жіберу керек.

```
    •    := 0; A[N+1]:=X;
REPEAT I:=I+1; UNTIL A[ I ]=X;

IF I > N THEN LOC := 0 ELSE LOC:= I;
```

Қарап шығудың орташа саны m = N / 2.

```
Массивтегі екілік (бинарлы) іздеу
```

Реттелген массивтер үшін қолданылады. Екілік іздеуде екіше бөлу әдісі пайдаланады. Алгоритмі:

```
I:=1;J:=N;
```

REPEAT

```
    •    := (I + J) DIV 2;
IF X > A[K] THEN I := K + 1 ELSE J := K – 1 UNTIL (A[K] = X) OR (I > J)
```

Қарап шығу саны m = log2 N. N = 100 болғанда: тізбектелген іздеу 50 қарап шығу ішінде орындалады (m = 50), екілік 6,62 ішінде (m = 6,62). Сызықты тізімде іздеу Файлдағыдай іздеу қатаң түрде ретпен орындалады. Ол элемент табылған кезде немесе тізімнің соңына жеткенде тоқтайды. Ең қарапайым шешім:

WHILE (P < > NIL) AND (P^.KEY < >X) DO P:=P^.NEXT

Егер элемент тізімде болмаса, онда P Nil-ге тең болса, келесі элементтің жоқ болғаны P^.KEY. Сондықтан тексерудің екінші бөлімін енгізу керек

```
WHILE P<> Nil DO
IF P^.KEY = X THEN { циклдан шығу }
ELSE P:=P^.NEXT;
Логикалық айнымалыны (жалаушаны) қолданған жөн:
    •    := TRUE;
WHILE (P <>Nil) AND B DO
IF P^.KEY = X THEN B :=FALSE
ELSE P := P^.NEXT;
Егер P = Nil және B = TRUE, онда элемент табылған жоқ.
```

Екілік ағашта іздеу Бинарлы ағаштар элементтері қандай да бір өзіне ғана тән, ерекше кілт бойынша ізделетін деректерді бейнелеуде жиі қолданылады. Егер ағаштың әрбір ti түйіні үшін сол жақ бағыныңқы ағаштың кілттері ti кілтінен кіші болса, ал оң жақ бағыныңқы ағаштың барлық кілттері ti кілтінен үлкен болса, онда бұл ағаш кілттер бойынша реттелген және іздеу ағашы деп аталады. Іздеу ағашында әрбір кілттің орнын тамырынан бастап, кілт мәніне байланысты оң жақ немесе сол жақ бағыныңқы ағашқа қарай өтіп табуға болады. N элементтен биіктігі log2 N көп емес екілік ағаш ұйымдасатынын көруге болады. Сондықтан егер ағаш идеалды блансалған боса, N эле-менттің ішінен іздеу кезінде log2 N көп емес теңестірулер керек. Іздеу үшін N теңестіру керек болатын сызықты тізімге қарағанда ағаштың артықшылығы осында. Реттелген екіік ағаш ішінде іздеу тамырынан ізделінген түйінге дейін бір ғана жолмен жүргізіеді (өйткені ол тамырынан ізделінетін түйінге дейін бір олмен жүреді) және LOC итерациялық функциясымен сипатталады.

```
FUNCTION LOC(X: INTEGER; T: REF): REF;
```

VAR FOUND: BOOLEAN; BEGIN

```
FOUND :=FALSE;
WHILE (T <> Nil) AND NOT FOUND DO
IF T^.KEY = X THEN FOUND := TRUE
ELSE IF T^.KEY > X THEN T:=T^.LEFT ELSE T :=T^.RIGHT
```

LOC:=T END; LOC(X, T) функциясы Nil мәніне ие болады, егер тамыры T болатын ағашта X мәнді кілт табылмаса. Циклден шығу шартын ағаштың барлық «жапырақтарын» тұйықтайтын тағы бір шектелген түйін ұйымдастырып жеңілдетуге болады Барлық жапырақтары бір зәкірге немесе бөгетке байланатын ағаш алдық. Онда іздеудің қарапайымдалған процедурасы былай жазылады:

```
FUNCTION LOC(X: INTEGER; T: REF): REF;
```

BEGIN

```
S^.KEY:=X;
WHILE T^.KEY <>X DO
IF X < T^KEY THEN T := T^.LEFT
ELSE T := T^.RIGHT;
```

LOC:=T END; Егер тамыры T болатын ағашта X мәні бар кілт табылмаса, онда бұл жағдайда LOC(X, T) S мәнін, яғни бөгетке сілтемені қабылдайды. S-ке сілтеме өзіне Nil сілтемесінің ролін алады. Ағашқа жаңа түйін қосу Ағашқа жаңа түйін қосудың көрнекті мысалы ретінде жиілік сөздігін құру болып табылады. Бұл жағдайда сөздер тізбегі берілген, әрбір сөздің кездесу жиілігін табу керек. Бұл әрбір бос ағаштан бастап, әр сөз ағаштан ізделінетіндігін білдіреді. Егер ол табылса, оның кездесу санағышы бір көрсеткішке ұлғаяды, табылмаса – осы сөз ағашқа енгізіледі (санағыштың бастапқы

мәні 1-ге тең. Мұны ағаштан қосып іздеу деп атауға болады. Қарапайымдылық үшін бастапқы мәтіннен сөздер ерекшеленіп, бүтін сандармен кодталған және енуші файлда орналасқан деп болжаймыз. Типтер былай сипатталған болсын:

```
TYPE REF = ^WORD;
WORD = RECORD
```

KEY: INTEGER; COUNT: INTEGER; LEFT, RIGHT: REF END; Кілттердің F бастапқы файлы бар деп есептейік, ал ROOT айнымалысы іздеу ағашының тамырын білдірсін:

```
RESET (F);
WHILE NOT EOF(F) DO BEGIN
READ (F, X);
SEARCH (X, ROOT)
```

END;

```
Мұнда SEARCH (X, ROOT) – іздеу процедурасы. Оның сипаттамасы:
PROCEDURE SEARCH (X: INTEGER; VAR P: REF); BEGIN IF P = Nil THEN BEGIN NEW(P);
```

WITH P^ DO BEGIN

```
KEY :=X; COUNT:=1;
LEFT:=Nil; RIGHT:=Nil
```

END END ELSE

```
IF X < P^.KEY THEN SEARCH(X, P^.LEFT) ELSE
IF X > P^.KEY THEN SEARCH(X, P^.RIGHT) ELSE
```

P^.COUNT := P^.COUNT +1

```
END {SEARCH }
```

- параметрі параметр-мән ретінде емес, параметр-айнымалы ретінде жіберілетіндігіне назар аударайық. Бұл маңызды, себебі оған сілтеменің жаңа мәні беріледі. Осыны есепке ала отырып, сонымен қоса сөздердің кілттері INPUT файлынан алынады десек, негізгі бағдарламаны былай жазуға болады : BEGIN ROOT :=Nil; WHILE NOT EOF DO BEGIN READ (K); SEARCH (K, ROOT); END; PRINTTREE (ROOT, 0) END. ROOT және K айнымалылары VAR ROOT: REF; K: INTEGER; деп сипатталу керек Салыстыру үшін SEARCH процедурасының версияларын рекурсияның қолданылуынсыз жасайық. Қосуды жасау үшін өтілген жолды кем дегенде алдындағы бір қадамды есте сақтау керек. Кезекті қосылатын компонентаны дұрыс қосу үшін оның арғы аталарына сілтемеміз болу керек және оның бағыныңқы ағаштардың қайсысы: сол жақ немесе оң жақ болып қосылатынын білуіміз керек. Ол үшін екі айнымалы енгізіледі: P2 мен D: PROCEDURE SEARCH (X: INTEGER; ROOT: REF); VAR PI, P2: REF; D: INTEGER; BEGIN P2 := ROOT; PI := P2^.RIGHT; D :=1; WHILE (PI <> Nil) AND (D <> 0) DO BEGIN P2:=P1; IFX<P1^.KEY THEN BEGIN P1:=P1^.LEFT; D:=-1 END ELSE IF X>P1^.KEY THEN

```
BEGIN P1:=Pl^.RIGHT; D := 1
```

