Әдістемелік нұсқауды
бекіту парағы

Нысан
ПМУ ҰС Н 7.18.1/05

БЕКІТЕМІН
ФМжАТФ деканы
_____ Ж.Қ. Нұрбекова
(қолы)
2010ж..
«___»_____

Құрастырушы: аға оқытушы _____ Ж.Б.Исабеков.
(қолы)

Есептеу техникасы және бағдарламалау кафедрасы

«Бағдарламаны әзірлеудің құрал-саймандары» пәні бойынша
050704 «Есептеу техникасы және бағдарламалық қамтама»
Мамандығының студенттеріне арналған
зертханалық сабақтар үшін
әдістемелік нұсқау

Кафедра отырысында **ұсынылған**
2010 ж. «___»_____, №__хаттама

Кафедра меңгерушісі _____ О.Г. Потапенко
(қолы)

ӘК **құпталған** «ФМжАТФ» факультеті
2010 ж. «___»_____, №__хаттама

ӘК төрағасы _____ А.Т. Кишубаева
(қолы)

№ 1 зертханалық жұмыс
Тақырыбы: «Plug-in модулдерінің жазылуы»

Соңғы уақытта көптеген бағдарламаларда қосымша модульдермен жұмыс жасау мүмкіндігі көбейіп кетті (plug-ins). Бұл тек сәнге табыну емес, бұл өз бағдарламаларын тәуелсіз әзірлеушілер есебінен қосымша мүмкіндіктермен толықтыру мүмкіндігі. Бір бағдарламашы немесе толық компания бәрін дербес жасай алмайды. Сондықтан да олар тәуелсіз әзірлеушілерге өз өнімдерін өз күштерімен жақсарту мүмкіндігін ұсынады.

Барлық бөлім бойы мен осы модульдерді салынбалы, қосымша немесе қосылатын модульдер деп атаймын, ал plug-in модульді ғана түсінетін боламын. plug-ins модульдері көбінесе серпінді кітапханалар түрінде орындалады, олар негізгі бағдарламаға қосыла алады. Сіздің жобаңыз жегілу барысында осындай файлдарды іздеу барысында белгілі орынды тексеретін болады. Егер осындайлар табылса, белгілі мәзірде модуль функцияларын пайдаланатын командалар пайда болады.

Plug-in жұмыс жасау үшін бағдарламалар құру

Меншікті plug-in модулі мен оған арналған тестілік бағдарлама құрып көрейік, тәжірибеде бәрін өз қолымызбен жасай алатын боламыз. Негізгі бағдарлама жазудан бастайық. Бұл үшін жай қосымшаның жаңа жобасын құрыңыз (Application). Басты нысанға біз MainMenu және Memo1 компоненттерін орналастырамыз. Мәзірде біз екі басты тармақ құрамыз: Файл мен Plug-ins. Бірінші тармақта Шығу кіші мәзірін құра аламыз, ал екіншісін бос қалдырамыз, сондықтан ол қосымша модульден тұратын модульдермен толықтырылатын болады.

Компонент Memo1 барлық нысан бойынша созуға болады.

type бөлімінің модулінде мына объектіні жариялау керек:

Plug-in модулін тестілеу

Бұл біздің бағдарлама қосылатын модульмен өзара әрекет ететін объект. Осы объектінің үш әдісі бар.

- GetAppiication — осы әдіс арқылы модуль бас бағдарлама қосымшасының объектісіне сілтеме ала алады. Өз мысалымда мен мұны келтірмеймін, алайда қосымша модульге сілтемені бас қосымшаға қалай тапсыру керектігін көрсетуді шештім. Бұл мүмкіндік Plug-ins модулі үшін керек болады;
- AddMenuitem — бас терезедегі осы әдіс қосылатын модульдің мүмкіндіктерін пайдалануға арналған мәзір тармағын құратын болады;
- Additem — осы әдіс арқылы қосымша модуль ақпаратты бас терезеге шығаратын болады, яғни Memol компонентіне.

Енді барлық осы әдістердің біздің қосымшада іске асырылуына қарайық. Бірінші кезекте GetAppiication:

Мұнда біз функциялар орындалу нәтижесіне бас қосымшаға сілтеме береміз. Осындай жай әдіспен еншілес модуль өзіне қажетінің барлығын біле алады.