END ELSE D := 0 END;

```
IF D = 0 THEN Pl^.COUNT := Pl^.COUNT + 1
```

ELSE BEGIN

```
NEW(Pl);
```

WITH P1^ DO BEGIN

```
KEY := X; LEFT :=Nil;
RUGHT :=Nil; COUNT := 1
```

END;

```
IF D < 0 THEN P2^.LEFT := P1
ELSE IF D > 0 THEN P2^.RIGHT := P1
```

END END; Мұнда екі сілтеме қолданылады: P 1 және P2, іздеу процесінде P2 әрқашан P1^ предикатына сілтейді. Бұл шартты іздеу басында қанағаттандыру үшін root сілтейтін қосымша фиктивті элемент енгізіледі. Нақты іздеу ағашының басы ROOT^.RIGHT сілтемесімен белгіленеді. Сондықтан бағдарлама келесі операторлардан басталу керек

```
NEW(ROOT); ROOT^.RIGHT :=Nil;
(бастапқы мән берудің орнына ROOT := Nil)
```

Ағаштан жою Жою – қосуға кері амал. Реттелген кілттері бар ағаштан X кілті бар түйінді жою алгоритмін құру керек. Элементті жою әдетте қосу сияқты оңай емес. Ол жойылатын элемент терминалды түйін болса немесе бір де ұрпағы болмаған жағдайда оңай болады Екі ұрпағы бар элементтерді жою кезінде қиыншылықтар туындайды, өйткені біз екі бағытта бір сілтемені қолдана алмаймыз. Бұл жағдайда жойылушы элементті оның сол жақ бағыныңқы ағашының оң жағындағы элементке ауыстыру керек немесе оның оң жақ бағыныңқы ағашының сол жағындағы элементке ауыстыру. Ондай элементтердің бірден көп ұрпағы болмайтыны анық. Жою процесін DELETE рекурсивті процедурасымен көрсетуге болады. Оның үш жағдай бар:

- X-ке тең кілті бар компонента жоқ;
- X-ке тең кілті бар компонентаның бірден көп емес ұрпағы бар;
- X-ке тең кілті бар компонентаның екі ұрпағы бар. PROCEDURE DELETE (X: INTEGER; VAR P: REF); VAR Q: REF; PROCEDURE DEL(VAR R: REF); BEGIN IF R^.RIGHT <> NIL THEN DEL(R^.RIGHT) ELSE BEGIN Q^.KEY := R^.KEY; Q^COUNT := R^.COUNT; Q := R; R := R^.LEFT END END { DEL } BEGIN { DELETE } IF P = NIL THEN WRITELN(‘ берілген кілтпен сөз табылған жоқ ') ELSE IF X < P^.CEY THEN DELETE(X, P^.LEFT) ELSE IF X > P^.KEY THEN DELETE (X, P^.RIGHT) ELSE BEGIN { P сілтемесі бойынша жою} Q:=P; IF Q^.RIGHT = NIL THEN P := Q^.LEFT ELSE IF Q^.LEFT = NIL THEN P := Q^.RIGHT ELSE DEL(Q^.LEFT); DISPOSE(Q) END END {DELETE } DEL қосымша рекурсивтік процедурасы үшінші жағдайда ғана шақырылады. Ол жойылатын Q^ түйінінің сол жақ бағыныңқы ағашының ең оң бұтағы бойымен «төмен түсіп», Q^-ғы бар ақпаратты (кілт пен санағыш)осы сол жақ бағыныңқы ағаштың R^ ең оң компонентасының сәйкес мәндеріне алмастырады, осыдан кейін R^-ден құтылуға болады. Бұл DISPOSE(Q) стандартты процедурасы арқылы жүзеге асады.

Процедура жұмысын 22-сур. бойынша тексеруге болады.

1. Аса бұтақталатын ағаштар Ақпараттың бинарлы ағаштар түрінде келтірілуі деректердің иерархиялық құрылымымен байланысты мәселелердің көпжақтылығын көрсетпейді. Бұл жағдай туыстық қатынастарды біз «жоғарылайтын сызық», яғни мысалы, бір адамды оның ата-анасы арқылы көрсеткіміз келгенде жеткілікті болады. Ал егер «төмендейтін сызық» арқылы көрсететін болсақ, онда ағашта көп бұтағы бар түйіндерден тұратын болады. Ондай ағаштар аса бұтақталған ағаштар деп аталады.

Мұндай жағдайларды түрлі жолмен түзетуге болады. Егер, мысалы, балалар санының абсолютті жоғарғы шегі берілсе, онда балаларды адамды бейнелейтін жазбадағы компонент-массив түрінде көрсетуге болады. Бұл жадының үнемделмеуіне соқтырады.

Сондықтан түйінде ең сол элементке (ең кіші немесе ең үлкен ұрпаққа) сілтеме болғанда, ал бір деңгейдің барлық элементтері тізім құрғанда аса бұтақталған ағаштың екілік көрінісін жиі пайдаланады (23-сур.).

Сурет 2. Аса бұтақталған ағаш

Бұл да нашар шешім, себебі мұндай сілтемелер функционалды түрде басқа мағынаға ие болады. Құрылымға басқа да туыстық қатынастарды қосу олардың бірнеше ағашпен бейнеленетін күрделі реляциялық деректер банкіне тез өсуіне әкеліп соғады. Осындай құрылымдармен жұмыс істейтін алгоритмдер олардың сипаттамаларымен тығыз байланысты, сондықтан олар үшін жұмыстың қандай да жалпы ережелерін анықтауды қажет етпейді. Алайда аса бұтақталған ағашты қолданудың жалпы қызығушылық туғызатын практикалық маңызды облыс бар. Ол – қосу мен жою қажеттілігі бар ірімасштабты іздеу ағаштарды құру мен қолдану, алайда оны ұзақ уақыт сақтауда оперативтік жады жеткіліксіз немесе қымбат болады. Ағаш сыртқы есте сақтау құрылғысында, мысалы, дискте сақталу керек делік. Осы мақсатта деректердің динамикалық құрылымдарын пайдаланған дұрыс. Мұндағы жаңалық – мұндағы сілтемелер оперативті жадыдағы емес, дисктегі адрестерді білдіреді. Дисктегі іздеу уақыты ұзағырақ, себебі дискке сұраныс беру бастиектерінің орын ауыстыры мен бұрылуына күту уақытын қажет етеді. Егер 1 млн . элементтен тұратын деректер жиынтығы бинарлы ағаш түрінде сақталса, онда элементті іздеу үшін орта есеппен log2106 = 20 іздеу қадамы керек. Сондықтан сұраныс берудің аз санын қажет ететін жадыны ұйымдастыру керек. Аса бұтақталған ағаш – берілген проблеманың идеалды шешімі. Егер сыртқы құрылғыда орналасқан жалғыз элементке сұраныс жүргізілсе, онда қосымша шығынсыз элементтердің тобына да сұраныс жасауға болады. Яғни ағашты барлық бағыныңқы ағаштар бір уақытта қол жеткізерлік деп алып бағыныңқы ағаштарға бөлу керек. Ондай бағыныңқы ағаштарды беттер деп атайық (24-сур.). Енді дискке бір сұраныс жасау үшін біз толық бір бетке сұраныс жасаймыз.

Сурет 3. Беттерге бөлінген бинарлы ағаш

Сонымен, беттегі түйіндер санын ұлғайтып, сұраныс санын азайтуға болады. Бір бетке 100 түйін орналастырдық делік. Онда 1 млн. түйіні бар іздеу ағашы беттеріне жасалатын сұраныс санының 20-ң орнына орташа есеппен алғанда log100106 = 3 сұраныс керек. Сонымен қоса аса бұтақталған ағаштар жағдайында олардың өсуін басқару схемасы керектігін ескеру керек, себебі егер ағаш кездейсоқ ретпен өсетін болса, онда ең қиын жағдайда сұраныс саны 104 болуын талап етуі мүмкін. Б-ағаштар Өсуді басқару критерийлері түрлі болу мүкін. Идеалды баланстықты бірден жоққа шығару керек, өйткені ол балансталуға үлкен шығын талап етеді. Одан жұмсағырақ китерийді Р. Бэйер ұсынды: әрбір бетте (біреуінен басқа) берілген n тұрақтысы жағдайында n-нен бастап 2n- е дейін түйін бар. N ағаштың реті деп аталады. Яғни N элементі бар ағашта ең жаман жағдай жолдардың максималды саны 2n түйін болғанда беттерге log2N сұраныс талап ететін болады. Мұнда жадыны қолдану коэффициенті 50%-дан кем емес, себебі беттер жартылай болса да толық келеді. Осы барлық артықшылықтарын атап өткенде берілген схема іздну, қосу және жоюдың қарапайым алгоритмдерін талап етеді. Қарастырылатын деректер беттері Б-ағаштар деп аталады. Б-ағаштардың қасиеттері:

```
    •    әрбір бетте 2n элементтен (кілттен) артық емес;
```

- әрбір бетте, түпкісінен басқа, n элементтен кем емес;
- әрбір бет не жапырақ болып келеді, яғни ұрпақтары жоқ, не m+1 ұрпағы бар болады, мұндағы m – онда орналасқан кілттер саны;
- барлық жапырақтар бір деңгейде орналасады.
- деңгейі бар, 2-ретті Б-ағаштың мысалы:

- ғана элементі бола алатын тамырынан басқа барлық беттердің 2, 3 не 4 элементі бар. Барлық жапырақтар 3-деңгейде. Егер жоғарыдағы бетте орналасқан кілттердің арасына ұрпақтарды қосқанда, ағаш бірдеңгейлі деп алғанда, кілттер солдан оңға қарай өсу ретімен орналасқан. Осындай орналасу кезінде Б-ағаш іздеудің бинарлы ағаштарын ұйымдастыру принципінің заңдылықты дамуын көрсетеді. Осыдан берілген кілт арқылы іздеу әдісі шығады. Мынандай түрде берілген бетті қарастырайық:

Яғни оның m кілті бар. Іздеудің X аргументі берілсін делік. Бет оперативті жадыға көшірілген деп алып, ki, ..., Км кілттері арасынан іздеудің белгілі әдістерін қолдана аламыз. (Үлкен m кезінде - бинарлы іздеу, үлкен емес кезінде - тізбектелген). Оперативті жадыдағы іздеуге кеткен уақыт бетті оперативті жадыға көшіруге кеткен уақытпен салыстырғанда әлдеқайда аз. Егер іздеу нәтижесіз болса, келесі жағдайлар болу мүмкін:

```
    •    1 <= i < m үшін ki < x < ki+1. Іздеуді pi^ бетінде жалғастырамыз;
    •    x > km. Іздеуді pm^бетінде жалғастырамыз;
```

- x < kl. Іздеуді p0^бетінде жалғастырамыз; Егер қандай да бір жағдайда сілтеме nil-ге тең болса, онда x кілтті элемент жоқ, іздеу тоқтатылады. Б-ағашқа қосу да өте оңай жүргізіледі. Егер элемент m<2n элементі бар бетке қойылса, онда қосу процесі тек осы бетпен ғана шектеледі. Егер бет толық болса, онда ағаш құрылымы өзгереді де жаңа беттер пайда болуы мүмкін. 2-ретті ағашты қарастырайық: Ағашқа 22 кілтін қосу керек делік. Қосу процесі келесі кезеңдерге бөлінеді:
- 22 кілті жоқ екені анықталады. C бетіне қосу мүмкін емес, яғни ол толық;
- C беті C және D беттеріне бөлінеді;
- m+1 кілттер саны C және D арасында тең бөлінеді, ал ортаңғы кілт бір деңгейге жоғары, A жоғарғы бетіне орналасады. Алатынымыз:

Алынған ағаш Б-ағаштың барлық қасиеттерін сақтап қалады. Дербес жағдайда, бөліну кезінде n элементі бар беттерді аламыз . Әрине, элементті жоғарғы бетке қосу сол беттің шектен тыс толып кетуіне соқтыруы мүмкін. Бөліну таралып, тамырына дейін жетуі мүмкін. Бұл жағдайда ағаш биіктігі ұлғаяды. Б-ағаш жапырақтарынан тамырына қарай өседі деп айтуға болады. Ендеше, қосу алгоритмін рекурсивтік әдіспен сипаттау дұрысырақ болатын сияқты , себебі бөліну процесі іздеу жолы бойымен кері қарай таралуы мүмкін. Алдымен элементтерді массив түрінде орналастырылатын бетті сипаттайық: TYPE PAGE = RECORD M: INDEX; P0: REF;

E: ARRAY[1..NN] OF ITEM END;

мұндағы

```
CONST NN = 2*N;
TYPE REF = TAGE;
```

INDEX - O..NN;

```
ITEM = RECORD
```

KEY: INTEGER; P:REF; COUNT: INTEGER END; COUNT компонентасы іздеу процесінде ешқандай роль орындамайтын ақпаратты алмастырады. Әрбір бетте 2n элемент үшін кеңістік бар. m өрісі шындығында қанша элемент орналасқанын көрсетеді. Қосу арқылы іздеу алгоритмі өте қарапайым және құрылымы жағынан бинарлы ағаштан іздеуге ұқсайды, бар айырмашылығы - әрі қарай жүретін жол мүмкін болатын екі бұтақтан аңдалмайды. Беттің ішіндегі іздеуді массивтегі бинарлы іздеу сияқты жасаған жөн. Қосу арқылы іздеу процедурасы схема түрінде былай болады:

```
PROCEDURE SEARCH(X: INTEGER; A: REF;
VARH: BOOLEAN; VAR U: ITEM);
```