Келесі әдіс — AddMenuItem:

Мұнда біз Plug-ins бас терезе мәзірінде тармақша құрамыз. Мәзір тармағының атауы ретінде бірінші тапсырылған параметр пайдаланылады. Onclick оқиғасын өңдеушісі ретінде құрылған тармақ үшін екінші параметрде тапсырылған рәсім көрсетіледі. Осылай еншілес модуль бас терезеде мәзір құрайды және оған өзінің оқиға өңдеушісін тағайындайды.

Және ең соңғы әдіс — AddItem :

Бұл жерде модульден Memo1 компонентіне тапсырылатын ақпарат шығарыладыБұл функцияны біздің әдіс қосымша модуль орындай алатын қандай да бір есептер нәтижелерін экранға шығару үшін пайдалана алады.

Барлық үш функциялар мен барлық сыныптың қосылатын модульдің бас бағдарламамен өзара қатынасын қамтамасыз ету үшін арналғанын естеріңізге саламын.

Енді implementation басты сөзінен кейін мынаны жариялаймыз:

Type бөлімінде ауыспалы TInitPlugin рәсім ретінде жарияланған. Осылай plug-in модулінен шығатын рәсім атауы мен параметрлері сипатталады. Ол осы бағдарламадан шақырылатын болады, ал енгізілетін модуль өз деректерін жазатын болады, мысалы, мәзірдің қажетті тармақтарын бас терезеге қосады.

const бөлімінде initpiugin мәні бар жол константы жарияланады. Бұл енгізілетін модульдің бастамашыл рәсімінің атауы.

Бірінші ауыспалы модульді көрсетеді және оған сілтемені сақтайтын болады. Бұл ауыспалы бағдарлама орындаудың барлық уақытында қол жетімді болады. Екінші ауыспалы дегеніміз - тізім, оған біз PInterface жегілген модулдеріне арналған барлық көрсеткіштерді енгізетін боламыз, бұл барлық сілтемелердің толық және бұзылмай сақталуына ықпал ететін болады.

Қосымша модулдермен жұмыс жасау үшін бізде бәрі дайын. Енді бағдарламаны іске қосу барысында біз іздеуді dll-файлдардың бар-жоғына арналған белгілі директорияға жібереміз. Бұл үшін onCreate оқиғалар өңдеушісіне мына кодты жазамыз:

Ең басында il тізімі келтіріледі. Одан кейін ауыспалы plugDir директорияны жібереміз, онда қосымша модульдерді іздеу керек болады.

Осы дайындықтардан кейін FindFirst функциясының көмегімен көрсетілген директориядан файлдарды іздеу іске қосылады. Егер файл табылса, LoadLibrary функциясының көмегімен табылған кітапхананы жадыға жегеміз. Келесі кезенде жегілген кітапханадан GetProcAddress функциясы арқылы іске қосу рәсімін іздейміз. Осы функцияның екі параметрі бар:

- жегілген кітапханаға арналған көрсеткіш;
- екінші параметр – табу керек рәсімнің аты. Мұнда біз GetProcAddress константын көрсетеміз, онда ' InitPlugin 1 атауы сақталады.

Егер GetProcAddress функциясын орындау нәтижесі нөлге тең болмаса, іске қосу рәсімі табылған болып есептеледі және оны орындау керек. Бұл үшін ең алдымен TiPiuginDemo үлгісіндегі объектіні құрамыз, бұл — интерфейс, оны біз қосымша модулдің бас бағдарламамен өзара қызмет етуі үшін құрдық. Содан кейін кітапханадан InitPlugin рәсімін шақырамыз, оған өзара әрекет ету үшін құрылған интерфейс көрсеткішін тапсырамыз. Сонымен қатар ауыспалы Pinterface іl тізіміне кейіннен пайдалану үшін қосамыз.

Егер іске қосу функциясы табылмаса, кітапхананы қайта жегеміз, өйткені ол біздің бағдарлама үшін plug-in-модулі болып табылмайтын кез-келген dll-файл бола алады. Бұл API-функциялар FreeLibrary көмегімен істелінеді..

onDestroy оқиғасы болған жағдайда біз мына тізімді жоюымыз керек:

procedure TPluginDemoForm.FormDestroy(Sender: TObject) ;

Бұл бас қосымша құру үшін plug-in модулдермен жұмыс жасай алатын негізгі іс-әрекет. Егер сізге бас бағдарлама мен қосымша модуль арасындағы бұдан да күрделірек ара-қатынастар керек болса, TiPiuginDemo объектісінің мүмкіндіктерін кеңейту қажет.