BEGIN

```
IF A = NIL THEN BEGIN {X ағашта жоқ}
```

U ағаш бойымен жоғары жіберілетіндігін көрсете отырып, x мәнін U элементіне беру және h-ті true деп орнату END ELSE

```
WITH A^ DO BEGIN {x–ті a" бетінен іздеу} массивтегі екілік іздеу;
```

IF табылса THEN элементтің шығу санағышын ұлғайту ELSE BEGIN

```
search(X, ұрпақ, H, U);
IF H THEN {U элементін жоғары беріп жіберу}
IF (A^–ғы элементтер саны)<2N
```

THEN A^ бетіне қосу және орнату

- false-те ELSE орташа элементті жоғары қарай жіберу және беттің бөлінуі END END END; Мұнда h бульдік айнымалысы U екінші шығу параметрі қосу бетіне беттелген жоғары қарай жіберілетін элемент (h=true болғанда) болып табылады. Әсіресе тамыр-беттің бөлінуі болатын жағдайды алдын-ала қарастыру керек. Бұл жағдайда жаңа тамырлы бетті құрып, U параметрі арқылы берілген бір элементті қосу керек. Сөйтіп, тамыр-бетінің тек бір элементі ғана болады. WITH операторын қолдану сол операторлардың құрамындағы беттің өрістері идентификаторлары автоматты түрде A^ бетіне жататындығын ғана білдірмейді. Егер беттер сыртқы жадыда орналасса, WITH операторы қосымша көрсетілген бетті оперативті жадыға жіберуді білдіреді. Сондықтан SEARCH-ке әрбір жолығу жадыда бір беттің орналасуын көрсетеді. Барлығы k = lognN рекурсивтік жолығулар керек. Мұндағы N – ағаш элементтерінің саны. Сөйтіп, біз жадыға K бет орналастыру мүмкіндігін жасауымыз керек. Бұл 2N бетінің өлшемдеріне шектеу салады. Бірақ орналасатын беттер саны K-дан көбірек болу мүмкін, себебі қосу олардың бөлінуіне әкеліп соғуы мүмкін. Тамырлы бетті әрдайым жадыда сақтаған жөн, себебі әрбір іздеу тамырдан басталады. Қосылатын кілттердің келесі тізбегі кіретін ағашты құру процесін қарастырайық (25-сур.): 20; 40; 10 30 15; 35 7 26 18 22; 5; 42 13 46 27 8 32; 38 24 45 25; Б -ағаштан элементті жою жалпы айтқанда оңай болғанымен, іс жүзінде қиынға түседі. Мұнда екі жағдайды бөліп айтуға болады. Жойылатын элемент жапырақ-бетте орналасқан, онда жою алгоритмі қарапайым және анық. Элемент жапырақта емес. Онда оны жапырақ-беттерде орналасқан және оңай жойылатын екі лексикографикалық іргелес элементтің біреуіне алмастыру керек. Екінші жағдайда іргелес кілтті іздеу сәйкес бинарлы ағаштағы іздеудің аналогы болады. Біз сол жақ бағыныңқы ағаштың оң жақ сілтемелері бойымен P жапырағына қарай төмен түсеміз, жойылатын элементті ең оң жақтағы P элементіне алмастырамыз, сосын P өлшемін 1-ге азайтамыз. Бірінші жағдайда да, екінші жағдайда да элементті жойғаннан кейін кішірейтілген беттегі m элемент санын тексеруіміз керек, өйткені m < n болғанда Б-ағаштардың негізгі қасиеті бұзылады. Егер сондай жағдай болса, біз көршілес орналасқан Q беттерден бір элементті алып қоюмыз керек. Бұл Q бетін оперативті жадыға көшіруді талап еткендіктен, бізге қымбатқа түседі, сондықтан Q-дан бірден көбірек элемент алуымызға тура келеді. Әдетте P және Q беттерінің элементері екі бетке де тең бөлінеді (бұл баланстану деп аталады). Алайда Q-дан элемент алуға болмайтын жағдай да туындауы мүмкін, өйткені ол өзінің минималды n өлшеміне жеткен. Бұл жағдайда P және Q беттеріндегі элементтердің жалпы саны 2n-1 тең және біз жоғарыдағы беттен бір элемент (орташа) қосып, бұл беттерді бір бетке қосуымызға болады. Процесс беттердің бөліну процесіне қарағанда кері болып табылады (шыққан ағаштан 22 кілтін жоюға талаптанып, сол процесті біздің мысалдан көруге болады Бірақ жоғарғы беттегі орташа кілтті жою оның өлшемін мүмкін n шегінен төмен азайтуға әкеліп соғуы мүмкін. Онда ары қарай жоғарғы деңгейдегі баланстау немесе бірігу қажет болады. Бұл процесс тамырға жүретін жол бойымен таралуы мүмкін. Егер тамыр 0-ге дейін кемісе, ол жойылады, ал ағаш биіктігі азаяды. Ағаштың ыдырау процесін мысал түрінде қарастырайық. Бастапқы екінші ретті ағаш болсын (27 а-сур.), одан тізбектеп келесі кілттерді жоюға болады: 25, 45, 24; 38, 32; 8, 27, 46, 13, 42; 5, 22; 18,26, 7, 35, 15; (; - секірістер, яғни беттерді босату)

Бақылау сұрақтары

- Ағаштан түйінді жою.
- Аса бұтақталған ағаштар.
- Б-ағаштар.
- Б-ағаштардың қасиеттері.

№ 13 тақырып Қатарды өңдеу алгоритмдері. Кестелерді толтыру алгоритмі. Ішкі жолдарды іздеу алгоритмдері. Қатарды өңдеу алгоритмдері, кестелерді толтыру алгоритміне кіріспе.

Жоспары:

- Қатарды өңдеу алгоритмдері
- Қатарларды түрлендіру

1. Қатарды өңдеу алгоритмдері Жолдық қатарлармен біріктіру және салыстыру амалдарын орындауға болады. Біріктіру амалы бірнеше жолдық қатарды біріктіріп шығару үшін қолданылады. Біріктірілетін жолдық қатардың ұзындығы 255-тен аспауы тиіс.Мысалы: A:=’Менің’; B:=’Қазақстанымның’; Writeln (A+’ ’+B) {Менің Қазақстанымның – тіркесі шығады}

X:=’Тәуелсіздігіне’; Y:=’10 жыл’; Z:=X+’ ’+Y; {Z Тәуелсіздігіне 10 жыл- тіркесін меншіктейді} Writeln (Z); {де, осы тіркесті экранға шығарады} Жолдық қатарларды салыстыру амалы екі жолдық қатарды салыстыру үшін қолданылады. Қатарларды салыстыру солдан оңға қарай ең бірінші кездесетін бірдей емес символға дейін жүргізіледі. Егер қай жолдың қатардың бірдей емес символының информация алмастыру стандартты таблицасындағы нөмірі үлкен болса, сол жолдық қатар үлкен деп есептеледі. Егер жолдық қатарлардың ұзындығы мен барлық символдары сәйкес келсе, олар тең деп есептеледі. АЛ егер жолдық қатардың ұзындықтары әр түрлі болып, ал символдары сәйкес болып келсе, онда үлкені болып ұзындығы үлкен жолдық қатар есептеледі. Жолдық қатарлардың салыстыру амалдарының нәтижесі әрқашан бульдік тип болады. ‘intel’>’INTEL’ {нәтижесі True} ‘Pentium’< ‘PENTIUM’ {нәтижесі False} ‘Duron’ = ‘Duron’ {нәтижесі True} ‘Celeron’<>’Celeron’ {нәтижесі False} ‘Hewlett’<= ‘Hewlett Packard’ {нәтижесі True} ‘Laser’ > = ‘Laserjet’ {нәтижесі False} Таңбалық қатарлар Таңбалық қатарларды баяндау. Таңбалық қатарларды бірнеше әдістер арқылы анықтауға болады. Келесі әдістер негізгі әдістерге жатады:

2. қатарлық константаларды қолдану;
3. таңбалық қатарлардан тұратын массивтерді және char типті массивтерді қолдану. Қатарлық константалар тырнақшаларға алынады. Тырнақшаларға алынған таңбадар және қатарлардың ең соңғы ’/0’ таңбаы жадының тізбектелген ұяшықтарында жазылады. Компилятор қатарды жадыға орналастырғанда жадының қажетті өлшемін анықтау үшін таңбадардың санын есептейді. қатарлық константаларды #define директивасының көмегімен анықтауға болады. Егер қатарда тырнақша таңбаын қолдану керек болса, онда бұл таңбадың алдына кері бөлшек сызығы жазылады. Мысалы: printf (“\” Сведения о сессии”\n”); қатарлық константа осы жол жазылған жадыдағы орынға сілтейтін көрсеткіш болып табылады. таңбалық қатарлардың массивін анықтағанда компилятор жадыдының қажетті өлшемін анықтау үшін массивті баяндағанда қатарлық константа арқылы инициалдауға болады. (Статикалық және сыртқы массивтер қолданылады). char c[ ]=”Определение максимального балла”; (сыртқы массив) Әдеттегі массивтерді қолданған жағдайдағы сияқты бұл массивтің аты c, осы массивтің 1-ші элементіне сілтейтін көрсеткіш болып табылады. c==&c[0]; *c==’0’, және *(c+1)==c[1]==’n’; қатарларды анықтау үшін көрсеткіштерді қолданамыз. Мысалы: char *c1=”\n ввод баллов”; Осы баяндалуға келесі баяндалу эквивалентті: static char c1[ ]=”\n ввод баллов”; қарастырылған қатардың екі баяндауы c1 қатардың көрсеткіші екенін көрсетеді. Жадының қажетті өлшемін айқын көрсетуге де болады. Сыртқы баяндауда келесі қатарды мына түрде жазуға болады: char c[35]=”определение максимального балла”; вместо char c[ ]=”определение максимального балла”; Таңбадарды өңдеу библиотекасы. таңбадарды өңдеу библиотекасы таңбалық мәліметтермен бір қатар пайдалы тексерістер мен операцияларды орындайтын бірнеше функциялардан тұрады. Әрбір функция аргумент ретінде int типін немесе EOF (файл соңы) индикаторын ұсынатын таңбады қабылдайды. таңбадарды өңдеу библиотекасының функцияларымен жұмыс істеу үшін <ctype.h> тақырыптық файлды қосамыз. Кестеде таңбадарды өңдеу библиотекасының функциялар тізімі келтірілген. Элементтердің саны қатардың ұзындығынан бір таңбаға артық болуы керек (нөль-таңбаын есептегенде). Басқа статикалық немесе сыртқы массивтердегідей кез келген қолданылған элементтер автоматты түрде нөлмен инициалданады (таңбалық түрде бұл нөль санының таңбаыны емес, нөль-таңбаы болып табылады).

Кесте 1.1. Таңбадарды өңдеу библиотекасының функциялары

Таңбалық қатарлардан тұратын массивтер. Әрбір жолы таңбалық массив болып табылатын таңбалық қатарлардан тұратын массивтерді қарастырайық. Статикалық массивтің баяндалуын келесі түрде келтірейік:

```
static char *m[4]={“регистр”,”ячейка”,”указатель”,”элемент”};
```

*m[4] массиві таңбалық қатарларға сілтейтін 4 көрсеткіштен тұрады. Сонымен, таңбалық қатарлар массивтер болып табылатын болса, онда осы массивтерге сілтейтін 4 көрсеткіш қарастырылады. 1-ші жолға сілтейтін 1-ші көрсеткіш болып m[0]-ші табылады. 2-ші жолға сілтейтін 2-ші көрсеткіш m[1]. Сонымен, әрбір көрсеткіш сәйкес қатардың ең бірінші таңбаына сілтейді. *m[0]==’р’; *m[1]==’я’; *m[2]==’у’; *m[3]==’э’; инициалдау массивтерге арналған ережелер бойынша орындалады. Тырнақшаларға жазылатын текстер жақшалы жазбаларға эквивалентті:

```
{{…},{…},…,{…}};
```

қатарлардан құрылған массивтерді баяндағанда таңбалық қатарлардың өлшемін көрсетуге де болады және бұл баяндалуда қатарлардың ұзындығы бірдей болады: static char m[4][10]; статикалық (сыртқы) массивтің қатарларының қолданылмаған (артық) элементтері ’/0’ (нөль-таңбаымен) таңбаымен инициалданады. Артық элементтер (яғни, жады тиімді жұмсалу үшін) болмас үшін келесі баяндалуларды қолдануға болады:

```
static char *[4];
```

Мұнда әрбір қатардың ұзындығы массивтің сәйкес қатарын инициалдайтын нақтылы қатармен анықталады.

1. Қатарларды түрлендіру Қатарларды түрлендіру функциялары. қатарларды түрлендіру функцияларымен жұмыс істеу үшін <stdlib..h> тақырыптық файлды қосу қажет. Бұл функциялар сандар қатарларын бүтін мәндерге және жылжымалы нүктелі мәндерге түрлендіреді. Кестеде қатарларды түрлендіру функциялар тізімі келтірілген. Кесте 1.2. Қатарларды түрлендіру функциялары

Стандартты енгізу/шығару библиотекасының функциялары. Стандартты енгізу/шығару библиотекасы <stdlib.h> таңбалық және қатарлық мәліметтермен жұмыс істеуге арналған бірнеше функциялардан тұрады. Кестеде стандартты енгізу/шығару библиотекасынан таңбадар мен қатарларды енгізу/шығару функцияларының тізімі келтірілген. Кесте 1.3. Стандартты енгізу/шығару библиотекасының таңбалық мен қатарларлық функциялары

Қатарларды өңдеу библиотекасындағы қатарларға қолданылатын функциялар.Қатарларды өңдеу библиотекасы қатарларды анықтау және қатарларды лексемаларға бөлу үшін,таңбадарды және басқа қатарларды іздеу үшін, қатарларды салыстыру үшін, қатарлық мәліметтерге қолданылатын операцияларды орындау үшін көптеген пайдалы функцияларды ұсынады. Қатарларды өңдеу библиотекасымен жұмыс істеу үшін <string.h> тақырыптық файлын қосу қажет. Кестеде бұл функциялардың тізімі келтірілген. Кесте 1.4. Қатарларды өңдеу библиотекасындағы функциялар

Қатарларды өңдеу библиотекасындағы қатарларды салыстыру функциялары. Кестеде функциялардың прототиптері мен әрқайсысының қысқаша баяндалуы келтірілген. Кесте 1.5. Салыстыру функциялары

Қатарларды өңдеу библиотекасындағы іздеу функциялары. Кестеде функциялардың прототиптері мен әрқайсысының қысқаша баяндалуы келтірілген. Кесте 1.6. Іздеу функциялары

Қатарларды өңдеу библиотекасындағы басқа функциялары Кестеде қатарды өңдеу библиотекасындағы басқа функиялардың прототиптері мен қысқаша баяндалуы келтірілген.