Plug-in модулдерін құру

Негізгі бағдарлама дайын. Енді оған қосылатын plug-in модулдің жазылуына кірісейік. Бұл үшін dll-файлдың жаңа жобасын құру керек. Delphiден File мәзірінен New кіші мәзірін ашыңыз, содан кейін Other тармағын таңдаңыз. Жаңа жоба құрудың пайда болған терезесінен DLL Wizard элементін таңдап, ОК кнопкасын басыңыз.

Құрылған модулдің кодын мына түрге келтіріңіз:

Осындағы ең негізгісі - mitpiugin рәсімі . Бұл қосымша модулдің іске қосылуы үшін шақырылатын рәсімнің өзі. Бірінші жолда ауспалы Demointerface модульдер арасындағы өзара қатынас интерфейсіне арналған көрсеткіш сақталады. Осы ауыспалыны біз әлі сипаттаған жоқпыз, бірақ оны жақын уақытта іске асырамыз. Келесі жолда Formi нысаны іске қосылады, оны да біз қазір құрамыз. Содан кйін негізгі бағдарлама мәзірінда AddMenuitem әдісінің PlugClass объектісін шақыру арқылы мәзірдің үш тармағы құрылады. Осы тармақтарға алдағы уақытта жазылатын олардың атаулары мен функция-өңдеушілері тапсырылады.

Сол сияқты біз рәсім құруымыз үшін оны сырткы бағдарламалар көруі үшін экспортталатын деп жариялауымыз керек. Бұл үшін рәсім export бөлімінде сипатталуы тиіс. Мұны міндетті түрде жасауымыз керек!!!

Енді біздің кітапханамызға жаңа нысанды қосамыз. Бұл үшін File мәзірінде New кіші мәзірін ашамыз, сосын Form тармағын таңдаймыз. Жаңа нысаннға үш кнопка орналастырамыз:

- диалогтық терезе көрсету;
- элемент қосу;
- терезені жасыру.

Енді plug-in модуль мен бас бағдарлама арасында өзара қарым-қатынас іске асырылатын интерфейстің сырткы түрін сипаттайық. Осы мақсатта type бөлімінде мына сөздерді жазыңыз:

Осы хабарландыру біз негізгі бағдарламада жазғанымызға ұқсас. Бір айырмашылығы әдістің әрбір атынан кейін abstract сөзі тұр. Бұл сөз әдістің осы модулде сипатталмайтынын және абстрактілі екендігін білдіреді. Себебі барлық объектіні іске асыру негізгі бағдарламада, ал мұнда біз интерфейс мүмкіндіктерімен пайдалануға болатын сипаттаманы ғана жасаймыз.

Var бөлімінде ауыспалы Demointerface жариялау керек, онда модулдер арасындағы өзара әрекет ету объектісінің көрсеткіші сақталатын болады. Осы ауыспалыны біз mitpiugin рәсімінде пайдаландық, ал мұнда оның сипаттамасын қосамыз:

Осында "Привет!" деген жазуы бар хабарламаның стандарттық терезесін көрсетеміз.

Элемент қосу кнопкасы үшін мына кодты енгіземіз:

Интерфейстің Additem әдісін біз жаңа жолдың модулінен бас бағдарламаның

Memo компонентіне қосу үшін қолданамыз.

Оқиғалар өңдеушісінде Терезені жасыру кнопкасын басу арқылы былай жазыңыз:

close емес, Hide әдісінің шақырылатынына назар аударыңыз. Осы әдістің көмегімен терезе жабылады, бірақ жойылмайды. Егер біз close деп жазсақ, терезені бірінші рет жапқанның өзінде ол жойылушы еді, ал келесі жолы біз оны бас терезеден көрсете алмаушы едік. Бірінші рет талпынғанда жадыға қол жеткізу қатесі пайда болды.

Сонымен қатар public бөлімінде біз пайдаланатын тағы екі әдісті сипаттауымыз керек:

Жинақталған dll-файлды негізгі бағдарламаның Plugins кіші директориясына орналастыруымыз керек. Бұдан кейін негізгі бағдарламаны іске қосып және Plug-ins мәзірін таңдап, біз қосымша модуль ұсынатын мүмкіндіктері бар мәзірдің тармақтарын көруіңіз тиіс.

Мәзірдің түрлі тармақтарын шақыру арқылы осы бағдарламамен жұмыс жасап көріңіз. Сіз негізгі бағдарламаның қалай жұмыс істейтіні мен оның plug-in модулімен қалай өзара әрекет ететінін түсінуіңіз керек..