Бақылау сұрақтары

- Кеңдікке іздеу әдісімен ерікті ағашты табу.
- Тереңдікке іздеу әдісімен ерікті ағашты табу.
- Графтағы циклдердің фундаменталды жиынтығы.

№ 14 тақырып Динамикалық программалау. Динамикалық программалауды үйрену.

Жоспары:

1. Динамикалық программа

1. Динамикалық программалау Динамикалық бағдарламалау - көп сатылы процестерге арнайы бейімделген оңтайлы басқаруды іздеудің математикалық әдісі. Мұндай процестің мысалын қарастырайық Динамикалық құрылымдардың негізгі қасиеттері Статикалық құрылым мен жартылайстатикалықтан айырмашылығы динамикалық құрылымдар келесі қасиеттерге ие:
2. тұрақсыздығы және оның өңдеу кезіндегі (элемент санының) күтілмеген өлшемі. Динамикалық құрылымның өлшемі 0-ден бастап қандай да бір міндеттің спецификасымен немесе машинаның қатынай алатын жадысының көлемімен анықталатын мәнге дейін өзгере алады.
3. жадыдағы құрылымның элементтерінің физикалық сыбайлылығының болмауы. Элементтердің логикалық тізбегі элементтердің өзінде сақталатын бір немесе бірнеше көрсеткіштердің (байланыстардың) көмегімен нақты түрде беріледі. Екінші қасиетпен байланысты, динамикалық құрылымды алатын жады үздіксіз облысты көрсетпейді. Динамикалық құрылымның элементтері ретсіз түрде шашылып тасталған. Бұның салдары статикалық және жартылайстатикалықпен салыстырғанда динамикалық құрылымның элементтеріне қатынаудың қиындай түсуі. Басқа жағынан, бұндай схема иілмелі және көбінесе жадыдағы жеңісті қамтамасыз етеді. Динамикалық құырылымдар әдетте алдында айтылған байланысты тізімдер ретінде көрсетіледі. Сондықтан оларды көбінесе тізімдік құрылымдар деп атайды. Бұндай тізімде әрбір элемент тізімнің келесі элементінің адресі немесе нөмірі көрсетілетін қосымша өріспен қамтамасыз етіледі

Сурет 4. Байланысты тізім

Байланысты тізім – элементтері ретінде элементтердің өзінде сақталатын көрсеткіш көмегімен бір-бірімен логикалық байланысқан бірдей форматты жазбалар болатын құрылым.

```
Қарапайым байланысты тізімдер – бірбайланысты (бір көрсеткіш), екібайланысты (екі).
```

Негізінде көрсеткіштер бұдан да көп болуы мүмкін. Бірбайланысты тізімде әрбір элемент екі өрістен тұрады: мазмұнды және көрсеткіш өрісі.

```
Мазмұнды өрісте мәліметтер сақталады (жалпы жағдайда бұл жазба).
```

Көрсеткіш өрісі тізімнің келесі элементінің адресін сақтайды. Көрсеткіш бойынша келесі элементке қатынау мүмкіндігін аламыз, ал одан кезекті элементке және т.с.с. Соңғы элементтің көрсеткіш өрісінде нөлдік немесе тізімнің соңын дәлелдейтін бос көрсеткіштің арнайы белгісі болуы керек (біздің мысалда 0). Сонымен қатар, тізім басының көрсеткіші болу керек. Ол оның логикалық құрылымының көрсеткіші болуы керек. ЭЕМ жадысында ақпаратты сақтаудың екі қарастырылған формаларын салыстыра отырып, мынаны ескеруге болады:

- байланысты тізімдер байланыстарды сақтау үшін қосымша жадыны талап етеді. Кейбір кезде бұл маңызды. Бірақ та көбінесе элементте байланыс өрісіне арналған бос орын бар болады, егер барлық слотты тұтастай алмаса. Сонымен қатар, бірнеше элементтерді қолдануда бір түйінге біріктіруге болады, сонда оларға тек қана бір байланыс қажет болады. Жадыдағы неғұрлым маңызды анық емес ұтыс кестенің жалпы бөліктерінің үйлесімділігіне мүмкіндік беретін байланысты тізімдерді қолдану кезінде пайда болады;
- байланысты тізімнің ішіндегі элементті оңай алып тастауға болады. Бұл үшін тек қана байланысты өзгерту керек (яғни, көрсеткіш мәнін);
- байланысты тізімнің кез келген жеріне элемент қосуға болады. Бұның бәрі тек байланыстың өзгеруіне ғана әкеледі. (тізбекті тізімдер үшін тізімнің бір бөлігін көшіру керек);
- байланысты тізімдегі k-ші элементке тек қана тізімнің басынан және алдындағы элементтерді көріп шыққан кейін ғана қатынау мүмкін. (бұл – олардың шашылып жатқаны үшін төлем);
- екі байланысты тізбектерді біріктіру немесе бір тізімді бірнеше бөліктерге бөлу оңай жүзеге асырылады;
- байланысты тізбімдердің схемасы тізбекті тізімдерге қарағанда күрделірек құрылымдарды бейнелеуге мүмкіндік береді. Бірбайланысты тізімнің сызықтығы оның элементтерінің барлығы логикалық түрде сызықты орналасқандығынан шығады, яғни әрбір элемент үшін (біріншісі мен соңғысынан басқа) жалғыз алдыңғы және жалғыз одан кейін жүретін элементтер бар. Жадыда бір тізбектің элементтерінің арасында басқа тізімдердің элементтері де болуы мүмкін. Бірбайланысты тізімнің дескрипторы ерекше жазба түрінде жүзеге асырылуы мүмкін және мынадай ақпараттарға ие:
- тізімнің коды;
- тізімнің аты;
- тізбект басының адресі (көрсеткіш);
- тізімдегі элементтердің ағымдағы саны;
- элементтің бейнеленуі. Дескриптор тізімнің элементтері орналасатын жадыда бола алады, немесе оған жадыдан басқа орын бөлінеді.