№ 2 зертханалық жұмыс

Тақырыбы: «Мұрағаттандырудың негізгі қағидалары. Әдістер сыныптамасы»

Келесі үлкен тақырып деректерді мұрағаттандыру болып табылады. Өздеріңізге белгілі бүгінгі деректер жазу форматтарының басым бөлігі пайдаланушылардың оңай оқуы, тез әрекет ету үшін ыңғайлы түрде сақталады. Алайда деректер оларды сақтау үшін нақты талап етілетін көлемнен көбірек орын алады. Деректер жазбасының артықтылығын жоятын алгоритмдер деректерді қысу алгоритмдері немесе мұрағаттандыру алгоритмдері деп аталады. Қазіргі уақытта бірнеше негізгі тәсілдерге негізделген деректер қысуға арналған көптеген бағдарламалар бар.

Криптографияға мұрағаттандыру неліктен керек? Себебі қазіргі криптоталдауда, яғни криптографиялар қайшылықтары туралы ғылымда деректерді белгілеріне қарай қысу алгоритмдерінің міндеттері артықшылықты жою болып табылады.

Барлық деректерді қысу алгоритмдері былай бөлінеді: 1) шығынсыз қысу алгоритмдері, оларды пайдалану барысында қабылдаудағы деректер ешбір өзгеріссіз қалпына келтіріледі және 2) шығынды қысу алгоритмдері, олар деректер ағымынан деректер желісіне сәл ғана әсер ететін немесе адаммен мүлдем қабылданбайтын ақпаратты жояды (осындай алгоритмдер қазір тек аудио, - бейне көріністер үшін ғана әзірленеді). Криптожүйелерде, әрине, алгоритмдердің тек бірінші тобы гана пайдаланылады.

Шығынсыз мұрағаттандырудың екі негізгі әдістері бар:

– Хаффман алгоритмі (ағылш. Huffman), өзара байланыспайтын байттар бірізділігін қысуға бағытталады;

– Лемпель-Зив алгоритмі (ағылш. Lempel, Ziv), сөздердің бірнеше рет қайталану – яғни байттардың бірізділігі фактілерін пайдаланатын мәтіндердің кез келген түрін қысуға бағытталған.

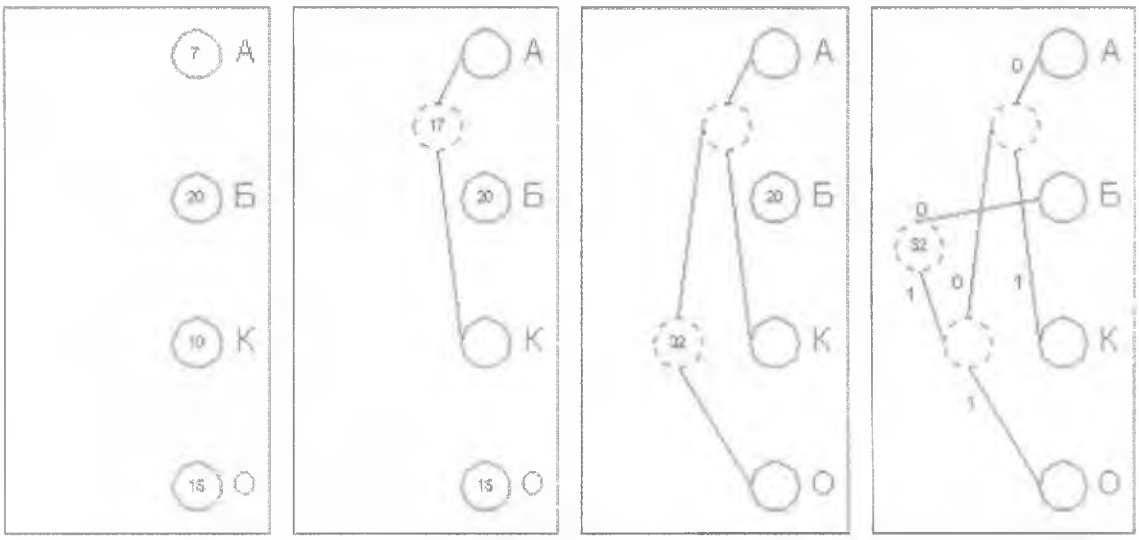
Шығынсыз мұрағаттандырудың барлық белгілі бағдарламалары (ARJ, RAR, ZIP және т.б.) осы екі әдістің бірігуін - LZH алгоритмін пайдаланады.

Хаффман алгоритмі

Алгоритм ерікті мәтінде стандарттық 256-символдық теру барысында кейбір символдардың қайталанудың орта кезеңінен жиі кездесетіндігін, ал кейбіреуінің сирегірек екендігіне негізделген. Тиісінше таралған символдардың жазбасына ұзындығы 8-ден аз бит бірізділігін пайдаланатын болсақ, сирек сиволдар жазбасына – ұзындарды пайдаланамыз, яғни сомалық файл көлемі азая түседі.

Хаффман энтропиясына өте жақын ұзындығы бар файл алу үшін қандай символды қандай кодпен жасыру керектігі жөнінде өте жеңіл белгілеу алгоритмін ұсынды. Бізде мәтінде кездесетін барлық символдар тізімі бар деп есептейік, ондағы әрбір символдың пайда болу саны белгілі дейік. Оларды көлденеңінен парақтың оң жағына ұяшықтар түрінде бір қатарға жазып алайық (1а-сурет). Мәтінде ең аз кездесетін екі символды тандап алайық.. Олардан солға қарай бағанның биігіне желі сызайық және

оған біріктіретін символдардың әрқайсысының жиілік сомасына тең мәнді жазып қоялық (2б-сурет). Осыдан бері қайталанудың ең төмен жиілігін іздеу барысында екі біріктірілген түйінді назарға алмайтын болайық, бірақ жаңа биіктікті екі біріккен биіктіктің пайда болу жиілігі сомасына тең пайда болу жиілігі бар толық ұяшық ретінде қарастыратын боламыз. Биіктіктерді біріктіру операциясын саны бар бір биіктікке келгенге дейін қайталаймыз (2в және 2г суреті). Тексеру үшін: онда кодталатын файлдың ұзындығы жазылатыны белгілі. Енді әрбір биіктіктен шығатын бағанның екі қабырғасына 0 және 1 биттерін орналастырамыз. Әрбір нақты әріптің кодын белгілеу үшін ағаш төбесінен оған дейін жүріп өту керек, жүру сызбасына қарай нөлдер мен бірліктер жазып алынып отырады. 4.5-суреті үшін "А" символы "000" кодын алады, "Б" символы – код "001", "О" символына "1" коды.



1 сурет Хаффман алгоритм

Ақпаратты код беру теориясында көрсетілгендей Хаффман коды префиксті болып табылады, яғни ешқандай символдың коды қандай да бір басқа символдың басы болып табылады. Бұны біздің мысалымызда тексеріңіз. Онда Хаффман коды әрбір тапсырылған символдың ұзындығы хабарланбаса да алушының қалпына келтіруге болатыны белгілі. Алушыға Хаффманның ағашы ыңғайлы түрде жіберіледі, содан соң символдар кодының кіріс бірізділігі ешқандай қосымша ақпаратсыз дербес іске асырылады. Мысалы "0100010100001" қабылдау барысында бірінші "Б" символы бөлектелінеді: "01-00010100001", содан соң ағаш биіктігінен бастап – "А" "01-000-10100001", одан кейін барлық жазба осылай қайта кодталынады "01-000-1-01-000-01" "БАОБАБ".

Лемпель-Зив алгоритмі

Лемпель-Зивтің – LZ77, өзінің жарияланған жылы бойынша осылай аталған классикалық алгоритмі барынша жеңіл. Ол былай түсіндіріледі: "егер бұдан бұрын өткен шығу ағымында байттардың осындай бірізділігі

байқалса, оның ұзындығы мен ағымдағы позициядан ауытқуы туралы жазба осы бірізділіктен қысқарақ болғанда, шығу файлына сілтеме жазылады. Осылай "КОЛОКОЛ_ОКОЛО_КОЛОКОЛЬНИ" фразасы осылайша белгіленеді: "КОЛО(-4,3)_(-5,4)О_(-14,7)ЬНИ".

RLE кең тараған қысу әдісі (ағылш. Run Length Encoding), бір символдың бірдей символдары бірізділігін жазу орнына осы алгоритмнің кіші сыныбы болып табылады. Мәселен, "АААААА" бірізділігін қарастырайық. RLE алгоритмінің көмегімен ол"(А,7)" ретінде белгіленетін болады, сонымен қатар оны LZ77: "А(-1,6)" алгоритмінің көмегімен де барынша жақсы қысуға болады. Расында да осындай бірізділікті қысу деңгейі төмен (шамамен 30-40%), бірақ LZ77 алгоритмі өз бетінше қолайлырақ және RLE мүлдем қысылмайтын әдіспен бірізділікті одан да жақсы өңдей алады..