Бірбайланысты тізімнің физикалық құрылымы – дескрипторлар мен жадының кейбір облысында ерікті орналасқан және көрсеткіштер көмегімен сызықты реттелген тізбек бойында бір-бірімен байланысқан өлшемі және форматы бойынша бірдей жазбалар жиынтығы. Мәліметтердің динамикалық структурасы - өлшемдері программаның орындалу барысында өзгеретін (өсетін немесе кемитін) мәліметтер структурасы. Байланысқан тізім - кез келген орнында қоюды және жоюды іске асыруға болатын мәліметтер элементтерінің «тізілген қатар» жиынтығы. Стектер қою жєне жоюдың орындалуына тек қана стектің бір ұшынан - үстінен рұқсат береді. Кезектер деп күту сызығын айтуға болады. Қою кезектің соңында, ал жою-кезектің басында іске асады. Екілік бұтағы тез іздеуді және мәліметтерді сұрыптауды жеңілдетеді, мәліметтер элементтерінің қайталануын тиімді жояды, каталогтардың файлдық структурасын және өрнектерді машиналық тілге компиляциялауды ұсынады. Мәліметтер құрылымы (Структура данных; data structure) — 1) мәліметтерді ұйымдастыру схемасы; мысалы жиым немесе жазба құрылымын сатылы түрде өрнектеу тәсілі; 2) қабылданған тәсілдердің бірі бойынша біріктірілген және реттелген мәліметтер элементтерінің жиыны; мәліметтердің арасындағы физикалық немесе логикалық қатынас. Мәліметтер - ақпараттың құрамдас бөлігі. Тіркелу әдісіне сәйкес мәліметтер әртүрлі тасуыштарда

сақталады және тасымалданады. Ең кең тараған мәліметтерді тасуыш қағаз болып табылады. Заттың оптикалық қасиеттерінің өзгерісі лазерлік сәулелердің көмегімен жазылатын тасуыштар CD-ROM-да қолданылады. Магниттік касиеттердің өзгерісін колданатын тасуыштар ретінде магниттік таспалар мен дискілерді алуға болады. Ақпараттық процесс барысында мәліметтер әдістердің көмегімен бір түрден екінші түрге өзгереді. Мәліметтерді өңдеу көптеген амалдардын жиынтығынан тұрады:

- мәліметтерді қалыптастыру
- мәліметтерді сүзгілеу;
- мәліметтерді сұрыптау;
- мәліметтерді топтастыру;
- мәліметтерді архивтеу;
- мәліметтерді қорғау;
- мәліметтерді тасымалдау;
- мәліметтерді түрлендіру. Жоғарыдағы мәліметтерге қолданылатын амалдар тізімі толық емес. Бүкіл әлемде миллиондаған адамдар мәліметтерді құру, өңдеу, түрлендіру және тасымалдаумен айналысады, әрбір жұмыс үстелінде әлеуметтік, экономикалық, ғылыми және мәдени процестерді басқару үшін қажет арнайы амалдар орындалып отырады. Барлық мүмкін болатын амалдардың толық тізімін құрастыру мүмкін емес және оның қажеті де жоқ. Бұдан ақпаратпен жұмыс істеуге көп уақыт кетеді, сондықтан оны автоматтандыру қажет деген корытындыға келеміз. Динамикалық құрылымдар анықтамасы бойынша жадыда құрылым элменттерінің физикалық шектестігінің болмауымен, оны өңдеу процесі кезінде құрылым өлшемінің тұрақсыздығымен сипатталады. Динамикалық құрылымның элементтері жадының алдын-ала айта алмайтын адрсетер бойынша орналасатындықтан, мұндай құрылымның элементінің адресі бастапқы немесе алдыңғы элементтен алынбайды. Динамикалық құрылымның элементтерінің арасында байланыс орнату үшін көрсеткіштер қолданылады, олар арқылы элементтер арасында анық байланыстар орнатылады. Деректерді жадыда бұлай көрсету байланысты деп аталады. Динамикалық құрылымның элементі екі өрістен тұрады: Ақпараттық өріс немесе деректер өрісі, мұнда құрылым құрылған деректер сақталады; жалпы жағдайда ақпараттық өрістің өзі жинақталған құрылым – вектор, массив, не басқа динамикалық құрылым болып табылады; Байланысу өрісі, берілген элементті құрылымның басқа элементтерімен байланыстыратын бір немесе бірнеше көрсеткіштерден тұрады. Деректерді байланысты көрсетудің артықшылығы – құрылымның едәуір өзгеруін қамтамасыз ету мүмкіндігі; Құрылымның өлшемі машиналық жадының қол жетерлік көлемімен ғана шектеледі; Құрылым элементтерінің логикалық тізбегін өзгерту кезінде жадыда деректерді орналастыру емес, көрсеткіштерді түзету ғана талап етіледі; Құрылымның иілгіштігі. Сонымен бірге, байланысты көрсетудің кемшіліктері де жоқ емес: Байланыс өрістеріне қосымша жады шығындалады; Байланысқан құрылым элементтеріне қатынас уақыт бойынша тиімсіздеу болуы мүмкін. Соңғы кемшілігі едәуір қиындықтар туғызатындықтан, деректерді байланысты көрсетуге соған бола шектеу қойылады. Егер деректерді аралас көрсетуде кез келген элементтің адресін есептеу үшін бізге барлық жағдайда элементтің нөмірі және құрылымның дескрипторындағы ақпарат жеткілікті болса, онда байланысты көрсету үшін элементтердің адресі бастапқы деректерден есептелінбейді. Байланысты құрылымның дескрипторы құрылымға кіруге мүмкіндік беретін бір немесе бірнеше көрсеткіштен тұрады, әрі қарай қажетті элементті іздеу элементтен элементке тізбек бойынша орындалады. Сондықтан байланысты көрсету деректердің логикалық құрылымы элемент нөмірі бойынша қатынас жасайтын вектор немесе массив түріндегі есептерде ешқашан қолданылмайды, бірақ логикалық құрылымы басқа бастапқы ақпараттарды (кестені, тізімді, ағашты және басқа) қажет ететін есептерде жиі қолданылады. Бақылау сұрақтары
- Динамикалық құрылымның элементі қандай өрістерден тұрады?
- Мәліметтердің динамикалық құрылымы деген не?
- Динамикалық бағдарламалаудың қандай түрлері бар?

№ 15 тақырып Технологиялық, техникалық қателер, программалық және жүйелік қателер, құжаттық қателер. Программаларды орындау барысында жүйенің қателер жайында хабарлары. Программалардың орындалуы.

Жоспары:

1. Программалық жабдықтарды жөндеу. Қателердің классифи-кациясы.
2. Программаны дұрыстау.

3 ПРОГРАММАНЫҢ ОРЫНДАЛУЫ.

1. Программалық жабдықтарды жөндеу. Қателердің классифи-кациясы Программалық жабдықты құру кезіндегі маңызды кезеңдердің бірі – программаны жөндеу кезеңі. Программаны жөндеу (Debugging -отладка) кезінде, программадағы қателер табылып, бөліп алынып жөнделеді. Программаны жөндеу үшін арнайы жөндеуші-программалар (отладчиктер) қолданылады. Программалау жүйелерінде кіріктірілген жөндеуші- программалар болады. Олар программистке программаны бақылап отыру мүмкіндігін береді, яғни қажет болған кезде тоқтату , қайта жүктеу, қадамдап орындау және т.б. сияқты әрекеттерді орындауды ұйымдастырады. Программист өзінің құрған қосымшасы орындалған кезде болуы мүмкін қателерді анықтап, ол қателер бола қалған жағдайда программаның қалай жұмыс жасауы керек екенін алдын-ала қамтамасыз етуі тиіс. Жалпы программалау кезінде жіберілетін қателерді келесі топтарға бөледі: синтаксистік қателер, логикалық қателер және динамикалық қателер. Синтаксистік қателерге программа мәтінін теру кезінде операторлардың қате жазылуы, операторларды айыру белгілерінің қойылмауы, программа соңының көрсетілмеуі және т.б. жатады. Әдеттте синтаксистік қателерді анықтау компилятордың қызметіне жатады, яғни программа синтаксистік қатесі жөнделмейінше компиляциядан өтпейді. Логикалық қателер, есеп алгоритмінің дұрыс құрылмауынан болады. Логикалық қатесі бар программалар түсініксіз жұмыс жасайды, мысалы, цикл алгоритмінде циклдан шығу шарты дұрыс құрылмаған болса, онда программа ешбір тоқтамастан қайталанып, нәтиже бермей жұмыс жасауы мүмкін, сол сияқты, есептеу алгоритмдерінде көбейтіндінің бастапқы мәнін нольге тең деп алғанда нәтижеде үнемі ноль шығуы мүмкін және т.б. . Мұндай қателерді программаны тестілеу, яғни әртүрлі мәндер үшін орындап көру арқылы табады. Динамикалық қателер бұл- программаның орындалуы кезінде пайда, болып оның орындалу тәртібінің бұзылуына немесе нәтижесіз тоқтап қалуына әкеліп соқтыратын қателер. Динамикалық қателерді немесе «орындау уақыты кезіндегі қателер» («ошибка времени выполнения», Runtime errors) деп те атайды. Динамикалық қателерге, мысалы, есептеу кезінде бөлшек бөлімінің нольге тең болуы, түбір астында теріс сан кездесіп қалуы, жады ресурстарының жетпей қалуы, программада көрсетілген маршрут бойынша файлдың табылмай қалуы, принтерде қағаздың бітіп қалуы және т.б. көптеген нәрселер жатады. Қосымшалардағы осындай динамикалық қателерге байланысты болатын жағдайларды «ерекше жағдайлар» деп атап, және олармен жұмыс жасау үшін программалау тілдерінде «ерекше жағдайларды өңдеу» түсінігі енгізілген.

1. Программаны дұрыстау Программа құруда семантикалық (мағыналық), синтаксистік жєне алгоритмдік қателер жиі кездеседі. Программаны компиляциялау кезінде синтаксистік қателерді компилятор µзі табады да, машина жұмысын тоқтатады. Ал, 15 орнына 25 енгізілген сияқты қатені компілятор еске алмай, программаның орындалуы аяқталған кезде қате нєтиже шығады. Программада қате жазылған оператор сияқты алгоритмдік қате де программаның орындалу нєтижесін дұрыс кµрсетпейтіні сµзсіз. Сондықтан программаны құрып болған соң он мұқият қайта тексеріп шығуы керек.

3 ПРОГРАММАНЫҢ ОРЫНДАЛУЫ

Турбо-Паскаль программасы экранға бірнеше терезені бір уақытта ашуға мүмкіндік береді. Орындау кезінде бірінші ағымдағы терезедегі программа орындалады. Ағымды терезенің белгісі – терезе жиегі қосарланған сызықтармен беріледі. Жазылған программаны орындау үшін мәзір жүйесіндегі Run бөліміне ену керек немесе F9 пернесін басу арқылы орындайды. Программаны орындауға жібергеннен кейін жүйе транслятор (интерпретатор) арқылы Паскаль тілінде жазылған программаны машина тілінде жазылған кодтарға ауыстырады да, программадан синтаксистік қателерді іздейді. Егер программа ішінде қате болса, онда программа орындалмайды, қайтадан программаны түзетуге ашады. Программаның жоғарғы жағында қызыл жолмен жазылған хабарлама шығады. Esc пернесін басқан соң ол жол кетеді де, курсор қатесі бар жолда орналасады. Қате туралы мәлімет алу үшін Ctrl+F1 пернелерін бірге басады. Қателер түзетілгеннен кейін ғана программа орындауды бастайды. Айнымалыларға мән беру үшін Ctrl+F9 пернесін басу керек. Нәтижесін көру үшін Alt+F5 пернелерін бірге басу керек. Қайтадан редактірлеу терезесіне шығу кез-келген пернені басу арқылы орындалад Бақылау сұрақтары

1. Паскаль программалау тілінде қателердің қандай классификациясы бар?
2. Бағдарламалық қателерді дұрыстау неден басталады?
3. Қателіктерді түзетуде компилятордың көмегі қандай?

Ұсынылған әдебиеттер Негізгі әдебиеттер

1 МАҚАШЕВ, Е.П. АЛГОРИТМДЕР ЖӘНЕ ПРОГРАММАЛАУ [МӘТІН] = ОҚУ ҚҰРАЛЫ / Е.П. МАҚАШЕВ - АЛМАТЫ, 2015.- 2 ЭКЗ

- Смайлова, Ұ.М. Программалау: Алгоритм құру технологиялары [Мәтін] = Оқу құралы / Ұ.М.Смайлова - Алматы, 2011.- 2 экз
- Васильев А.Н. Python на примерах. Практический курс ·по программированию. - СПб.: Наука и Техника, 2016. - 432 с.

Қосымша әдебиеттер

1. Поляков К.Ю., Еремин Е.А.. Информатика, 10-11 класс, учеб. пособие, изд.БИНОМ, 2015 г.
2. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений.--:- СПб.: БХВ-Петербург, 2016.-704 с.
3. Шапошникова С. Лаборатория юного линуксоида http://younglinux.info, 2011
4. 050704- Есептеу техникасы және бағдарламамен қамтамасыз ету [Мәтін] = Алгоритмдер теориясы., Сараптау жіне интелектуальды жүйелер. Интернет технологиялары. Комьпютерлік желілер.: Оқу-әдістемелік құралы / Құрас. аға оқытушы Шалбаева Ш., Алтынбеков Ш.Е., Абдрахманова М.Б, п.ғ.к. Жайдақбаева Л.Қ., ж.т.б.- Шымкент: ХГТУ, 2012.- 228 б.-40экз
5. Манакова, И.П. М23. Информационные системы и технологии. Языки программирования высокого уровня. Программирование на языке Python [Электронный ресурс] : учеб.-метод. пособие / И.П. Манакова; М-во образования и науки РФ; ФГАОУ ВО «УрФУ им. первого Президента России Б.Н. Ельцина», Нижнетагил. технол. ин-т (филиал). – Нижний Тагил : НТИ (филиал) УрФУ, 2016. – 37 с. Мультимедиялық қамтамасыз ету және электрондық ресурстар https://sirdariya.kz/kk/node/316 https://stud.kz/referat/show/69095 https://bilim-all.kz/jospar/5349 https://infourok.ru/saba-zhospari-algoritm-negizderi-algoritmdeu-programmalau-zholdari-884255.html